FLC Cheatsheet

rbino, Lapo, ray

Disclaimer

Gli autori non si assumono nessuna responsabilità per quanto riguarda la correttezza di tutto ciò che segue, questo documento è stato creato nei pomeriggi di studio matto e disperatissimo e non ambiva ad essere diffuso al mondo intero. Se avete correzioni, aggiunte, consigli aprite una issue o mandate una pull request sul repo Git[1].

Licenza

1 Regular Expressions and Finite Automata

1.1 Berry-Sethi

1. Numerare i caratteri della grammatica e aggiungere il terminatore ($\dashv$)
2. Calcolare il set Ini
   (a) Tutti i simboli con cui la stringa può iniziare
3. Calcolare la look-up table per ogni carattere
   (a) Per ogni simbolo, ricavare i followers, ovvero tutti i caratteri che lo possono seguire
4. Disegno l'automa
   (a) Il primo stato ha come nome il set Ini
   (b) Per ogni carattere nel nome dello stato, si guardano i followers e per ogni carattere (escludendo il pedice) si fa un arco in uscita che accetta quel carattere e finisce in uno stato con il nome dei followers di quel carattere
      i. Se ci sono due caratteri uguali ma con pedice diverso (i.e. a_1 e a_2) lo stato di arrivo è l'unione dei followers
      ii. Se ci si accorge che il nome dello stato risultante è uguale ad uno stato già esistente si riutilizza quello stato
      iii. Se tra i follower c'è anche il terminatore ($\dashv$), lo stato viene segnato come terminale

[1] `https://github.com/rbino/flccheatsheet`

1.2 Minimizzazione

1. Fare una matrice con righe e colonne con i nomi degli stati, rinominati con nomi umani (il triangolo superiore è inutile dato che è simmetrica)
2. Al primo step, segnare gli stati sicuramente distinguibili tra loro, ovvero le coppie finale-non finale
3. Per gli stati rimasti, per ogni carattere sugli archi d'uscita, segnare le coppie degli stati d'arrivo
4. Se una coppia contiene due stati distinguibili, marcare la casella come distinguibile
   (a) Stati di errore: se finiscono nello stato d'errore con la stessa variabile oppure uno finisce nello stato d'errore e l'altro ha un autoanello, sono non-distinguibili
5. Gli stati rimasti alla fine non marcati come distinguibili sono indistinguibili
6. Ricostruire l'automa fondendo gli stati indistinguibili

1.3 BMC (Brzozowski) (da FSA a Regex)

1. Aggiungere all'FSA uno stato iniziale senza archi entranti e uno stato finale senza archi uscenti, che rispettivamente con delle ϵ-mosse vanno agli stati iniziali e ricevono gli stati finali
2. Eliminare un nodo intermedio alla volta fondendo gli archi entranti e uscenti con opportune Regex
   (a) Remember: autoanelli = *
   (b) Se ci sono più alternative mettere |
3. Continuare fino a quando non rimangono solo stato iniziale e finale. La Regex sull'arco che li unisce è la Regex equivalente all'FSA

1.4 Intersezione tra linguaggi (FSA) - metodo veloce

1. Prendo i due FSA e rinomino opportunamente gli stati in modo da usare due set di caratteri distinti (e.g. alfabeto greco e lettere)
2. Prendo gli stati iniziali dei due FSA e li unisco in un unico stato (e.g. 1α)
3. Prendo gli archi uscenti comuni (ovvero, le lettere che vengono accettate da tutti e due i nodi) e creo un arco che accetta la lettera accettata da entrambi e finisce in uno stato "fusione" tra gli stati d'arrivo (e.g. se da 1 accetto c finendo nello stato 2 e da α accetto c finendo nello stato γ, avrò un arco che da 1α finisce in 2γ)
4. Ripeto l'operazione per tutti i nodi creati fino a quando posso fare qualcosa
   (a) Se un nodo che creo non è finale e non ha archi uscenti in comune, è uno stato pozzo quindi posso buttarlo
5. Uno stato è finale solo se entrambi gli stati sono finali

1.5 Intersezione tra linguaggi (FSA) - prodotto cartesiano

1. Dispongo i due FSA "a matrice"
2. Riempio la matrice con il prodotto cartesiano di tutti gli stati
3. Vedi metodo veloce per gli archi

1.6 Creare una grammatica da un FSA

1. Chiamo gli stati dell'FSA con lettere maiuscole (non-terminali)
2. Seguo l'FSA creando per ogni stato una regola: $A \to aB$ dove A è lo stato, a è il carattere sull'arco e B è lo stato su cui finisce l'arco.
   (a) Se ci sono più archi in uscita, mettere le regole in or (|)

1.7 Invertire un FSA

1. Completo l'FSA con gli stati di errore nel caso manchino archi uscenti in qualche stato
2. Gli stati finali diventano non finali e viceversa

1.8 Linguaggi locali

1. Per controllare se un linguaggio è local o locally testable prima di tutto controlliamo se vale la condizione sufficiente, cioè: ogni stato deve avere una sola lettera in ingresso per tutti gli archi e non ci devono essere due stati che hanno la stessa lettera in ingresso. Se questa è valida, allora è locale. Altrimenti bisogna verificare ulteriormente: creo il set Ini in cui metto i caratteri con cui può iniziare il linguaggio e il set Fin dei caratteri con cui può finire il linguaggio. Poi creo il set Dig che include tutti i possibili digrammi (e.g. coppie di caratteri) generabili a partire da qualsiasi stato dell'FSA. Il linguaggio è locale se, e solo se, l'FSA accetta qualsiasi stringa che può essere generata concatenando un numero arbitrario di digrammi presi da Dig purché la stringa inizi con un carattere di Ini e termini con un carattere di Fin

1.9 Eliminare ϵ-mosse (Cut spontaneous transitions)

1. Elimino un arco con un'epsilon mossa
2. Copio gli archi uscenti dal nodo di arrivo sul nodo di partenza della ϵ-mossa
   (a) Occhio se ce n'è più di una in fila, vanno considerate tutte

1.10 Linear language equations

1. Scrivere le regole come se fossero un sistema di equazioni
2. Risolvere il sistema
3. Usare la Arden identity: $L_A = L_x | L_y L_A$ diventa $L_A = L_y^* L_x$
   (a) Promemoria: per usare la Arden identity bisogna prima raccogliere tutte le L_A: $aL_A \cup bL_A = \{a, b\}L_A$

2 Free Grammars and Pushdown Automata

2.1 Grammatiche note

2.1.1 Dyck Language

Riconosce le parentesi bilanciate (e.g. (()())())

$$S \to (S)S|\epsilon$$

2.1.2 Palindromi

Con centro (e.g. aabcbaa)

$$S \to aSa|bSb|c$$

Senza centro (e.g. aabbaa)

$$S \to aSa|bSb|\epsilon$$

2.1.3 Varie traduzioni di Regex

$$\begin{aligned} a^* &= S \to aS|\epsilon \\ a^+ &= S \to aS|a \end{aligned}$$

2.2 EBNF

2.2.1 Sintassi

Le regole EBNF hanno una sintassi del tipo
<EXPR>→<NUM> (<OP> <NUM>)*
Elementi opzionali con le quadre
<EXPR>→[<LPAREN>]<PIPPO>[<RPAREN>]
Soliti operatori (*, +, |) più numeri come apice per il numero di ripetizioni.
I caratteri singoli vanno inseriti tra apici.
I set di caratteri possono essere indicati con la notazione (a...z) etc.

2.2.2 Svolgimento

Di solito viene descritto un qualche tipo di linguaggio buffo, conviene partire dalla parte finale del testo (di solito dice "Un programma è una lista di blabla"), a questo punto si definisce il programma (o quello che è) come lista dei blabla e si scende definendo i blabla fin quando si arriva ai terminali.

3 Syntax Analysis and Parsing Methodologies

3.1 ELR(1)

3.1.1 Costruire l'automa pilota

1. Ogni stato dell'automa pilota è diviso in due parti: sotto si inseriscono le computazioni che iniziano a quello stato, sopra si seguono quelle in corso
2. Si inseriscono nel primo macrostato dell'automa pilota tutti gli stati raggiungibili con ϵ-mossa dallo stato iniziale dell'FSA, esso compreso
3. Le produzioni che iniziano a quello stato vengono inserite insieme al loro lookahead nella seconda metà dello stato dell'automa pilota. Il lookahead è composto dall'insieme dei caratteri ($\dashv$ compreso) che possono essere ricevuti al termine della produzione
4. Per ogni carattere che può essere ricevuto dagli stati compresi nel macrostato corrente, calcola il macrostato successivo composto da tutti gli stati raggiungibili col carattere dato partendo dagli stati del macrostato precedente (il lookahead non cambia) e le eventuali nuovi produzioni che iniziano in questo macrostato. Collegare i due macrostati con un arco con etichetta del carattere accettato. Nel caso più di uno stato produca la stessa transizione (stesso carattere, stesso stato di arrivo), aggiungere più frecce per controllare convergence conflicts

5. Gli stati che terminano produzioni vengono cerchiati
6. (Riciclare gli stati se ne escono di uguali a qualcuno che abbiamo già)
7. Una volta finiti tutti i macrostati, controllare eventuali presenze di conflitti

3.1.2 Conflitti

- Convergence conflict: se ci sono più frecce con lo stesso carattere tra una coppia di stati
- Reduce-reduce conflict: se abbiamo nel macrostato più di uno stato finale con caratteri condivisi nel lookahead
- Shift-reduce conflict: se abbiamo un macrostato con uno stato finale e una freccia uscente con un carattere nel lookahead del suddetto stato

Se ci sono conflitti, l'automa **NON** è ELR(1)

3.1.3 Tabellozzo

1. Si parte dallo stato iniziale e lo si mette nello stack. In input si mette la stringa da leggere (con $\dashv$ alla fine)
2. Si legge il primo carattere della stringa. In "move executed" si segna "shifted $c". Sullo stack si segna il carattere letto e lo stato in cui si arriva.
3. Si continua fin quando non si arriva allo stato finale di una produzione. Quando succede, si poppa dallo stack fin quando si torna allo stato in cui è iniziata (controllare sia che sia lo stato iniziale che il lookahead che dev'essere lo stesso). Quando si fa questa operazione si segne in "move executed" "reduced $sarcazzo"
4. Dopo una reduce, è come se "leggessimo" il non-terminale che abbiamo ridotto, quindi facciamo la shift sul non-terminale
5. Continuare fino alla fine della stringa
6. Se alla fine abbiamo nello stack solo lo stato iniziale, in input solo il terminatore e svolgiamo una reduce dell'assioma, allora la stringa viene riconosciuta

3.2 ELL(n)

3.2.1 Verifica su FSA

1. Per ogni nodo che ha un arco uscente che riconosce un non-terminale, si traccia un arco tratteggiato che unisce quel nodo al nodo iniziale della macchina che riconosce il non-terminale
2. Per ogni arco , si computa il guide set, che è l'insieme degli n-grammi che è possibile ricevere partendo dallo stato di partenza dell'arco (se è tratteggiato, dallo stato di arrivo), incluso il terminatore
   (a) In caso di ELL(1) basta farlo per ogni arco tratteggiato e finale (i guide sets degli altri archi sono semplicemente le etichette)
3. Se i guide set degli archi uscenti da un nodo sono non disgiunti, il linguaggio non è ELL(n)

3.2.2 Tabellozzo

1. Si parte dallo stato iniziale della macchina assiomatica (S), che si mette nello stack. In input di mette la stringa.
2. Ad ogni azione si cambia lo stato sullo stack
3. Se si legge un terminale, l'azione è "scanned a"
4. Se si legge un non-terminale, l'azione è "Call X" e viene pushato sullo stack lo stato iniziale del non-terminale
5. Al passo successivo di quando si arriva ad uno stato finale di un non-terminale, si mette un'azione "returned from X" e si poppa lo stato finale
6. Se alla fine ci si trova nello stato finale dell'assioma e l'input è solo il terminatore, la stringa viene accettata

3.3 Earley

È simile a ELR solo che si seguono tutte le produzioni in parallelo (ed è fatto su una stringa data)

1. Si lavora sull'FSA
2. Nel primo stato si segnano le produzioni che possono iniziare nel primo stato dell'FSA. Se un non-terminale è nullable (lo stato iniziale è anche finale), bisogna considerare subito la sua riduzione. Sulla seconda colonna si segna il macrostato a cui è iniziata la produzione
3. Si legge il primo carattere e si seguono solo le produzioni che lo accettano. Si segna lo stato di arrivo (che continuerà ad avere inizio 0) ed eventuali nuove produzioni che cominciano (con inizio 1)
4. Si continua fino alla fine della stringa. Quando una produzione secondaria raggiunge lo stato finale, nello stesso macrostato bisogna aggiungere lo stato successivo del suo genitore (con inizio del genitore)
5. Se si arriva alla fine della stringa con uno stato finale della macchina assiomatica (S) che ha inizio 0, la stringa viene riconosciuta, altrimenti no

4 Language Translation and Semantic Analysis

4.1 Traduzione di linguaggi

1. Per ogni regola di riconoscimento del linguaggio sorgente, scrivere la regola del linguaggio destinazione
2. Attenzione: bisogna tenere **tutte** le regole del linguaggio sorgente
3. Potrebbe chiedere di modificare anche il linguaggio sorgente

4.2 Grammatiche con attributi

Servono per "fare cose" durante il parsing usando attributi (aka, delle variabili) che vengono modificati durante l'esecuzione delle regole della grammatica. Gli attributi possono essere di tipo left, ovvero che vengono propagati dalle foglie verso la radice, oppure right, ovvero che vengono propagati dalla radice verso le foglie.

1. Per ogni regola, definire quali operazioni vengono eseguite. Conviene guardare l'albero e capire da quello se conviene definire gli attributi come right o left (non per forza tutti uguali)

2. Nella tabella degli attributi bisogna scrivere tipo (left o right), significato (cosa mi significa quell'attributo), non-terminali (quelli in cui il valore viene usato) e il dominio (il tipo della variabile)
3. Per ogni regola, indicare come viene assegnata l'attributo. Utilizzare i pedici delle lettere per stabilire a chi appartiene l'attributo che si sta usando
4. Quando si sono scritte tutte le regole, decorare l'albero (prendere le regole e fare i conti sull'albero)

4.2.1 One-sweep e L-condition

1. Tracciare delle frecce per indicare le dipendenze tra i vari attributi sull'albero decorato
2. La grammatica è one-sweep se e solo se:
   (a) Non ci sono cicli nel grafo delle dipendenze
   (b) In un nodo non ci devono essere attributi left che dipendono da attributi right associati allo stesso non-terminale destro
   (c) Nel grafo non ci devono essere dipendenze tra un attributo sinistra associato ad un non-terminale sinistro e un destro di un non-terminale destro
   (d) Il grafo dei figli dello stesso nodo non deve avere loop
3. La grammatica soddisfa la L-condition se e solo se:
   (a) È one-sweep
   (b) Il grafo dei figli dello stesso non ha archi tra attributi di nodi più a destra e nodi più a sinistra

4.3 Dataflow Analysis

4.3.1 Liveness analysis

1. Riempire la tabella di def e use con le variabili
   (a) Va in def: se è a sinistra dell'uguale in un assegnamento oppure è un argomento di una read
   (b) Va in use: se è a destra dell'uguale in un assegnamento, è un argomento di una write oppure viene usata per confronti
2. Impostare le equazioni di In e Out dei nodi
   (a) $In(n) = \{use(n)\} \cup \{out(n) \backslash def(n)\}$
   (b) $Out(n) = \bigcup_{\forall p \in succ(n)} \{In(p)\}$
3. Per risolvere iterativamente, partire dalle equazioni di Out mettendo tutti gli insiemi vuoti, l'equazione di In avrà come risultato l'insieme use. Da qui in poi risolvere alternatamente le equazioni Out e In usando i risultati del passo precedente. Continuare fin quando si giunge ad un punto fisso (e.g. due iterazioni successive danno insiemi uguali).

4.3.2 Reaching definitions

Per ogni stato richiesto, si controlla per ogni variabile da quali stati possono arrivare le definizioni che lo raggiungono. Praticamente, si percorre il grafo all'indietro fino a raggiungere ogni possibile definizione che non viene soppressa da una successiva.