Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2012/2013

Linguaggi formali e compilazione

Perché sono importanti

- Diffusissimi in ogni “angolo” dell’Informatica.
- Tipici pattern di ricerca all’interno di documenti definiscono linguaggi regolari.
- Rivestono poi un ruolo cruciale nei linguaggi di programmazione.
- Sono infatti regolari, ad esempio:
  - l’insieme degli identificatori di funzione e di variabile;
  - l’insieme di tutte le costanti numeriche (integer o float).
- Sono inoltre 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 $\Sigma = \{a, b, c\}$, i linguaggi unitari su Σ sono: $\{a\}$, $\{b\}$ e $\{c\}$.
- Un linguaggio L su un alfabeto $\Sigma = \{a_1, \ldots, a_n\}$ 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 $\{a_1\}, \ldots, \{a_n\}$.
- Più precisamente:
  - $\{a_1\}, \ldots, \{a_n\}$ 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 $X = X_1 X_2 \ldots X_n$ una generica stringa di Σ^*. Il linguaggio $\{X\}$ è regolare in quanto esprimibile come concatenazione dei linguaggi unitari $\{X_1\}, \{X_2\}, \ldots, \{X_n\}$.
- Ad esempio $\{\texttt{C++}\}$ è concatenazione dei linguaggi unitari $\{\texttt{C}\}$, $\{+\}$ e $\{+\}$, mentre $\{\texttt{Python}\}$ 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 generiche sull'alfabeto ASCII è regolare perché esprimibile come unione dei linguaggi regolari $\{X\}, \{Y\}$, e $\{Z\}$.

Esempi di linguaggi regolari (continua)

- Il linguaggio $\{\texttt{C++}, \texttt{Python}\}$ 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

- $\{ab, c\}^2$ è (un linguaggio) regolare perché:

$$\{ab, c\}^2 = (\{a\}\{b\} \cup \{c\})(\{a\}\{b\} \cup \{c\})$$

- $L = \{a^n | n \geq 0\}$ è regolare perché $L = \{a\}^*$.
- Ma allora anche il linguaggio $L_5 = \{a^n b^m | n, m \geq 0\}$ è regolare poiché $L_5 = \{a\}^*\{b\}^*$, cioè è la concatenazione di due linguaggi regolari.
- Il linguaggio $\{a\}^+ = \{a^n | n \geq 1\}$ è regolare perché $\{a\}^+ = \{a\}\{a\}^*$.
- $\left(\{ab, c\}^2\right)^R$ è regolare poiché $\left(\{ab, c\}^2\right)^R = \{ba, c\}^2$.
- In generale L^R è regolare se (e solo se) L è regolare.
- Il linguaggio $L_6 = \{a^n b^n | n \geq 0\}$ non è regolare.
- Il linguaggio $L_{10} = \{a^n | n \text{ primo}\}$ non è regolare.

Un esempio relativo al Web

- È regolare il linguaggio di tutti i possibili *url* (indirizzi web).
- Perché è regolare?
- Dobbiamo preliminarmente chiederci: come è fatto un url?
  - un nome di dominio, cioè una serie di stringhe separate da punti, ...
  - ... seguito da un simbolo “/” e da una serie opzionale di nomi di cartelle ...
  - ... seguita dal nome completo (nome e tipo) di un documento.
- Possiamo specificare ogni singola parte, e poi un intero url, come linguaggio regolare?

Un esempio relativo al Web (continua)

- Sia $\Sigma = \{\mathtt{a}, \ldots, \mathtt{z}, \mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\}$.
- Allora $S = \Sigma^*$ identifica stringhe su Σ di qualsiasi lunghezza.
- Per identificare nomi di dominio di composti da due sole stringhe, ad esempio `stackoverflow.com`, possiamo usare il linguaggio:

$$S\{.\}S$$

- Per identificare nomi di dominio composti da due, tre, o quattro stringhe possiamo invece usare il linguaggio:

$$L_{DN} = S\{.\}S \cup S\{.\}S\{.\}S \cup S\{.\}S\{.\}S\{.\}S$$

- Se volessimo identificare nomi di dominio composti da un numero arbitrario di stringhe (a patto che possano esistere) potremmo definire S_{DN} nel modo seguente:

$$L_{DN} = S\{.\}S\,(\{.\}S)^*$$

Un esempio relativo al Web (continua)

- Per identificare una sequenza (eventualmente vuota) di cartelle possiamo definire, analogamente, il linguaggio L_C:

$$L_C = (S\{/\})^*$$

- Definiamo ancora il linguaggio I di tutti i possibili identificatori di file:

$$I = (\Sigma \cup \{.\})^*$$

- Infine il linguaggio L_U di tutti i possibili `url` può essere definito come:

$$L_U = L_{DN}\{/\}L_C I$$

Un esempio relativo al Web (continua)

- Qualcuno potrebbe obiettare che L_U contiene stringhe, come ad esempio, `www.unimore.pippo/fasullo.html`, che non rappresentano `url` validi.
- Vero! Però va osservato come l'obiezione non riguardi un "difetto strutturale" della specifica del linguaggio.
- Possiamo comunque risolvere agevolmente il problema, definendo dapprima il linguaggio L_{DPL} di tutti i *domini di primo livello* validi (o meglio, "attualmente validi"):

$$L_{DPL} = \{com, org, edu, net, eu, it, fr, ...\}$$

- ... e quindi ridefinire L_{DN} come:

$$L_{DN} = S\left(\{.\}S\right)^*\{.\}L_{DPL}$$

Espressioni regolari

- Le *espressioni regolari* su un alfabeto Σ sono un formalismo (cioè a loro volta sono linguaggi) per definire linguaggi regolari.
- Definiremo dapprima le espressioni regolari (e.r.) nella forma matematicamente più pulita.
- In seguito presenteremo “abbreviazioni” linguistiche comunemente riconosciute da molti strumenti/ambienti (da `MS Word`® a `grep`).
- Negli esercizi useremo quasi esclusivamente le espressioni nella forma base.

Espressioni regolari di base

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

  Base
  - $\mathbf{\Phi}$ è un'espressione regolare che denota il linguaggio vuoto;
  - per ogni $a \in \Sigma$, $\mathbf{a}$ è un'e.r. che denota il linguaggio unitario $\{a\}$.

  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 $\mathcal{E}|\mathcal{F}$) 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

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

serve a forzare un ordine di composizione delle espressioni diverso da quello standard (in base al quale chiusura precede concatenazione che precede unione).

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 $R_1 = \{\mathtt{0}, \mathtt{10}, \mathtt{110}, \mathtt{1110}, \ldots\}$.
- 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 $\mathtt{10}$.
- Posto $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, 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 altri caratteri a.
- La scrittura $(\mathbf{1} + \mathbf{01})^*(\mathbf{0} + \mathbf{1} + \mathbf{01})$ denota il linguaggio delle stringhe su $\mathcal{B}$ di lunghezza almeno 1 che non contengono due caratteri $\mathtt{0}$ consecutivi.

Abbreviazioni di uso pratico

- Il simbolo ϵ si usa per indicare l'insieme $\{\epsilon\}$. (Attenzione! Non è l'insieme vuoto.)
- La scrittura $[\mathcal{E}]$ si può utilizzare al posto della e.r. $\mathcal{E}+\epsilon$ e l'operatore [ ] prende il nome di *opzione*.
- Se è definito un ordinamento fra i caratteri di Σ, allora si possono utilizzare convenzioni specifiche per denotare intervalli di caratteri. Ad esempio, la scrittura [**a** – **f**] denota i caratteri compresi fra a ed f (opzione su un intervallo).
- Le scritture `[xyz]` e `[^xyz]` indicano rispettivamente un qualunque carattere appartenente o non appartenente all'insieme `{x,y,z}`.

Abbreviazioni di uso pratico

- Poiché $L^+ = LL^*$, l'operatore di chiusura (non riflessiva) è ammesso nelle e.r. dove si intende che $\mathcal{E}^+ = \mathcal{E}\mathcal{E}^*$ (in tal caso l'operatore di unione o alternativa viene sostituito da $|$).
- Poiché $L^n = \overbrace{LL\ldots L}^{n}$, l'operatore di elevamento a potenza è ammesso nelle e.r. e si intende che $\mathcal{E}^n = \overbrace{\mathcal{E}\mathcal{E}\ldots\mathcal{E}}^{n}$.
- La scrittura $[\mathcal{E}]_i^j$ si può utilizzare al posto della e.r. $\mathcal{E}^i + \mathcal{E}^{i+1} + \ldots + \mathcal{E}^j$.

Definizione di numeri e di identificatori

- Per alcuni insiemi di caratteri di particolare importanza (cifre, lettere, caratteri alfanumerici, caratteri di spaziatura, ...) si possono usare espressioni specifiche, come ad esempio (prendendo a prestito la notazione dalle espressioni riconosciute dal comando `grep` di Linux):
  [: *digit* :], [: *alpha* :], [: *alnum* :], [: *space* :], . . .
- L'espressione regolare [**1** – **9**][: *digit* :]* denota l'insieme delle stringhe che rappresentano (nella consueta rappresentazione in base 10) i numeri interi positivi.
- L'espressione regolare
  [: *alpha* :]([: *alpha* :]|[: *digit* :])* denota l'insieme degli identificatori legali in alcuni linguaggi di programmazione (soprattutto fra i più vecchi).

Applicazioni diffuse che usano le espressioni regolari

- Innanzitutto gli editori di testo, più o meno sofisticati (`MS Word`®, `Libre Office Writer`, `Emacs`, ...)
- Molte applicazioni che manipolano file eseguibili da linea di comando in ambiente Unix/Linux (ad esempio, `grep`, `find` e `sed`)
- Utility per la costruzione di analizzatori lessicali (come `Lex`).
- In tutti questi casi, la sintassi per le espressioni regolari è molto più ampia, e (pur non aumentando il potere espressivo) rende la definizione dei pattern molto più semplice.

Espressioni regolari tipicamente usate in applicazioni di Linux

- . ha un match con qualsiasi singolo carattere, ad eccezione di ‘\ n’.
- * ha un match con 0 o più copie della precedente espressione.
- + ha un match con 1 o più copie della precedente espressione.
- ? ha un match con 0 o 1 copia della precedente espressione.
- $\{n\}$ e $\{n, m\}$, dove *n* ed *m* sono numeri, hanno un match con la precedente espressione rispettivamente *n* volte (prima forma) oppure fra *n* ed *m* volte (seconda forma).
- [ ] ha un match con qualunque carattere incluso tra le parentesi; se il primo carattere è ^, allora c’è un match con qualunque carattere non incluso fra le parentesi.

Espressioni regolari ... sotto Linux (continua)

- Un simbolo - entro le parentesi quadre serve per indicare un intervallo di caratteri, come nel caso di [0 – 9].
- ^ come primo carattere di una e.r. ha un match con l'inizio di una linea.
- $ come ultimo carattere di una e.r. ha un match con la fine di una linea.
- \ è il classico carattere di escape.
- | è il simbolo di alternativa.
- ( ),o \(\), servono per il raggruppamento di e.r. e per il loro eventuale riferimento.

Esempi di uso di e.r. in ambiente Libre Office

Esercizi di uso delle espressioni regolari.

1) Sostituire sequenze costituite da più spazi e/o tabulazioni con un singolo carattere spazio. Ad esempio:
La vispa Teresa fra l'erbetta rincorre la farfalletta!
Soluzione: ([:space:]|\t)([:space:]|\t)+ → “ ”

2) Inserire in ogni url l'indicazione dello schema (protocollo) http://
Ad esempio www.unimore.it deve diventare http://www.unimore.it.
Soluzione: [:alnum:]+(\.[:alnum:]+)*\.(it|com|org|net|edu) → http://$0

3) http://www.unimore.it non deve però diventare
http://http://www.unimore.it
Soluzione: ([:space:])(([:alnum:]+\.)+(it|com|org|net|edu)) → $1http://$2

4) Eliminare ogni zero non significativo dai numeri, ad esempio, trasformare 01 e 0.10 in 1 e 0.1 evitando invece di modificare 10 e 0.101. Idem per 05 e 0.781700. Tuttavia, vorremmo che 1.0000 diventasse 1.0 e non 1, perché vogliamo che rimanga un numero float.
Sol.: ([:space:])0([0-9]) → $1$2, (\.[0-9]*[1-9])0+([^0-9]) → $1$2 e \.[0]+ → \.0

Man pages di grep

Anchoring

The caret **^** and the dollar sign **$** are meta-characters that respectively match the empty string at the beginning and end of a line.

The Backslash Character and Special Expressions

The symbols **\<** and **\>** respectively match the empty string at the beginning and end of a word. The symbol **\b** matches the empty string at the edge of a word, and **\B** matches the empty string provided it's not at the edge of a word. The symbol **\w** is a synonym for **[[:alnum:]]** and **\W** is a synonym for **[^[:alnum:]]**.

Repetition

A regular expression may be followed by one of several repetition operators:

?	The preceding item is optional and matched at most once.
*****	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
{n}	The preceding item is matched exactly n times.
{n,}	The preceding item is matched n or more times.
{,m}	The preceding item is matched at most m times.
{n,m}	The preceding item is matched at least n times, but not more than m times.

Esempi di uso di grep

- `grep -e grep parte*.tex`, o anche solo `grep grep parte*.tex`
  ricerca la stringa “`grep`” nei file con nome `parte*.tex`.
- `grep -e grep -e sed parte*.tex`
  ricerca entrambe le stringhe “`grep`” e “`sed`”.
- `grep -c -e '^\\end[{]f' parte*.tex`
  conta tutte le righe (in ogni file che corrisponde a `parte*.tex`) che iniziano con la la stringa `\end{f`

Che cosa è Sed

- `sed` è uno `stream editor`, cioè un editor utilizzabile per eseguire trasformazioni su uno stream di input (un file, un input da terminale o come risultato di un comando precedente).
- `sed` opera con un singolo passo (linea dopo linea) sull'input.
- Non daremo la sintassi precisa del comando `sed`; vedremo piuttosto alcuni esempi.

Esempi di uso di sed

- `sed 'd' test.txt` Legge tutte le linee del file test.txt e produce un output in cui esse sono tutte cancellate
- `sed '2d' test.txt` Cancella da test.txt la sola riga 2
- `sed '1,3d' test.txt` Cancella da test.txt le righe da 1 a 3
- `sed '/^1/d' test.txt` Cancella le righe che iniziano con 1
- `sed '/1$/d' test.txt` Cancella le righe che terminano con 1

Esempi di uso di sed

- `sed '/\( 1\)\1/d' test.txt` Cancella le righe che presentano almeno due ripetizioni consecutive della sequenza " 1"
- `sed 's/abc/xyz/' test.txt` Sostituisce in ogni riga le prime occorrenze di `abc` con `xyz`
- `sed 's/abc/xyz/g' test.txt` Sostituisce tutte le occorrenze di `abc` con `xyz`
- `sed -n 's/abc/xyz/p' test.txt` Sostituisce in ogni riga le prime occorrenze di `abc` con `xyz`; produce in output solo le righe modificate
- `sed -n 's/abc/xyz/gp' test.txt` Sostituisce tutte le occorrenze di `abc` con `xyz`; produce in output solo le righe modificate

Esempi di uso di sed

- `sed 's/a/b/g;s/b/c/g' test.txt`
  Sostituisce prima tutte le a con b e poi tutte le b (incluse quelle derivanti dalla precedente trasformazione) in c
- `sed 's/.$//' test.txt` Utile se si legge sotto Linux un file DOS/Windows
- `sed "s/E'/È/g;s/a'/à/g;s/i'/ì/g; \`
  `s/u'/ù/g;s/o'/ò/g;s/é/è/g; \`
  `s/e'/è/g;s/chè/ché/g" test.txt`
  Corregge le accentazioni e cambia apostrofi "impropri" in accenti in un documento scritto in italiano.

Esempi di uso di sed

- `sed 's/^[ \t]*//;s/[ \t]*$//' test.txt`
  Toglie spazi e tabulazioni ad inizio e fine riga
- `sed -n 's/.*href="\([^"]*\)".*/\1/p' test.txt` Restituisce solo la parte riconosciuta dalla (sotto)espressione regolare inclusa fra \(e \), in questo caso un url
- `sed 's/<[^>]*>//g' test.html` Rimuove i tag in un file html
- `sed 's/[[:cntrl:]]//g' test.txt` Elimina tutti i caratteri di controllo.
- `sed 's:/usr/local:/usr:g; \`
  `s:/usr:/usr/local:g' test.txt`
  Sostituisce tutte le ocorrenze di `/usr` ma non di `/usr/local` con `/usr/local`
- `'s/<[^>]*>//g;s/^[ \t]*//;s/[ \t]*$//;\`
  `s/[[:cntrl:]]//g;/^$/d' test.txt`
  Quale effetto ha?

Qualche esercizio di prova

- Scrivere un'e.r. per il seguente linguaggio sull'alfabeto $\{a, b, c\}$:

$$E_1 = \{a^n b^m c^k | m = 0 \Rightarrow k = 3\}$$

- Scrivere un'e.r. per il linguaggio E_2, sull'alfabeto $\{a, b\}$, delle stringhe contenenti al più due a.
- Scrivere un'e.r. per il linguaggio E_3, sull'alfabeto $\{a, b\}$, delle stringhe contenenti un numero dispari di b.
- Scrivere un'e.r. per il linguaggio E_4, sull'alfabeto $\{a, b\}$, definito ricorsivamente nel modo seguente:
  1. $\epsilon \in E_4$;
  2. Se $x \in E_4$ allora anche $abax \in E_4$ e $xaa \in E_4$.

  Inoltre, solo stringhe ottenibili in questo modo appartengono a E_4.

Ancora qualche esercizio

- Scrivere un'e.r. per il linguaggio E_5, sull'alfabeto $\{a, b, c\}$, costituito dalle stringhe in cui ogni occorrenza di b è seguita da almeno un'occorrenza di c.
- Descrivere nel modo più semplice possibile, in Italiano, il linguaggio corrispondente alla seguente espressione regolare: $((\mathbf{a}|\mathbf{b})^3)^*(\mathbf{a}|\mathbf{b})$.
- Si dica qual è la stringa più corta che non appartiene al linguaggio descritto dall'espressione regolare $\mathbf{a}^*(\mathbf{ab})^*\mathbf{b}^*$.

Alcuni esercizi dati all'esame

- Si scriva un'espressione regolare per definire il linguaggio delle stringhe sull'alfabeto $\{0, 1\}$ che non contengono tre 1 di fila.
- Si consideri l'espressione regolare

$$\mathbf{b}^*\mathbf{aa}\,(\mathbf{ba} \mid \mathbf{b})^*\,\mathbf{b}$$

  e si descriva "a parole" il linguaggio da essa rappresentato.
- Si scriva un'espressione regolare per il linguaggio su $\{0, 1\}$ costituito dalle stringhe che iniziano con 00 oppure terminano con 01.