Indice

1 Introduzione

1.1 Motivazione

Un linguaggio è uno strumento per descrivere come risolvere i problemi in maniera rigorosa, in modo tale che sia eseguibile da un calcolatore Perché è utile studiare come creare un linguaggio di programmazione?

- non rimanere degli utilizzatori passivi
- capire il funzionamento dietro le quinte di un linguaggio
- domain-specific language (DSL): è un linguaggio pensato per uno specifico problema
- model drivern software development: modo complesso per dire UML e simili
- model checking

1.2 Definizioni base

Un linguaggio è composto da:

- lessico e sintassi
- compilatore: parser + generatore di codice oggetto

La generazione automatica di codice può essere dichiarativa lessico (espressioni regolari o automa a stati finite) o sintassi(grammatiche o automa a pile). Un automa a stati finiti consuma informazioni una alla volta, ne salva una quantità finita. Alcuni esempi di applicazione di automa a stati finiti: software di progettazione di circuiti, analizzatore lessicale, ricerca di parole sul web e protocolli di comunicazione.

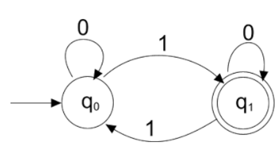

Figura 1: Semplice automa

1.3 Contenuti del corso

- Linguaggi formali e Automi:
  - Automi a stati finiti, espressioni regolari, grammatiche libere, automi a pila, Macchine di Turing, calcolabilità
- Compilatori:
  - Analisi lessicale, analisi sintattica, analisi semantica, generazione di codice
- Logica di base:
  - Logica delle proposizioni e dei predicati
- Modelli computazionali:
  - Specifica di sistemi tramite sistemi di transizione, logiche temporali per la specifica e verifica di proprietà dei sistemi (model checking), sistemi concorrenti (algebre di processi e reti di Petri)

1.4 Informazioni utili

Parte integrante del corso:

- Supporto alla parte teorica usando tool specifici.
  - JFLAP 7.1: http://www.jflap.org (automi/grammatiche)
  - Tina 3.7.5: http://projects.laas.fr/tina (model checking di sistemi di transizione e reti di Petri)
  - LTSA 3.0: http://www.doc.ic.ac.uk/ltsa (sistemi di transizione definiti tramite algebre di processi)
- Nel resto del corso utilizzeremo un ambiente di sviluppo per generare parser/compilatori
  - IntelliJ esteso con plug-in ANTLRv4, ultima versione 1.20 (generatore ANTLR: http://www.antlr.org/)

Libri di testo suggeriti:

- J. E. Hopcroft, R. Motwani e J. D. Ullman: Automi, linguaggi e calcolabilita', Addison-Wesley, Terza Edizione, 2009. Cap. 1–9
- A. V. Aho, M. S. Lam, R. Sethi e J. D. Ullman: Compilatori: principi tecniche e strumenti, Addison Wesley, Seconda Edizione, 2009. Cap. 1–5
- M. Huth e M. Ryan: Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, Second Edition, 2004. Cap. 1–3

2 Linguaggi regolari

2.1 Alfabeti

Un *alfabeto* è un insieme finito e non vuoto di simboli, comunemente indicato con Σ. Seguono alcuni esempi di alfabeti:

- $\Sigma = \{0,1\}$ alfabeto binario
- $\Sigma = \{a,b,...,z\}$ alfabeto di tutte lettere minuscole
- L'insieme ASCII

2.1.1 Stringhe

Una stringa/parola è un insieme di simboli di un alfabeto, 0010 è una stringa che appartiene $\Sigma = \{0,1\}$.
La *stringa vuota* è una stringa composta da 0 simboli.
La lunghezza della stringa sono il numeri di caratteri che la compongono (non devono essere unici). La sintassi per la lunghezza di una stringa w è $|w|$, quindi $|001| = 3$ oppure $|\epsilon| = 0$ (nota bene, $\epsilon \neq 0$ ma è di lunghezza 0).

Potenze di un alfabeto

Se Σ è un alfabeto si può esprimere l'insieme di tutte le stringhe di una certa lunghezza con una notazione esponenziale: Σ^k denota tutte le stringhe di lunghezza k con simboli che appartengono a Σ.
Per esempio:
$\Sigma^1 = \{0,1\}$
$\Sigma^2 = \{00, 01, 10, 11\}$
$\Sigma^2 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
L'insieme delle stringhe meno quella vuota è segnato come Σ^+, mentre l'insieme che include la stringa vuota è Σ^*,

2.1.2 Concatenazione di stringhe

Siano x e y stringhe, dove i è la lunghezza di x e j è la lunghezza di y, la stringa xy è la stringa risultata dalla concatenazione delle stringhe xy di lunghezza i+j.

2.2 Definizione di linguaggio

Un insieme di stringhe a scelta L $\subseteq \Sigma^*$ si definisce linguaggio su Σ.
Un modo formale per definire un alfabeto è il seguente {w | enunciato su w}, che si traduce in "w tale che enunciato su w".
$\{0^n 1^n | n \geq 1\}$ si traduce in "l'insieme di 0 elevato alla n, 1 alla n tale che n è maggiore o uguale a 1"

3 Automa a stati finiti

Un automa a stati finiti deterministico consiste in:

1. Un insieme di stati finiti Q
2. Un insieme di simboli di input, Σ
3. Una funzione di transizione, che prende in input uno stato e un simbolo e restituisce uno stato. Tale funzione è spesso indicato con δ ed è usata per rappresentare i archi nella rappresentazione grafica. Ovvero sia q uno stato, a un input allora δ(q,a) è lo stato p tale che esista un arco da q a p.
4. Uno stato iniziale (naturalmente che appartiene a Q)
5. Un insieme di stati accettati finali F. Questo è un sottoinsieme di Q.

Un automa a stati finiti deterministico è spesso chiamato con l'acronimo DFA e viene può essere rappresentato nella seguente maniera concisa:

$$A = (Q, \Sigma, \delta, q_0, F)$$

Dove A rappresenta il DFA.

3.1 Elaborazione di stringhe

Per elaborare una stringa è si definisce lo stato iniziale, quello finale e una serie di regole di transizione per poterci arrivare. Se dovessi decodificare la stringa 01 il DFA risulterebbe:

$$A = (Q = \{q1, q2, q3\}, \{0, 1\}, \delta, q_0, \{q1\})$$

I stati sono i sequenti:
$\delta(q_0, 1) = q_0$: leggo come primo stato 1, nessun progresso fatto
$\delta(q_0, 0) = q_2$: leggo come primo stato 0, posso andare avanti e cercare un 1
$\delta(q_2, 1) = q_1$: leggo 1 dopo lo 0, ho trovato la stringa
$\delta(q_2, 0) = q_2$: leggo 0 dopo lo 0, non ho fatto progresso
Nota bene: questa è una notazione arbitraria del libro, q1 e q2 si possono invertire.

3.1.1 Notazioni semplici per DFA

Diagramma di transizione

Dato un DFA $A = (Q, \Sigma, \delta, q_0, F)$ un suo diagramma di transizione è composto da:

- Ogni stato Q è un nodo
- Ogni funzione δ è una freccia
- La freccia Start che denota il primo input
- Gli stati accettati F hanno un doppio cerchio

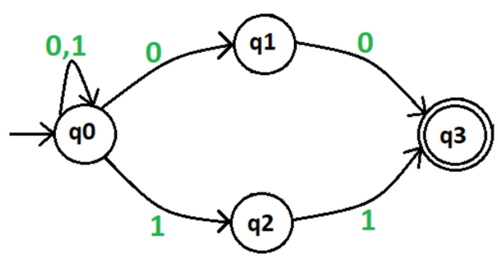

Figura 2: Diagramma di transizione

Tabelle di transizione

Una tabella di transizione è costituita nelle riga dalle funzioni δ e nelle colonne dagli input. Ogni incrocio equivale a uno stato della funzione δ con un input generico a.

	0	1
$\rightarrow$ q$_0$	q$_2$	q$_0$
*q$_1$	q$_1$	q$_1$
q$_2$	q$_2$	q$_1$

Tabella 1: Esempio di tabella

La freccia è lo start e l'asterisco è lo stato finale.

3.1.2 Estensione della funzione di transizione di stringhe

Allo scopo di poter seguire una sequenza di input ci serve definire una funzione di transizione estesa. Se δ è una funzione di transizione, chiameremo $\hat{\delta}$ la sua funzione estesa. La funzione estesa prende in input q e una stringa w e ritorna uno stato p.
Ogni stato viene calcolato grazie allo stato esteso precedente:

$$\hat{\delta}(q,w) = \delta(\hat{\delta}(q,x), a)$$

Esempio

L = { w | w ha un numero pari di 0 e di 1 }
Nota bene: 0 è pari quindi conta come stato accettato, ed è l'unico stato accettato.

q_0: 0 e 1 sono pari
q_1: 0 pari 1 dispari
q_2: 1 pari 0 dispari
q_3: 0 dispari 1 dispari

$$A = (\{q1, q2, q3, q4\}, \{0, 1\}, \delta, q_0, \{q_0\})$$