Espressioni Regolari, GNFA e Pumping Lemma

Consigli e conversioni

Tutorato 3: GNFA-ER e conversioni, Pumping Lemma per Linguaggi Regolari

Gabriel Rovesti
Corso di Laurea in Informatica - Università degli Studi di Padova

Anno Accademico 2024-2025

Contents

1 Espressioni Regolari

1.1 Concetti di base

Un'espressione regolare (RE) è un modo dichiarativo per descrivere linguaggi regolari, mentre gli automi a stati finiti (DFA, NFA) rappresentano il metodo costruttivo.

Concetto chiave

Un linguaggio è *regolare* se e solo se può essere rappresentato da un'espressione regolare, e allo stesso tempo se e solo se può essere riconosciuto da un automa a stati finiti.

Le espressioni regolari sono utilizzate in molti contesti pratici:

- Comandi UNIX (come `grep`)
- Strumenti per l'analisi lessicale (`lex`, `flex`)
- Editor di testo
- Pattern matching in linguaggi di programmazione

1.2 Sintassi formale

Le espressioni regolari sono costruite utilizzando:

1. **Costanti di base**:
   - ε per la stringa vuota
   - $\emptyset$ per il linguaggio vuoto
   - a, b, ... per i simboli $a, b, \ldots \in \Sigma$
2. **Operatori**:
   - $+$ per l'unione
   - $\cdot$ per la concatenazione (spesso omesso)
   - $*$ per la chiusura di Kleene
3. **Parentesi**: ( ) per il raggruppamento

1.3 Semantica delle espressioni regolari

Se E è un'espressione regolare, allora $\mathcal{L}(E)$ è il linguaggio rappresentato da E. La definizione è induttiva:

- **Caso base**:

$$\mathcal{L}(\varepsilon) = \{\varepsilon\} \tag{1}$$
$$\mathcal{L}(\emptyset) = \emptyset \tag{2}$$
$$\mathcal{L}(a) = \{a\} \tag{3}$$

- **Caso induttivo**:

$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F) \tag{4}$$
$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F) \tag{5}$$
$$\mathcal{L}(E^*) = \mathcal{L}(E)^* \tag{6}$$
$$\mathcal{L}((E)) = \mathcal{L}(E) \tag{7}$$

1.4 Regole di precedenza

Come nelle espressioni aritmetiche, anche per le espressioni regolari esistono regole di precedenza:

1. Chiusura di Kleene ($*$) (più alta)
2. Concatenazione ($\cdot$)
3. Unione ($+$) (più bassa)

Errore comune

Molti errori derivano dalla mancata considerazione delle regole di precedenza. Ad esempio, $01^* + 1$ viene interpretato come $(0(1^*)) + 1$, non come $(01)^* + 1$.

1.5 Esempi di espressioni regolari

1. Stringhe con 0 e 1 alternati: $(01)^* + (10)^* + 1(01)^* + 0(10)^*$ oppure $(+1)(01)^*(+0)$
2. Stringhe con numero pari di a: $(b^*ab^*a)^*b^*$
3. Stringhe che contengono la sottostringa 101: $\Sigma^*101\Sigma^*$
4. Stringhe che non contengono la sottostringa 101: questo è più complesso e richiede scomposizione in casi

2 Equivalenza tra Espressioni Regolari e Automi

Un risultato fondamentale nella teoria dei linguaggi regolari è che automi a stati finiti (FA) ed espressioni regolari (RE) hanno lo stesso potere espressivo.

Concetto chiave

L'equivalenza tra espressioni regolari e automi a stati finiti può essere dimostrata in due direzioni:

1. Per ogni espressione regolare R esiste un NFA A tale che $\mathcal{L}(A) = \mathcal{L}(R)$
2. Per ogni NFA A possiamo costruire un'espressione regolare R tale che $\mathcal{L}(R) = \mathcal{L}(A)$

2.1 Da espressione regolare a NFA (ε-NFA)

2.1.1 Approccio costruttivo

La dimostrazione è per induzione strutturale su R:

- **Caso base**:
  - Per ε: un automa con un solo stato, sia iniziale che finale
  - Per $\emptyset$: un automa con un solo stato non finale
  - Per a: un automa con due stati, il secondo finale, connessi da una transizione etichettata a
- **Caso induttivo**:
  - Per $R + S$: un nuovo stato iniziale con ε-transizioni verso gli stati iniziali degli automi per R e S
  - Per RS: gli stati finali dell'automa per R connessi con ε-transizioni allo stato iniziale dell'automa per S
  - Per R^*: un nuovo stato iniziale (anche finale) con ε-transizioni allo stato iniziale dell'automa per R, e ε-transizioni dagli stati finali di R al suo stato iniziale

Suggerimento

Questa costruzione produce sempre un ε-NFA, che può poi essere convertito in un NFA standard, e infine in un DFA se necessario.

2.2 Da NFA a espressione regolare

2.2.1 Automi Nondeterministici Generalizzati (GNFA)

La conversione da NFA a espressione regolare utilizza un passaggio intermedio attraverso GNFA.

Concetto chiave

Un **GNFA** (Generalized Nondeterministic Finite Automaton) è un automa non deterministico dove:

- Le transizioni sono etichettate con *espressioni regolari* anziché singoli simboli
- L'automa legge blocchi di simboli dall'input che appartengono al linguaggio dell'espressione regolare sull'arco

2.2.2 Forma standard di un GNFA

Un GNFA in forma standard ha le seguenti caratteristiche:

1. Lo stato iniziale ha transizioni verso ogni altro stato, ma nessuna transizione entrante

2. Un unico stato finale, senza transizioni uscenti, con una transizione proveniente da ogni altro stato

3. Esiste sempre una transizione per ogni coppia di stati, e un self-loop per ogni stato (eccetto iniziale e finale)

2.2.3 Conversione da NFA a GNFA

La conversione da NFA a GNFA di forma standard richiede:

1. Aggiungere un nuovo stato iniziale q_{start} con transizioni ε verso lo stato iniziale originale

2. Aggiungere un nuovo stato finale q_{accept} con transizioni ε dagli stati finali originali

3. Aggiungere transizioni mancanti etichettate con $\emptyset$ (linguaggio vuoto)

4. Unire transizioni multiple tra stati con operatore di unione

2.2.4 Eliminazione degli stati (State Elimination)

Una volta ottenuto un GNFA, si riducono gli stati uno alla volta:

Procedimento di risoluzione

Algoritmo di eliminazione degli stati:

1. Iniziare con un GNFA con k stati

2. Se $k = 2$ (solo stato iniziale e finale), l'etichetta sulla transizione è l'espressione regolare cercata

3. Altrimenti, scegliere uno stato q_{rip} da eliminare (né iniziale né finale)

4. Per ogni coppia di stati q_i e q_j tali che esistono transizioni da q_i a q_{rip} e da q_{rip} a q_j:

   - Sia R_1 l'espressione sulla transizione $q_i \to q_{rip}$
   - Sia R_2 l'espressione sulla transizione $q_{rip} \to q_{rip}$ (self-loop)
   - Sia R_3 l'espressione sulla transizione $q_{rip} \to q_j$
   - Sia R_4 l'espressione sulla transizione $q_i \to q_j$ (se esiste, altrimenti $\emptyset$)
   - Creare una nuova transizione $q_i \to q_j$ con espressione $(R_1(R_2)^*R_3) + R_4$

5. Rimuovere q_{rip} e tutte le transizioni ad esso connesse

6. Ripetere finché non rimangono solo due stati

Errore comune

In fase di eliminazione degli stati, dimenticare di considerare i self-loop o l'esistenza di transizioni dirette tra stati porta a espressioni regolari errate.

3 Il Pumping Lemma per Linguaggi Regolari

Il Pumping Lemma è uno strumento fondamentale per dimostrare che certi linguaggi non sono regolari.

3.1 Enunciato del Lemma

Concetto chiave

Pumping Lemma: Se L è un linguaggio regolare, allora esiste una costante $p > 0$ (pumping length) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere scomposta in $s = xyz$ dove:

1. $|y| > 0$ (il secondo pezzo è non vuoto)
2. $|xy| \leq p$ (i primi due pezzi insieme hanno lunghezza al massimo p)
3. $\forall i \geq 0$, $xy^iz \in L$ (la stringa ottenuta ripetendo y un numero arbitrario di volte appartiene ancora a L)

3.2 Dimostrazione del Lemma

Procedimento di risoluzione

1. Se L è regolare, esiste un DFA A con un certo numero p di stati che lo riconosce
2. Consideriamo una stringa $w \in L$ con lunghezza $|w| \geq p$
3. Consideriamo gli stati $p_0, p_1, \ldots, p_{|w|}$ visitati durante il riconoscimento di w
4. Tra i primi $p + 1$ stati $(p_0, p_1, \ldots, p_p)$ deve esserci almeno una ripetizione (principio dei cassetti)
5. Sia $p_l = p_m$ con $0 \leq l < m \leq p$ la ripetizione
6. Definiamo $x = w[1 \ldots l]$, $y = w[l + 1 \ldots m]$, $z = w[m + 1 \ldots |w|]$
7. Allora $w = xyz$ con $|y| > 0$ (perché $l < m$) e $|xy| \leq p$ (perché $m \leq p$)
8. Per ogni $i \geq 0$, la stringa xy^iz viene accettata dall'automa, perché ripercorre il loop di stati che include $p_l = p_m$

3.3 Il Pumping Lemma come gioco

Una strategia utile per comprendere e applicare il Pumping Lemma è vederlo come un gioco tra due giocatori.

Concetto chiave

Gioco del Pumping Lemma:

1. **L'avversario** (che sostiene che il linguaggio sia regolare) sceglie una lunghezza p
2. **Noi** scegliamo una stringa $w \in L$ con $|w| \geq p$
3. **L'avversario** spezza w in xyz con $|y| > 0$ e $|xy| \leq p$
4. **Noi** scegliamo un valore $i \geq 0$ tale che $xy^iz \notin L$
5. Se troviamo un valore i tale che $xy^iz \notin L$, allora **abbiamo vinto**, dimostrando che L non è regolare

3.4 Utilizzo del Pumping Lemma

Per dimostrare che un linguaggio L non è regolare:

Procedimento di risoluzione

1. **Assumere per assurdo** che L sia regolare
2. Quindi, per il Pumping Lemma, esiste una costante $p > 0$
3. **Scegliere sapientemente** una stringa $w \in L$ con $|w| \geq p$
4. **Considerare tutte le possibili** scomposizioni $w = xyz$ con $|y| > 0$ e $|xy| \leq p$
5. **Dimostrare** che per ogni scomposizione, esiste un valore $i \geq 0$ tale che $xy^iz \notin L$
6. **Concludere** che L non può essere regolare (per contraddizione)

Suggerimento

Nella scelta della stringa w, cercare una stringa con una struttura che viene facilmente alterata dalla ripetizione di y. Spesso, $i = 0$ (rimozione di y) o $i = 2$ (duplicazione di y) sono scelte efficaci.

3.5 Esempi classici

3.5.1 Esempio 1: $L = \{0^n 1^n \mid n \geq 0\}$

Procedimento di risoluzione

1. Assumiamo per assurdo che L sia regolare, quindi esiste una costante $p > 0$ del Pumping Lemma
2. Scegliamo $w = 0^p 1^p \in L$ (lunghezza $2p \geq p$)
3. Consideriamo una qualsiasi scomposizione $w = xyz$ con $|y| > 0$ e $|xy| \leq p$
4. Poiché $|xy| \leq p$, la stringa y è fatta solo di 0 (cade nella prima metà della stringa)
5. Consideriamo $xy^2z = x0^{|y|}0^{|y|}z = 0^{p+|y|}1^p$
6. Questa stringa ha più 0 che 1, quindi $xy^2z \notin L$
7. Contraddizione, quindi L non è regolare

3.5.2 Esempio 2: $L = \{ww^R \mid w \in \{a, b\}^*\}$ (stringhe palindromi doppie)

Procedimento di risoluzione

1. Assumiamo per assurdo che L sia regolare con costante $p > 0$
2. Scegliamo $w = a^p b a^p = a^p (ba^p)^R \in L$
3. Qualsiasi scomposizione $w = xyz$ con $|xy| \leq p$ e $|y| > 0$ avrà y contenente solo a (nella prima parte della stringa)
4. Consideriamo $xy^0z = xz$ che rimuove alcune a dalla prima parte
5. Questa stringa non può essere scritta nella forma uu^R per alcun u
6. Quindi $xy^0z \notin L$, contraddizione

3.6 Limiti del Pumping Lemma

Concetto chiave

Il Pumping Lemma fornisce una condizione *necessaria* ma *non sufficiente* per i linguaggi regolari:

- Tutti i linguaggi regolari soddisfano il Pumping Lemma
- Esistono linguaggi non regolari che soddisfano comunque il Pumping Lemma

Errore comune

Un errore comune è credere che se un linguaggio soddisfa il Pumping Lemma, allora sia regolare. Questo non è vero, come dimostra l'esempio $L = \{a^\ell b^m c^n \mid \ell, m, n \geq 0 \text{ e se } \ell = 1 \text{ allora } m = n\}$ che non è regolare ma soddisfa il Pumping Lemma per un'appropriata scelta di p.

4 Esercizi Risolti

4.1 Linguaggi che non sono regolari

4.1.1 $L_{ab} = \{w \in \{a, b\}^* \mid$ **numero di** a **uguale al numero di** $b\}$

Procedimento di risoluzione

1. Assumiamo per assurdo che L_{ab} sia regolare con costante $p > 0$
2. Scegliamo $w = a^p b^p \in L_{ab}$
3. Per ogni scomposizione $w = xyz$ con $|y| > 0$ e $|xy| \leq p$, la stringa y contiene solo a
4. Consideriamo xy^2z, che contiene più a che b
5. Quindi $xy^2z \notin L_{ab}$, contraddizione

4.1.2 $L_p = \{1^p \mid p$ **è primo**$\}$

Procedimento di risoluzione

1. Assumiamo per assurdo che L_p sia regolare con costante $p > 0$
2. Scegliamo $w = 1^q$ dove q è un numero primo e $q > p + 2$
3. Per ogni scomposizione $w = xyz$ con $|y| > 0$ e $|xy| \leq p$, definiamo $|y| = m > 0$
4. Consideriamo $xy^{q-m}z$ che ha lunghezza $q - m + (q - m) \cdot m = (q - m)(m + 1)$
5. Poiché $m > 0$, abbiamo $m + 1 > 1$
6. Poiché $m \leq |xy| \leq p < q - 2$, abbiamo $q - m > 2 > 1$
7. Quindi $(q - m)(m + 1)$ è il prodotto di due numeri maggiori di 1, quindi non è primo
8. Così $xy^{q-m}z \notin L_p$, contraddizione

4.2 Linguaggi regolari

4.2.1 $L_{nm} = \{a^n b^m \mid n$ è dispari oppure m è pari$\}$

Questo linguaggio è regolare e può essere rappresentato dall'espressione regolare: $a(aa)^*b^* + a^*(bb)^*$

Equivalentemente, può essere riconosciuto da un automa che traccia la parità di n e m utilizzando 4 stati.

5 Conclusioni

La teoria dei linguaggi regolari offre strumenti potenti per descrivere e analizzare linguaggi:

- **Espressioni regolari**: descrizione dichiarativa e concisa
- **Automi a stati finiti**: modelli computazionali che riconoscono linguaggi
- **GNFA**: ponte concettuale per convertire automi in espressioni regolari
- **Pumping Lemma**: strumento per dimostrare che un linguaggio non è regolare

Concetto chiave

L'equivalenza tra espressioni regolari e automi a stati finiti è un risultato fondamentale che dimostra che queste diverse rappresentazioni hanno lo stesso potere espressivo.

La comprensione di questi concetti e strumenti è essenziale per la teoria dei linguaggi formali e ha applicazioni pratiche in molti campi dell'informatica, dalla compilazione all'analisi di testi, alla verifica di sistemi.

Suggerimento

Quando si studia un nuovo linguaggio, è utile chiedersi:

- È regolare?
- Se sì, quale automa lo riconosce? Quale espressione regolare lo descrive?
- Se no, posso utilizzare il Pumping Lemma per dimostrarlo?