Logica in Informatica

Nicolò Felicioni[1]

Dipartimento di Elettronica e Informazione
Politecnico di Milano

nicolo . felicioni @ polimi . it

April 13, 2021

[1] Materiale sulla logica del primo ordine fornito principalmente da Achille Frigeri, materiale sulla logica monadica fornito da Nicholas Mainardi.

Logica in Informatica - Definizioni

Cosa significa definire un predicato P di arità n?

Significa trovare una formula vera del tipo

$$\forall x_1 \ldots \forall x_n \ (P(x_1, \ldots, x_n) \Leftrightarrow \mathcal{F})$$

dove $\mathcal{F}$ è una formula chiusa che contiene solo predicati e funzioni la cui definizione è già nota o in cui compare P ma con argomenti "più semplici" di quelli che compaiono nella parte sinistra (in tal caso si parla di definizione ricorsiva).

Se P ha arità 1, cioè rappresenta l'appartenenza ad un insieme, possiamo scrivere $x \in P$ al posto del formalmente corretto $P(x)$.

Logica in Informatica - Descrizione di linguaggi

Specificare in logica del primo ordine il linguaggio $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$. Nella specifica si possono usare esclusivamente i seguenti predicati e funzioni:

- il predicato di uguaglianza "$=$"
- la funzione di concatenazione "$\cdot$"
- i caratteri alfabetici a, b, c e la stringa vuota ε come costanti
- il predicato "$L(x)$" per indicare $x \in \mathcal{L}$
- eventuali altri predicati definibili esclusivamente usando i simboli sopra introdotti

quindi **non si può** utilizzare la funzione di elevamento a potenza.

Logica in Informatica - Descrizione di linguaggi

Specificare in logica del primo ordine il linguaggio $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$.

Idea: definiamo dei predicati (cioè linguaggi) più semplici da usare per definire $\mathcal{L}$, potrebbero essere comodi:

- $L_a(x)$ per indicare $x \in a^*$
- $L_b(x)$ per indicare $x \in b^*$
- $L_c(x)$ per indicare $x \in c^*$
- $L_1(x)$ per indicare $x \in \{a^n b^n \mid n \geq 0\}$
- $L_2(x)$ per indicare $x \in \{b^n c^n \mid n \geq 0\}$

Logica in Informatica - Descrizione di linguaggi

- $L_a(x)$ per indicare $x \in a^*$

$$\forall x(L_a(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \wedge L_a(y))))$$

Logica in Informatica - Descrizione di linguaggi

- $L_a(x)$ per indicare $x \in a^*$

$$\forall x(L_a(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \wedge L_a(y))))$$

- $L_b(x)$ per indicare $x \in b^*$

$$\forall x(L_b(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = b \cdot y \wedge L_b(y))))$$

Logica in Informatica - Descrizione di linguaggi

- $L_a(x)$ per indicare $x \in a^*$

$$\forall x(L_a(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \wedge L_a(y)))$$

- $L_b(x)$ per indicare $x \in b^*$

$$\forall x(L_b(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = b \cdot y \wedge L_b(y)))$$

- $L_c(x)$ per indicare $x \in c^*$

$$\forall x(L_c(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = c \cdot y \wedge L_c(y)))$$

Logica in Informatica - Descrizione di linguaggi

- $L_a(x)$ per indicare $x \in a^*$

$$\forall x(L_a(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \wedge L_a(y)))$$

- $L_b(x)$ per indicare $x \in b^*$

$$\forall x(L_b(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = b \cdot y \wedge L_b(y)))$$

- $L_c(x)$ per indicare $x \in c^*$

$$\forall x(L_c(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = c \cdot y \wedge L_c(y)))$$

- $L_1(x)$ per indicare $x \in \{a^n b^n \mid n \geq 0\}$

$$\forall x(L_1(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \cdot b \wedge L_1(y)))$$

Logica in Informatica - Descrizione di linguaggi

- $L_a(x)$ per indicare $x \in a^*$

$$\forall x(L_a(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \wedge L_a(y)))$$

- $L_b(x)$ per indicare $x \in b^*$

$$\forall x(L_b(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = b \cdot y \wedge L_b(y)))$$

- $L_c(x)$ per indicare $x \in c^*$

$$\forall x(L_c(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = c \cdot y \wedge L_c(y)))$$

- $L_1(x)$ per indicare $x \in \{a^n b^n \mid n \geq 0\}$

$$\forall x(L_1(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = a \cdot y \cdot b \wedge L_1(y)))$$

- $L_2(x)$ per indicare $x \in \{b^n c^n \mid n \geq 0\}$

$$\forall x(L_2(x) \Leftrightarrow (x = \varepsilon \vee \exists y(x = b \cdot y \cdot c \wedge L_2(y)))$$

Logica in Informatica - Descrizione di linguaggi

Specificare in logica del primo ordine il linguaggio $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$.

Quindi

$$\forall x(L(x) \Leftrightarrow (x = \varepsilon \vee \exists y \exists z \exists w(x = y \cdot z \cdot w \wedge L_1(y \cdot z) \wedge L_2(z \cdot w) \wedge L_a(y) \wedge L_b(z) \wedge L_c(w)))).$$

Logica in Informatica - Descrizione di linguaggi

Specificare in logica del primo ordine il linguaggio $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$.

Quindi

$$\forall x(L(x) \Leftrightarrow (x = \varepsilon \vee \exists y \exists z \exists w(x = y \cdot z \cdot w \wedge L_1(y \cdot z) \wedge L_2(z \cdot w) \wedge L_a(y) \wedge L_b(z) \wedge L_c(w)))).$$

Ma serve davvero tutto? Non basterebbe

$$\forall x(L(x) \Leftrightarrow (x = \varepsilon \vee \exists y \exists z \exists w(x = y \cdot z \cdot w \wedge L_1(y \cdot z) \wedge L_2(z \cdot w))))?$$

Logica in Informatica - Descrizione di linguaggi

Specificare in logica del primo ordine il linguaggio $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$.

Quindi

$$\forall x(L(x) \Leftrightarrow (x = \varepsilon \vee \exists y \exists z \exists w(x = y \cdot z \cdot w \wedge L_1(y \cdot z) \wedge L_2(z \cdot w) \wedge L_a(y) \wedge L_b(z) \wedge L_c(w))))).$$

Ma serve davvero tutto? Non basterebbe

$$\forall x(L(x) \Leftrightarrow (x = \varepsilon \vee \exists y \exists z \exists w(x = y \cdot z \cdot w \wedge L_1(y \cdot z) \wedge L_2(z \cdot w))))?$$

No, infatti se $x = aaabbbcc$, la formula sopra sarebbe soddisfatta con $y = aaab$, $z = bb$, $w = cc$ (cioè $x = \underbrace{aaab}_{y}\underbrace{bb}_{z}\underbrace{cc}_{w}$) ma $x \notin \mathcal{L}$.

Logica in Informatica - Descrizione di proprietà - 1

1. Specificare, **usando solo il predicato di uguaglianza e la funzione di concatenazione**, il predicato binario sulle stringhe substring(x, y) che indica che x è una sottostringa di y, cioè una sequenza di caratteri consecutivi che compare in y (ε è sottostringa di qualsiasi stringa).
   Ad esempio substring$(ac, abcd)$ è falso, mentre substring$(bc, abcd)$ è vero.

Logica in Informatica - Descrizione di proprietà - 1

1. Specificare, **usando solo il predicato di uguaglianza e la funzione di concatenazione**, il predicato binario sulle stringhe substring(x, y) che indica che x è una sottostringa di y, cioè una sequenza di caratteri consecutivi che compare in y (ε è sottostringa di qualsiasi stringa).
   Ad esempio substring$(ac, abcd)$ è falso, mentre substring$(bc, abcd)$ è vero.
2. Specificare substring2(x, y, i, j) che indica che x e uguale alla sottostringa di y che va dal carattere in posizione i a quello in posizione j di y (estremi inclusi). Per una stringa di lunghezza n, le posizioni dei suoi caratteri vanno da 0 a $n - 1$ (x non può essere ε).
   Per la specifica si possono usare anche le 4 operazioni aritmetiche, le costanti intere e la funzione unaria "len" che restituisce la lunghezza di una stringa.

Logica in Informatica - Descrizione di proprietà - 1

1. Specificare, **usando solo il predicato di uguaglianza e la funzione di concatenazione**, il predicato binario sulle stringhe substring(x, y) che indica che x è una sottostringa di y, cioè una sequenza di caratteri consecutivi che compare in y (ε è sottostringa di qualsiasi stringa).
   Ad esempio substring$(ac, abcd)$ è falso, mentre substring$(bc, abcd)$ è vero.
2. Specificare substring2(x, y, i, j) che indica che x e uguale alla sottostringa di y che va dal carattere in posizione i a quello in posizione j di y (estremi inclusi). Per una stringa di lunghezza n, le posizioni dei suoi caratteri vanno da 0 a $n-1$ (x non può essere ε). Per la specifica si possono usare anche le 4 operazioni aritmetiche, le costanti intere e la funzione unaria "len" che restituisce la lunghezza di una stringa.
3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Logica in Informatica - Descrizione di proprietà - 1

1. Specificare, **usando solo il predicato di uguaglianza e la funzione di concatenazione**, il predicato binario sulle stringhe substring(x, y) che indica che x è una sottostringa di y, cioè una sequenza di caratteri consecutivi che compare in y.(la stringa nulla è sottostringa di qualsiasi stringa).
   Ad esempio substring$(ac, abcd)$ è falso, mentre
   substring $substring(bc, abcd)$ è vero.

Logica in Informatica - Descrizione di proprietà - 1

1. Specificare, **usando solo il predicato di uguaglianza e la funzione di concatenazione**, il predicato binario sulle stringhe substring(x, y) che indica che x è una sottostringa di y, cioè una sequenza di caratteri consecutivi che compare in y.(la stringa nulla è sottostringa di qualsiasi stringa).
   Ad esempio substring$(ac, abcd)$ è falso, mentre
   substring *substring*$(bc, abcd)$ è vero.

Questo è facile:

$$\forall x \forall y(\text{substring}(x, y) \Leftrightarrow \exists p \exists r(y = p \cdot x \cdot r)).$$

Funziona anche se $x = \varepsilon$, basta porre $p = y$ e $r = \varepsilon$.

Logica in Informatica - Descrizione di proprietà - 1

2. Specificare substring2(x, y, i, j) che indica che x e uguale alla sottostringa di y che va dal carattere in posizione i a quello in posizione j di y (estremi inclusi). Per una stringa di lunghezza n, le posizioni dei suoi caratteri vanno da 0 a $n-1$ (x non può essere ε). Per la specifica si possono usare anche le 4 operazioni aritmetiche, le costanti intere e la funzione unaria "len" che restituisce la lunghezza di una stringa.

Logica in Informatica - Descrizione di proprietà - 1

2. Specificare substring2(x, y, i, j) che indica che x e uguale alla sottostringa di y che va dal carattere in posizione i a quello in posizione j di y (estremi inclusi). Per una stringa di lunghezza n, le posizioni dei suoi caratteri vanno da 0 a $n-1$ (x non può essere ε). Per la specifica si possono usare anche le 4 operazioni aritmetiche, le costanti intere e la funzione unaria "len" che restituisce la lunghezza di una stringa.

È meno facile, però abbiamo a disposizione più simboli:

$$\forall x \forall y \forall i \forall j (\text{substring2}(x, y, i, j) \Leftrightarrow \\ \exists p \exists r (y = p \cdot x \cdot r \wedge \text{len}(p) = i \wedge \text{len}(x) = j - i + 1))$$

N.B.: non serve dire che $i \leq j$, perché se così non fosse $len(x)$ sarebbe negativa.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Simile al problema di riconoscere ww^R.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Più difficile, posso provare per via diretta oppure ricorsivamente.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Più difficile, posso provare per via diretta oppure ricorsivamente.

Via diretta: dobbiamo ragionare sulle posizioni dei caratteri. Immaginiamo due stringhe x e y: *abbc*, *cbba*. Il carattere in posizione 0 di x dev'essere in posizione $len(y) - 1$ di y, il carattere in posizione 1 di x dev'essere in posizione $len(y) - 2$ di y, e così via.

- $0 \rightarrow len(y) - 1$
- $1 \rightarrow len(y) - 2$
- $2 \rightarrow len(y) - 3$
- ...

Viceversa per i caratteri in y:

- $0 \rightarrow len(x) - 1$
- $1 \rightarrow len(x) - 2$
- $2 \rightarrow len(x) - 3$

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Più difficile, posso provare per via diretta oppure ricorsivamente.

Via diretta:

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow$$
$$(\text{len}(x) = \text{len}(y) = 0) \vee$$
$$\big(\forall c \forall i(\text{substring2}(c, x, i, i) \Rightarrow \text{substring2}(c, y, \text{len}(y) - 1 - i, \text{len}(y) - 1 - i)) \wedge$$
$$\forall c \forall i(\text{substring2}(c, y, i, i) \Rightarrow \text{substring2}(c, x, \text{len}(x) - 1 - i, \text{len}(x) - 1 - i))\big).$$

Potrei evitare di inserire la sottoformula

$$\forall c \forall i(\text{substring2}(c, y, i, i) \Rightarrow \text{substring2}(c, x, \text{len}(x) - 1 - i, \text{len}(x) - 1 - i))?$$

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Più difficile, posso provare per via diretta oppure ricorsivamente.

Via diretta:

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow$$
$$(\text{len}(x) = \text{len}(y) = 0) \vee$$
$$\big(\forall c \forall i(\text{substring2}(c, x, i, i) \Rightarrow \text{substring2}(c, y, \text{len}(y) - 1 - i, \text{len}(y) - 1 - i)) \wedge$$
$$\forall c \forall i(\text{substring2}(c, y, i, i) \Rightarrow \text{substring2}(c, x, \text{len}(x) - 1 - i, \text{len}(x) - 1 - i))\big).$$

Potrei evitare di inserire la sottoformula

$$\forall c \forall i(\text{substring2}(c, y, i, i) \Rightarrow \text{substring2}(c, x, \text{len}(x) - 1 - i, \text{len}(x) - 1 - i))?$$

No, perché in tal caso la proprietà sarebbe verificata con $x = a$ e $y = ba$.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Ricorsivamente:

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow$$
$$(\text{len}(x) = \text{len}(y) = 0) \vee$$
$$(\text{len}(x) = \text{len}(y) = 1 \wedge x = y) \vee$$
$$\big(\exists z \exists w \exists r \exists q(x = z \cdot w \wedge y = r \cdot q \wedge$$
$$\text{len}(z) = \text{len}(q) \wedge \text{len}(w) = \text{len}(r) \wedge \text{len}(z) \geq 1 \wedge \text{len}(w) \geq 1 \wedge$$
$$\text{reverse}(z, q) \wedge \text{reverse}(w, r))\big).$$

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Ricorsivamente:

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow$$
$$(\text{len}(x) = \text{len}(y) = 0) \vee$$
$$(\text{len}(x) = \text{len}(y) = 1 \wedge x = y) \vee$$
$$\big(\exists z \exists w \exists r \exists q(x = z \cdot w \wedge y = r \cdot q \wedge$$
$$\text{len}(z) = \text{len}(q) \wedge \text{len}(w) = \text{len}(r) \wedge \text{len}(z) \geq 1 \wedge \text{len}(w) \geq 1 \wedge$$
$$\text{reverse}(z, q) \wedge \text{reverse}(w, r))\big).$$

Potrei evitare di inserire la sottoformula $\text{len}(z) \geq 1 \wedge \text{len}(w) \geq 1$?

Logica in Informatica - Descrizione di proprietà - 1

3. Con le stesse restrizioni del precedente, specificare il predicato binario reverse(x, y) che indica che la stringa x è l'inversa della stringa y. Ad esempio reverse$(abcd, dcba)$ è vero, mentre reverse$(abcd, cba)$ è falso.

Ricorsivamente:

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow$$
$$(\text{len}(x) = \text{len}(y) = 0) \vee$$
$$(\text{len}(x) = \text{len}(y) = 1 \wedge x = y) \vee$$
$$\big(\exists z \exists w \exists r \exists q(x = z \cdot w \wedge y = r \cdot q \wedge$$
$$\text{len}(z) = \text{len}(q) \wedge \text{len}(w) = \text{len}(r) \wedge \text{len}(z) \geq 1 \wedge \text{len}(w) \geq 1 \wedge$$
$$\text{reverse}(z, q) \wedge \text{reverse}(w, r))\big).$$

Potrei evitare di inserire la sottoformula $\text{len}(z) \geq 1 \wedge \text{len}(w) \geq 1$? No, perché in tal caso non si avrebbe ricorsione, infatti ponendo $w = r = \varepsilon$, la formula si ridurrebbe a

$$\forall x \forall y(\text{reverse}(x, y) \Leftrightarrow (\text{len}(x) = \text{len}(y) = 0) \vee$$
$$(\text{len}(x) = \text{len}(y) = 1 \wedge x = y) \vee \text{reverse}(x, y).$$

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ naturali (dalla posizione 1 a $2n$, la condizione che la lunghezza di *in* e *out* è $2n$ e rimane costante si ritiene sottintesa). P copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

1. Usando solo il predicato di uguaglianza e i simboli delle operazioni di somma e prodotto, specificare i predicati odd(n) e even(n), dove odd(n) è vero se e solo se n è dispari, e even(n) se e solo se n è pari.
2. Specificare le precondizioni di P, che sono
   1. ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*;
   2. tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari.
3. Specificare le postcondizioni di P, che sono
   1. tutti gli elementi dalla posizione 1 a n dell'array *out* sono uguali a qualche elemento dispari di *in*;
   2. tutti gli elementi dalla posizione $n+1$ a $2n$ dell'array *out* sono uguali a qualche elemento pari di *in*.
4. Valutare se ci sono ulteriori pre/post condizioni da aggiungere per rendere la specifica della procedura perfettamente coerente con la

Logica in Informatica - Pre/Post condizioni - 1

1. Usando solo il predicato di uguaglianza e i simboli delle operazioni di somma e prodotto, specificare i predicati odd(n) e even(n), dove odd(n)) è vero se e solo se n è dispari, e even(n) se e solo se n è pari.

Logica in Informatica - Pre/Post condizioni - 1

1. Usando solo il predicato di uguaglianza e i simboli delle operazioni di somma e prodotto, specificare i predicati odd(n) e even(n), dove odd(n)) è vero se e solo se n è dispari, e even(n) se e solo se n è pari.

Si ha:

$$\forall x(\text{odd}(x) \Leftrightarrow \exists k(x = 2k + 1));$$
$$\forall x(\text{even}(x) \Leftrightarrow \exists k(x = 2k)).$$

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti. Possiamo supporre di poter accedere all'elemento in posizione i-esima di *in* tramite la funzione in[i].

2. Specificare le precondizioni di P, che sono
   2.1. ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*

   2.2. tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti. Possiamo supporre di poter accedere all'elemento in posizione i-esima di *in* tramite la funzione $\text{in}[i]$.

2. Specificare le precondizioni di P, che sono
   2.1. ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*

$$\forall i(1 \leq i \leq 2n \Rightarrow \neg\exists j(1 \leq j \leq 2n \wedge \neg(i = j) \wedge \text{in}[i] = \text{in}[j]))$$

   2.2. tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti. Possiamo supporre di poter accedere all'elemento in posizione i-esima di *in* tramite la funzione $\text{in}[i]$.

2. Specificare le precondizioni di P, che sono
   2.1. ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*

$$\forall i(1 \leq i \leq 2n \Rightarrow \neg\exists j(1 \leq j \leq 2n \wedge \neg(i = j) \wedge \text{in}[i] = \text{in}[j]))$$

   2.2. tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari

$$\forall i(1 \leq i \leq 2n \Rightarrow (\text{odd}(i) \Rightarrow \text{odd}(\text{in}[i])) \wedge (\text{even}(i) \Rightarrow \text{even}(\text{in}[i])))$$

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

3. Specificare le postcondizioni di P, che sono
   - 3.1. tutti gli elementi dalla posizione 1 a n dell'array *out* sono uguali a qualche elemento dispari di *in*
   - 3.2. tutti gli elementi dalla posizione $n+1$ a $2n$ dell'array *out* sono uguali a qualche elemento pari di *in*

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

3. Specificare le postcondizioni di P, che sono
   3.1. tutti gli elementi dalla posizione 1 a n dell'array *out* sono uguali a qualche elemento dispari di *in*

$$\forall i(1 \leq i \leq n \Rightarrow \exists j(1 \leq j \leq 2n \wedge \text{odd}(\text{in}[j]) \wedge \text{out}[i] = \text{in}[j]))$$

   3.2. tutti gli elementi dalla posizione $n+1$ a $2n$ dell'array *out* sono uguali a qualche elemento pari di *in*

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

3. Specificare le postcondizioni di P, che sono
   3.1. tutti gli elementi dalla posizione 1 a n dell'array *out* sono uguali a qualche elemento dispari di *in*

$$\forall i(1 \leq i \leq n \Rightarrow \exists j(1 \leq j \leq 2n \wedge \text{odd}(\text{in}[j]) \wedge \text{out}[i] = \text{in}[j]))$$

   3.2. tutti gli elementi dalla posizione $n+1$ a $2n$ dell'array *out* sono uguali a qualche elemento pari di *in*

$$\forall i(n+1 \leq i \leq 2n \Rightarrow \exists j(1 \leq j \leq 2n \wedge \text{even}(\text{in}[j]) \wedge \text{out}[i] = \text{in}[j]))$$

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

4. Valutare se ci sono ulteriori pre/post condizioni da aggiungere per rendere la specifica della procedura perfettamente coerente con la sua definizione informale fornita sopra.

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

4. Valutare se ci sono ulteriori pre/post condizioni da aggiungere per rendere la specifica della procedura perfettamente coerente con la sua definizione informale fornita sopra.

Non abbiamo specificato come post-condizione che alla fine del processo i valori contenuti in *out* devono essere **tutti** quelli di *in*, infatti le pre/post-condizioni individuate prima sono verificate anche nel caso in cui

$$in = [1,\ 2,\ 3,\ 4] \qquad out = [1,\ 1,\ 2,\ 2].$$

Logica in Informatica - Pre/Post condizioni - 1

Una procedura P riceve in ingresso un naturale $n > 1$ e due array, *in* e *out*, contenenti $2n$ numeri naturali (dalla posizione 1 a $2n$). La procedura copia tutti gli elementi dell'array *in* nell'array *out*, inserendo i valori dispari nelle prime n posizioni e quelli pari nelle n posizioni seguenti.

4. Valutare se ci sono ulteriori pre/post condizioni da aggiungere per rendere la specifica della procedura perfettamente coerente con la sua definizione informale fornita sopra.

Non abbiamo specificato come post-condizione che alla fine del processo i valori contenuti in *out* devono essere **tutti** quelli di *in*, infatti le pre/post-condizioni individuate prima sono verificate anche nel caso in cui

$$in = [1,\ 2,\ 3,\ 4] \qquad out = [1,\ 1,\ 2,\ 2].$$

Si deve quindi aggiungere la seguente post-condizione

$$\forall i(1 \leq i \leq 2n \Rightarrow \exists j(1 \leq j \leq 2n \wedge \mathrm{out}[i] = \mathrm{in}[j])).$$

In generale non basterebbe, poiché in tal modo specifichiamo solo che l'array *out* contiene ogni elemento di *in*; ma poiché *in* e *out* hanno la stessa lunghezza e *in* non ha elementi ripetuti, è sufficiente questa

Logica in Informatica - Pre/Post condizioni - 2

Un programma P riceve in ingresso due parole (stringhe) x e y del vocabolario italiano tali che l'insieme dei caratteri di x coincida con quello di y. Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 (vero) il valore della lettera proposizionale ESITO, altrimenti a 0.
Ad esempio se x è "basso" e y e "asso", le stringhe x e y non soddisfano le precondizioni; se x è "sasso" e y e "asso", le precondizioni sono soddisfatte e P pone ESITO$= 1$; se x è "sasso" e y e "ossa", le precondizioni sono soddisfatte e P pone ESITO$= 0$.
Formulare in logica del primo ordine le precondizioni e postcondizioni di P servendosi esclusivamente del predicato di uguaglianza, del predicato ESITO della funzione di concatenazione e di un predicato unario "char" che e vero se e solo se il suo argomento è una stringa costituita da un unico carattere.

Logica in Informatica - Pre/Post condizioni - 2

Il programma P riceve in ingresso due parole (stringhe) x e y del vocabolario italiano tali che l'insieme dei caratteri di x coincida con quello di y. Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 il valore della variabile ESITO, altrimenti a 0.

Logica in Informatica - Pre/Post condizioni - 2

Il programma P riceve in ingresso due parole (stringhe) x e y del vocabolario italiano tali che l'insieme dei caratteri di x coincida con quello di y. Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 il valore della variabile ESITO, altrimenti a 0.

Definisco per convenienza il predicato substring:

$$\forall x \forall y(\text{substring}(x, y) \Leftrightarrow \exists s \exists t(s \cdot x \cdot t = y))$$

Logica in Informatica - Pre/Post condizioni - 2

Il programma P riceve in ingresso due parole (stringhe) x e y del vocabolario italiano tali che l'insieme dei caratteri di x coincida con quello di y. Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 il valore della variabile ESITO, altrimenti a 0.

Definisco per convenienza il predicato substring:

$$\forall x \forall y(\text{substring}(x, y) \Leftrightarrow \exists s \exists t(s \cdot x \cdot t = y))$$

Precondizioni:

$$\forall c(\text{char}(c) \Rightarrow (\text{substring}(c, x) \Leftrightarrow \text{substring}(c, y)))$$

Logica in Informatica - Pre/Post condizioni - 2

Il programma P riceve in ingresso due parole (stringhe) x e y del vocabolario italiano tali che l'insieme dei caratteri di x coincida con quello di y. Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 il valore della variabile ESITO, altrimenti a 0.

Definisco per convenienza il predicato substring:

$$\forall x \forall y(\text{substring}(x, y) \Leftrightarrow \exists s \exists t(s \cdot x \cdot t = y))$$

Precondizioni:

$$\forall c(\text{char}(c) \Rightarrow (\text{substring}(c, x) \Leftrightarrow \text{substring}(c, y)))$$

Postcondizioni:

$$\text{ESITO} \Leftrightarrow (\text{substring}(x, y) \vee \text{substring}(y, x))$$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.

Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n, m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n, m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $\text{coprimi}(m, n) \implies z = 1$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n, m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $\text{coprimi}(m, n) \implies z = 1$
Ma così non si impone $z \neq 1$ se non sono coprimi

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n, m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $coprimi(n, m) \iff z = 1 \vee z = 0$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n,m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n,m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $coprimi(n,m) \iff z = 1 \vee z = 0$
Ma così `return 0` soddisfa la specifica

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n, m) = \begin{cases} 1, & \text{se } n, m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$

Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n, m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $(coprimi(n, m) \wedge z = 1) \vee (\neg coprimi(n, m) \wedge z = 0)$

Logica in Informatica - Pre/Post condizioni - 3

Sia dato il seguente programma: $A(n,m) = \begin{cases} 1, & \text{se } n,m \text{ coprimi} \\ 0, & \text{altrimenti} \end{cases}$
Scrivere pre e post condizioni in logica del primo ordine.
Coprimi vuol dire che il massimo comun divisore (MCD) dev'essere 1.

PRE: $n, m \in \mathbb{N}$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall n, m(\text{coprimi}(n,m) \iff \neg\exists k, x_1, x_2(k > 1 \wedge n = kx_1 \wedge m = kx_2))$$

POST: $(coprimi(n,m) \wedge z = 1) \vee (\neg coprimi(n,m) \wedge z = 0)$
Alternativamente: $(coprimi(n,m) \leftrightarrow z = 1) \wedge (\neg(z = 1) \Rightarrow z = 0)$

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

Il *minimo comune multiplo* è il più piccolo numero intero positivo multiplo sia di x sia di y. Per cui, decidiamo che non ha senso se uno dei due numeri in input 0.

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

PRE: $x, y \neq 0$

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

PRE: $x, y \neq 0$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall a, b, c(\texttt{CommonMultiple}(a, b, c) \iff \exists k_1, k_2 \in \mathbb{N}(c = k_1 a \wedge c = k_2 b))$$

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

PRE: $x, y \neq 0$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall a, b, c(\texttt{CommonMultiple}(a, b, c) \iff \exists k_1, k_2 \in \mathbb{N}(c = k_1 a \wedge c = k_2 b))$$

POST:
$\texttt{CommonMultiple}(x, y, z) \wedge \neg\exists h \in \mathbb{N}(h < z \wedge \texttt{CommonMultiple}(x, y, h))$

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

PRE: $x, y \neq 0$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$$\forall a, b, c(\texttt{CommonMultiple}(a, b, c) \iff \exists k_1, k_2 \in \mathbb{N}(c = k_1 a \wedge c = k_2 b))$$

POST:
$\texttt{CommonMultiple}(x, y, z) \wedge \neg\exists h \in \mathbb{N}(h < z \wedge \texttt{CommonMultiple}(x, y, h))$
Ma così $\texttt{CommonMultiple}(x, y, 0)$ è vera per ogni possibile coppia x, y

Logica in Informatica - Pre/Post condizioni - 4

L'esercizio precedente aveva una particolarità che ci ha aiutato: l'output aveva due soli valori possibili (i.e. la funzione aveva un codominio finito). In questo caso, l'algoritmo A di cui fare la specifica è il seguente:

$$A(x, y) = z, \quad z = mcm(x, y)$$

PRE: $x, y \neq 0$

Per le post-condizioni, definiamo un predicato che ci è d'aiuto:

$\forall a, b, c(\texttt{CommonMultiple}(a, b, c) \iff \exists k_1, k_2 \in \mathbb{N}(c = k_1 a \wedge c = k_2 b))$

POST:
$\texttt{CommonMultiple}(x, y, z) \wedge \neg\exists h \in \mathbb{N}(h < z \wedge \texttt{CommonMultiple}(x, y, h))$
Ma così $\texttt{CommonMultiple}(x, y, 0)$ è vera per ogni possibile coppia x, y In questo caso, basta semplicemente ridefinire CommonMultiple:
$\forall a, b, c(\texttt{CommonMultiple}(a, b, c) \iff \exists k_1, k_2 \in \mathbb{N}(c = k_1 a \wedge c = k_2 b \wedge k_1 > 0 \wedge k_2 > 0))$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

Formalizzare tramite logica del primo ordine una formula che specifichi tutti i requisiti del problema (e quindi sia vera se e solo se i pezzi sulla scacchiera corrispondono a una soluzione).

Suggerimento: usare l'ordine e le operazioni su $\mathbb{N}$ e un predicato $p(n, x, y)$ che è vero se e solo se l'n-esima regina si trova sulla x-esima riga e y-esima colonna.

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

Poiché il numero di righe, colonne e regine è lo stesso, posso supporre che il dominio sia l'insieme dei naturali da 1 a 8.

1. almeno una posizione sulla scacchiera per la regina n

$$\forall n \exists x \exists y (p(n, x, y))$$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

Poiché il numero di righe, colonne e regine è lo stesso, posso supporre che il dominio sia l'insieme dei naturali da 1 a 8.

1. almeno una posizione sulla scacchiera per la regina n

$$\forall n \exists x \exists y (p(n, x, y))$$

2. al piu una posizione sulla scacchiera per la regina n

$$\forall n \forall x_1 \forall x_2 \forall y_1 \forall y_2 ((p(n, x_1, y_1) \wedge p(n, x_2, y_2)) \Rightarrow (x_1 = x_2 \wedge y_1 = y_2))$$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

Poiché il numero di righe, colonne e regine è lo stesso, posso supporre che il dominio sia l'insieme dei naturali da 1 a 8.

1. almeno una posizione sulla scacchiera per la regina n

$$\forall n \exists x \exists y (p(n, x, y))$$

2. al piu una posizione sulla scacchiera per la regina n

$$\forall n \forall x_1 \forall x_2 \forall y_1 \forall y_2 ((p(n, x_1, y_1) \wedge p(n, x_2, y_2)) \Rightarrow (x_1 = x_2 \wedge y_1 = y_2))$$

3. al piu una regina in riga x

$$\forall n_1 \forall n_2 \forall x \forall y_1 \forall y_2 ((p(n_1, x, y_1) \wedge p(n_2, x, y_2)) \Rightarrow (n_1 = n_2 \wedge y_1 = y_2))$$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

Poiché il numero di righe, colonne e regine è lo stesso, posso supporre che il dominio sia l'insieme dei naturali da 1 a 8.

1. almeno una posizione sulla scacchiera per la regina n

$$\forall n \exists x \exists y (p(n, x, y))$$

2. al piu una posizione sulla scacchiera per la regina n

$$\forall n \forall x_1 \forall x_2 \forall y_1 \forall y_2 ((p(n, x_1, y_1) \wedge p(n, x_2, y_2)) \Rightarrow (x_1 = x_2 \wedge y_1 = y_2))$$

3. al piu una regina in riga x

$$\forall n_1 \forall n_2 \forall x \forall y_1 \forall y_2 ((p(n_1, x, y_1) \wedge p(n_2, x, y_2)) \Rightarrow (n_1 = n_2 \wedge y_1 = y_2))$$

4. al piu una regina in colonna y

$$\forall n_1 \forall n_2 \forall x_1 \forall x_2 \forall y ((p(n_1, x_1, y) \wedge p(n_2, x_2, y)) \Rightarrow (n_1 = n_2 \wedge x_1 = x_2))$$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

5. due regine diverse non sono mai sulla stessa diagonale NW-SE

$$\forall n_1 \forall n_2 \forall x_1 \forall x_2 \forall y_1 \forall_2((p(n_1, x_1, y_1) \wedge p(n_2, x_2, y_2) \wedge \neg(n_1 = n_2)) \Rightarrow \neg(x_1 + y_2 = x_2 + y_1))$$

Logica in Informatica - Specifica di sistemi

Nel "problema delle 8 regine" si vogliono posizionare 8 regine su una scacchiera 8 per 8 (inizialmente vuota) in modo che nessuna regina possa attaccare (ossia si trovi sulla stessa riga, colonna o diagonale di) un'altra regina.

5. due regine diverse non sono mai sulla stessa diagonale NW-SE

$$\forall n_1 \forall n_2 \forall x_1 \forall x_2 \forall y_1 \forall_2((p(n_1, x_1, y_1) \wedge p(n_2, x_2, y_2) \wedge \neg(n_1 = n_2)) \Rightarrow \neg(x_1 + y_2 = x_2 + y_1))$$

6. due regine diverse non sono mai sulla stessa diagonale NE-SW

$$\forall n_1 \forall n_2 \forall x_1 \forall x_2 \forall y_1 \forall_2((p(n_1, x_1, y_1) \wedge p(n_2, x_2, y_2) \wedge \neg(n_1 = n_2)) \Rightarrow \neg(x_1 + y_1 = x_2 + y_2))$$

MFO - Introduzione

La logica monadica serve per definire i linguaggi. Introduciamo la logica monadica del primo ordine (Monadic First Order Logic, o MFO). N.B.: è una restrizione rispetto alla logica del primo ordine, ha potere descrittivo minore.

- Le variabili sono definite su un sottoinsieme finito di $\mathbb{N}$, rappresentano gli indici delle stringhe descritte, partendo da 0
- Se la lunghezza di una stringa è n, il dominio delle variabili sarà $\{0, 1, \ldots, n-1\}$
- Una formula generica φ può essere:
  - $\neg\varphi$
  - $\varphi_1 \wedge \varphi_2$
  - $\forall x(\varphi)$
  - $a(x)$, predicato che esiste per ogni lettera dell'alfabeto ($\forall a \in I$), che è vero se e solo se la x-esima lettera è a
  - $x < y$, cioè la variabile x strettamente minore della variabile y

MFO - Introduzione

Da qui, ci sono predicati derivati:

- $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $\exists x(\varphi) \equiv \neg\forall x(\neg\varphi)$

MFO - Introduzione

Predicati ausiliari:

- $x = y \equiv \neg(x < y) \wedge \neg(y < x)$
- $x \leq y \equiv \neg(y < x)$
- La constante *zero*: $x = 0 \equiv \forall y(x \leq y)$
- Il predicato di successore: $y = S(x) \equiv x < y \wedge \neg\exists z(x < z \wedge z < y)$
- Vincoli numerici: $1 = S(0), 2 = S(1) \ldots$
- $last(x) \equiv \neg\exists y(x < y)$

Notazione compatta:

- Possiamo scrivere $y = x + 1$ al posto di $y = S(x)$
- Allo stesso modo, $y = x + k$ al posto di $y = S({}^k x)^k$, i.e., k volte il successore di x
- Viceversa, $y = x - 1$ al posto di $x = S(y)$
- $y = x - k$ al posto di $x = y + k$

MFO- Esercizio 1

Linguaggio formato da stringhe che iniziano con a e hanno almeno 3 b.

$$a(0) \wedge \exists x_1, x_2, x_3 (b(x_1) \wedge b(x_2) \wedge b(x_3) \wedge (x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_1 \neq x_3))$$

MFO - Esercizio 2

$L = a^+ b^+$

Ci sono diversi modi per farlo:

- Primo carattere è a, ultimo b, c'è un punto in cui vale il fatto che tutti i caratteri prima sono a e tutti quelli dopo sono b (separatore)
- Primo carattere è a, ultimo b, per ogni coppia a, b che si incontra, le a vengono prima delle b
- Sfruttare il successore

MFO - Esercizio 2

$L = a^+ b^+$

Ci sono diversi modi per farlo:

- Primo carattere è a, ultimo b, c'è un punto in cui vale il fatto che tutti i caratteri prima sono a e tutti quelli dopo sono b (separatore)
- Primo carattere è a, ultimo b, per ogni coppia a, b che si incontra, le a vengono prima delle b
- Sfruttare il successore

Utilizziamo il primo approccio:

$\exists l(last(l) \wedge b(l) \wedge a(0) \wedge$
$\exists x(x \neq 0 \wedge b(x) \wedge$
$\forall y(y < x \Rightarrow a(y)) \wedge$
$\forall z(x \leq z \leq l \Rightarrow b(z))))$

MFO - Esercizio 2

$L = a^+ b^+$
Ci sono diversi modi per farlo:

- Primo carattere è a, ultimo b, c'è un punto in cui vale il fatto che tutti i caratteri prima sono a e tutti quelli dopo sono b (separatore)
- Primo carattere è a, ultimo b, per ogni coppia a, b che si incontra, le a vengono prima delle b
- Sfruttare il successore

Utilizziamo il primo approccio:
$\exists l(last(l) \wedge b(l) \wedge a(0) \wedge$
$\exists x(x \neq 0 \wedge b(x) \wedge$
$\forall y(y < x \Rightarrow a(y)) \wedge$
$\forall z(x \leq z \leq l \Rightarrow b(z))))$

N.B.: Di solito, con gli $\exists$ si usano gli *"and"*, mentre con i $\forall$ si usano gli *"implica"*

MFO - Esercizio 2 - nota

$L = a^+ b^+$

MFO non è chiusa rispetto alla $*$ di Kleene (quindi nemmeno rispetto al $+$).

Allora come mai siamo riusciti ad esprimere L con la MFO?

MFO - Esercizio 2 - nota

$L = a^+ b^+$

MFO non è chiusa rispetto alla $*$ di Kleene (quindi nemmeno rispetto al $+$).

Allora come mai siamo riusciti ad esprimere L con la MFO?

Se riusciamo ad esprimere le stringhe di un linguaggio usando solo i successori, possiamo usare MFO.

MSO - A volte non basta la MFO

$$L = a^{2k+1}$$

Questo linguaggio fa capire che non possiamo definire tutti i linguaggi regolari con la MFO.
Non abbiamo per esempio la possibilità di esprimere parità e disparità.
La logica MFO non è chiusa rispetto alla star: $L = a(aa)^*$

MSO - A volte non basta la MFO

$$L = a^{2k+1}$$

Questo linguaggio fa capire che non possiamo definire tutti i linguaggi regolari con la MFO.
Non abbiamo per esempio la possibilità di esprimere parità e disparità.
La logica MFO non è chiusa rispetto alla star: $L = a(aa)^*$

Proviamo la MSO:
Possiamo definire dei predicati da applicare sulle variabili naturali, proviamo a definire il predicato P di parità e sfruttiamolo per verificare che la lunghezza della stringa sia dispari.

MSO - A volte non basta la MFO

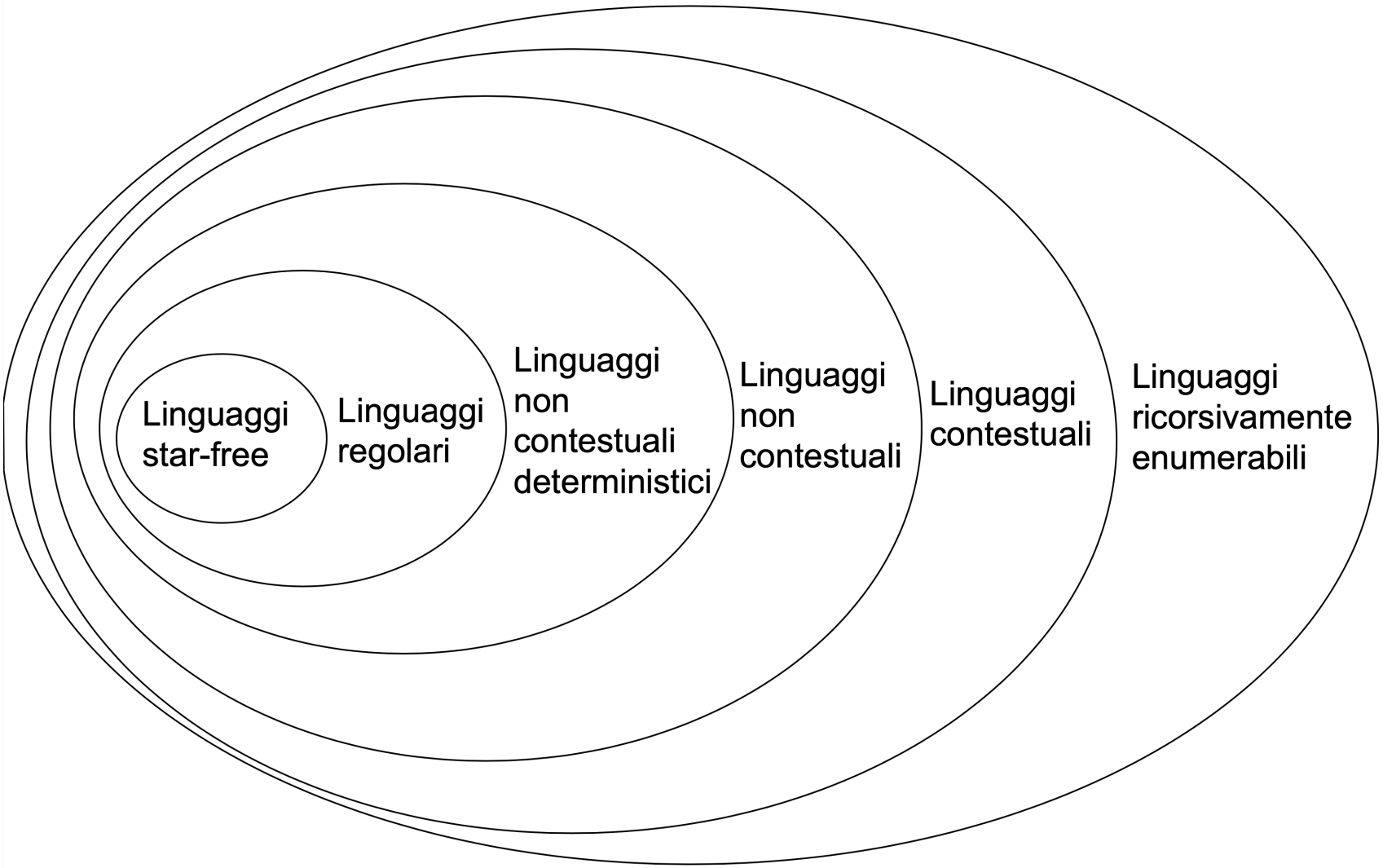
$$L = a^{2k+1}$$

Questo linguaggio fa capire che non possiamo definire tutti i linguaggi regolari con la MFO.
Non abbiamo per esempio la possibilità di esprimere parità e disparità.
La logica MFO non è chiusa rispetto alla star: $L = a(aa)^*$

Proviamo la MSO:
Possiamo definire dei predicati da applicare sulle variabili naturali, proviamo a definire il predicato P di parità e sfruttiamolo per verificare che la lunghezza della stringa sia dispari.

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

Il bersaglio



Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^{+}h \mid la)^{+}$.

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^+h \mid la)^+$.

Possiamo trovare una formula MFO?

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^{+}h \mid la)^{+}$.

Possiamo trovare una formula MFO?

Potremmo pensare che non sia possibile, dato che la MFO non è chiusa rispetto alla stella $*$ di Kleene

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^+h \mid la)^+$.

Possiamo trovare una formula MFO?

Potremmo pensare che non sia possibile, dato che la MFO non è chiusa rispetto alla stella $*$ di Kleene

Però, il fatto che il linguaggio abbia una star non vuol dire che non possa essere specificato con la MFO

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^+h \mid la)^+$.

Idea: Possiamo scrivere una formula specificando i possibili successori per ogni lettera dell'alfabeto. A questo punto, specificare anche i caratteri di inizio e di fine. Questi vincoli saranno soddisfatti da tutte e sole le stringhe in L.

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^+h \mid la)^+$.

Idea: Possiamo scrivere una formula specificando i possibili successori per ogni lettera dell'alfabeto. A questo punto, specificare anche i caratteri di inizio e di fine. Questi vincoli saranno soddisfatti da tutte e sole le stringhe in L.

- $\forall x(e(x) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee h(y))))$. Ogni e è seguita o da e o da h
- $\forall x(l(x) \Rightarrow \exists y(y = x + 1 \wedge a(y)))$. Una l può essere seguita solo da a
- $\forall x(h(x) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee l(y))))$. Una h può essere seguita da e o l
- $\forall x(a(x) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee l(y))))$. Una a può essere seguita da e o l
- $e(0) \vee l(0)$. Il primo carattere non può essere h o a
- $\exists x(last(x) \wedge (h(x) \vee a(x)))$. Specifico qual è l'ultimo carattere

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^{+}h \mid la)^{+}$.

Problema: non posso dire che per ogni x, se x è una h il suo successore deve essere e o l. Perché? perché x potrebbe essere l'ultimo carattere.

Scegliere la logica per le canzoni di Vasco

Consideriamo di nuovo le canzoni di Vasco Rossi: $L = (e^+h \mid la)^+$.

Problema: non posso dire che per ogni x, se x è una h il suo successore deve essere e o l. Perché? perché x potrebbe essere l'ultimo carattere.

- $\forall x(e(x) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee h(y))))$. Ogni e è seguita o da e o da h
- $\forall x(l(x) \Rightarrow \exists y(y = x + 1 \wedge a(y)))$. Una l può essere seguita solo da a
- $\forall x(h(x) \wedge \neg(last(x)) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee l(y))))$. Una h può essere seguita da e o l, **se non è l'ultimo carattere**
- $\forall x(a(x) \wedge \neg(last(x)) \Rightarrow \exists y(y = x + 1 \wedge (e(y) \vee l(y))))$. Una a può essere seguita da e o l, **se non è l'ultimo carattere**
- $e(0) \vee l(0)$. Il primo carattere non può essere h o a
- $\exists x(last(x) \wedge (h(x) \vee a(x)))$. Specifico qual è l'ultimo carattere

Adesso, l'ultima clausola sull'ultimo carattere in realtà diventa superflua.

Scegliere la logica monadica per il linguaggio - 1

$$L = \{(ab)^+ \mid (aabb)^+ \mid (aaabbb)^+\}$$

- MFO o MSO?
- Possiamo specificare il linguaggio usando solo la relazione di successore?

Scegliere la logica monadica per il linguaggio - 1

$$L = \{(ab)^+ \mid (aabb)^+ \mid (aaabbb)^+\}$$

- MFO o MSO?
- Possiamo specificare il linguaggio usando solo la relazione di successore?

Idea: Proviamo a gestire i 3 casi separatamente:

- $(a(0) \wedge b(1) \wedge \forall x(b(x) \wedge \neg last(x) \Rightarrow a(x+1)) \wedge \forall y(a(y) \Rightarrow b(y+1))) \vee$
- $(a(0) \wedge a(1) \wedge b(2) \wedge \forall x(a(x) \wedge b(x+1) \Rightarrow (b(x+2) \wedge (\neg last(x+2) \Rightarrow (a(x+3) \wedge a(x+4) \wedge b(x+5)))))) \vee$
- $(a(0) \wedge a(1) \wedge a(2) \wedge b(3) \wedge \forall x(a(x) \wedge b(x+1) \Rightarrow (b(x+2) \wedge b(x+3) \wedge (\neg last(x+3) \Rightarrow (a(x+4) \wedge a(x+5) \wedge a(x+6) \wedge b(x+7))))))$

Scegliere la logica monadica per il linguaggio - 2

Descrivere le stringhe sull'alfabeto $\mathbf{I} = \{a, b\}$ formate da una sequenza di sottostringhe, ognuna con un prefisso di una cifra che indica la lunghezza della sottostringa

- $2ab3aaa6abbaba, 3aba5bbbbb0, 4abaa05abbba$ sono valide
- $2a3aba, 3abab, 1aba$ non sono valide
- MFO/MSO?

Scegliere la logica monadica per il linguaggio - 2

Descrivere le stringhe sull'alfabeto $\mathbf{I} = \{a, b\}$ formate da una sequenza di sottostringhe, ognuna con un prefisso di una cifra che indica la lunghezza della sottostringa

- $2ab3aaa6abbaba, 3aba5bbbbb0, 4abaa05abbba$ sono valide
- $2a3aba, 3abab, 1aba$ non sono valide

- MFO/MSO?
- Se c'è una cifra i, allora i seguenti i caratteri devono essere a o b, mentre l'$i + 1$-esimo sarà un'altra cifra (a meno che la stringa non sia conclusa)

Scegliere la logica monadica per il linguaggio - 2

Descrivere le stringhe sull'alfabeto $\mathbf{I} = \{a, b\}$ formate da una sequenza di sottostringhe, ognuna con un prefisso di una cifra che indica la lunghezza della sottostringa

- $2ab3aaa6abbaba, 3aba5bbbbb0, 4abaa05abbba$ sono valide
- $2a3aba, 3abab, 1aba$ non sono valide

- MFO/MSO?
- Se c'è una cifra i, allora i seguenti i caratteri devono essere a o b, mentre l'$i+1$-esimo sarà un'altra cifra (a meno che la stringa non sia conclusa)
  - ↪ Possiamo esprimere questo linguaggio solo con i successori, per cui MFO sembra sufficiente!

Scegliere la logica monadica per il linguaggio - 2

Descrivere le stringhe sull'alfabeto $\mathbf{I} = \{a, b\}$ formate da una sequenza di sottostringhe, ognuna con un prefisso di una cifra che indica la lunghezza della sottostringa
Struttura della formula: $\forall x(\bigwedge_{k=0}^{9} k(x) \Rightarrow (\forall y(x < y \leq x + k \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + k + 1) \wedge \neg b(x + k + 1)))$
Per cui, la formula reale sarà: $\forall x$:

- $0(x) \Rightarrow (\forall y(x < y \leq x \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + 1) \wedge \neg b(x + 1))$
- $1(x) \Rightarrow (\forall y(x < y \leq x + 1 \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + 2) \wedge \neg b(x + 2))$
- $2(x) \Rightarrow (\forall y(x < y \leq x + 2 \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + 3) \wedge \neg b(x + 3))$
- $3(x) \Rightarrow (\forall y(x < y \leq x + 3 \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + 4) \wedge \neg b(x + 4))$
- ...
- $9(x) \Rightarrow (\forall y(x < y \leq x + 9 \Rightarrow a(y) \vee b(y)) \wedge \neg a(x + 10) \wedge \neg b(x + 10))$

Inoltre, dobbiamo indicare che la stringa deve iniziare con una cifra: $\bigvee_{k=0}^{9} k(0)$

Scegliere la logica monadica per il linguaggio - 3

$L = a^* b^* c^*$. Possiamo descrivere questo linguaggio specificando i successori:

- $\forall x(b(x) \wedge \neg last(x) \Rightarrow b(x+1) \vee c(x+1))$. Se b non è l'ultimo, allora sarà seguito o da b o da c
- $\forall x(c(x) \wedge \neg last(x) \Rightarrow c(x+1))$. Se c non è l'ultimo, sarà seguito per forza da c
- Non abbiamo bisogno di specificare primo e ultimo carattere, dato che possono essere tutti i caratteri dell'alfabeto
- a può essere seguita da ogni carattere dell'alfabeto, per cui non specifichiamo nessun vincolo sui suoi successori
- C'è bisogno di aggungere esplicitamente la stringa vuota?

Scegliere la logica monadica per il linguaggio - 3

$L = a^* b^* c^*$. Possiamo descrivere questo linguaggio specificando i successori:

- $\forall x(b(x) \wedge \neg last(x) \Rightarrow b(x+1) \vee c(x+1))$. Se b non è l'ultimo, allora sarà seguito o da b o da c
- $\forall x(c(x) \wedge \neg last(x) \Rightarrow c(x+1))$. Se c non è l'ultimo, sarà seguito per forza da c
- Non abbiamo bisogno di specificare primo e ultimo carattere, dato che possono essere tutti i caratteri dell'alfabeto
- a può essere seguita da ogni carattere dell'alfabeto, per cui non specifichiamo nessun vincolo sui suoi successori
- C'è bisogno di aggungere esplicitamente la stringa vuota?
- No, dato che la formula $\forall x((b(x) \wedge \neg last(x) \Rightarrow b(x+1) \vee c(x+1)) \wedge (c(x) \wedge \neg last(x) \Rightarrow c(x+1)))$ risulta vera se non ci sono caratteri nella stringa

Scegliere la logica monadica per il linguaggio - 4

$L = \{y \in \{a,b\}^* \mid \#a = 2k \wedge \#b = 2h+1 \wedge h,k \geq 0\}$

- Abbiamo bisogno di due predicati:
  1. Predicato A vero quando il numero di a lette finora è pari
  2. Predicato B vero quando il numero di b lette finora è dispari
- Quindi usiamo MSO
- Condizioni iniziali: $A(0) \wedge \neg B(0)$
- Ora specifichiamo come ogni predicato cambia al variare del carattere letto:
  - $\forall x(a(x) \wedge \neg last(x) \Rightarrow (A(x) \Leftrightarrow \neg A(x+1)) \wedge (B(x) \Leftrightarrow B(x+1)))$ (se il carattere è a, A cambia mentre B no)
  - $\forall x(b(x) \wedge \neg last(x) \Rightarrow (A(x) \Leftrightarrow A(x+1)) \wedge (B(x) \Leftrightarrow \neg B(x+1)))$ (se il carattere è b, B cambia mentre A no)
  - $\forall x(a(x) \wedge last(x) \Rightarrow \neg A(x) \wedge B(x))$ (se a ultimo carattere, vuol dire che sia il numero di a che il numero di b lette finora devono essere dispari)
  - $\forall x(b(x) \wedge last(x) \Rightarrow A(x) \wedge \neg B(x))$ (se b ultimo carattere, vuol dire che sia il numero di a che il numero di b lette finora devono essere pari)
- Problema: la formula viene soddisfatta anche da ϵ! $\Rightarrow$ Aggiungiamo $a(0) \vee b(0)$

Algoritmo FSA $\rightarrow$ MSO

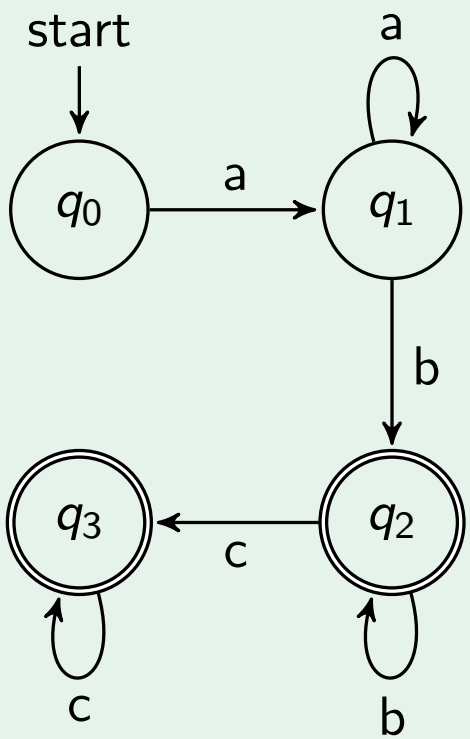
Si può sempre passare da FSA a MSO:

Da FSA a MSO

- Consideriamo un FSA con $\mathbf{Q} = \{q_0, q_1, \ldots, q_K\}$ e un alfabeto di input $\mathbf{I}$
- Esiste un predicato $i(x), \forall i \in \mathbf{I}$
- Possiamo scrivere una formula in MSO equivalente alla sua struttura:
  $\exists Q_0, Q_1, \ldots, Q_K (\forall z (\bigwedge_{i \neq j} \neg (Q_i(z) \wedge Q_j(z))) \wedge Q_0(0) \wedge$
  $\forall x (\neg (last(x)) \Rightarrow (\bigvee_{\delta(q_i, i) = q_j} Q_i(x) \wedge i(x) \wedge Q_j(x+1)) \wedge$
  $last(x) \Rightarrow (\bigvee_{\delta(q_i, i) = q_j, q_j \in \mathbf{F}} Q_i(x) \wedge i(x))))$

Algoritmo FSA $\rightarrow$ MSO

FSA



MSO

$$\exists Q_0, Q_1, Q_2, Q_3(\forall z(\neg(Q_0(z) \wedge Q_1(z)) \wedge$$
$$\neg(Q_0(z) \wedge Q_2(z)) \wedge \neg(Q_0(z) \wedge Q_3(z)) \wedge$$
$$\neg(Q_1(z) \wedge Q_2(z)) \wedge \neg(Q_1(z) \wedge Q_3(z)) \wedge$$
$$\neg(Q_2(z) \wedge Q_3(z))) \wedge Q_0(0) \wedge (\forall x(\neg(last(x)) \Rightarrow$$
$$(Q_0(x) \wedge a(x) \wedge Q_1(x+1) \vee$$
$$Q_1(x) \wedge a(x) \wedge Q_1(x+1) \vee$$
$$Q_1(x) \wedge b(x) \wedge Q_2(x+1) \vee$$
$$Q_2(x) \wedge b(x) \wedge Q_2(x+1) \vee$$
$$Q_2(x) \wedge c(x) \wedge Q_3(x+1) \vee$$
$$Q_3(x) \wedge c(x) \wedge Q_3(x+1)) \wedge$$
$$last(x) \Rightarrow (Q_1(x) \wedge b(x) \vee$$
$$Q_2(x) \wedge b(x) \vee$$
$$Q_2(x) \wedge c(x) \vee$$
$$Q_3(x) \wedge c(x)))))$$