המרת ביטוי רגולרי לאוטומט סופי דטרמיניסטי

והמרת אוטומט סופי דטרמיניסטי לביטוי רגולרי

על סמך משפט Kleene

מגיש	ליאור וונש
מס' זהות	
סמסטר	סמסטר ב' 2020
מחלקה	הנדסת תוכנה
מנחה	ד"ר רודה יואב

תוכן עניינים

רקע כללי

אוטומט

תיאוריית האוטומטים היא לימוד מכונות ואוטומטים מופשטים, כמו גם בעיות חישוב שניתן לפתור באמצעותן. זוהי תיאוריה במדעי המחשב התיאורטיים ובמתמטיקה דיסקרטית. אוטומט מורכב ממצבים (המיוצגים על ידי מעגלים) ומעברים (המיוצגים על ידי חצים). כאשר האוטומט רואה סמל קלט, הוא מבצע מעבר (או קפיצה) למצב אחר, על פי פונקציית המעבר שלו, שלוקחת את המצב הנוכחי ואת הסמל האחרון ככניסותיו. תיאוריית האוטומטים קשורה קשר הדוק לתורת השפה הפורמלית.

אוטומט הוא ייצוג סופי של שפה רשמית שעשויה להיות סדרה אינסופית של מילים. אוטומטים מסווגים לרוב לפי סוג השפות הפורמליות שהם יכולים לקבל, שמומחשות בדרך כלל על ידי ההיררכיה של חומסקי, המתארת את היחסים בין שפות שונות וסוגי לוגיקה פורמלית. אוטומטים ממלאים תפקיד מרכזי בתיאוריה של חישוב, בניית מהדרים, בינה מלאכותית, ניתוח ואימות תוכנה. אוטומט הוא מבנה העשוי ממצבים שנועדו לקבוע אם יש לקבל או לדחות את הקלט. זה נראה כמו משחק לוח בסיסי בו כל חלל בלוח מייצג מדינה. לכל מדינה יש מידע על מה לעשות כאשר מתקבל קלט על ידי המכונה. כאשר המכונה מקבלת קלט חדש, היא מסתכלת על המצב ובוחרת נקודה חדשה על סמך המידע מה לעשות כאשר היא מקבלת קלט זה במצב זה. כשאין כניסות נוספות, האוטומט נעצר והמרחב שהוא נמצא בו כשהוא מסתיים קובע אם האוטומט מקבל או דוחה את מערך הקלטים המסוים הזה.

ביטוי רגולרי

ביטוי רגולרי הוא רצף של תווים המגדירים דפוס חיפוש. בדרך כלל משתמשים בתבניות כאלה על ידי אלגוריתמי חיפוש מחרוזות לצורך פעולות "מצא" או "מצא והחלף" על מחרוזות, או לצורך אימות קלט.

זוהי טכניקה שפותחה במדעי המחשב התיאורטיים ותורת השפה הפורמלית. הרעיון עלה בשנות החמישים כאשר המתמטיקאי האמריקני סטיבן קול קליין הפך את התיאור של שפה רגולרית לרשמי. הרעיון נכנס לשימוש נפוץ עם כלי עיבוד טקסטים של מערכת ההפעלה יוניקס. תחבירים שונים לכתיבת ביטויים רגילים קיימים מאז שנות השמונים, האחד הוא תקן POSIX ואחר, בשימוש נרחב, הוא תחביר Perl. משתמשים בביטויים רגולריים במנועי חיפוש, תהליכי חיפוש והחלפה של מעבדי תמלילים ועורכי טקסט, בכלי עיבוד טקסטים כגון sed ו- AWK ובניתוח לקסיקלי. שפות תכנות רבות מספקות יכולות מובנות או ספריות עבור ביטויים רגולריים. הביטוי ביטויים רגולריים, המכונים גם regexes, משמש לעתים קרובות למשמעות התחביר הטקסטואלי הספציפי, לייצוג תבניות להתאמת טקסט. כל תו בביטוי רגולרי (כלומר, כל תו במחרוזת המתאר את הדפוס שלה) הוא או תו-מטבע, בעל משמעות מיוחדת, או תו רגיל שיש לו משמעות מילולית.
במחקר שלנו נשתמש בביטויים ובאוטומטים עבור האלפבית 'a', 'b'.

רקע תיאורטי

אוטומט סופי דטרמיניסטי

אוטומט סופי דטרמיניסטי (אס"ד, DFA) מוגדר ע"י קבוצה של 5 איברים $< Q, \Sigma, \delta, q_0, F >$:

- Q = קבוצה סופית של מצבים.
- Σ = קבוצה סופית של סמלים, האלפית של האוטומט – אצלנו ‘a’, ‘b’.
- δ = פונקציית המעברים, כלומר $\delta: Q \times \Sigma \to Q$.
- q_0 = המצב ההתחלתי, כלומר המצב של האוטומט לפני שמעבדים קלט כלשהו.
- F = קבוצת המצבים המקבלים של האוטומט, כאשר $F \subseteq Q$.

אוטומט קורא מחרוזת סופית של סמלים $a_1, a_2, \ldots, a_n$, כאשר $a_i \in \Sigma$, המכונה מילת קלט. קבוצת המילים מסומנת על ידי Σ^*. רצף של מצבים $q_0, q_1, \ldots, q_n$, כאשר $q_i \in Q$, כך ש-q_0 הוא מצב ההתחלתי ולכל 0<i<n+1 מתקיים $q_i = \delta(q_i - 1, a_i)$, הוא ריצה של האוטומט במילת קלט $w = a_1 a_2 \ldots a_n \in \Sigma^*$.
במילים אחרות, בהתחלה האוטומט נמצא במצב ההתחלתי q_0, ואז האוטומט קורא סמלים של מילת הקלט ברצף. כאשר האוטומט קורא את סמל a_i הוא קופץ למצב $q_i = \delta(q_i - 1, a_i)$.

אומרים כי q_n הוא המצב הסופי של הריצה. מילה $w = a_1 a_2 \ldots a_n \in \Sigma^*$ מתקבלת ע"י האוטומט אם $q_n \in F$.
אוטומט יכול לזהות שפה פורמלית.
השפה $L \subseteq \Sigma^*$ המוכרת על ידי אוטומט היא הסט של כל המילים שמתקבלות על ידי האוטומט.
השפות הניתנות לזיהוי הן קבוצת השפות המוכרות על ידי אוטומט כלשהו.
השפות הניתנות לזיהוי הן שפות רגולריות, כלומר שפות שבנויות מביטויים רגולריים.

ביטוי רגולרי

ביטויים רגילים מורכבים מקבועים, המסמנים קבוצות של מחרוזות, וסמלי אופרטורים המציינים פעולות על מערכות אלה.

בהינתן אלף-בית סופי Σ, הקבועים הבאים מוגדרים כביטויים רגולריים:

- $\emptyset$ = קבוצה ריקה.
- ε = הקבוצה המכילה רק את המחרוזת "הריקה", שאין לה תווים כלל.
- $\sigma \in \Sigma$ = הקבוצה המכילה רק את התו σ.

בהינתן ביטויים רגולריים R ו- S, הפעולות הבאות עליהם מוגדרות לייצור ביטויים רגולריים:

- (שרשור) RS מציין את מערך המחרוזות שניתן להשיג על ידי שרשור מחרוזת ב- R ומחרוזת ב- S.
  לדוגמה, בהינתן {"R = {"ab", "c, ו- {"S = {"d", "ef
  ניתן להגדיר את השרשור כך - {"RS = {"abd", "abef", "cd", "cef.

- (התחלפות) R | S מציין את האיחוד הקבוע של קבוצות שתוארו על ידי R ו- S.
  לדוגמה, בהינתן R המתאר את {"ab", "c"} ו- S המתאר את {"ab", "d", "ef"}
  ניתן להגדיר את הביטוי R | S כך - {"ab", "c", "d", "ef"}.

- (כוכב Kleene) *R מציין את קבוצת-העל הקטנה ביותר של הקבוצה המתוארת על ידי R המכילה את ε וסגורה תחת שרשור מחרוזות. זאת הקבוצה של כל המחרוזות הניתנות לייצור על ידי שרשור של כל מחרוזת סופית (כולל מחרוזת בגודל אפס) של המחרוזות מהקבוצה המתוארה על ידי R.
  לדוגמא, *{"1" ,"0"} היא הקבוצה של כל המחרוזות הבינאריים הסופיים (כולל המחרוזת הריקה).

כדי להימנע מסוגריים ההנחה היא שלכוכב קליין יש את העדיפות הגבוהה ביותר, לאחר מכן שרשור ואז התחלפות. אם אין עמימות, ייתכן שהסוגריים מושמטים.

לדוגמה, c (ab) יכול להיות כתוב כ- abc, ו- ((* a | (b (c יכול להיות כתוב כ- *a | bc.

אוטומט סופי לא דטרמיניסטי

אוטומט סופי לא דטרמיניסטי ללא מסעי אפסילון (NFA) מוגדר ע"י $< Q, \Sigma, \delta, q_0, F >$:

- Q = קבוצה סופית של מצבים.
- Σ = קבוצה סופית של סמלים, האלפית של האוטומט – אצלנו ‘a’, ‘b’.
- δ = פונקציית המעברים, כלומר $\delta: Q \times \Sigma \rightarrow P(Q)$
  כאשר P מסמן קבוצת החזקה.
- q_0 = המצב ההתחלתי, כלומר המצב של האוטומט לפני שמעבדים קלט כלשהו.
- F = קבוצת המצבים המקבלים של האוטומט, כאשר $F \subseteq Q$.

בהינתן $w = a_1 a_2 \dots a_n$ מחרוזת מעל Σ, האוטומט NFA מקבל את המחרוזת אם קיימת סדרת מצבים $r_0, r_1, \dots, r_n \in Q$ כך ש-

- $r_0 \in q_0$
- $\forall i = 0, 1, \dots, n-1 : r_{i+1} \in \delta(r_i, a_{i+1})$
- $r_n \in F$

המרה של NFA ל- DFA:

בהינתן אסל"ד $NFA =< Q, \Sigma, \delta, q_0, F >$ המוגדר בעמוד הקודם, ניתן לבנות אס"ד $DFA =< Q_D, \Sigma, \delta_D, q_0^D, F_D >$ באופן הבא –

$$Q_D = P(Q)$$
$$q_0^D = \{q_0\}$$
$$F_D = \{B \in P(Q) \mid B \cap F \neq \emptyset\}$$
$$\forall B \in Q_D, \sigma \in \Sigma : \ \delta_D(B, \sigma) = \bigcup_{p \in B} \delta(p, \sigma)$$

אוטומט סופי לא דטרמיניסטי עם מסעי אפסילון

אוטומט סופי לא דטרמיניסטי עם מסעי אפסילון (εNFA) מוגדר ע"י $< Q, \Sigma, \delta, q_0, F >$:

- Q = קבוצה סופית של מצבים.
- Σ = קבוצה סופית של סמלים, האלפבית של האוטומט – אצלנו 'a', 'b'.
- δ = פונקציית המעברים, כלומר $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$
  כאשר P מסמן קבוצת החזקה ו-ε את המילה הריקה.
- q_0 = המצב ההתחלתי, כלומר המצב של האוטומט לפני שמעבדים קלט כלשהו.
- F = קבוצת המצבים המקבלים של האוטומט, כאשר $F \subseteq Q$.

עבור $q \in Q$, נגדיר פונקציה $eClosure(q)$ שמציינת את קבוצת המצבים הישיגים (הניתנים להשגה) ממצב q ע"י מעקב אחר מעברי-ε, כלומר עבור $p \in Q$ יתקיים $p \in eClosure(q)$ אם קיימת סדרת מצבים $q_1, q_2, \dots, q_k$ כך ש-

- $q_1 = q$
- $\forall\, 1 \leq i < k :\ q_{i+1} \in \delta(q_i, \varepsilon)$
- $q_k = p$

בהינתן $w = a_1 a_2 \dots a_n$ מחרוזת מעל Σ, האוטומט εNFA מקבל את המחרוזת אם קיימת סדרת מצבים $r_0, r_1, \dots, r_n \in Q$ כך ש-

- $r_0 \in eClosure(q_0)$
- $\forall i = 0, 1, \dots, n-1 : r_{i+1} \in eClosure(r') \text{ where } r' \in \delta(r_i\,, a_{i+1})$
- $r_n \in F$

המרה של εNFA ל- NFA:

בהינתן אוטומט $\varepsilon NFA =< Q, \Sigma, \delta, q_0, F >$ המוגדר בעמוד הקודם, ניתן לבנות אסל"ד ללא מסעי אפסילון $NFA = < Q, \Sigma, \delta', q_0, F' >$ באופן הבא –

$$F' = \begin{cases} F \cup \{q_0\}, & eClosure(q_0) \cap F = \emptyset \\ F, & otherwise \end{cases}$$

$$\forall q \in Q, \sigma \in \Sigma :\ \delta'(q, \sigma) = \bigcup_{p \in eClosure(q)} \bigcup_{r \in \delta(p,\sigma)} eClosure(r)$$

משפט Kleene

לכל שפה $L \subseteq \Sigma^*$ מתקיים:

L רגולרית

<=>

קיים ביטוי רגולרי $r \in R(\Sigma)$, כך ש- $L[r] = L$

המרת ביטוי רגולרי לאוטומט סופי דטרמיניסטי (Regex-DFA)

מקרי בסיס:

- אם $r = \emptyset$, אזי $L[r] = \emptyset$.
- אם $r = \varepsilon$, אזי $L[r] = \{\varepsilon\}$.
- אם $r = \sigma$ עבור $\sigma \in \Sigma$, אזי $L[r] = \{\sigma\}$.

מקרים כלליים:

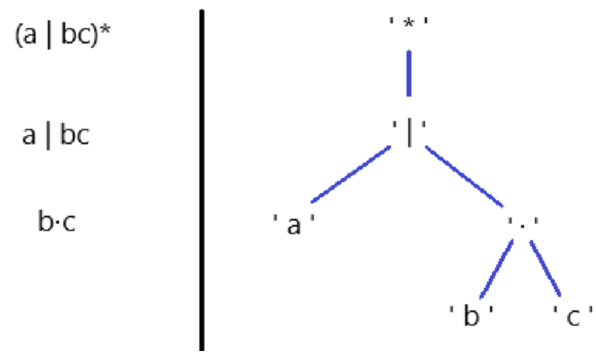
יהיו $r, r_1, r_2 \in R(\Sigma)$ ונניח כי $L[r] = L,\ L[r_1] = L_1,\ L[r_2] = L_2$.

- $L[r_1 \mid r_2] = L[r_1] \cup L[r_2] = L_1 \cup L_2$
- $L[r_1 \cdot r_2] = L[r_1] \cdot L[r_2] = L_1 \cdot L_2$
- $L[r^*] = (L[r])^* = L^*$

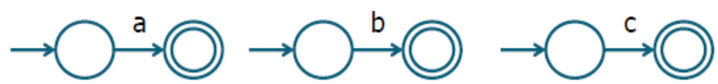
דוגמא להמרת ביטוי רגולרי לאוטומט סופי דטרמיניסטי

$$r = (a \mid bc)^*$$

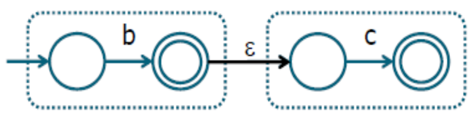
ניתן לייצג את הביטוי באמצעות עץ יצירה לפי סריקת in-order.



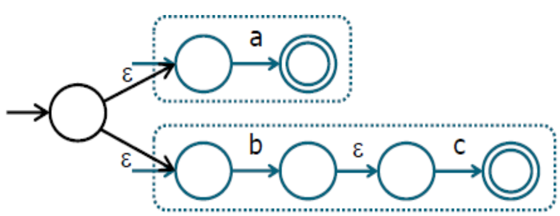
נתחיל ממקרי הבסיס:



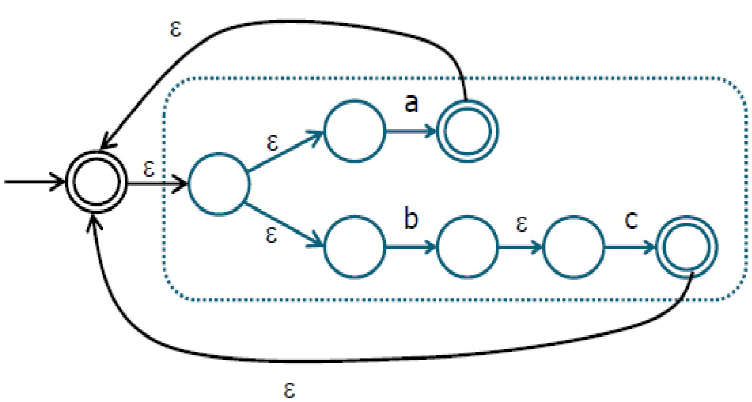
אוטומט עבור $b \cdot c$:



אוטומט עבור $a \mid bc$:



אוטומט עבור $r = (a \mid bc)^*$:



<u>המרת אוטומט סופי דטרמיניסטי לביטוי רגולרי</u> (DFA-Regex)

נניח שכל המצבים שלנו ממוספרים ע"י $\{0, 1, \dots, n\}$ והמצב הוא ההתחלתי הוא 0.

נסמן ב- $R(i, j, k)$ את קבוצת המילים שעוברות ממצב q_i ל-q_j בלי לעבור באף מצב שמספרו $k + 1$ או יותר (חוץ מ-i ו-j עצמם שיכולים להיות גדולים יותר).

לפי ההגדרה של אוטומט ניתן לראות כי:

$i = j \land \exists\sigma{:}\ \delta(i, \sigma) = j \Rightarrow R(i, j, -1) = \varepsilon \mid \sigma$

$i = j \land \nexists\sigma{:}\ \delta(i, \sigma) = j \Rightarrow R(i, j, -1) = \varepsilon$

$i \neq j \land \exists\sigma{:}\ \delta(i, \sigma) = j \Rightarrow R(i, j, -1) = \sigma$

$i \neq j \land \nexists\sigma{:}\ \delta(i, \sigma) = j \Rightarrow R(i, j, -1) = \emptyset$

מסקנה:

$R(i, j, n) =$ קבוצת כל המילים שניתן להגיע בעזרתן ממצב i למצב j ולסיים בו, כאשר ניתן לעבור בכל המצבים בלי הגבלה. לכן, על מנת למצוא את קבוצת כל המילים או בעצם את הביטוי הרגולרי עבור אוטומט מסוים, נאחד בין כל ה-$R(i, j, k)$ כך ש-$i = 0$ מסמן את המצב ההתחלתי ו-j מסמן כל פעם מצב מקבל אחר של האוטומט, עבור $k = n$.

המטרה שלנו היא להוכיח שכל $R(i, j, k)$ יכול להיות מבוטא כ"י ביטוי רגולרי ובכך להראות שקיים ביטוי רגולרי עבור השפה של האוטומט שלנו.

ניתן לעשות זאת ע"י אינדוקציה על k.

בסיס: $k = -1$: $R(i, j, -1)$ יש לו ביטוי רגולרי כפי שהראינו מעלה.

צעד: נניח כי לכל $R(i, j, k)$ יש ביטוי רגולרי, נוכיח כי יש ביטוי רגולרי עבור $R(i, j, k + 1)$ שלוקח אותנו ממצב i למצב j ובדרך משתמש רק במצבים $\{1, 2, \dots, k + 1\}$.

$$i \rightsquigarrow k+1 \rightsquigarrow k+1 \rightsquigarrow k+1 \rightsquigarrow j$$

ניתן לראות כי תתי-המילים שמובילים בין כל שני מצבים משתמשים רק במצבים $\{0, \dots, k\}$.

$$i \overset{R(i,k+1,k)}{\rightsquigarrow} k+1 \overset{R(k+1,k+1,k)}{\rightsquigarrow} k+1 \overset{R(k+1,k+1,k)}{\rightsquigarrow} k+1 \overset{R(k+1,j,k)}{\rightsquigarrow} j$$

לכן, ניתן לבטא את קבוצת המילים $R(i, j, k + 1)$ כך –

$R(i, j, k + 1) = R(i, j, k) \mid R(i, k + 1, k) \cdot R^*(k + 1, k + 1, k) \cdot R(k + 1, j, k)$

לפי הנחת האינדוקציה, לכל ה-R בצד ימין ובצד שמאל של ה- 'או' יש ביטויים רגולריים והשתמשנו באופרטורים של ביטויים רגולריים – ולכן כל הביטוי מהווה ביטוי רגולרי.

צמצום ביטויים רגולריים

מוסכמות לפישוט ביטויים רגולריים בהם נשתמש:

- נשמיט את אופרטור השרשור.
  לדוגמה, נרשום ab במקום $a \cdot b$.

- נוותר על סוגריים ברצף של פעולות שרשור וברצף של פעולות איחוד.
  לדוגמה, במקום $(a \mid b) \mid c$ נרשום $a \mid b \mid c$.

- נקבע סדר קדימויות: כוכבית קליין תהיה ראשונה, שרשור שני ואיחוד אחרון.
  לדוגמה, $(ab) \mid c(d^*) \equiv ab \mid cd^*$.

- נשתמש בזהויות הבאות:
  - $a \mid b \equiv b \mid a$
  - $(\varepsilon \mid a)^* \equiv a^*$
  - $(\varepsilon \mid a) \cdot a^* \equiv a^* \cdot (\varepsilon \mid a) \equiv a^*$
  - $\emptyset \cdot \sigma = \emptyset$
  - $\emptyset \mid \sigma = \sigma$

- נשתמש בהשוואה והכלה של ביטויים רגולריים פשוטים בצורה הבאה:

$$\text{regex1} \mid \text{regex2} \equiv \begin{cases} \text{regex1} & \mid \text{regex1} - \text{היא מחרוזת שמתקבלת מ } \text{regex2} \\ \text{regex2} & \mid \text{regex2} - \text{היא מחרוזת שמתקבלת מ } \text{regex1} \end{cases}$$

מטרת המחקר

המטרה שלנו היא לתכנת את הטרנספורמציה מאס"ד לביטוי רגולרי ואת הטרנספורמציה מביטוי רגולרי לאס"ד ולהדגים בעזרת שימוש במשפט Kleene שהן אכן עובדות.

בנוסף, נשווה בין גדלי המחרוזות של הביטויים הרגולריים המתקבלים מהמרת אס"ד לביטוי רגולרי עם צמצום ביטויים רגולריים לפי המסוכמות שקבענו ובלי צמצום ביטויים רגולריים.

פיתוח תוכנה

- נשתמש בתוכנה Eclipse ובשפה Java.
- נשתמש במקור (6) ע"מ להפוך את קוד מבנה ה-DFA לויזואלי.
- נכתוב את הפונקציות העיקריות הבאות:

• Thompson.compile – פונקציה שמקבלת כקלט ביטוי רגולרי ומחזירה אוטומט סופי לא-דטרמיניסטי עם מסעי אפסילון eNFA שמתאים לאותו ביטוי רגולרי, הפונקציה מבוססת על אלגוריתם שפותח ממשפט Kleene.

• NFA.eNFAtoNFA – פונקציה שמקבלת כקלט אוטומט סופי לא-דטרמיניסטי עם מסעי אפסילון eNFA ומחזירה אוטומט סופי לא-דטרמיניסטי ללא מסעי אפסילון NFA המתאים לו.

• DFA.NFAtoDFA – פונקציה שמקבלת כקלט אוטומט סופי לא-דטרמיניסטי ללא מסעי אפסילון NFA ומחזירה אוטומט סופי דטרמיניסטי DFA המתאים לו.

• DFA.getDelta – פונקציה שמקבלת כקלט אוטומט כלשהו, מצב כלשהו באוטומט ואות ומחזירה את קבוצת כל המצבים שניתן להגיע אליהם מאותו מצב בעזרת האות הזו.

• NFA.getEClosure – פונקציה שמקבלת כקלט אוטומט סופי לא-דטרמיניסטי עם מסעי אפסילון eNFA ומצב כלשהו באוטומט מחזיר את קבוצת כל המצבים שניתן להגיע אליהם מאותו מצב בלי קריאת אות כלשהי (מעבר חופשי).

• Regex.R – פונקציה שמקבלת כקלט אוטומט סופי דטרמיניסטי DFA, רשימת המצבים שלו ממוספרים מ-0 עד n ופרמטרים i,j,k ומחזירה ביטוי רגולרי שמבטא את אותו R, הפונקציה מבוססת על אלגוריתם שפותח ממשפט Kleene.

• Regex.DFAtoRegex – פונקציה שמקבלת כקלט אוטומט סופי דטרמיניסטי DFA ומחזירה את הביטוי הרגולרי המתאים לו.

• Regex.isAccepted – פונקציה שמקבלת כקלט אוטומט סופי דטרמיניסטי DFA ומחרוזת מעל האלפבית שלו ומחזירה האם המחרוזת מתקבלת מקריאתה ע"י האוטומט.

• RunAssignment.DFAequalsRegex – פונקציה שמקבלת כקלט אוטומט סופי דטרמיניסטי DFA וביטוי רגולרי ומחזירה האם האוטומט שקול לביטוי הרגולרי ע"י הרצת כל המילים עד 10 תוים מעל האלפבית המוגדר על האוטומט ועל הביטוי במקביל והשוואה של תוצאת כל הרצת מילה.

פיתוח בדיקה

נשתמש ב-Junit Testing, נבנה מחלקת בדיקות ובתוכה המתודה FinalTest.testTransformations.
נבנה רשימה ארוכה של ביטויים רגולריים מוכנים מראש כקלטים תקינים למערכת.

עבור כל ביטוי רגולרי ברשימה:

1. נדפיס את הקלט, הביטוי הרגולרי.

2. נריץ את האלגוריתם להמרת ביטוי רגולרי Regex לאוטומט סופי דטרמיניסטי DFA
תוך שימוש במתודות:
Thompson.compile, NFA.eNFAtoNFA, DFA.NFAtoDFA.

3. נדפיס את האוטומט הסופי דטרמיניסטי שנוצר כתוצאה מאותו ביטוי רגולרי כקוד מבנה להעתקה למקור (6) ע"מ לקבל מחוון ויזואלי.

4. נשווה באמצעות המתודה RunAssignment.DFAequalsRegex בין האוטומט שנוצר לבין הביטוי הרגולרי שיצר אותו ונדפיס את תוצאת ההשוואה.

עבור האוטומט הסופי דטרמיניסטי שהתקבל:

5. נריץ את האלגוריתם להמרת אוטומט סופי דטרמיניסטי DFA לביטוי רגולרי Regex,
תוך שימוש במתודות:
Regex.DFAtoRegex, Regex.R.

* נבצע כאן השוואה של גדלי המחרוזות של הביטויים הרגולריים כאשר משתמשים בצמצום ביטויים רגולריים לפי המוסכמות שקבענו וכאשר לא משתמשים בצמצום זה.

6. נדפיס את הביטוי הרגולרי שנוצר כתוצאה מאותו אוטומט סופי דטרמיניסטי.

7. נשווה באמצעות המתודה RunAssignment.DFAequalsRegex בין האוטומט הקלט לבין הביטוי הרגולרי שנוצר ממנו ונדפיס את תוצאת ההשוואה.

תוצאות הבדיקה

```
Input Regex: e
Output DFA:
#states
1
0
#initial
0
#accepting
0
#alphabet
e
a
b
#transitions
0:a>1
1:a>1
0:b>1
1:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: e
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (e|(e(e)*e))
Unreduced Output Regex equals Input DFA ? true


Input Regex: a
Output DFA:
#states
1
0
2
#initial
0
#accepting
2
#alphabet
e
a
b
#transitions
1:a>1
0:b>1
1:b>1
0:a>2
2:a>1
2:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: a
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (a|(e(e)*a)|(a|(e(e)*a)(e)*e))
Unreduced Output Regex equals Input DFA ? true
```

```
Input Regex: (a)
Output DFA:
#states
1
0
2
#initial
0
#accepting
2
#alphabet
e
a
b
#transitions
1:a>1
0:b>1
1:b>1
0:a>2
2:a>1
2:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: a
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (a|(e(e)*a)|(a|(e(e)*a)(e)*e))
Unreduced Output Regex equals Input DFA ? true


Input Regex: a*
Output DFA:
#states
1
0
2
#initial
0
#accepting
0
2
#alphabet
e
a
b
#transitions
2:b>1
0:a>2
0:b>1
2:a>2
1:a>1
1:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: (e|(a|a((a))*a))
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (e|(e(e)*e))|(a|(e(e)*a)|(a|(e(e)*a)(e|a)*e|a))
Unreduced Output Regex equals Input DFA ? true
```

```
Input Regex: (a)*
Output DFA:
#states
1
0
2
#initial
0
#accepting
0
2
#alphabet
e
a
b
#transitions
2:b>1
0:a>2
0:b>1
2:a>2
1:a>1
1:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: (e|(a|a((a))*a))
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (e|(e(e)*e))|(a|(e(e)*a)|(a|(e(e)*a)(e|a)*e|a))
Unreduced Output Regex equals Input DFA ? true


Input Regex: (a*)
Output DFA:
#states
1
0
2
#initial
0
#accepting
0
2
#alphabet
e
a
b
#transitions
2:b>1
0:a>2
0:b>1
2:a>2
1:a>1
1:b>1
Output DFA equals Input Regex ? true
Reduced Output Regex: (e|(a|a((a))*a))
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: (e|(e(e)*e))|(a|(e(e)*a)|(a|(e(e)*a)(e|a)*e|a))
Unreduced Output Regex equals Input DFA ? true
```

```
Input Regex: aa
Output DFA:
#states
1
2
0
3
#initial
0
#accepting
3
#alphabet
e
a
b
#transitions
0:a>2
3:a>1
3:b>1
0:b>1
2:b>1
1:a>1
1:b>1
2:a>3
Output DFA equals Input Regex ? true
Reduced Output Regex: aa
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: ((a|(e(e)*a)(e)*a)|((a|(e(e)*a)(e)*a)(e)*e))
Unreduced Output Regex equals Input DFA ? true
```

```
Input Regex: (a)a
Output DFA:
#states
1
2
0
3
#initial
0
#accepting
3
#alphabet
e
a
b
#transitions
0:a>2
3:a>1
3:b>1
0:b>1
2:b>1
1:a>1
1:b>1
2:a>3
Output DFA equals Input Regex ? true
Reduced Output Regex: aa
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: ((a|(e(e)*a)(e)*a)|((a|(e(e)*a)(e)*a)(e)*e))
Unreduced Output Regex equals Input DFA ? true
```

```
Input Regex: a(a)
Output DFA:
#states
1
2
0
3
#initial
0
#accepting
3
#alphabet
e
a
b
#transitions
0:a>2
3:a>1
3:b>1
0:b>1
2:b>1
1:a>1
1:b>1
2:a>3
Output DFA equals Input Regex ? true
Reduced Output Regex: aa
Reduced Output Regex equals Input DFA ? true
Unreduced Output Regex: ((a|(e(e)*a)(e)*a)|((a|(e(e)*a)(e)*a)(e)*e))
Unreduced Output Regex equals Input DFA ? true
```

<u>הערה</u>

רשימת הביטויים הרגולריים נרחבת הרבה יותר ממה שראינו כאן, אך הניסויים בוצעו ונבדקו עבור כל הביטויים הרגולריים ברשימה.

מסקנות

באמצעות משפט Kleene, אכן הצלחנו להראות שהטרנספורמציות בין ביטוי רגולרי לאס"ד עובדות. הצלחנו ליצור אלגוריתם נכון להמרת ביטוי רגולרי לאוטומט סופי דטרמיניסטי ואלגוריתם נכון נוסף להמרת אוטומט סופי דטרמיניסטי לביטוי רגולרי וגם לאשש את נכונותם ע"י בדיקת השוואה המבוססת על Brute-Force.

בנוסף, ניתן לראות את ההבדל המשמעותי בין ביטויים רגולריים כאשר משתמשים במוסכמות הצמצום שהגדרנו וכאשר לא משתמשים בהן.

מקורות

1. אורט בראודה. מצגת בנושא פעולות רגולריות וביטויים רגולריים.
2. אורט בראודה. מצגת בנושא שקילות אס"ד ואסל"ד.
3. Automata and Formal Languages, Yoav Rodeh.
4. https://brilliant.org/wiki/regular-languages/
5. https://brilliant.org/wiki/finite-state-machines/
6. http://ivanzuzak.info/noam/webapps/fsm2regex/