Compilatori

Corso di Laurea in Informatica

Mauro Leoncini

A.A. 2024/2025

Compilatori

Importanza dei linguaggi regolari

- Sono "pervasivi", presenti in molti ambiti dell'Informatica
- Ad esempio, tipici pattern di ricerca all'interno di documenti definiscono linguaggi regolari.
- Come tali, vengono utilizzati negli editor di testo ma anche in molti programmi di utilità disponibili in ambiente Unix/Linux (`grep`, `sed`, `awk`, ...)
- Rivestono un ruolo cruciale nei linguaggi di programmazione.
- Ad esempio, sono linguaggi regolari:
  - l'insieme degli identificatori (di funzione e di variabile);
  - l'insieme di tutte le costanti numeriche (integer o float).
- Sono comunque regolari tutti i linguaggi *finiti*, cioè costituiti da un numero finito di stringhe.

Definizione di linguaggio regolare

- Dato un alfabeto Σ cominciamo col definire *unitario su* Σ ogni linguaggio costituito da un singolo carattere di Σ
- Ad esempio, se $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, i linguaggi unitari su Σ sono: $\{\mathtt{a}\}$, $\{\mathtt{b}\}$ e $\{\mathtt{c}\}$.
- Un linguaggio L su un alfabeto $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ si dice *regolare* se può essere espresso usando un numero finito di operazioni di *concatenazione*, *unione* e *chiusura riflessiva* a partire dai suoi linguaggi unitari $\{\mathtt{a}_1\}, \ldots, \{\mathtt{a}_n\}$
- Più precisamente:
  - $\{\epsilon\}, \{\mathtt{a}_1\}, \ldots, \{\mathtt{a}_n\}$ sono linguaggi regolari
  - se R_1 ed R_2 sono linguaggi regolari, allora $R_1 \cup R_2$ e $R_1 R_2$ sono linguaggi regolari
  - se R è un linguaggio regolare allora R^* è un linguaggio regolare

Esempi di linguaggi regolari

- Sia Σ l'alfabeto ASCII e sia $x = x_1 x_2 \dots x_n$ una generica stringa di Σ^*. Il linguaggio $\{x\}$ è regolare in quanto esprimibile come concatenazione dei linguaggi unitari $\{x_1\}, \{x_2\}, \dots, \{x_n\}$
- Esempio: $\{\texttt{C++}\}$ è concatenazione dei linguaggi unitari $\{\texttt{C}\}$, $\{+\}$ e $\{+\}$, $\{\texttt{Python}\}$ è concatenazione dei linguaggi unitari $\{\texttt{P}\}$, $\{\texttt{y}\}$, $\{\texttt{t}\}$, $\{\texttt{h}\}$, $\{\texttt{o}\}$ e $\{\texttt{n}\}$
- Il linguaggio $\{x, y, z\}$, dove x, y e z sono stringhe sull'alfabeto ASCII è regolare perché esprimibile come unione dei linguaggi regolari $\{x\}, \{y\}$, e $\{z\}$
- Esempio: $\{\texttt{C++}, \texttt{Python}\}$ è regolare perché unione di due linguaggi che sappiamo essere regolari
- Generalizzando gli esempi precedenti si dimostra facilmente come ogni linguaggio finito sia esprimibile come unione di concatenazioni di linguaggi unitari

Altri esempi di linguaggi (regolari e non)

- $L = \{\mathtt{a}^n | n \geq 0\}$ è regolare perché $L = \{\mathtt{a}\}^*$
- Anche il linguaggio $L_{n,m} = \{\mathtt{a}^n\mathtt{b}^m | n, m \geq 0\}$ è regolare poiché $L_{n,m} = \{\mathtt{a}\}^*\{\mathtt{b}\}^*$, cioè è la concatenazione di due linguaggi regolari
- Il linguaggio $\{\mathtt{a}\}^+ = \{\mathtt{a}^n | n \geq 1\}$ è regolare perché $\{\mathtt{a}\}^+ = \{\mathtt{a}\}\{\mathtt{a}\}^*$
- Il L^R è regolare se (e solo se) L è regolare.
- Il linguaggio $L_{n,n} = \{\mathtt{a}^n\mathtt{b}^n | n \geq 0\}$ non è regolare
- Il c.d. linguaggio delle *repliche*

$$L_{\mathtt{rep}} = \{\beta \in \Sigma^* | \beta = \alpha\alpha, \alpha \in \Sigma^*\}$$

non è regolare

- Il linguaggio $L_{prime} = \{\mathtt{a}^n | n \text{ primo}\}$ non è regolare

Compilatori

Espressioni regolari

- Le *espressioni regolari* su un alfabeto Σ sono un formalismo (cioè a loro volta sono linguaggi) per definire linguaggi regolari
- Introdurremo dapprima le espressioni regolari "pure" (e.r.), poi le espressioni come vengono comunemente usate in strumenti/applicazioni reali (da `MS Word`® a `grep`)
- Le espressioni regolari pure sono definite in modo matematicamente semplice e pulito ma sono però poco pratiche
- In concreto si usano espressioni regolari con una notazione molto estesa (in ambiente Linux si distingue fra *Basic Regular Expression* ed *Extended Regular Expression*)

Espressioni regolari di base

- Le e.r. su un alfabeto Σ riflettono le costruzioni usate nella definizione dei linguaggi regolari su Σ

Base

- ϵ è un'espressione regolare che denota il linguaggio composto dalla sola stringa vuota ϵ
- per ogni $\mathtt{a} \in \Sigma$, $\mathbf{a}$ è un'e.r. che denota il linguaggio unitario $\{\mathtt{a}\}$

Ricorsione Se $\mathcal{E}$ ed $\mathcal{F}$ sono e.r. che denotano, rispettivamente, i linguaggi E ed F, allora la scrittura:

- $\mathcal{E}\mathcal{F}$ è un'e.r. che denota il linguaggio EF (*concatenazione*)
- $\mathcal{E}|\mathcal{F}$ (o anche $\mathcal{E} + \mathcal{F}$) è un'e.r. che denota il linguaggio $E \cup F$ (*unione*)
- $\mathcal{E}^*$ è un'e.r. che denota il linguaggio E^* (*chiusura riflessiva*)

Espressioni regolari

- Consideriamo poi un'ulteriore regola:

  Parentesi. Se $\mathcal{E}$ è un'e.r., la scrittura $(\mathcal{E})$ è un'e.r. *equivalente* alla prima, cioè che denota lo stesso insieme di stringhe

  che serve a forzare un ordine di composizione delle espressioni diverso da quello standard
- L'ordine standard prevede che la chiusura precede la concatenazione che precede unione
- Se pensiamo a concatenazione come moltiplicazione, chiusura come esponenziazione e unione come addizione, le regole di precedenza coincidono con quelle dell'aritmetica
- In generale, il linguaggio denotato da un'espressione regolare $\mathcal{E}$ potrà essere indicato scrivendo $L(\mathcal{E})$

Esempi

- L'espressione regolare $\mathbf{0}|\mathbf{1}^*\mathbf{10}$ su $\mathcal{B}$ (interpretabile come $\mathbf{0}|((\mathbf{1}^*)\mathbf{10})$, in base alle regole di precedenza) denota il linguaggio $R_1 = \{\mathbf{0}, \mathbf{10}, \mathbf{110}, \mathbf{1110}, \ldots\}$
- Il linguaggio R_1 è chiaramente differente dal linguaggio R_2 su $\mathcal{B}$ definito dall'espressione regolare $(\mathbf{0}|\mathbf{1})^*\mathbf{10}$, che consiste di tutte le stringhe binarie che terminano con 10
- Posto $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, l'espressione regolare $\mathbf{a}(\mathbf{b}|\mathbf{c})^*\mathbf{a}$ denota il linguaggio R_3 su Σ costituito dalle stringhe che iniziano e terminano con il carattere a e che non contengono altre a
- La scrittura $(\mathbf{1}|\mathbf{01})^*(\mathbf{0}|\boldsymbol{\epsilon})$ denota il linguaggio delle stringhe su $\mathcal{B}$ di lunghezza almeno 1 che non contengono due caratteri 0 consecutivi

Problemi pratici

- Nelle regole di composizione delle e.r. non esiste la possibilità di esprimere la negazione di un carattere a
- Con ciò intendiamo "tutti i caratteri dell'alfabeto escluso a"
- Si provi a scrivere un'e.r. pura, sull'alfabeto ASCII per denotare tutte le stringhe che non terminano con 1.
- Allo stesso modo non esiste un operatore per indicare una potenza *finita* di un insieme di stringhe.
- Si provi qui a scrivere un'espressione regolare pura per il semplice linguaggio $\{a^i | 0 \leq i \leq 30\}$
- Le "estensioni" che andremo ora a vedere, e che mirano a rendere le e.r. utilizzabili in concreto, caratterizzano le c.d. *espressioni regolari estese* secondo lo standard *POSIX* (*Portable Operating System Interface for Unix*)

Metacaratteri e regole aggiuntive

- Come sappiamo, i simboli dell'alfabeto sono detti anche *caratteri*
- Nella descrizione formale del linguaggio si usano però anche altri simboli, che non fanno parte dell'alfabeto.
- Nel caso delle e.r., esempi di simboli usati per scopi descrittivi sono le parentesi e i simboli $*$ e $|$
- Tali simboli sono detti *metacaratteri*
- Nelle e.r. estese (definite, ad es. sull'alfabeto ASCII) esistono metacaratteri che denotano opportuni sotto-insiemi di caratteri dell'alfabeto. Ad esempio:
  - la scrittura `[:alpha:]` denota l'insieme dei caratteri alfabetici (rispetto al *locale* corrente)
  - `[:digit:]` denota l'insieme delle cifre decimali
  - `[:alnum:]` denota l'insieme dei caratteri alfabetici e numerici (le cifre)
- Dagli esempi si evince che un metacarattere è un simbolo *astratto*

Metacaratteri e regole aggiuntive

- Le parentesi quadre sono metacaratteri.
- Caratteri e metacaratteri inclusi fra parentesi quadre si intendono in or
- Ad esempio `[_[:alnum:]]` denota il l'insieme dei caratteri alfanumerici unito il "trattino basso" (*underscore*)
- Il simbolo ^come primo carattere all'interno di una coppia di parentesi quadre denota l'insieme dei caratteri dell'alfabeto, ad esclusione dei simboli che seguono
- Ad esempio `[^[:alnum:]]` denota l'insieme di tutti i caratteri che non sono alfanumerici
- Altri metacaratteri importanti servono a specificare potenze di insiemi:
  - la scrittura $\mathcal{E}?$ denota l'insieme $\{\epsilon\} \cup L(\mathcal{E})$
  - la scritture $\mathcal{E}\{m, n\}$ denota l'insieme

$$L(\mathcal{E})^m \cup L(\mathcal{E})^{m+1} \cup \ldots \cup L(\mathcal{E})^n$$

Metacaratteri e regole aggiuntive

- Il simbolo * conserva il significato che ha nelle espressioni pure

$$\mathcal{E}^* = \bigcup_{i=0}^{\infty} L(\mathcal{E})^i$$

- Poiché $L^+ = LL^*$, l'operatore di chiusura (non riflessiva) è ammesso nelle e.r. estese e si intende che

$$\mathcal{E}^+ = \mathcal{E}\mathcal{E}*$$

- Se è definito un ordinamento fra i caratteri dell'alfabeto, allora si possono utilizzare convenzioni specifiche per denotare intervalli di caratteri. Ad esempio, la scrittura $[\mathtt{a} - \mathtt{f}]$ denota i caratteri compresi fra `a` ed `f`
- Con alfabeto ASCII e locale C una definizione alternativa di $[[: \mathtt{alnum} :]]$ è quindi $[0 - 9\mathtt{A} - \mathtt{Za} - \mathtt{z}]$

Problemi di *matching*

- I programmi `C++` che abbiamo visto alla fine di un precedente set di slide sono esempi di programmi *decisori* di linguaggi
- Il decisore ci permette cioè di definire un linguaggio esattamente come l'insieme di tutte e sole le stringhe per cui il decisore risponde `True` (o `Yes`)
- Nella pratica, molti problemi che coinvolgono le espressioni regolari sono posti in modo diverso (ma con un pressoché identico "substrato" algoritmico)
- Solitamente, dunque, i programmi guidati da e.r. ricevono in input una stringa di testo e un'espressione regolare e devono trovare, nel testo, le occorrenze di detta espressione regolare
- Ad esempio, questo è il caso delle funzioni di ricerca di molti *text editor* "classici" o di altre utlity come `grep` e `sed`

Problemi di *matching*

- Il caso del front-end di compilatori ed interpreti è ancora diverso (come poi vedremo abbondantemente)
- In questo caso il programma (*analizzatore lessicale* o *lexer*) riceve in input il testo (il programma da compilare/eseguire) e un *insieme di espressioni regolari*.
- Essenzialmente il lexer deve "dire" quale espressione regolare descrive un opportuno prefisso (che non è sempre il primo) del testo ancora da analizzare.
- Ad esempio, se l'input ancora da esaminare fosse

  ```
  formula = 2*x**2
  ```

  il prossimo match trovato dal lexer dovrebbe essere `formula` (un caso di identificatore) piuttosto che `for` (un caso di parola chiave)

Qualche esercizio di prova

- Scrivere e.r. per i linguaggi sull'alfabeto $\{\mathtt{a}, \mathtt{b}\}$ composti dalle stringhe contenenti: (1) al più due a, (2) un numero dispari di b.
- Scrivere un'e.r. per il linguaggio sull'alfabeto $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ così definito

$$\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^k | m = 0 \Rightarrow k = 3\}$$

- Scrivere un'e.r. per il linguaggio sull'alfabeto $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, costituito dalle stringhe in cui ogni occorrenza di b è seguita da almeno una c
- Qual è il linguaggio descritto dalla seguente espressione regolare estesa: $[\mathbf{ab}]\{3\} * [\mathbf{ab}]$
- Scrivere un'espressione regolare per definire il linguaggio delle stringhe sull'alfabeto $\{0, 1\}$ che non contengono tre 1 di fila

Espressioni regolari che ci saranno utili

- E.r. estesa per identificatori composti dal carattere *underscore*, lettere e numeri e inizianti per lettera o underscore

$$[\mathbf{a-zA-Z\_}][\mathbf{a-zA-Z0-9\_}]*$$

- E.r. per numeri in virgola fissa (interi inclusi)

$$(\mathbf{0}|[\mathbf{1-9}][\mathbf{0-9}]*)\backslash.?[\mathbf{0-9}]*$$

- E.r. per numeri in virgola mobile

$$(\mathbf{0}|[\mathbf{1-9}][\mathbf{0-9}]*)\backslash.?[\mathbf{0-9}]*([\mathrm{eE}][-+]?[\mathbf{0-9}]+)$$