Laboratorio di Linguaggi Formali e Traduttori
Corso di Studi in Informatica
A.A. 2021/2022

Luigi Di Caro, Viviana Patti e Jeremy Sproston
Dipartimento di Informatica — Università degli Studi di Torino

Versione del 18 ottobre 2021

Sommario

Questo documento descrive le esercitazioni di laboratorio e le modalità d'esame del corso di *Linguaggi Formali e Traduttori* per l'A.A. 2021/2022.

Svolgimento e valutazione del progetto di laboratorio

È consigliato sostenere l'esame nella prima sessione d'esame dopo il corso.

Supporto on-line al corso e forum di discussione

Sulla piattaforma I-learn sono disponibili due forum: il primo è dedicato alla pubblicazioni di annunci e notizie di carattere generale, mentre il secondo è un forum di discussione dedicato per gli argomenti affrontati durante il corso. L'iscrizone al forum annunci è effettuata automaticamente, è possibile disiscriversi ma è consigliabile farlo solo a seguito del superamento dell'esame per poter sempre ricevere in modo tempestivo le comunicazioni effettuate dal docente.

Progetto di laboratorio

Il progetto di laboratorio consiste in una serie di esercitazioni assistite mirate allo sviluppo di un semplice traduttore. Il corretto svolgimento di tali esercitazioni presuppone una buona conoscenza del linguaggio di programmazione Java e degli argomenti di teoria del corso Linguaggi Formali e Traduttori.

Modalità dell'esame di laboratorio

Per sostenere l'esame a un appello è necessario prenotarsi. L'esame di laboratorio è **orale** e **individuale**, anche se il codice è stato sviluppato in collaborazione con altri studenti. Durante l'esame vengono accertati: il corretto svolgimento della prova di laboratorio; la comprensione della sua struttura e del suo funzionamento; la comprensione delle parti di teoria correlata al laboratorio stesso.

Note importanti

- Per poter discutere il laboratorio è *necessario* aver prima superato la prova scritta relativa al modulo di teoria. L'esame di laboratorio deve essere superato nella sessione d'esame in cui viene superato lo scritto, altrimenti lo scritto deve essere sostenuto nuovamente.

- La presentazione di codice "funzionante" non è condizione sufficiente per il superamento della prova di laboratorio. In altri termini, è possibile essere respinti presentando codice funzionante (se lo studente dimostra di non avere adeguata familiarità con il codice e i concetti correlati).
- Il progetto di laboratorio può essere svolto individualmente o in gruppi formati da al massimo 3 studenti. Anche se il codice è stato sviluppato in collaborazione con altri studenti, i punteggi ottenuti dai singoli studenti sono indipendenti. Per esempio, a parità di codice presentato, è possibile che uno studente meriti 30, un altro 25 e un altro ancora sia respinto.
- Dal momento che durante la prova è possibile che venga richiesto di apportare modifiche al codice del progetto, è opportuno presentarsi all'esame con un'adeguata conoscenza del progetto e degli argomenti di teoria correlati.

Calcolo del voto finale

I voti della prova scritta e della prova di laboratorio sono espressi in trentesimi. Il voto finale è determinato calcolando la media pesata del voto della prova scritta e del laboratorio , secondo il loro contributo in CFU (con una eventuale modifica nel caso in cui lo studente ha scelto di sostenere una prova orale), e cioè

$$\text{voto finale} = \frac{\text{voto dello scritto} \times 2 + \text{voto del laboratorio}}{3} \pm \text{eventuale esito orale}$$

Validità del presente testo di laboratorio

Il presente testo di laboratorio è valido sino alla sessione di febbraio 2023.

1 Implementazione di un DFA in Java

Lo scopo di questo esercizio è l'implementazione di un metodo Java che sia in grado di discriminare le stringhe del linguaggio riconosciuto da un automa a stati finiti deterministico (DFA) dato. Il primo automa che prendiamo in considerazione, mostrato in Figura 1, è definito sull'alfabeto $\{0, 1\}$ e riconosce le stringhe in cui compaiono almeno 3 zeri consecutivi.

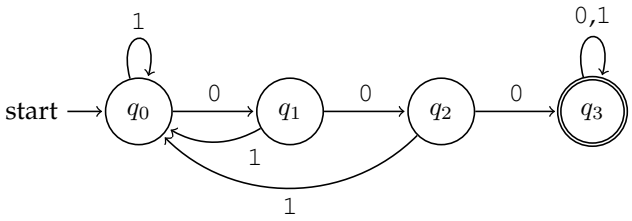

Figura 1: DFA che riconosce stringhe con 3 zeri consecutivi.

L'implementazione Java del DFA di Figura 1 è mostrata in Listing 1. L'automa è implementato nel metodo `scan` che accetta una stringa `s` e restituisce un valore booleano che indica se la stringa appartiene o meno al linguaggio riconosciuto dall'automa. Lo stato dell'automa è rappresentato per mezzo di una variabile intera `state`, mentre la variabile `i` contiene l'indice del prossimo carattere della stringa `s` da analizzare. Il corpo principale del metodo è un ciclo che, analizzando il contenuto della stringa `s` un carattere alla volta, effettua un cambiamento dello stato dell'automa secondo la sua funzione di transizione. Notare che l'implementazione assegna il valore -1 alla variabile `state` se viene incontrato un simbolo diverso da `0` e `1`. Tale valore non è uno stato valido, ma rappresenta una condizione di errore irrecuperabile.

Listing 1: Implementazione Java del DFA di Figura 1.

```
public class TreZeri
{
    public static boolean scan(String s)
    {
        int state = 0;
        int i = 0;
        while (state >= 0 && i < s.length()) {
            final char ch = s.charAt(i++);

            switch (state) {
            case 0:
                if (ch == '0')
                    state = 1;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 1:
                if (ch == '0')
                    state = 2;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 2:
                if (ch == '0')
                    state = 3;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 3:
                if (ch == '0' || ch == '1')
                    state = 3;
                else
                    state = -1;
                break;
            }
        }
        return state == 3;
    }

    public static void main(String[] args)
    {
        System.out.println(scan(args[0]) ? "OK" : "NOPE");
    }
}
```

Esercizio 1.1. Copiare il codice in Listing 1, compilarlo e testarlo su un insieme significativo di stringhe, per es. "`010101`", "`1100011001`", "`10214`", ecc.

Come deve essere modificato il DFA in Figure 1 per riconoscere il linguaggio complementare, ovvero il linguaggio delle stringhe di `0` e `1` che **non** contengono 3 zeri consecutivi? Progettare e implementare il DFA modificato, e testare il suo funzionamento.

Esercizio 1.2. Progettare e implementare un DFA che riconosca il linguaggio degli identificatori in un linguaggio in stile Java: un identificatore è una sequenza non vuota di lettere, numeri, ed il simbolo di "underscore" _ che non comincia con un numero e che non può essere composto solo dal simbolo _. Compilare e testare il suo funzionamento su un insieme significativo di esempi.
Esempi di stringhe accettate: "`x`", "`flag1`", "`x2y2`", "`x_1`", "`lft_lab`", "`_temp`", "`x_1_y_2`", "`x___`", "`__5`"
Esempi di stringhe non accettate: "`5`", "`221B`", "`123`", "`9_to_5`", "`___`"

Esercizio 1.3. Progettare e implementare un DFA che riconosca il linguaggio di stringhe che contengono un numero di matricola seguito (subito) da un cognome, dove la combinazione di matricola e cognome corrisponde a studenti del corso A che hanno un numero di matricola pari oppure a studenti del corso B che hanno un numero di matricola dispari. Si ricorda che gli studenti del corso A sono quelli con cognomi la cui iniziale è compresa tra A e K, e gli studenti del corso B sono quelli con cognomi la cui iniziale è compresa tra L e Z.

Per esempio, "`123456Bianchi`" e "`654321Rossi`" sono stringhe del linguaggio, mentre "`654321Bianchi`" e "`123456Rossi`" no. Nel contesto di questo esercizio, un numero di matricola non ha un numero prestabilito di cifre (ma deve essere composto di almeno una cifra). Un cognome corrisponde a una sequenza di lettere, e deve essere composto di almeno una lettera. Quindi l'automa deve accettare le stringhe "`2Bianchi`" e "`122B`" ma non "`654322`" e "`Rossi`". Assicurarsi che il DFA sia minimo.

Esercizio 1.4. Modificare l'automa dell'esercizio precedente in modo che riconosca le combinazioni di matricola e cognome degli studenti del corso A che hanno un numero di matricola pari oppure a studenti del corso B che hanno un numero di matricola dispari, dove il numero di matricola e il cognome possono essere separati da una sequenza di spazi, e possono essere precedute e/o seguite da sequenze eventualmente vuote di spazi. Per esempio, l'automa deve accettare la stringa "`654321 Rossi`" e "` 123456 Bianchi `" (dove, nel secondo esempio, ci sono spazi prima del primo carattere e dopo l'ultimo carattere), ma non "`1234 56Bianchi`" e "`123456Bia nchi`". Per questo esercizio, i cognomi composti (con un numero arbitrario di parti) possono essere accettati: per esempio, la stringa "`123456De Gasperi`" deve essere accettato. Modificare l'implementazione Java dell'automa di conseguenza.

Esercizio 1.5. Progettare e implementare un DFA che, come in Esercizio 1.3, riconosca il linguaggio di stringhe che contengono matricola e cognome degli studenti del corso A che hanno un numero di matricola pari oppure a studenti del corso B che hanno un numero di matricola dispari, ma in cui il cognome precede il numero di matricola (in altre parole, le posizioni del cognome e matricola sono scambiate rispetto all'Esercizio 1.3). Assicurarsi che il DFA sia minimo.

Esercizio 1.6. Progettare e implementare un DFA che riconosca il linguaggio di stringhe che contengono un numero di matricola seguito (subito) da un cognome, dove la combinazione di matricola e cognome corrisponde a studenti del turno T2 o del turno T3 del laboratorio di Linguaggi Formali e Traduttori. Si ricorda le regole per suddivisione di studenti in turni:

- Turno T1: cognomi la cui iniziale è compresa tra A e K, e la penultima cifra del numero di matricola è dispari;
- Turno T2: cognomi la cui iniziale è compresa tra A e K, e la penultima cifra del numero di matricola è pari;
- Turno T3: cognomi la cui iniziale è compresa tra L e Z, e la penultima cifra del numero di matricola è dispari;
- Turno T4: cognomi la cui iniziale è compresa tra L e Z, e la penultima cifra del numero di matricola è pari.

Un numero di matricola deve essere composto di almeno due cifre, ma (come in Esercizio 1.3) non ha un numero massimo prestabilito di cifre. Assicurarsi che il DFA sia minimo.
Esempi di stringhe accettate: "`654321Bianchi`", "`123456Rossi`", "`221B`"
Esempi di stringhe non accettate: "`123456Bianchi`", "`654321Rossi`", "`5`", "`654322`", "`Rossi`", "`2Bianchi`"

Esercizio 1.7. Progettare e implementare un DFA che riconosca il linguaggio di stringhe che contengono il tuo nome e tutte le stringhe ottenute dopo la sostituzione di un carattere del nome con un altro qualsiasi. Ad esempio, nel caso di uno studente che si chiama Paolo, il DFA deve accettare la stringa "`Paolo`" (cioè il nome scritto correttamente), ma anche le stringhe "`Pjolo`", "`caolo`", "`Pa%lo`", "`Paola`" e "`Parlo`" (il nome dopo la sostituzione di un carattere), ma non "`Eva`", "`Perro`", "`Pietro`" oppure "`P*o*o`".

Esercizio 1.8. Progettare e implementare un DFA che riconosca il linguaggio delle costanti numeriche in virgola mobile utilizzando la notazione scientifica dove il simbolo `e` indica la funzione esponenziale con base 10. L'alfabeto del DFA contiene i seguenti elementi: le cifre numeriche `0`, `1`, ..., `9`, il segno `.` (punto) che precede una eventuale parte decimale, i segni `+` (più) e `-` (meno) per indicare positività o negatività, e il simbolo `e`.

Le stringhe accettate devono seguire le solite regole per la scrittura delle costanti numeriche. In particolare, una costante numerica consiste di due segmenti, la seconda di quale è opzionale: il primo segmento è una sequenza di cifre numeriche che (1) può essere preceduta da un segno `+` o meno `-`, (2) può essere seguita da un segno punto `.`, che a sua volta deve essere seguito da una sequenza non vuota di cifre numeriche; il secondo segmento inizia con il simbolo `e`, che a sua volta è seguito da una sequenza di cifre numeriche che soddisfa i punti (1) e (2) scritti per il primo segmento. Si nota che, sia nel primo segmento, sia in un eventuale secondo segmento, un segno punto `.` non deve essere preceduto per forza da una cifra numerica.
Esempi di stringhe accettate: "`123`", "`123.5`", "`.567`", "`+7.5`", "`-.7`", "`67e10`", "`1e-2`", "`-.7e2`", "`1e2.3`"
Esempi di stringhe non accettate: "`.`", "`e3`", "`123.`", "`+e6`", "`1.2.3`", "`4e5e6`", "`++3`"

Esercizio 1.9. Progettare e implementare un DFA con alfabeto {`/`, `*`, `a`} che riconosca il linguaggio di "commenti" delimitati da `/*` (all'inizio) e `*/` (alla fine): cioè l'automa deve accettare le stringhe che contengono almeno 4 caratteri che iniziano con `/*`, che finiscono con `*/`, e che contengono una sola occorrenza della sequenza `*/`, quella finale (dove l'asterisco della sequenza `*/` non deve essere in comune con quello della sequenza `/*` all'inizio).
Esempi di stringhe accettate: "`/****/`", "`/*a*a*/`", "`/*a/**/`", "`/**a///a/a**/`", "`/**/`", "`/*/*/`"
Esempi di stringhe non accettate: "`/*/`", "`/**/***/`"

Esercizio 1.10. Modificare l'automa dell'esercizio precedente in modo che riconosca il linguaggio di stringhe (sull'alfabeto {`/`, `*`, `a`}) che contengono "commenti" delimitati da `/*` e `*/`, ma con la possibilità di avere stringhe prima e dopo come specificato qui di seguito. L'idea è che sia possibile avere eventualmente commenti (anche multipli) immersi in una sequenza di simboli dell'alfabeto. Quindi l'unico vincolo è che l'automa deve accettare le stringhe in cui un'occorrenza della sequenza `/*` deve essere seguita (anche non immediatamente) da un'occorrenza della sequenza `*/`. Le stringhe del linguaggio possono **non** avere nessuna occorrenza della sequenza `/*` (caso della sequenza di simboli senza commenti). Implementare l'automa seguendo la costruzione vista in Listing 1.
Esempi di stringhe accettate: "`aaa/****/aa`", "`aa/*a*a*/`", "`aaaa`", "`/****/`", "`/*aa*/`", "`*/a`", "`a/**/***a`", "`a/**/***/a`", "`a/**/aa/***/a`"
Esempi di stringhe non accettate: "`aaa/*/aa`", "`a/**//***a`", "`aa/*aa`"