TDDD14 - Home assignment 2

Jesper Wrang (jeswr740) - 960619-8472 - jeswr740@student.liu.se

2019-05-15

1

(a) $G_{np} = (\{YP, NP, IP\}, \{a, b\}, Rules, YP)$. Förkortningarna står för:

$$YP = \text{Yttre Palindrom}$$
$$NP = \text{Non Palindrom}$$
$$IP = \text{Inre Palindrom}$$

Rules är då följande regler:

$$YP \rightarrow aYPa \mid bYPb \mid NP$$
$$NP \rightarrow aIPb \mid bIPa$$
$$IP \rightarrow NP \mid aIPa \mid bIPb \mid a \mid b \mid \epsilon$$

YP används i fall att de yttre delarna av strängen skulle i ut som ett palindrom. NP ser till att vi faktiskt bryter mot definitionen av ett palindrom någonstans i strängen. Sist så kan vi ha godtyckligt med saker innerst i strängen efter att vi har garanterat att det inte är ett palindrom längre. Detta görs med IP.

(b) $G_3 = (\{Start, A, B, C, X\}, \{a, b, c\}, Rules, Start)$ där *Rules* är följande regler:

$$Start \rightarrow aAa \mid bBb \mid cCc \mid X$$
$$A \rightarrow XAX \mid a$$
$$B \rightarrow XBX \mid b$$
$$C \rightarrow XCX \mid c$$
$$X \rightarrow a \mid b \mid c$$

Start i detta fall är en symbol för att välja vilken bokstav som ska vara på första, sista och mellersta positionen. När en bokstav har valt så går man vidare till antingen A, B eller C. Där så kan man lägga till godtyckligt med bokstäver mellan första och mellersta samt sista och mellersta, vilket görs med hjälp av X. Det enda sättet detta kan terminera på är att välja mittenbokstav till rätt symbol (antingen a, b eller c beroende på vad man valt innan för start- och slutsymbol).

2

(a) LL skulle vara en sträng i L konkatenerat men en annan sträng i L. Dock så betyder det andra språket att det ska vara samma sträng konkatenerat med sig själv, alltså skiljer sig språken. Ett fall där de är samma skulle vara om L bara innehöll en sträng.

Svar: Språken skiljer sig

(b) $L_1^R L_2^R$ är en konkatenering av en omvänd sträng i L_1 med en omvänd sträng i L_2. Denna ihopslagna sträng börjar alltså på sista bokstaven i ordet från L_1 . Dock så börjar ordet $(L_1 L_2)^R$ på den sista bokstaven i ordet från L_2. De är alltså inte samma språk. Ett fall då de är samma kan vara om L_1 och L_2 båda innehåller ett och samma ord.

Svar: Språken skiljer sig

(c) Här är $L_1^R L_2^R$ samma som i förra uppgiften, alltså en omvänd sträng i L_1 konkatenerat men en om omvänd sträng i L_2. Denna sträng börjar på sista bokstaven av ordet i L_1, i mitten har vi första bokstaven av L_1 sedan kommer sista karaktären av order i L_2, i slutet av ordet av vi första bokstaven av ordet i L_2.

$(L_2 L_1)^R$ följer samma struktur som ovan: starta på sista bokstaven i L_1 och sluta på första i L_2. I mitten av ordet får vi också samma sak: första bokstaven av L_1 och sista från L_2. Det är alltså tydligt att ändarna på strängar gör samma transformation av de båda språket. För alla karaktärer mellan första och sista i L_1 och L_2 kan man göra samma argument.

Svar: Språken är samma

(d) L^* är alla möjliga kombinationer av konkateneringar av alla strängar i L. $L^* L$ är samma sak fast man tvingar med en sträng i L på slutet. Dock är dessa språk inte ekvivalenta. Till exempel med $L = \{a\}$ så beskriver L^* språket: $\{\epsilon, a, aa, ...\}$ medan $L^* L$ bara beskriver $\{a, aa, ...\}$. Ett fall då de är samma är då $\epsilon \in L$

Svar: Språken skiljer sig

3

(a) Börja med att välja $s = a^p\, b^{p+1}\, c^{p+2}$ där p är pumplängden. Nu ska vi enligt pumplemmat kunna skriva $s = uvwxy$ för någon uppsättning av u, v, w, x och y så att $|vwx| \leq p$ och $|vx| \geq 1$.

Vi får nu 5 olika fall för vad vwx kan innehålla, och i varje fall kan vi skapa en motsägelse så att den pumpade strängen inte tillhör L_1:

- vwx innehåller bara a:n. Här kan vi pumpa uppåt tills antalet a:n är fler än eller lika med antalet b:n.
- vwx innehåller en blandning av a:n och b:n. Här kan vi pumpa uppåt tills antalet a:n är fler än eller lika med antalet c:n.
- vwx innehåller bara b:n. Här kan vi pumpa uppåt tills antalet b:n är fler än eller lika med antalet c:n.
- vwx innehåller en blandning av b:n och c:n. Här kan vi pumpa neråt tills antalet b:n eller antalet c:n är mindre än eller lika med antalet a:n.
- vwx innehåller bara c:n. Här kan vi pumpa neråt tills antalet c:n är mindre än eller lika med antalet b:n.

(b) Välj $s = 10^p\, 20^p\, 30^p$. Nu vill vi dela upp s så att $s = uvwxy$, $|vwx| \leq p$ och $|vx| \geq 1$. Här får vi flera olika fall beroende på vart vwx hamnar i s:

- $vwx = 10^i$ för något $i < p$. Pumpas detta neråt blir vi av med 1:an och vi får en sträng som innehåller en 2:a, en 3:a och en massa nollor. Denna sträng kan omöjligt vara i L.
- $vwx = 20^i$, samma argument som ovan.
- $vwx = 30^i$, samma argument som ovan.
- $vwx = 0^i$, $i \leq p$. Detta är oberoende av vilken "hög av nollor" man är i. Om detta pumpas neråt förlorar man nollor i en av termerna vilket gör det omöjligt för denna sträng att vara i L. Det går även inte att ändra termerna så att man får en korrekt summa.
- $vwx = 0^i 20^j$, $i+j < p$. Om vi pumpar detta uttryck neråt så förlorar vi nollor i första och/eller andra uttrycket. Det tredje uttrycket har kvar sina nollor vilket gör att detta aldrig kan skrivas som en korrekt summa. Det kan även hända att vi förlorar 2:an då vi pumpar, och då kan man använda argumentet som gjordes i första punkten.
- $vwx = 0^i 30^j$, samma argument som ovan.

Inget av fallen gav en giltig sträng, alltså är L_2 inte kontextfritt.

4

(a) En av strängarna är *acc*. Resten av strängarna följer mönstret: börja på a, sedan n antal b:n och sist $n+1$ antal c:n. Alltså kan språket skrivas som:

$$L(G) = \{acc\} \cup \{ab^n c^{n+1} \mid n \geq 0\}$$

(b) Ta till exempel strängen *abcc*. Detta kan med vänsterderivering skapas på flera olika sätt, exempelvis:

$$S \implies aB \implies abBc \implies abcc$$
$$S \implies aDc \implies abcc$$

Eftersom det finns flera sätt att generera samma sträng på så är G tvetydig.

(c) Vi har redan visat att G är tvetydig (för båda vänster- och högerderivering), alltså kan det inte heller vara en $LR(1)$ grammatik.

(d) Definiera G' som ett språk med följande regler:

$$S \to acc \mid aB$$
$$B \to bBc \mid c$$

Detta är den formella beskrivingen av det som beskriv i (a)-uppgiften. Den beskriver alltså samma språk som G, dock så kan man i varje steg bestämma vilken regel man ska använda, alltså att grammatiken är deterministisk.

5

(a) Att lägga till en extra stack till en PDA gör så att man kan beskriva mer språk än en vanlig PDA som endast en stack. Till exempel kan en 2-stack PDA känna igen språket:

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

genom att lägga till antalet a:n i båda stackarna, och sedan jämföra första stacken med antalet b:n och andra stacken med antalet c:n. Detta är inte möjligt med en PDA som endast har 1 stack.

(b) En Turingmaskin med en stack kan beskriva exakt samma språk som en Turingmaskin utan en separat stack. Detta är eftersom man alltid kan simulera en stack på en Turingmaskin. Det ger alltså inget att lägga till en stack till en Turingmaskin.

6

(a) Antag att funktionen f är en mappning från L till sig själv, alltså att:

$$x \in L \iff f(x) \in L$$

Eftersom $f(x) = x$ så är detta reflexiv relation.

(b) Antag att funktionen f är en mappning från L_1 till L_2 och att g är en mappning från L_2 till L_1, alltså att:

$$x \in L_1 \iff f(x) \in L_2$$
$$x \in L_2 \iff g(x) \in L_1$$

Dock så kan vi inte garantera att detta detta är symmetrisk för språken L_1 och L_2 då detta kräver att de följande också är sant:

$$x \in L_1 \iff f(x) \in L_1$$
$$x \in L_2 \iff g(x) \in L_2$$

Vilket endast är sant då $L_1 = L_2$, detta är alltså inte (generellt) en symmetrisk relation.

(c) Antag att funktionen f är en mappning från L_1 till L_2 och att g är en mappning från L_2 till L_3, alltså att:

$$x \in L_1 \iff f(x) \in L_2$$
$$x \in L_2 \iff g(x) \in L_3$$

Vi kan nu skapa funktionen $h(x) = f(g(x))$ som är en (transitiv) mappning från L_1 till L_3, då:

$$\begin{aligned} x \in L_1 &\iff f(x) \in L_2 \iff \\ &\iff g(f(x)) \in L_3 \iff \\ &\iff h(x) \in L_3 \end{aligned}$$

Mappningsreduktion är alltså en transitiv relation.