קומפילציה – השלב הראשון:

ניתוח לקסיקלי = וידוא שכל המילים שייכות לשפה

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  6;`
`}`	

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  6;`
`}`
	Build: 1 succeeded

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  6;`
`}`
	Build: 1 succeeded

`void f(int a)`
`{`
`  6b;`
`}`	

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

Build: 1 succeeded	`void f(int a)`
	`{`
	`  6;`
	`}`

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier 'b‘
error C2065: 'b' : undeclared identifier
	`void f(int a)`
	`{`
	`  6b;`
	`}`

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  0r;`
`}`	

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier ‘r‘
error C2065: ‘r' : undeclared identifier
	`void f(int a)`
	`{`
	`  0r;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier ‘r‘
error C2065: ‘r' : undeclared identifier
	`void f(int a)`
	`{`
	`  0r;`
	`}`

	`void f(int a)`
	`{`
	`  0x;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier ‘r‘
error C2065: ‘r' : undeclared identifier
	`void f(int a)`
	`{`
	`  0r;`
	`}`

error C2153: hex constants must have at least one hex digit	`void f(int a)`
	`{`
	`  0x;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier ‘r‘
error C2065: ‘r' : undeclared identifier
	`void f(int a)`
	`{`
	`  0r;`
	`}`

error C2153: hex constants must have at least one hex digit	`void f(int a)`
	`{`
	`  0x;`
	`}`

	`void f(int a)`
	`{`
	`  0u;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2059: 'bad suffix on number‘
error C2146: missing ';' before identifier ‘r‘
error C2065: ‘r' : undeclared identifier
	`void f(int a)`
	`{`
	`  0r;`
	`}`

error C2153: hex constants must have at least one hex digit	`void f(int a)`
	`{`
	`  0x;`
	`}`

Build: 1 succeeded	`void f(int a)`
	`{`
	`  0u;`
	`}`

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  9000000000000000000;`
`}`	

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  9000000000000000000;`
`}`
	error C2177: constant too big

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  9000000000000000000;`
`}`
	error C2177: constant too big

`void f(int a)`
`{`
`  int @gmail=0;`
`}`	

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2177: constant too big	`void f(int a)`
	`{`
	`  9000000000000000000;`
	`}`

error C2018: unknown character '0x40'	`void f(int a)`
	`{`
	`  int @gmail=0;`
	`}`

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  123.45.67;`
`}`	

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  123.45.67;`
`}`
	error C2143: missing ';' before 'constant'

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(int a)`
`{`
`  123.45.67;`
`}`
	error C2143: missing ';' before 'constant'

`void f(int a)`
`{`
`  123e;`
`}`	

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

error C2143: missing ';' before 'constant'	`void f(int a)`
	`{`
	`  123.45.67;`
	`}`

error C2021: expected exponent value, not ';'	`void f(int a)`
	`{`
	`  123e;`
	`}`

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(void)`
`{`
`  0x123456789;`
`}`	

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(void)`
`{`
`  0x123456789;`
`}`
	Build: 1 succeeded

ניתוח לקסיקלי – חימום

קוד	שגיאות (?)

`void f(void)`
`{`
`  0x123456789;`
`}`
	Build: 1 succeeded

`void f(void)`
`{`
`  int a=0x123456789;`
`}`	

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

Build: 1 succeeded	`void f(void)`
	`{`
	`  0x123456789;`
	`}`

warning C4305: truncation from '__int64' to 'int'
warning C4309: truncation of constant value
	`void f(void)`
	`{`
	`  int a=0x123456789;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

	`void f(void)`
	`{`
	`  int a=0x00000000000000000000000007;`
	`}`

ניתוח לקסיקלי – חימום

שגיאות (?)	קוד

Build: 1 succeeded	`void f(void)`
	`{`
	`  int a=0x00000000000000000000000007;`
	`}`

ניתוח לקסיקלי – תיאור

- קובץ C הוא למעשה משפט שמורכב ממילים בשפה
- תפקיד המנתח הלקסיקלי, השלב הראשון בקומפילציה, הוא לוודא שאכן כל המילים בשפה. וכך גם בעברית: מי שמצפלב חושב שיש סרפפ – שימו לב שגם תוכנת המצגת עושה את זה, אבל המשימה שלה יותר קלה – למה?
- מה הן המילים בשפת C?

מילים חוקיות בשפת C

Legal Tokens	Example
Constants	123, 90, 222, … 19.7, 13e+8, … 0x80, 0xabcd, …
Identifiers	numStudentsInMTA, strcpy,
Reserved Keywords	Int, float, char, double, … If, while, do, goto, … struct, class, typedef, …
Parentheses	{,},(,),[,]
Binary Operators	+,-,*,/,=,->,==, …
Unary operators	-,*,++,
Comments	/* … */, //

דרוש: מנגנון (יעיל) המאפשר זיהוי מספר בשפת C

- למה אי אפשר לעשות משהו כמו מילון אם רוצים לבדוק האם לפנינו מספר חוקי?
- רצף של ספרות הוא תמיד מספר? 000088
- חיובי\שלילי (מה אם אין סימן?)
- נקודה עשרונית (מותרת נקודה אחת ולא יותר)
- ייצוג כבסיס + מעריך (מה לגבי 34.56E+5.66)
- ייצוג אקסהדצימלי (מה לגבי 0xAAA)

<u>ביטויים רגולריים</u> כמנגנון ייצוג של מספר אינסופי* (נו, בערך) של מילים בשפת C

- למשל, ביטוי רגולרי המאפשר זיהוי של identifiers בשפת C:
  [_ a-z A-Z][0-9 a-z _ A-Z]*
- או למשל מספר בייצוג הקסה-דצימלי: (מה האורך המותר?)
  [0][xX][0-9 a-f A-F]+
- מה הביטוי הרגולרי שמגדיר float עם בסיס ואקספוננט?
- ומה לגבי מספר unsigned?
- אפשר להשתכנע בקלות שאכן <u>מילים בשפת C</u> (ובכל שפת תכנות קיימת) <u>ניתנות לייצוג על ידי ביטוי רגולרי מתאים</u>. נו יופי, אז ייצגנו כביטוי רגולרי. עכשיו מה עושים?

ביטויים רגולריים – תזכורת

בהינתן אלפבית כלשהו Σ, שפה מעליו, היא אוסף כלשהו של מחרוזות (מחרוזת היא <u>תמיד סופית</u>).

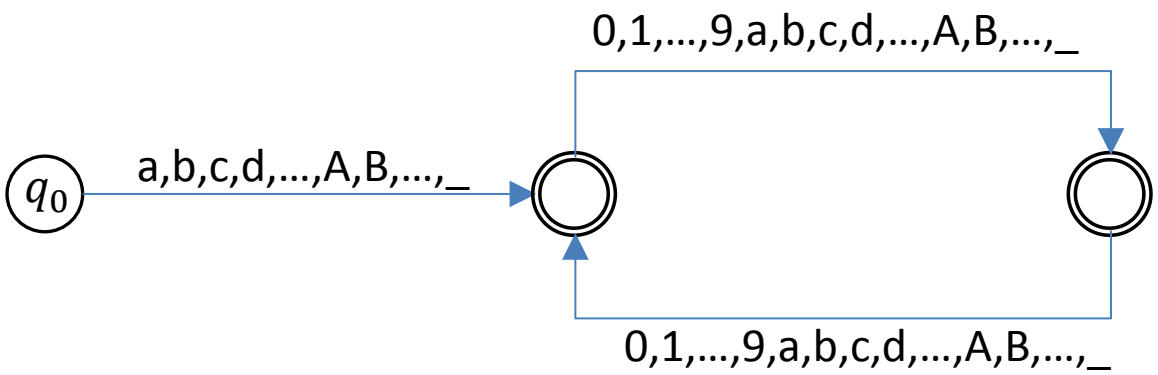
- בהינתן אלפבית כלשהו Σ, ביטוי רגולרי R מייצג שפה L(R) מעל האלפבית באופן הבא:
  - הביטוי הרגולרי a מייצג את השפה $\{a\}$
  - הביטוי הרגולרי ϵ (המחרוזת הריקה) מייצג את השפה $\{\epsilon\}$
  - הביטוי הרגולרי $\emptyset$ מייצג את השפה הריקה
- בהינתן שני ביטויים רגולריים R_1, R_2 הפעולות הבאות מוגדרות ליצירת ביטוי רגולרי חדש:
  - שרשור: R_1R_2 מייצג את השפה $\{w_1w_2 \mid w_i \in L(R_i)\}$
  - איחוד: $R_1 \mid R_2$ מייצג את השפה $L(R_1) \cup L(R_2)$
  - הסגור של קלין: R_1^* מייצג שפה המורכבת משרשורים (סופיים!) כלשהם של מילים מתוך $L(R_1)$

החבר הכי טוב של ביטוי רגולרי – אוטומט דטרמיניסטי סופי

- מסתבר שלכל ביטוי רגולרי קיים אוטומט דטרמיניסטי סופי שמקבל בדיוק את אותה שפה שהביטוי הרגולרי מייצג.
- ההוכחה לטענה מעלה היא קונסטרוקטיבית, כלומר, בהינתן ביטוי רגולרי, קיים מתכון אשר בונה את האוטומט הנ"ל.
- נו, אז עכשיו יש אוטומט דטרמיניסטי סופי שמקבל את אותה שפה שהביטוי הרגולרי המקורי ייצג. מה עכשיו?

אוטומט דטרמיניסטי סופי – תזכורת

- אוטומט דטרמיניסטי סופי ('אוטומט' בקיצור) הוא <u>גרף מכוון</u>, ובו <u>קודקוד התחלה</u> (q_0). בהינתן קלט כלשהו (מילה) נקרא אותו משמאל לימין וכל אות נעבור לקודקוד חדש על פי פונקציית מעבר הנמצאת על הקשתות. אם סיימנו במצב מקבל (עיגול כפול בציור) אז המילה בשפה.
- למשל, אוטומט המאפשר זיהוי של identifiers בשפת C:

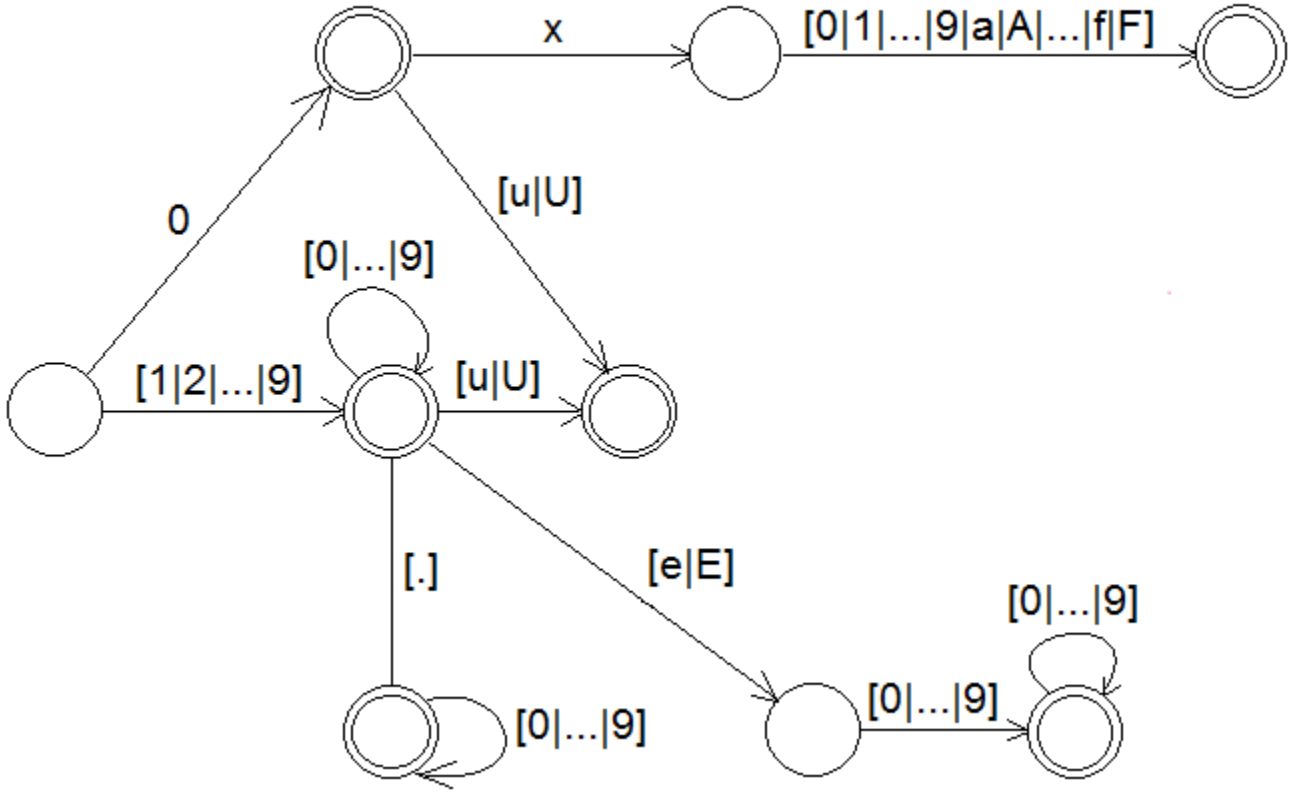


- מה קורה אם האות הראשונה היא # לאן עוברים??

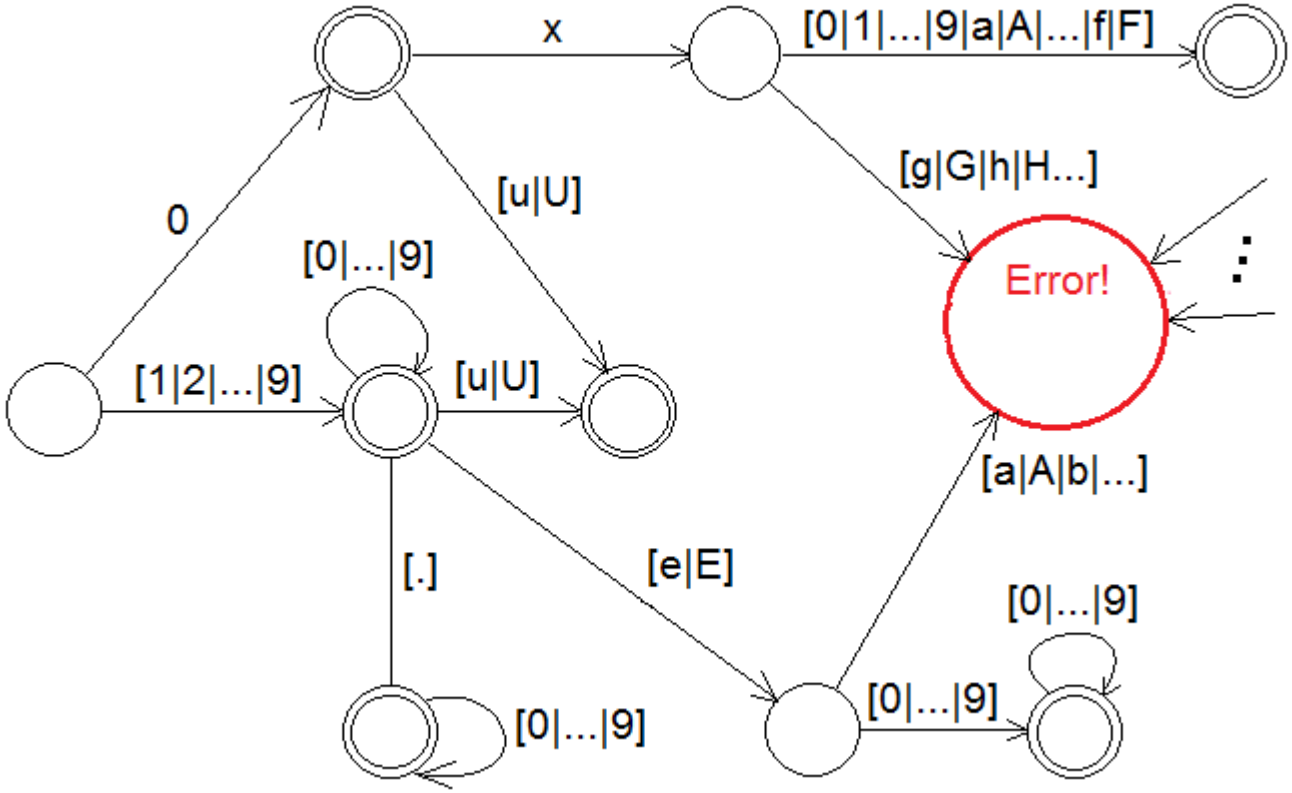
אוטומט דטרמיניסטי סופי – הגדרה פורמלית

- אוטומט דטרמיניסטי סופי הוא חמישיה סדורה $(Q, \Sigma, \delta, q_0, F)$ כאשר Q היא קבוצת המצבים, Σ הוא האלפבית, $\delta: V \times \Sigma \to V$ היא פונקציית המעבר, $q_0 \in Q$ הוא המצב ההתחלתי, ו $F \subseteq Q$ הוא קבוצת המצבים המקבלים
- נניח שקראנו i אותיות מהקלט והגענו לקודקוד q', ונניח שהוא הבאה בקלט היא σ אז מסתכלים בפונקציית המעבר $\delta: V \times \Sigma \to V$ כדי לדעת לאיזה קודקוד לעבור. נניח שכאן $\delta(q', \sigma) = q''$ אז נעבור למצב q'' ונמשיך לקרוא את האות הבאה בקלט

אוטומט דטרמיניסטי סופי – דוגמא
זיהוי מספר בשפת C (האם חסר כאן משהו?)



אוטומט דטרמיניסטי סופי – דוגמא
לא לשכוח טיפול בשגיאות!!!

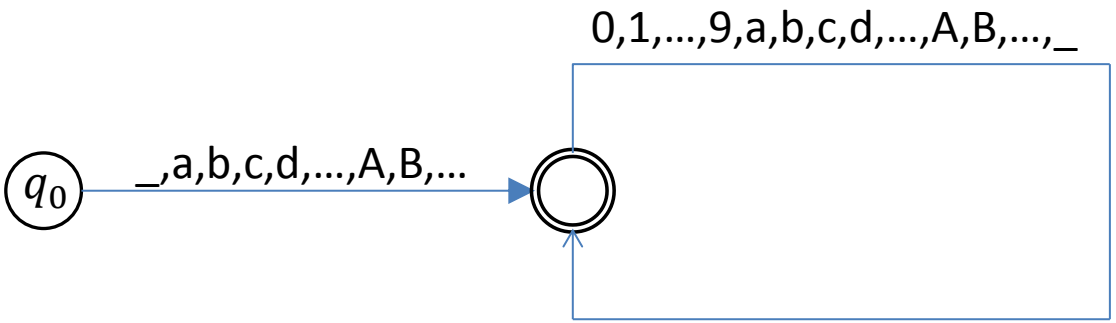


אוטומט דטרמיניסטי סופי – מימוש

- איך הייתם מממשים אוטומט דטרמיניסטי סופי בתוכנה?
- אוטומט דטרמיניסטי סופי הוא פשוט טבלה דו מימדית! השורות זה המצבים, והעמודות זה האותיות.
- כמה גדולה תהיה הטבלה שלנו? האם באמת צריך לשמור עמודה נפרדת עבור הספרה 4 והספרה 5?

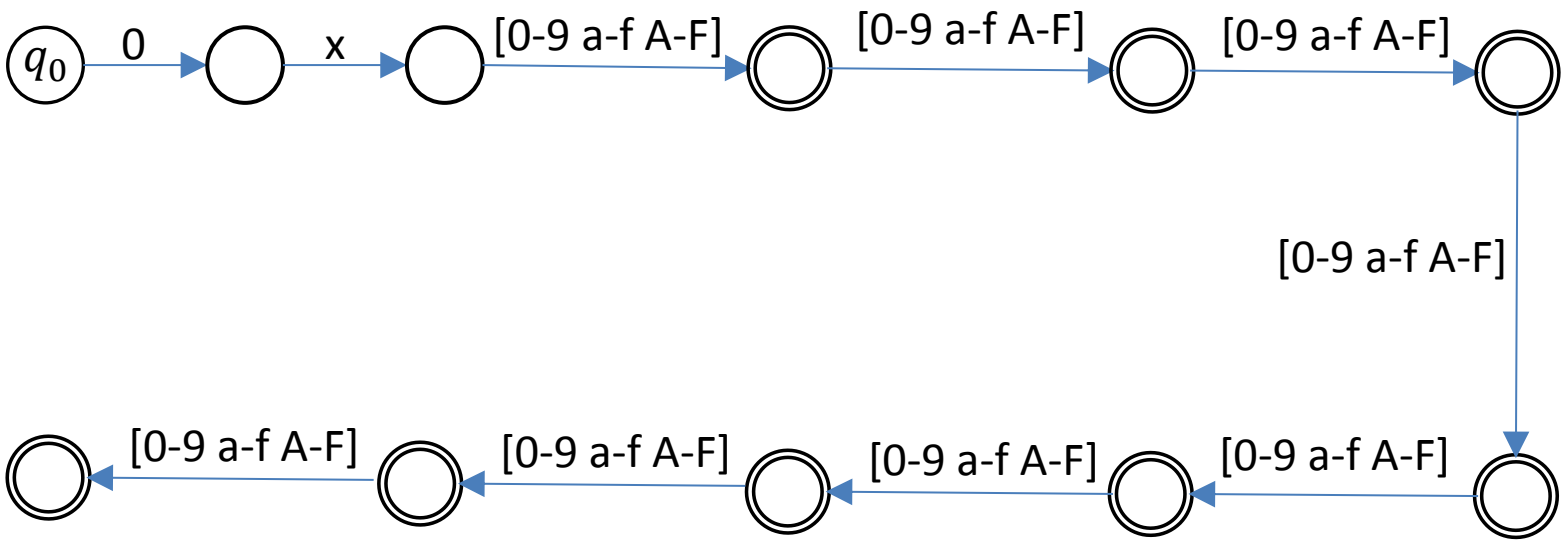
ייצוג ביטוי רגולרי כאוטומט דטרמיניסטי סופי – דוגמא

- אם ניקח את הביטוי שמתאר identifiers בשפת C, הרי שדי קל להפוך אותו לאוטומט:
- הביטוי: *[_ a-z A-Z][0-9 _ a-z A-Z]
- האוטומט:



ייצוג ביטוי רגולרי כאוטומט דטרמיניסטי סופי – עוד דוגמא

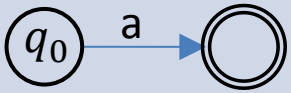
- נסתכל על ייצוג Hex בשפת C. גם אותו די קל להפוך לאוטומט:
- **הביטוי:** [0][x][0-9 a-f A-F][0-9 a-f A-F]? … [0-9 a-f A-F]?
- **האוטומט:**



- מה יכול להיות החיסרון של כזה ייצוג?
- האם כך באמת מיוצגים מספרי HEX בשפת C? (עוד נתייחס לזה בהמשך)

ייצוג ביטוי רגולרי כאוטומט דטרמיניסטי סופי – המקרה הכללי

- ביטויים רגולריים הם חיה רקורסיבית, ולכן, מספיק להראות איך הופכים כל ביטוי רגולרי פרימיטיבי לאוטומט, ואיך מבטאים את שלושת הפעולות באמצעותן מרכיבים ביטויים רגולריים: שרשור, איחוד, והסגור של קלין
- ביטויים רגולריים פרימיטיביים כאוטומטים:

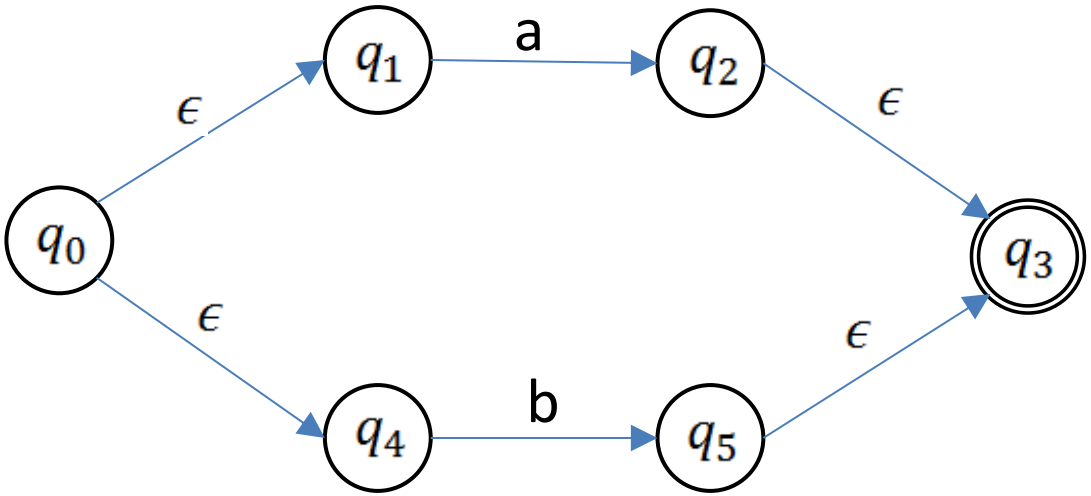
ביטוי רגולרי	ϵ	$\emptyset$	a
האוטומט המתאים			

ייצוג ביטוי רגולרי כאוטומט דטרמיניסטי סופי – המקרה הכללי

- כדי להסביר כיצד מייצגים ביטוי רגולרי כלשהו על ידי אוטומט, אנחנו צריכים להציג את הנושא של מעברי אפסילון, ואוטומטים לא דטרמיניסטיים.
- מעבר אפסילון באוטומט היא קשת מכוונת בין שני מצבים, המאפשרת מעבר (מכוון) בין מצב אחד לשני, גם מבלי שנקראה אף אות מהקלט. אוטומט שיש בו מעברי אפסילון נהפך ללא דטרמיניסטי, מכיוון שבמהלך קריאת הקלט משמאל לימין, יתכן וישנה נקודת זמן בה נמצא ביותר ממצב אחד.
- כלומר, בכל נקודת זמן של קריאת הקלט אנחנו בקבוצת מצבים אפשריים, ולא רק במצב יחיד כמו באוטומט דטרמיניסטי רגיל
- בשקף הבא נראה דוגמא פשוטה למעברי אפסילון, ולאוטומט לא דטרמיניסטי

מעברי אפסילון ואוטומטים לא דטרמיניסטיים

- נניח שאנו רוצים למצוא אוטומט לא דטרמיניסטי המייצג את הביטוי הרגולרי: $a \mid b$
- נוכל לעשות זאת די בקלות, אם מותר להשתמש במעברי אפסילון ובאי דטרמיניזם: ϵ



אוטומטים לא דטרמיניסטיים סופיים – הצגה פורמלית

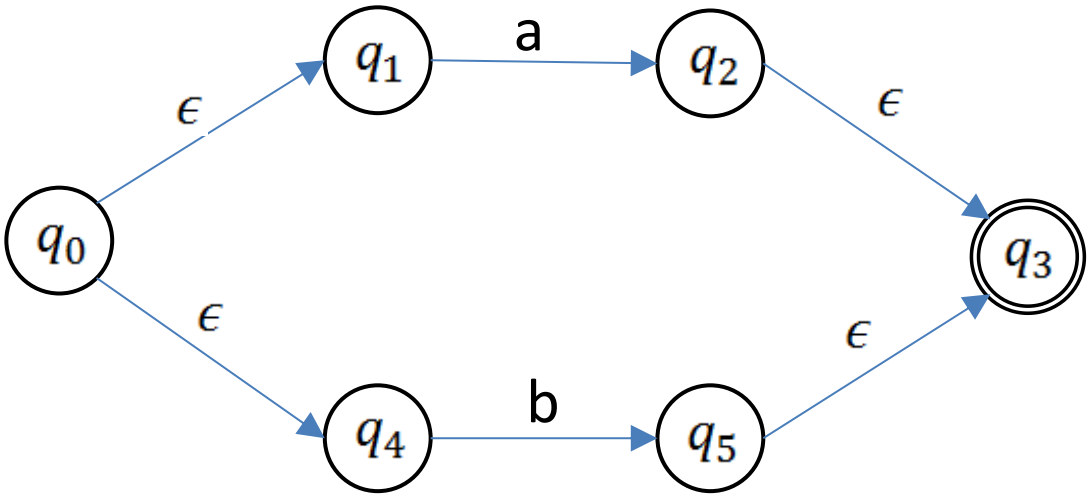
- אוטומט לא דטרמיניסטי סופי הוא חמשיה סדורה: $\{Q, \Sigma, \Delta, q_0, F\}$ כאשר Q היא קבוצת המצבים, Σ הוא האלפבית, $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$ היא פונקציית המעברים, $q_0 \in Q$ הוא המצב ההתחלתי, ו $F \subseteq Q$ היא קבוצת המצבים המקבלים. ϵ הוא הקלט הריק.
- כל אוטומט לא דטרמיניסטי סופי מגדיר שפה מעל האלפבית Σ – שפת המילים שהוא מקבל. כלומר, מילים שאם נריץ אותן על האוטומט נסיים <u>בקבוצת מצבים שמכילה מצב מקבל</u>.

דטרמיניזציה של אוטומטים!

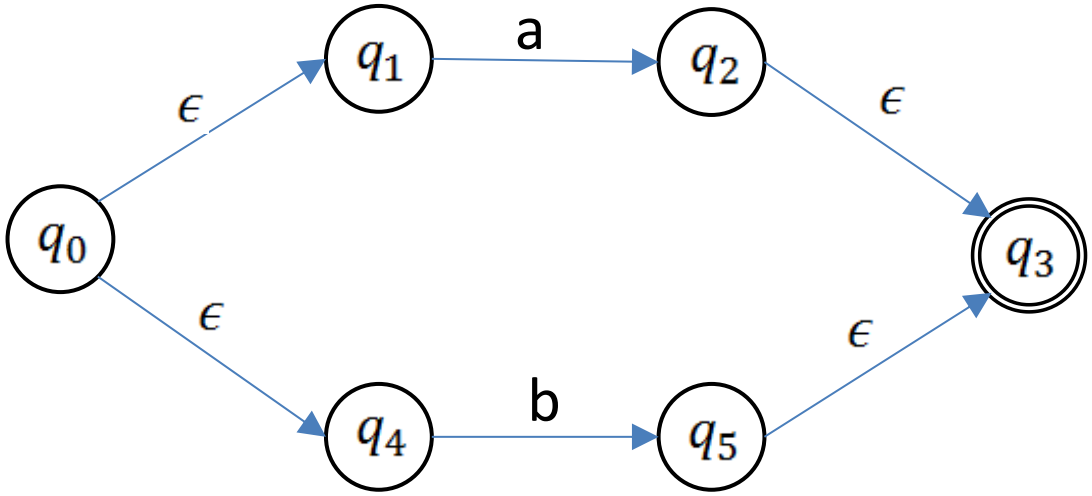
- נו יופי, רצינו להגיע מביטויים רגולריים לאוטומטים דטרמיניסטיים סופיים, והגענו מביטויים רגולריים לאוטומטים לא דטרמיניסטיים סופיים. מה עכשיו?
- מסתבר שכל אוטומט לא דטרמיניסטי, אפשר להפוך לאוטומט דטרמיניסטי!
- איך? בשקף הבא!

דטרמיניזציה של אוטומטים!

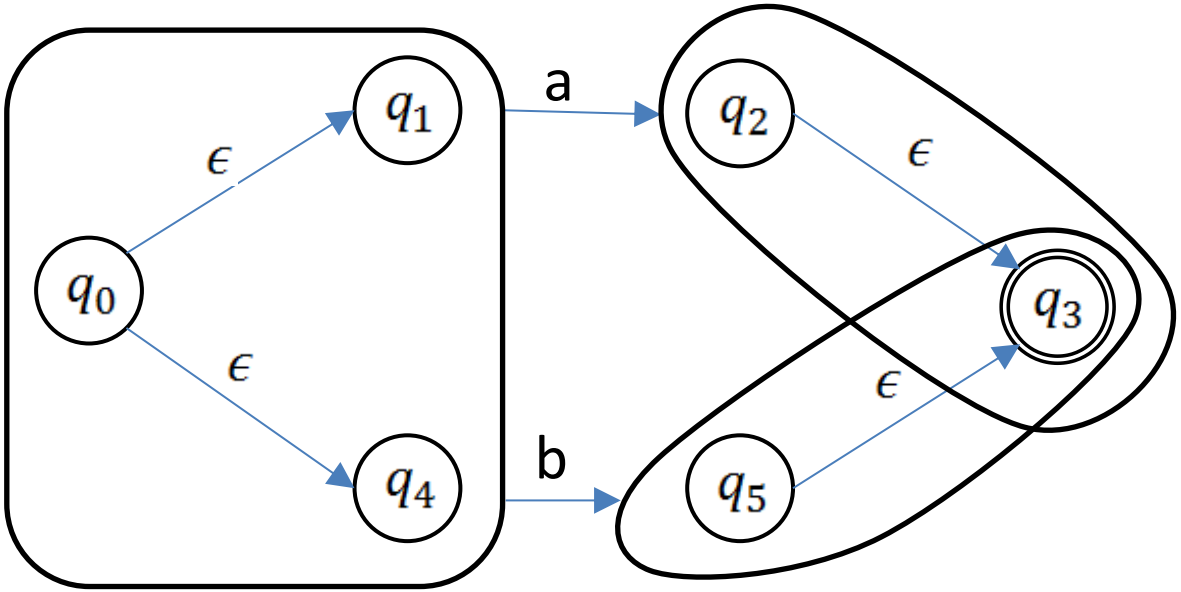
- נשתמש בדוגמא משני שקפים קודם:

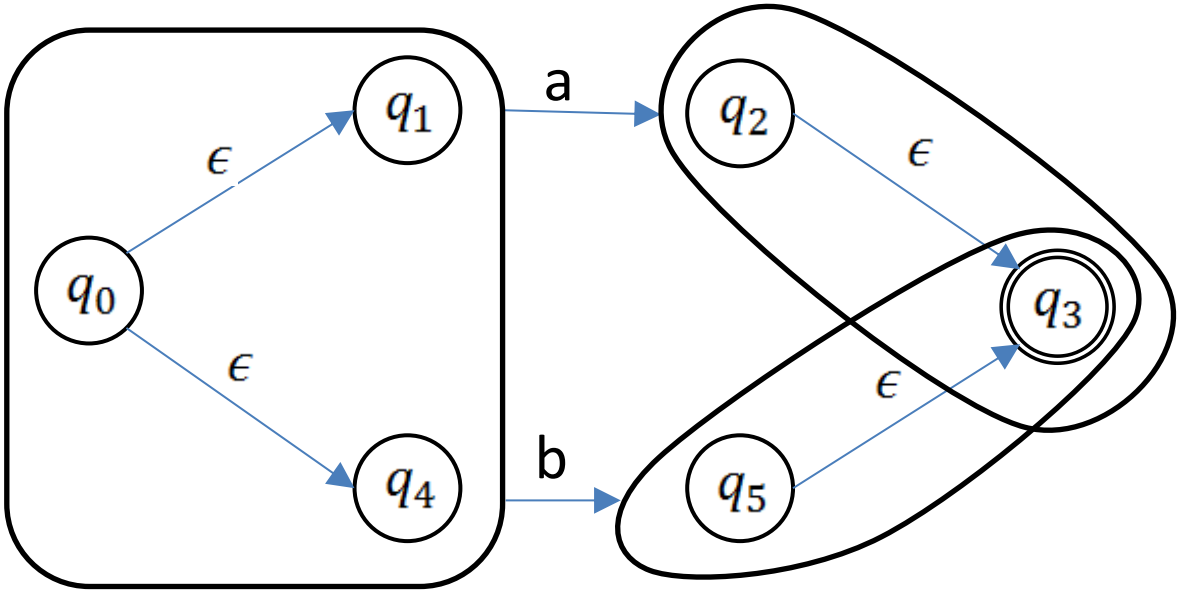


- בהתחלת הריצה, אנחנו יכולים להיות במצבים 0,1,4.
- אם ראינו את האות a אז אנחנו במצב 2 או 3
- אם ראינו את האות b אז אנחנו במצב 5 או 3
- אם הסתיים הקלט, ואנחנו בקבוצת מצבים שיש בה מצב מקבל, אז נקבל את המילה.
- בכל מצב אחר לדחות.

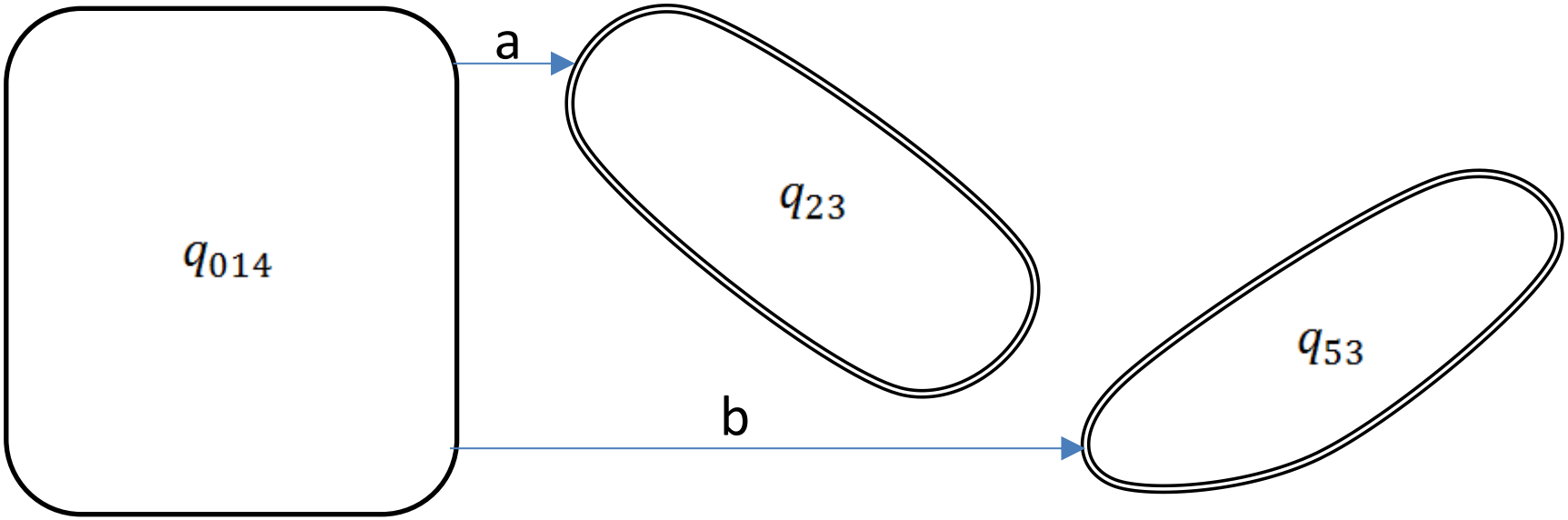


- כלומר עברנו מהאוטומט הלא דטרמיניסטי מעלה, לאוטומט הדטרמיניסטי למטה





- ואותו אפשר לרשום בנוחות כך:



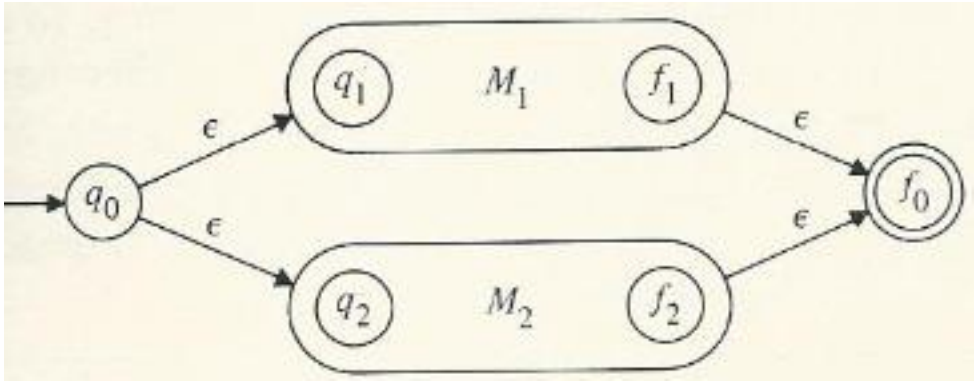
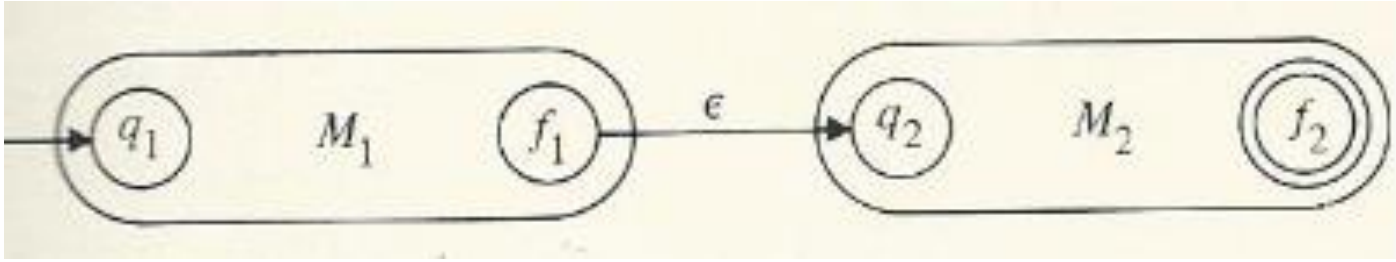
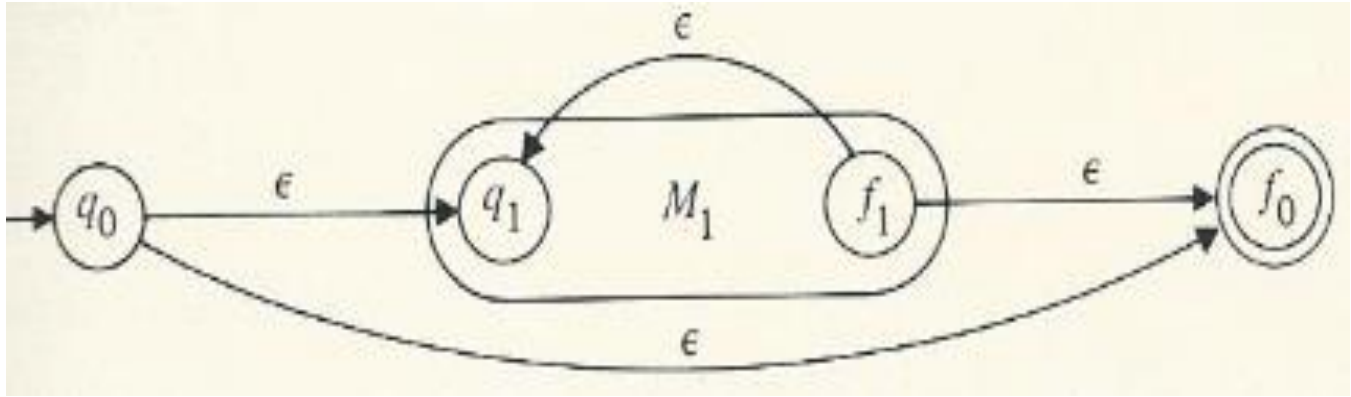
מביטוי רגולרי לאוטומט דטרמיניסטי סופי – תוכנית פעולה

1. בהינתן ביטוי רגולרי, נמצא אוטומט לא דטרמיניסטי סופי שמייצג אותו
2. ניקח את האוטומט הלא דטרמיניסטי שמצאנו, ונעשה לו דטרמיניזציה
3. כשיהיה לנו ביד אוטומט דטרמיניסטי סופי, נוכל לממש אותו בקלות בכל שפת תכנות, על ידי טבלה דו מימדית!

מביטוי רגולרי לאוטומט לא דטרמיניסטי (שלב 1 בשקף הקודם)

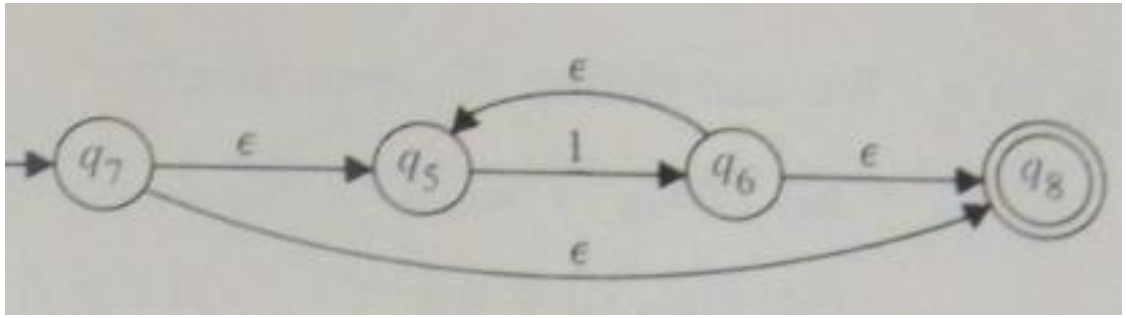
- כאמור, ביטויים רגולריים הם חיה רקורסיבית, אשר מורכבים מהפעלה חוזרת של אחת משלוש פעולות על ביטויים רגולריים פרימיטיביים.
- ראינו קודם מה האוטומט המתאים לכל אחד מהביטויים הפרימיטיביים, ולכן נותר לתאר איך כל אחת משלוש הפעולות (שרשור, איחוד והסגור של קלין) ניתנת לביטוי כפעולה בין אוטומטים.
- השקף הבא עושה בדיוק את זה

מביטוי רגולרי לאוטומט לא דטרמיניסטי

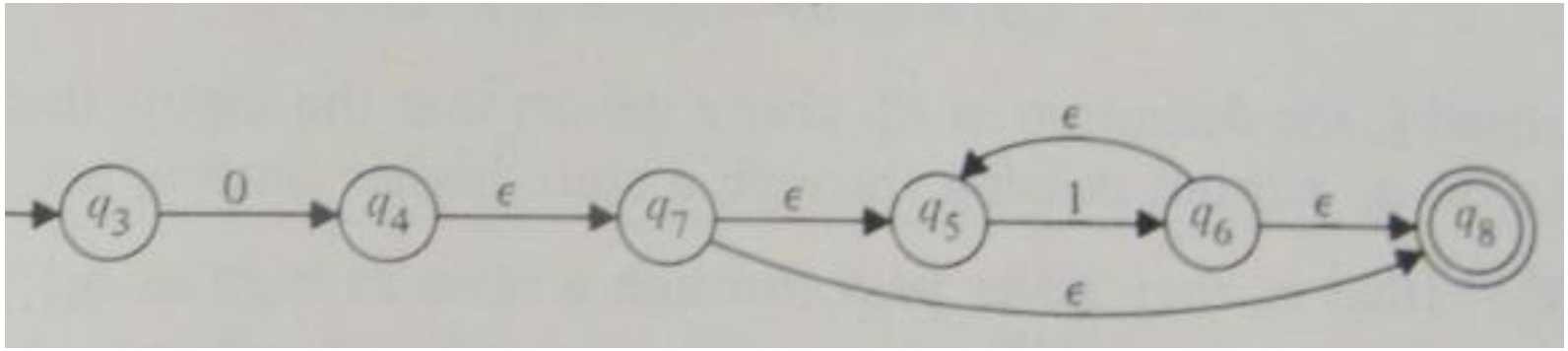
	איחוד של שני ביטויים רגולריים
	שרשור של שני ביטויים רגולריים
	הסגור של קלין של ביטוי רגולרי

תרגיל כיתה: הפכו את הביטויים הבאים לאוטומטים לא דטרמיניסטיים סופיים

- $R_1 = 1^*$

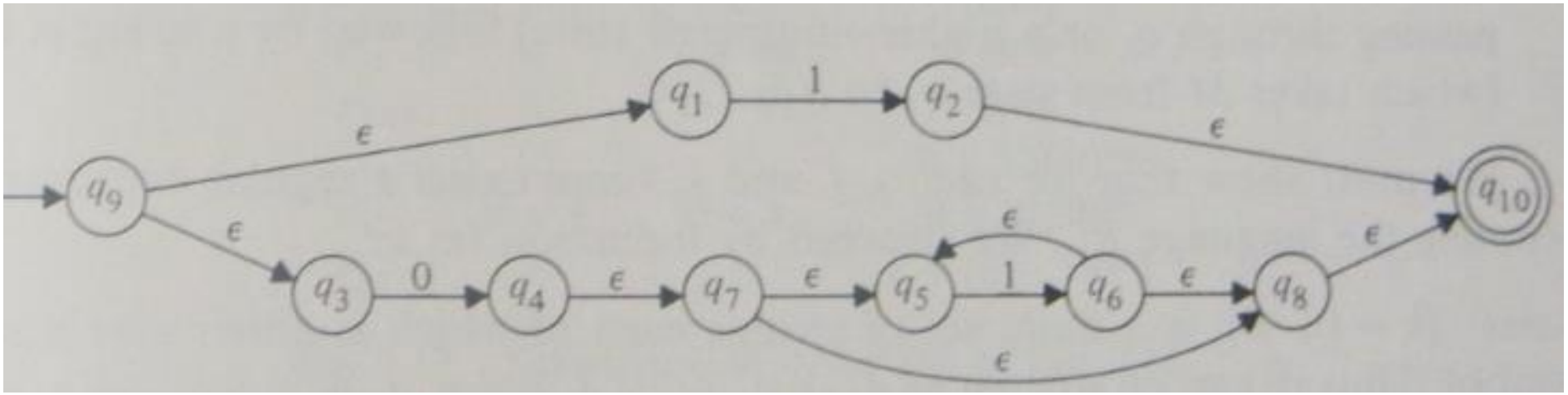


- $R_2 = 01^*$



תרגיל כיתה: הפכו את הביטויים הבאים לאוטומטים לא דטרמיניסטיים סופיים

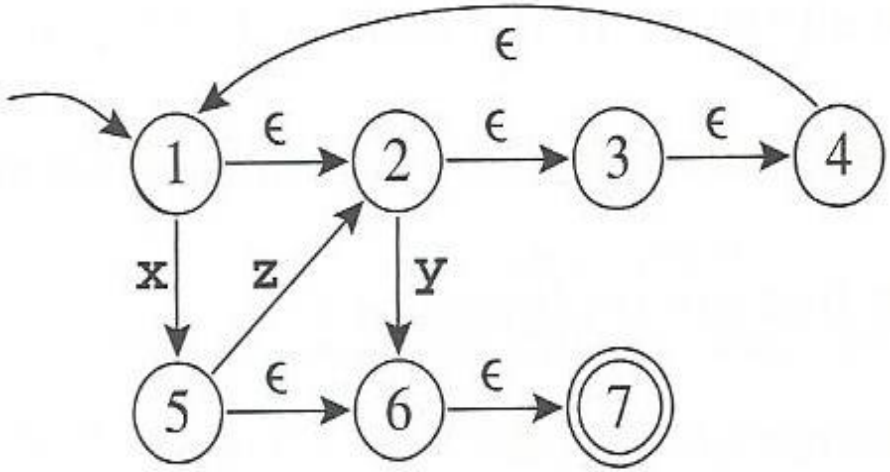
- $R_3 = 01^*|1$



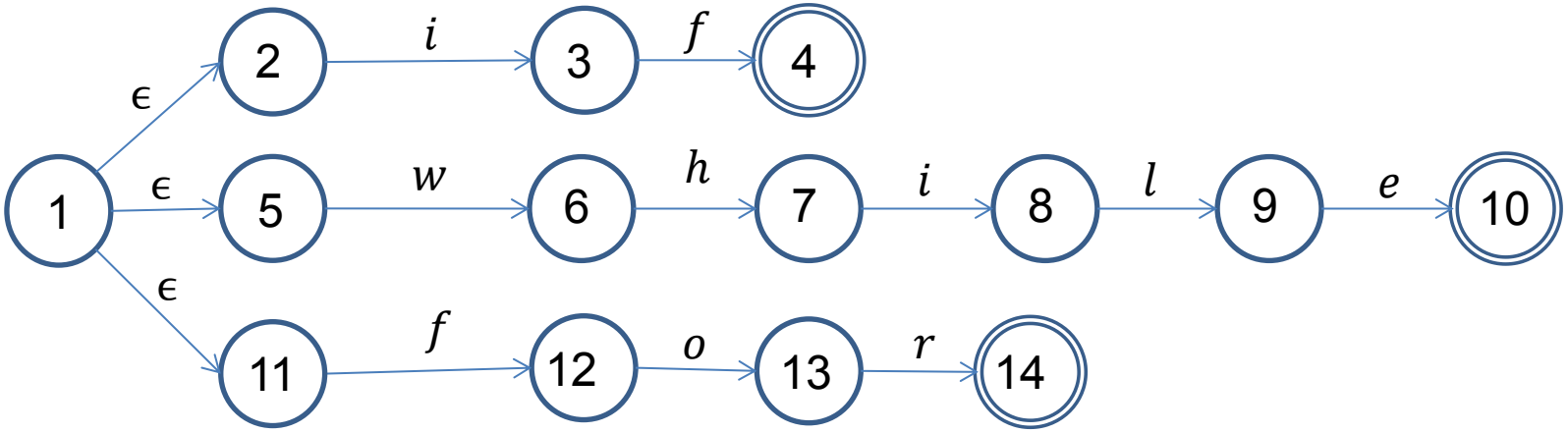
מאוטומט לא דטרמיניסטי סופי לאוטומט דטרמיניסטי סופי

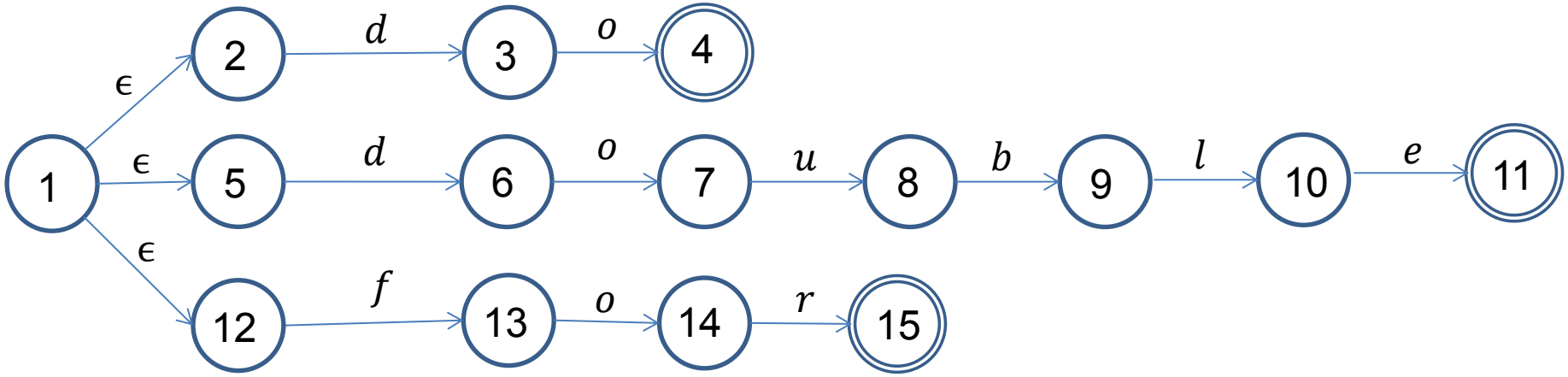
1. קבוצת המצבים היא קבוצת החזקה של המצבים באוטומט הלא דטרמיניסטי
2. המצב ההתחלתי הוא הסגור-אפסילון של המצב ההתחלתי
3. פונקציית המעבר תעבוד כך: לכל מצב בקבוצה, נחשב את קבוצת המצבים שהוא עובר אליה וניקח את הסגור אפסילון שלה. אחר כך ניקח את האיחוד של כל הקבוצות האלה.
4. מצב מקבל הוא קבוצה של מצבים באוטומט הלא דטרמיניסטי, שהכילה מצב מקבל
5. תרגיל כיתה: הפכו את האוטומטים הבאים לדטרמיניסטיים

דוגמאות לאוטומטים לא דטרמיניסטיים סופיים



דוגמאות לאוטומטים לא דטרמיניסטיים סופיים





בניית מנתח לקסיקלי מהחיים האמתיים – סיכום

- בנו ביטויים רגולריים המתארים את סוגי המילים השונות הקיימות בשפת התכנות (identifiers, reserved keywords, hexadecimal numbers, float numbers, strings ועוד). שימו לב: הסיווג נעשה כדי לבנות ביטויים פשוטים ככל הניתן.
- במידה ויש התנגשויות, כלומר אותה מילה יכולה להתקבל על ידי יותר מאוטומט אחד (המילה for), יש להגדיר עדיפות!

מילים בשפת C הן ביטויים רגולריים!

- הנה הקובץ שמתאר את הביטויים הרגולריים של שפת C:
- http://www.lysator.liu.se/c/ANSI-C-grammar-l.html
- שימו לב כמה פשוט הוא כל ביטוי. שימו לב לגמישות התיאור: למשל, אם רוצים לאסור על מספר שלם להתחיל באפסים, או שבמקום הסימן = נשתמש בחץ להשמה (a ← 80) השינויים האלה יכולים להיעשות מהר מאוד.

איך זה קורה במציאות:
ביטויים רגולריים ←
אוטומט לא דטרמיניסטי סופי ←
אוטומט דטרמיניסטי סופי

- בכל שפת תכנות שימושית כיום, המילים החוקיות הן ביטויים רגולריים.
- הצעדים הנ"ל הם פשוטים מספיק להיעשות באופן אוטומטי: המשתמש (ממציא השפה) יתאר את המילים החוקיות כאוסף של ביטויים רגולריים. התוכנה (Flex) תהפוך את האוסף לאוטומט לא דטרמיניסטי סופי. אחר כך, היא תהפוך אותו לאוטומט דטרמיניסטי סופי ותוציא כפלט את פונקציית המעברים כטבלה

Flex – Fast Lexical Analyzer

- Flex הוא מנתח לקסיקלי שמקבל קובץ של ביטויים רגולריים המתארים מילים חוקיות בשפה, ומחזיר קובץ בשפת C המממש פונקציה אחת בלבד ()yylex. פונקציה זאת קוראת מהקלט מילה ומחזירה האם היא בשפה.
- ליתר דיוק, כל ביטוי רגולרי מתאר מילה חוקית בשפה <u>מסוג מסוים</u> (identifier, float, int, string etc.) והפונקציה ()yylex הממומשת על ידי Flex תחזיר את סוג המילה, במידה והיא בשפה, או שגיאה, אם המילה לא בשפה.

דוגמא לקלט של Flex

```
" "                                  {adjust(); continue;}
\n                                   {adjust(); EM_newline(); continue;}
","                                  {adjust(); return COMMA;}
";"                                  {adjust(); return SEMICOLON;}
for                                  {adjust(); return FOR;}
while                                {adjust(); return WHILE;}
"("                                  {adjust(); return LPAREN;}
")"                                  {adjust(); return RPAREN;}
"["                                  {adjust(); return LBRACK;}
"]"                                  {adjust(); return RBRACK;}
"{"                                  {adjust(); return LBRACE;}
"}"                                  {adjust(); return RBRACE;}
"+"                                  {adjust(); return PLUS;}
"-"                                  {adjust(); return MINUS;}
"*"                                  {adjust(); return TIMES;}
"/"                                  {adjust(); return DIVIDE;}
"="                                  {adjust(); return EQ;}
[0-9]+[0-9]*                         {adjust(); yylval.ival=atoi(yytext); return INT;}
[a-zA-Z]+[0-9a-zA-Z]*                {adjust(); yylval.sval=String(yytext); return ID;}
@                                    {adjust(); EM_error(EM_tokPos,"illegal token");}
```

קלט של Flex – נקודות נוספות

```
%{
#include <string.h>
#include <math.h>
#include "../../COMMON_H_FILES/util.h"
#include "../../COMMON_H_FILES/errormsg.h"
#include "../../BISON_OUTPUT_FILES/tiger.tab.h"
int charPos=1;

int yywrap(void)
{
 charPos=1;
 return 1;
}


void adjust(void)
{
 EM_tokPos=charPos;
 charPos+=yyleng;
}

%}
```

הקרביים של Flex – יצירת ביטוי "או" משני ביטויים

nfa.c × ecs.c dfa.c ccl.c Input.lex

(Global Scope)

```c
/* mkbranch - make a machine that branches to two machines
 *
 * synopsis
 *
 *   branch = mkbranch( first, second );
 *
 *     branch - a machine which matches either first's pattern or second's
 *     first, second - machines whose patterns are to be or'ed (the | operator)
 *
 * Note that first and second are NEITHER destroyed by the operation.  Also,
 * the resulting machine CANNOT be used with any other "mk" operation except
 * more mkbranch's.  Compare with mkor()
 */

int mkbranch( first, second )
int first, second;
    {
    int eps;

    if ( first == NO_TRANSITION )
        return second;

    else if ( second == NO_TRANSITION )
        return first;

    eps = mkstate( SYM_EPSILON );

    mkxtion( eps, first );
    mkxtion( eps, second );

    return eps;
    }
```

הקרביים של Flex – תכונות נוספות

ecs.c × Input.lex

lobal Scope)

```
/* ccl2ecl - convert character classes to set of equivalence classes */

void ccl2ecl()
    {
    int i, ich, newlen, cclp, ccls, cclmec;

    for ( i = 1; i <= lastccl; ++i )
        {
        /* We loop through each character class, and for each character
         * in the class, add the character's equivalence class to the
         * new "character" class we are creating.  Thus when we are all
         * done, character classes will really consist of collections
         * of equivalence classes
         */

        newlen = 0;
        cclp = cclmap[i];

        for ( ccls = 0; ccls < ccllen[i]; ++ccls )
            {
            ich = ccltbl[cclp + ccls];
            cclmec = ecgroup[ich];

            if ( cclmec > 0 )
                {
                ccltbl[cclp + newlen] = cclmec;
                ++newlen;
                }
            }

        ccllen[i] = newlen;
        }
    }
```

הקרביים של Flex – תכונות נוספות

dfa.c × nfa.c ecs.c Input.lex

(Global Scope)

```c
/* epsclosure - construct the epsilon closure of a set of ndfa states
 *
 * synopsis
 *    int *epsclosure( int t[num_states], int *numstates_addr,
 *          int accset[num_rules+1], int *nacc_addr,
 *          int *hashval_addr );
 *
 * NOTES
 *  The epsilon closure is the set of all states reachable by an arbitrary
 *  number of epsilon transitions, which themselves do not have epsilon
 *  transitions going out, unioned with the set of states which have non-null
 *  accepting numbers.  t is an array of size numstates of nfa state numbers.
 *  Upon return, t holds the epsilon closure and *numstates_addr is updated.
 *  accset holds a list of the accepting numbers, and the size of accset is
 *  given by *nacc_addr.  t may be subjected to reallocation if it is not
 *  large enough to hold the epsilon closure.
 *
 *  hashval is the hash value for the dfa corresponding to the state set.
 */

int *epsclosure( t, ns_addr, accset, nacc_addr, hv_addr )
int *t, *ns_addr, accset[], *nacc_addr, *hv_addr;
    {
    register int stkpos, ns, tsp;
    int numstates = *ns_addr, nacc, hashval, transsym, nfaccnum;
    int stkend, nstate;
    static int did_stk_init = false, *stk;

#define MARK_STATE(state) \
trans1[state] = trans1[state] - MARKER_DIFFERENCE;

#define IS_MARKED(state) (trans1[state] < 0)
```

איך משתמשים בפלט של Flex?

- Flex יוצר קובץ C בשם lex.yy.c-i default ובו המימוש של אוטומט דטרמיניסטי סופי. למעשה, הפונקציה היחידה שמיוצאת היא yylex() שמחזירה מספר לפי ה define-ים שבקובץ tiger.tab.h ולפיו אפשר לדעת איזו סוג מילה זוהתה. בסוף הקלט yylex() מחזיר 0.
- שימו לב, אצלכם הקבצים הנ"ל כבר שולבו בפרוייקטי השלד שתקבלו.

הסבר כללי על תרגילי הבית

- קבצים, סביבה, קוד קיים
- הגשה
- עבודה בצוותים
- מקורות באינטרנט