Azzolini Riccardo 2020-12-07

Equivalenza tra CFG e PDA

1 Forme sentenziali

Data una CFG $G = \langle V, T, \Gamma, S \rangle$, si chiamano **forme sentenziali** di G le stringhe $\alpha \in (V \cup T)^*$ tali che $S \overset{*}{\Rightarrow} \alpha$, cioè le stringhe di terminali e non-terminali derivabili a partire dal simbolo iniziale. In particolare, si definiscono

- **forme sentenziali sinistre** le stringhe $\alpha \in (V \cup T)^*$ tali che $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$;
- **forme sentenziali destre** le stringhe $\alpha \in (V \cup T)^*$ tali che $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$.

Osservazione: Rispetto alle stringhe del linguaggio $L(G)$ generato da G, che sono formate solo da simboli terminali, nelle forme sentenziali possono esserci anche simboli non-terminali.

2 Dalle grammatiche ai PDA

Si vuole dimostrare che, data una qualunque CFG $G = \langle V, T, \Gamma, S \rangle$, esiste un PDA che riconosce il linguaggio generato da G. L'idea è quella di costruire un PDA che, su un input $w \in T^*$, simuli la sequenza delle forme sentenziali sinistre usate dalla grammatica per cercare di generare w.

Sia $w = xy$ (con $x, y \in T^*$), dove x è la stringa già letta dal PDA a un certo punto dell'esecuzione, e sia $\alpha \in (V \cup T)^*$ il contenuto dello stack in tale momento. Allora, $x\alpha$ è una forma sentenziale sinistra della grammatica ($S \overset{*}{\underset{lm}{\Rightarrow}} x\alpha$). Questa configurazione del PDA è caratterizzata dall'ID (q, y, α).

Il singolo passo di computazione del PDA dipende dal tipo di simbolo in cima allo stack:

- Se il simbolo in cima è un non-terminale, cioè lo stack contiene $A\alpha$ con $A \in V$, il PDA applica sostanzialmente una regola di produzione $A \to \beta$, scelta non-deterministicamente tra le regole della grammatica aventi A come testa. La mossa compiuta è un'ϵ-transizione che sostituisce il simbolo A in cima allo stack con la stringa β (il corpo della regola di produzione):

$$(q, y, A\alpha) \vdash (q, y, \beta\alpha)$$

- Se invece il simbolo in cima è un terminale, cioè lo stack contiene $a\alpha$ con $a \in T$, il PDA verifica che a sia effettivamente il prossimo simbolo presente nella stringa in input, ed elimina a dalla cima dello stack. Dunque, se $y = az$, la mossa compiuta è:

  $$(q, az, a\alpha) \vdash (q, z, \alpha)$$

  Se invece il prossimo simbolo in input è diverso dal terminale in cima allo stack, cioè $y = bz$ con $b \neq a$, allora la computazione si blocca, e di conseguenza non è una computazione accettante.

Siccome il PDA opera sempre sul simbolo in cima allo stack, la simulazione dell'applicazione delle regole di produzione avviene sempre sul non-terminale più a sinistra, ovvero segue una strategia leftmost, con la quale, come anticipato, si generano nello stack (insieme alla parte letta dell'input) le forme sentenziali sinistre della grammatica.

2.1 Formalizzazione

Data una CFG $G = \langle V, T, \Gamma, S \rangle$, si costruisce il PDA $P_G = \langle \{q\}, T, V \cup T, \delta, q, S \rangle$, che:

- ha un unico stato q, che in quanto tale è anche lo stato iniziale;
- ha come alfabeto di input l'insieme dei terminali della grammatica, poiché le stringhe da riconoscere — quelle del linguaggio $L(G)$ — sono costruite appunto sull'insieme T;
- ha come alfabeto di stack $V \cup T$, perché lo stack deve contenere forme sentenziali sinistre della grammatica, che sono stringhe $\alpha \in (V \cup T)^*$;
- ha come simbolo iniziale di stack il simbolo iniziale della grammatica, dato che le forme sentenziali sono appunto le stringhe derivabili a partire da tale simbolo;
- accetta per stack vuoto, quindi non si indica l'insieme di stati finali, che è irrilevante;
- ha una funzione di transizione δ definita secondo due famiglie di regole:
  - (R1) le regole che simulano le produzioni della grammatica:

    $$\forall A \in V \quad \delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \in \Gamma\}$$

  - (R2) le regole di match, che verificano la corrispondenza tra terminali in cima allo stack e simboli in input:

    $$\forall a \in T \quad \delta(q, a, a) = \{(q, \epsilon)\}$$

Teorema: $N(P_G) = L(G)$, cioè il linguaggio $N(P_G)$ accettato per stack vuoto dal PDA P_G è uguale al linguaggio $L(G)$ generato dalla CFG G.

Osservazione: La costruzione del PDA P_G a partire dalla CFG G richiede tempo lineare nella dimensione della grammatica (ma il PDA ottenuto è non deterministico).

2.2 Esempio: grammatica dei palindromi

Si consideri la solita grammatica dei palindromi su $\{0, 1\}$:

$$G_{pal} = \langle \{P\}, \{0, 1\}, \Gamma, P \rangle \qquad \Gamma = \{P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1\}$$

Secondo la costruzione appena descritta, il PDA corrispondente è

$$P_{pal} = \langle \{q\}, \{0, 1\}, \{0, 1, P\}, \delta, q, P \rangle$$

dove δ definisce le seguenti transizioni:

- transizioni generate da (R1):

$$\delta(q, \epsilon, P) = \{(q, \epsilon), (q, 0), (q, 1), (q, 0P0), (q, 1P1)\}$$

- transizioni generate da (R2):

$$\delta(q, 0, 0) = \{(q, \epsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \epsilon)\}$$

Una delle possibili computazioni di P_{pal} sull'input 0110 è la seguente:

Passo di computazione	Transizione applicata	Regola simulata
$(q, 0110, P)$		
$\vdash (q, 0110, 0P0)$	$\delta(q, \epsilon, P) \ni (q, 0P0)$	$P \to 0P0$
$\vdash (q, 110, P0)$	$\delta(q, 0, 0) \ni (q, \epsilon)$	match su 0
$\vdash (q, 110, 1P10)$	$\delta(q, \epsilon, P) \ni (q, 1P1)$	$P \to 1P1$
$\vdash (q, 10, P10)$	$\delta(q, 1, 1) \ni (q, \epsilon)$	match su 1
$\vdash (q, 10, 10)$	$\delta(q, \epsilon, P) \ni (q, \epsilon)$	$P \to \epsilon$
$\vdash (q, 0, 0)$	$\delta(q, 1, 1) \ni (q, \epsilon)$	match su 1
$\vdash (q, \epsilon, \epsilon)$	$\delta(q, 0, 0) \ni (q, \epsilon)$	match su 0

Siccome P_{pal} accetta per stack vuoto, la stringa 0110 è accettata: $0110 \in N(P_{pal}) = L(G_{pal})$.

2.3 Esempio: grammatica delle espressioni

Si consideri la grammatica delle espressioni semplificate:

$$G_{\text{Exp}} = \langle \{E, I\}, \{+, *, (,), a, b, 0, 1\}, \Gamma, E \rangle$$
$$\Gamma = \left\{ \begin{array}{l} E \to I \mid E + E \mid E * E \mid (E) \\ I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array} \right\}$$

Il PDA corrispondente è

$$P_{\text{Exp}} = \langle \{q\}, \Sigma = \{+, *, (,), a, b, 0, 1\}, \Sigma \cup \{E, I\}, \delta, q, E \rangle$$

con le transizioni

- generate da (R1):

$$\begin{aligned} \delta(q, \epsilon, E) &= \{(q, I), (q, E + E), (q, E * E), (q, (E))\} \\ \delta(q, \epsilon, I) &= \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\} \end{aligned}$$

- generate da (R2):

$$\begin{aligned} \delta(q, +, +) &= \{(q, \epsilon)\} & \delta(q, a, a) &= \{(q, \epsilon)\} \\ \delta(q, *, *) &= \{(q, \epsilon)\} & \delta(q, b, b) &= \{(q, \epsilon)\} \\ \delta(q, (, () &= \{(q, \epsilon)\} & \delta(q, 0, 0) &= \{(q, \epsilon)\} \\ \delta(q,),)) &= \{(q, \epsilon)\} & \delta(q, 1, 1) &= \{(q, \epsilon)\} \end{aligned}$$

Una computazione accettante di P_{Exp} sull'input $a * (a + b)$ è la seguente:

Passo di computazione	Transizione applicata	Regola simulata
$(q, a * (a + b), E)$		
$\vdash (q, a * (a + b), E * E)$	$\delta(q, \epsilon, E) \ni (q, E * E)$	$E \to E * E$
$\vdash (q, a * (a + b), I * E)$	$\delta(q, \epsilon, E) \ni (q, I)$	$E \to I$
$\vdash (q, a * (a + b), a * E)$	$\delta(q, \epsilon, I) \ni (q, a)$	$I \to A$
$\vdash (q, * (a + b), * E)$	$\delta(q, a, a) \ni (q, \epsilon)$	match su a
$\vdash (q, (a + b), E)$	$\delta(q, *, *) \ni (q, \epsilon)$	match su $*$
$\vdash (q, (a + b), (E))$	$\delta(q, \epsilon, E) \ni (q, (E))$	$E \to (E)$
$\vdash (q, a + b), E))$	$\delta(q, (, () \ni (q, \epsilon)$	match su (
$\vdash (q, a + b), E + E))$	$\delta(q, \epsilon, E) \ni (q, E + E)$	$E \to E + E$
$\vdash (q, a + b), I + E))$	$\delta(q, \epsilon, E) \ni (q, I)$	$E \to I$
$\vdash (q, a + b), a + E))$	$\delta(q, \epsilon, I) \ni (q, a)$	$I \to a$
$\vdash (q, + b), + E))$	$\delta(q, a, a) \ni (q, \epsilon)$	match su a
$\vdash (q, b), E))$	$\delta(q, +, +) \ni (q, \epsilon)$	match su $+$
$\vdash (q, b), I))$	$\delta(q, \epsilon, E) \ni (q, I)$	$E \to I$
$\vdash (q, b), b))$	$\delta(q, \epsilon, I) \ni (q, b)$	$I \to b$
$\vdash (q,),))$	$\delta(q, b, b) \ni (q, \epsilon)$	match su b
$\vdash (q, \epsilon, \epsilon)$	$\delta(q,),)) \ni (q, \epsilon)$	match su)

3 Dai PDA alle grammatiche

L'altro verso dell'equivalenza tra CFG e PDA è espresso dal seguente teorema:

Teorema: Dato un PDA $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0 \rangle$ (che accetta per stack vuoto), esiste una CFG G che genera il linguaggio riconosciuto da P, ovvero tale che $L(G) = N(P)$.

Anche per questo teorema si ha una dimostrazione costruttiva, che però non verrà mostrata. Infatti, questo verso dell'equivalenza è poco interessante dal punto di vista applicativo, così come lo era il passaggio dalle espressioni regolari agli automi a stati finiti nell'ambito dei linguaggi regolari.