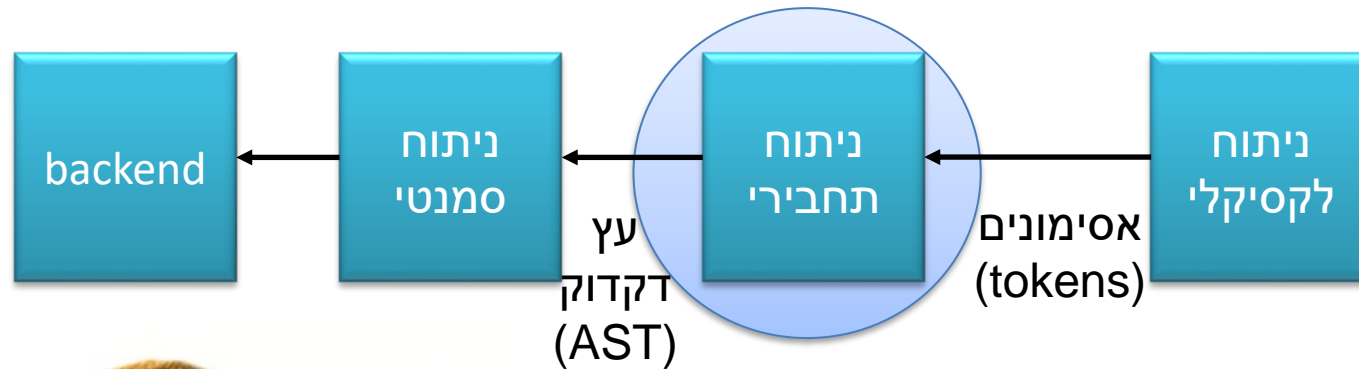


תזכורת מתרגולים אחרונים

- מבנה סכמתי של קומפיילר



- ניתוח תחבירי:

LL(1): Top Down –

:Bottom up –

LR(0) •

SLR •

LR(1) •



LR(1)

- נשתמש ב-lookahead יותר חכם מ-SLR
- נפרק את מצבי LR(0) למצבים עדינים יותר, המכילים גם מידע על lookahead
- עידון של ההקשר: מה מגיע אחרי במסלול ספציפי במקום בכל מקום אפשרי בשפה

$$A \rightarrow \alpha \bullet B \beta \Rightarrow A \rightarrow \alpha \bullet B \beta, t$$

פריטי LR(1)

• ההבדל בין פריטי LR(0) ו-LR(1) נעוץ רק ב-closure.

- פריט LR(0) הוא $(A \rightarrow \alpha \bullet \beta)$ כאשר $A \rightarrow \alpha \beta \in P$
- סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:
 - בסיס: $\text{closure}(I) = I$
 - צעד: אם $(A \rightarrow \alpha \bullet B \beta) \in \text{closure}(I)$ אז לכל $B \rightarrow \gamma \in P$ גם $(B \rightarrow \bullet \gamma) \in \text{closure}(I)$
- פונקצית המעברים של האוטומט:
$$\delta(I, X) = \bigcup \left\{ \text{closure}(A \rightarrow \alpha X \bullet \beta) \mid (A \rightarrow \alpha \bullet X \beta) \in I \right\}$$

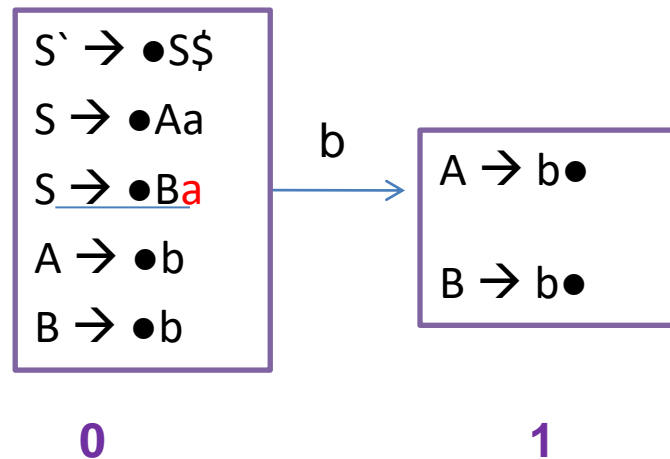
- פריט LR(1) הוא $(A \rightarrow \alpha \bullet \beta, t)$ כאשר $A \rightarrow \alpha \beta \in P, t \in T \cup \{\$ \}$
- סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:
 - בסיס: $\text{closure}(I) = I$
 - צעד: אם $(A \rightarrow \alpha \bullet B \beta, t) \in \text{closure}(I)$ אז לכל $B \rightarrow \gamma \in P$ ולכל $x \in \text{first}(\beta t)$, גם $(B \rightarrow \bullet \gamma, x) \in \text{closure}(I)$
- פונקצית המעברים של האוטומט:
$$\delta(I, X) = \bigcup \left\{ \text{closure}(A \rightarrow \alpha X \bullet \beta, t) \mid (A \rightarrow \alpha \bullet X \beta, t) \in I \right\}$$

משמעות פריט LR(1): זיהינו את מה שמשמאל לנקודה, אנו מצפים למצוא את מה שמימין לה, ולאחר מכן את האסימון המצורף לפריט.

R\R דוגמא - קונפליקט

• אבל עבור הדקדוק הבא, עדיין יש בעיה:

0. $S' \rightarrow S\$$
1. $S \rightarrow Aa$
2. $S \rightarrow Ba$
3. $A \rightarrow b$
4. $B \rightarrow b$



SLR

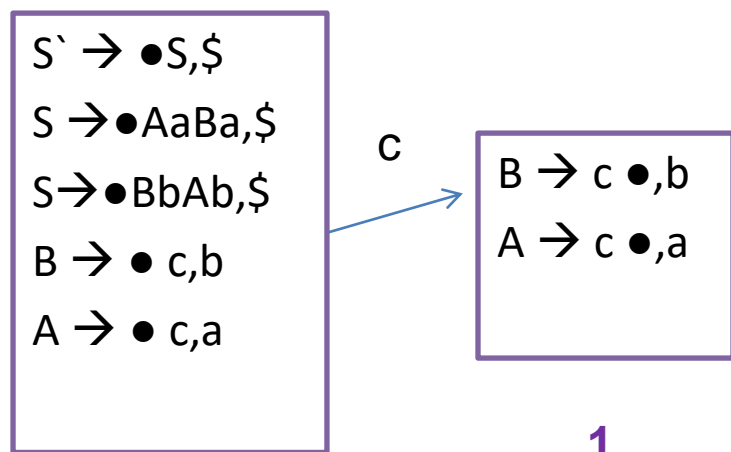
Follow(A)={a}
Follow(B)={a}

	a	b	\$
0		s1	
1	r3, r4		
...			
...			
...			

אינו פותר את הקונפליקט. SLR בדוגמא זו מעבר ל

LR(1) – פתרון קונפליקט R\R

0. $S' \rightarrow S$
1. $S \rightarrow AaBb$
2. $S \rightarrow BbAb$
3. $B \rightarrow c$
4. $A \rightarrow c$



0

1

LR(1)

	a	b	c	\$
0			s1	
1	r4	r3		
...				
...				
...				

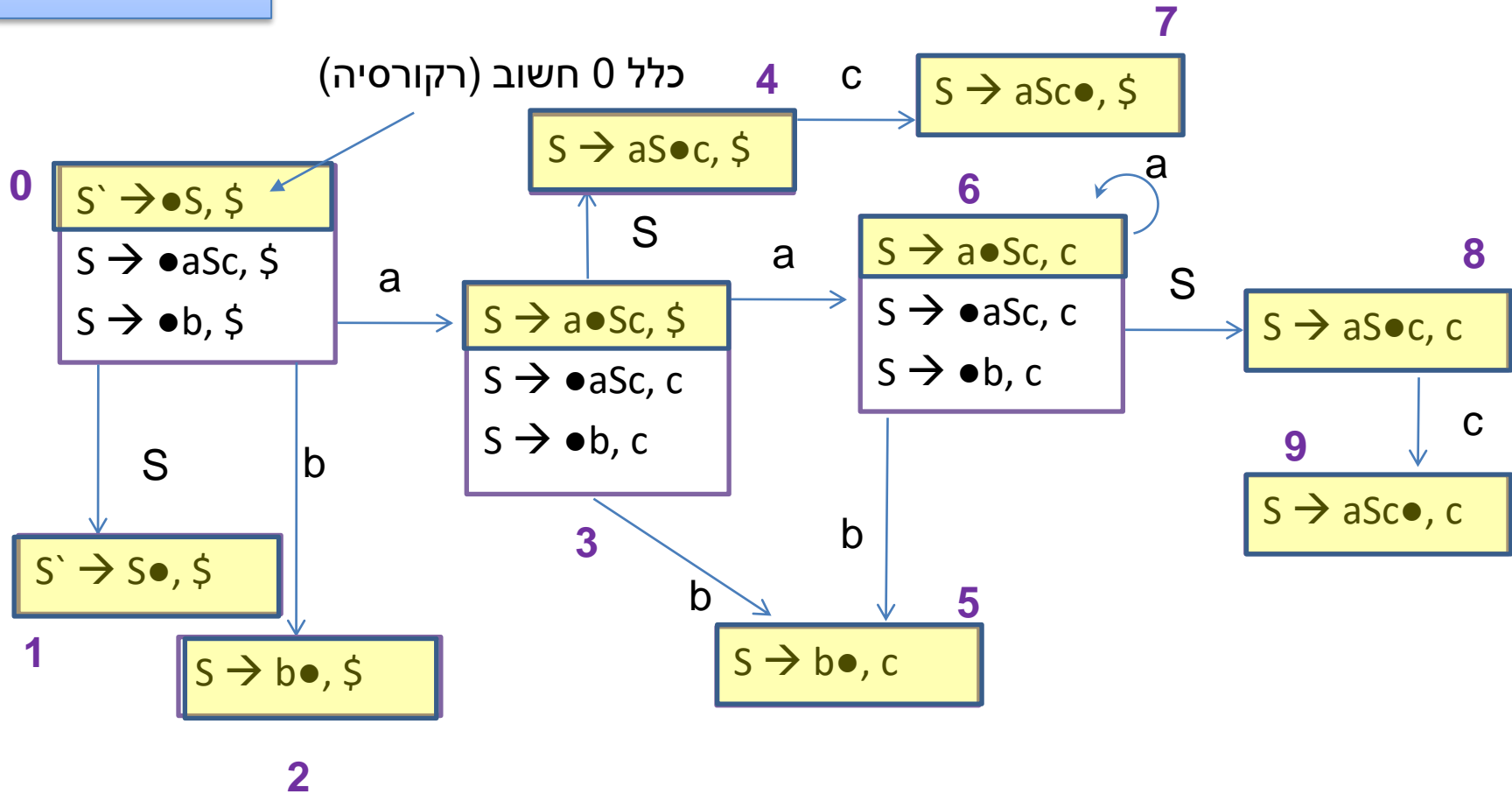
בניית טבלת הניתוח

- LR(0)/SLR כמו goto וטבלת Action בונים טבלת

$$\text{action}[i, t] = \begin{cases} \text{SHIFT}_j & \delta(I_i, t) = I_j \\ \text{REDUCE}_k & \text{rule } k \text{ is } A \rightarrow \alpha, \quad (A \rightarrow \alpha \bullet, t) \in I_i \\ \text{ACCEPT} & (S' \rightarrow S \bullet, \$) \in I_i \text{ and } t = \$ \\ \text{ERROR} & \text{otherwise} \end{cases}$$

1. $S \rightarrow aSc$
2. $S \rightarrow b$

דוגמא נוספת: רקורסיה



*החלקים בצהוב הם הגרעין של המצב.

0. $S' \rightarrow S$

1. $S \rightarrow aSc$

2. $S \rightarrow b$

דוגמא (1) LR- הרצה

הרצה באופן דומה ל LR(0)

actions					goto
	a	b	c	\$	S
0	s3	s2			1
1				acc	
2				r2	
3	s6	s5			4
4			s7		
5			r2		
6	s6	s5			8
7				r1	
8			s9		
9			r1		

מחסנית	פעולה	קלט
(0,)	s3	aabcc\$
(0,)(3,a)	s6	abcc\$
(0,)(3,a)(6,a)	s5	bcc\$
(0,)(3,a)(6,a)(5,b)	r2	cc\$
(0,)(3,a)(6,a)(goto(6,S),S)		
(0,)(3,a)(6,a)(8,S)	s9	cc\$
(0,)(3,a)(6,a)(8,S)(9,c)	r1	c\$
(0,)(3,a)(goto(3,S),S)		
(0,)(3,a)(4,S)	s7	c\$
(0,)(3,a)(4,S)(7,c)	r1	\$
(0,)(goto(0,S),S)		
(0,)(1,S)	acc	\$

Bottom-up בעולם האמיתי

- הכלים הנפוצים לייצור קומפיילרים (compiler)
compilers הם Bottom-up:



Bison



YACC



bison

- bison הוא כלי אוטומטי לייצור מהדרים (RTFM).

– קלט:

- קובץ מקור בשפת bison המגדיר את סכימת התרגום.

– פלט:

- ניתוח תחבירי ב: שיטת Bottom-up

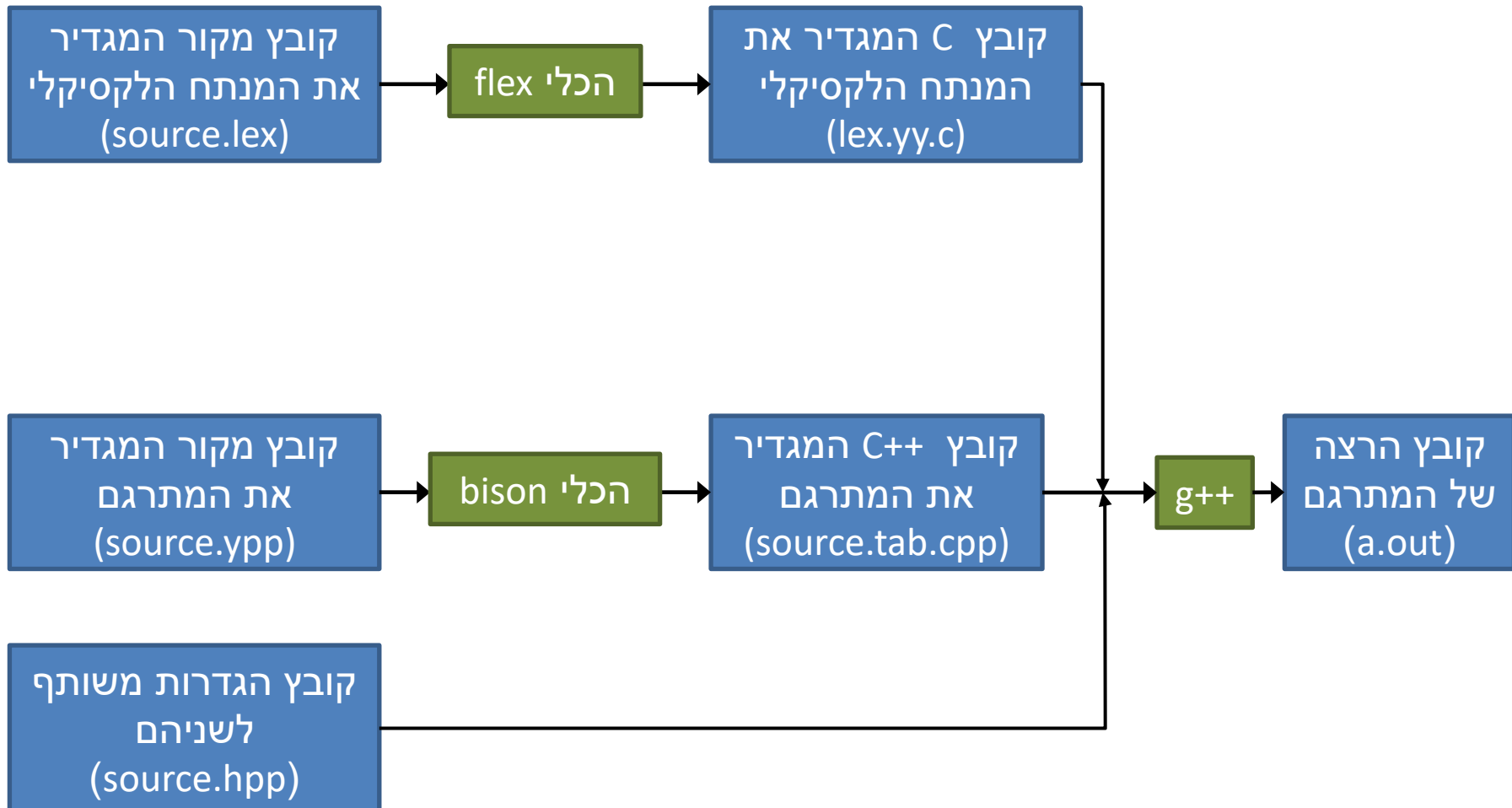
+ הרחבה ל: פתרון קונפליקטים.

– מצפה לניתוח לקסיקלי:

– יכול להיות מסופק ע"י כלי Lex (yylex)

– יכול להיות מוגדר בדרכים אחרות

סכימת השימוש ב bison בעזרת Lex



המנתח הנוצר על ידי bison

- מכיל את הפונקציות הבאות:
 - **yylex()** : פונקציית הניתוח הלקסיקלי.
 - נוצרת ע"י כלי Lex.
 - **yyparse()** : פונקציית הניתוח התחבירי הראשית.
 - נוצרת אוטומטית ע"י bison.
 - מחזירה 0 אם הניתוח הסתיים בהצלחה או 1 אחרת.
 - **yyerror()** : פונקציה ש yyparse קוראת לה בכל פעם שמתגלית שגיאה.
 - המשתמש יכול לדרוס אותה ע"מ לטפל בעצמו בשגיאות.
 - **main()** : מטרתה העיקרית היא קריאה ל yyparse.
 - ניתן לנצלה לשם איתחולים ופעולת סיום מיוחדת של המנתח.

מבנה קובץ הקלט של bison

%{

C user declarations חלק 1:

%}

declarations

%%

חלק 2:

rules

%%

חלק 3:

C user routines

- קובץ מופרד ל-3 חלקים: הצהרות, חוקים, וקוד C++.
- שתי השורות המפרידות בין החלקים חייבות להופיע (מודגשות בקו תחתון).

מבנה קובץ הקלט של bison

%{

user declarations

חלק 1:

%}

declarations

%%

rules

חלק 2:

%%

user routines

חלק 3:

- על השגרות המופיעות בחלק 3 יש להכריז בחלק 1 בין %{ ל- %} .
- כל הרשום בחלק 3 ובחלק 1 בין %{ ל- %} (כתוב בשפת C++) מועתק כמו שהוא לקובץ המקור שנוצר ע"י bison.

דוגמא

```
%{  
    #include <iostream>  
    #include "calc.h"  
    using namespace std;  
  
    #define YYSTYPE Node*  
    int yylex();  
    void yyerror(const char*);  
%}
```

```
%token NUM  
%left PLUS MINUS  
%left MULT  
%right LPAR RPAR
```

```
%%
```

```
prog :      prog line  
      | /*epsilon*/  
      ;
```

```
line : exp '\n' {cout << $1->prettyPrint() << endl; }  
      ;
```

```
exp :      NUM { $$ = $1; }  
      | exp PLUS exp { $$ = new Add($1,$3); }  
      | exp MINUS exp { $$ = new Sub($1,$3); }  
      | exp MULT exp { $$ = new Mul($1,$3); }  
      | LPAR exp RPAR { $$ = $2; }  
      ;
```

```
%%
```

```
void yyerror(const char*) { cout << "syntax error" << endl; }  
  
int main() {  
    return yyparse();  
}
```

declarations

%{

C user declarations חלק 1:

%}

declarations

%%

חלק 2:

rules

%%

C user routines

חלק 3:

- מיועד להצהרה על האסימונים של הדקדוק.
- בחלק זה גם תוגדר עדיפות לבחירת אסימון. (על כך בהמשך.)

דוגמא

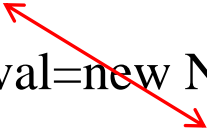
```
%{  
    #include <iostream>  
    #include "calc.h"  
    using namespace std;  
  
    #define YYSTYPE Node*  
    int yylex();  
    void yyerror(const char*);  
%}  
  
%token NUM  
%left PLUS MINUS  
%left MULT  
%right LPAR RPAR  
  
%%  
  
prog :  
    | /*epsilon*/  
;  
  
line : exp '\n' {cout << $1->prettyPrint() << endl; }  
;  
  
exp :  
    NUM { $$ = $1; }  
    | exp PLUS exp { $$ = new Add($1,$3); }  
    | exp MINUS exp { $$ = new Sub($1,$3); }  
    | exp MULT exp { $$ = new Mul($1,$3); }  
    | LPAR exp RPAR { $$ = $2; }  
;  
  
%%  
  
void yyerror(const char*) { cout << "syntax error" << endl; }  
  
int main() {  
    return yyparse();  
}
```

קישור בין flex ל-bison

תאום אסימונים

bison מפעיל את המנתח הלקסיקלי של flex ע"י קריאה ל-yylex.
צריך לתאם בין האסימונים ש-flex מייצר לבין האסימונים ש-bison קורא.

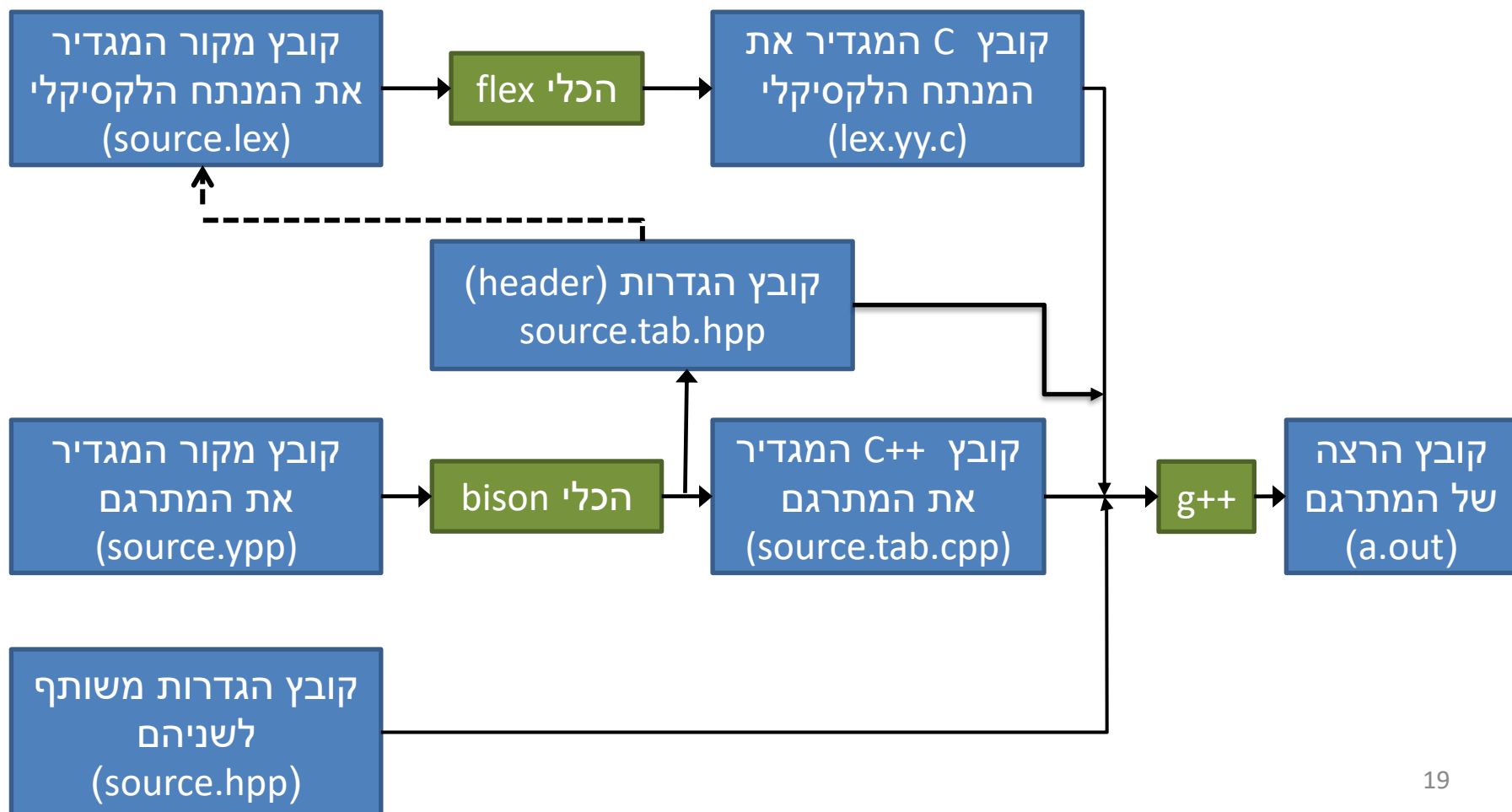
<u>Bison</u> :	%token NUM
<u>Lex</u> :	[0-9]+ {yyval=new Num(yytext) return NUM ; }



- כל האסימונים שהוגדרו בקובץ ה bison יופיעו ב header שנוצר ע"י הכלי (source.tab.hpp).
- קובץ הגדרות ה Lex צריך לצרוך (include) את הקובץ source.tab.hpp על מנת ליצור את התיאום.

קישור בין flex ל-bison

תיאום בין האסימונים ש-flex מייצר לבין האסימונים ש-bison קורא.
ע"י source.tab.hpp



קישור בין flex ל-bison

קריאת אסימונים

- :bison

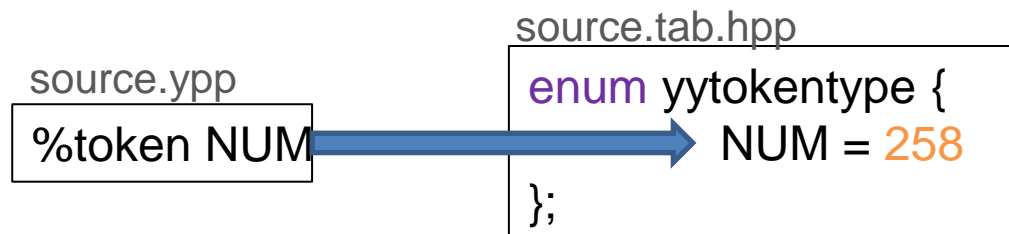
- מניח את קיומה של yylex.

- המחזירה בכל פעם קבוע המתאים לאחד האסימונים שהוגדרו ע"י %token.

- :Lex

- כל מחרוזת שהוגדרה ב bison ע"י X %token מתורגמת לקבוע ב source.tab.hpp

- מנתח לקסיקלי המחזיר את המזהה הקבוע X, יחזיר את אותו הקבוע שמוגדר בקובץ ההגדרות, ולכן יפורש ע"י המנתח התחבירי כאסימון X.



קישור בין flex ל-bison

תכונות סמנטיות לאסימונים

Bison: %token NUM

Lex: [0-9]+ { **yyval**=new Num(yytext);
return NUM; }

- **yyval** – משתנה גלובלי מטיפוס **YYSTYPE** המוגדר ב

[.source.tab.hpp](#)

- מכיל את התכונות הסמנטיות של האסימון האחרון שנקרא.

טיפוסי החזרה ב-Bison

- תכונות כל משתני הדקדוק הן מאותו הטיפוס – YYSTYPE.
 - זהו גם הטיפוס של המשתנה הגלובלי `yylval`.
 - ברירת המחדל לטיפוס YYSTYPE היא `int`.
 - ניתן לדרוס את YYSTYPE כדי להגדיר תכונות בעלות טיפוסים שונים, (בקובץ `source.hpp`) ע"י למשל:

```
struct Node {  
    virtual std::string prettyPrint();  
};  
class Num : public Node { //...  
};  
class Add : public Node { //...  
};  
#define YYSTYPE Node*
```

Lex:

```
[0-9]+ {yylval=new Num(yytext)  
return NUM;}
```

חוקים – כללי הגזירה

%{

C user declarations חלק 1:

%}

declarations

%%

חלק 2:

rules

%%

חלק 3:

C user routines

כללי הגזירה

- כלל גזירה ירשם בצורה הבאה:

– עבור $A \rightarrow B C \text{ NUM } D E$

אופציונלי

A: B C NUM D E  `{/* action */}`;

- ניתן לשרשר מספר כללים על ידי |

A: B C NUM D E `{/*action*/}`

| E F `{/*action*/}`

;

- המשתנה באגף שמאל של החוק הראשון הוא המשתנה התחילי של הדקדוק.

זיהוי דקדוק ב-Bison

%%

```
prog : prog line  
      | /*epsilon*/  
      ;
```

```
line : exp '\n';
```

```
exp : NUM  
     | exp PLUS exp  
     | exp MINUS exp  
     | exp MULT exp  
     | LPAR exp RPAR  
     ;
```

%%

מנתח זה ירוץ על קלט
ויסיים אם הקלט בשפה או
יקרא ל-yyerror אם לא

כלומר: מזהה דקדוק בלבד

בניית AST

```
A:  B C NUM D E  { /* action */  
  |  E F          { /*action*/  
  ;
```

- בתור `/*action*/` נכתוב

קוד שנרצה להריץ בסוף הכלל

– מועתק (כמעט) כמו שהוא לקובץ הפלט

- כדי לבנות את ה-AST:

– "להחזיר" (=לשים במחסנית) ערך מכל כלל – את תת
העץ שהוא מייצג

– בהתבסס על תתי העצים של הילדים (או חלקם)

- יש ב-Bison גישה לאיברי הכלל במחסנית

פעולות עבור כללי הגזירה

```
A:  B C NUM D E  { /* action */
    |  E F          { /*action*/
;

```

- גישה לאובייקטים (מסוג YYSTYPE) של משתנים וטרמינלים במחסנית:

– \$\$ - המשתנה המופיע באגף שמאל של החוק

– \$n - הסימן ה-n באגף ימין של החוק ($n > 0$)

$A \rightarrow B C N U M D E$

\$\$	\$1	\$2	\$3	\$4	\$5	הסימן
A	B	C	NUM	D	E	מתייחס ל-

ביצוע פעולות

%%

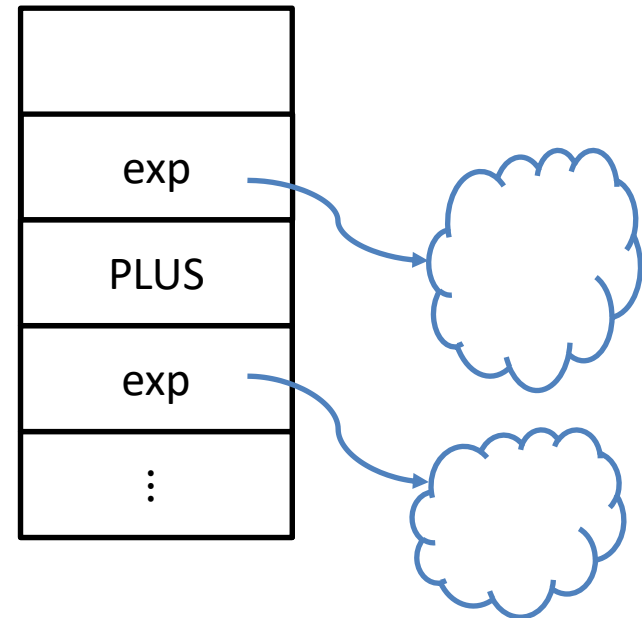
```
prog :  prog line
      | /*epsilon*/
;
```

```
line : exp '\n' {cout << $1->prettyPrint() << endl; }
;
```

```
exp :  NUM { $$ = $1; }
      | exp PLUS exp { $$ = new Add($1,$3); }
      | exp MINUS exp { $$ = new Sub($1,$3); }
      | exp MULT exp { $$ = new Mul($1,$3); }
      | LPAR exp RPAR { $$ = $2; }
;
```

%%

Reducing exp \rightarrow exp PLUS exp:



ביצוע פעולות

%%

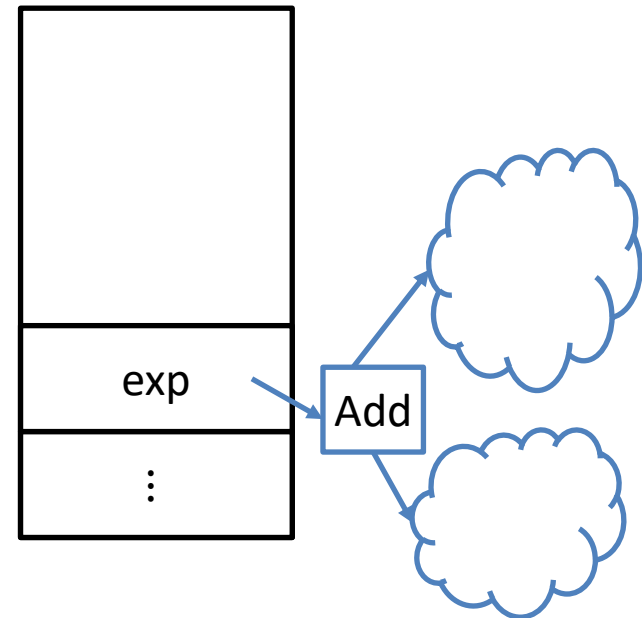
```
prog : prog line
      | /*epsilon*/
;
```

```
line : exp '\n' {cout << $1->prettyPrint() << endl; }
;
```

```
exp : NUM { $$ = $1; }
     | exp PLUS exp { $$ = new Add($1,$3); }
     | exp MINUS exp { $$ = new Sub($1,$3); }
     | exp MULT exp { $$ = new Mul($1,$3); }
     | LPAR exp RPAR { $$ = $2; }
;
```

%%

Reducing exp \rightarrow exp PLUS exp:



שאלה: למה המימוש של הכלל הראשון והאחרון הוא ע"י $$$ = \1 ו- $$$ = \2 ?

טיפול בדקדוקים רב משמעיים

- ישנם אלמנטים בשפות תכנות המתוארים ע"י דקדוק דו משמעי.

$$E \rightarrow E \pm E$$

$$E \rightarrow E * E$$

$$E \rightarrow \text{num}$$

– לדוגמא, ביטויים אריתמטיים עם חיבור וכפל:

– דקדוק זה הוא דו משמעי על "1+2*3".

לאחר קריאת "1+2" ישנן 2 התנהגויות:

1. בצע shift, המשך לקרוא את הקלט ובצע reduce בסופו ע"פ הכלל השלישי.

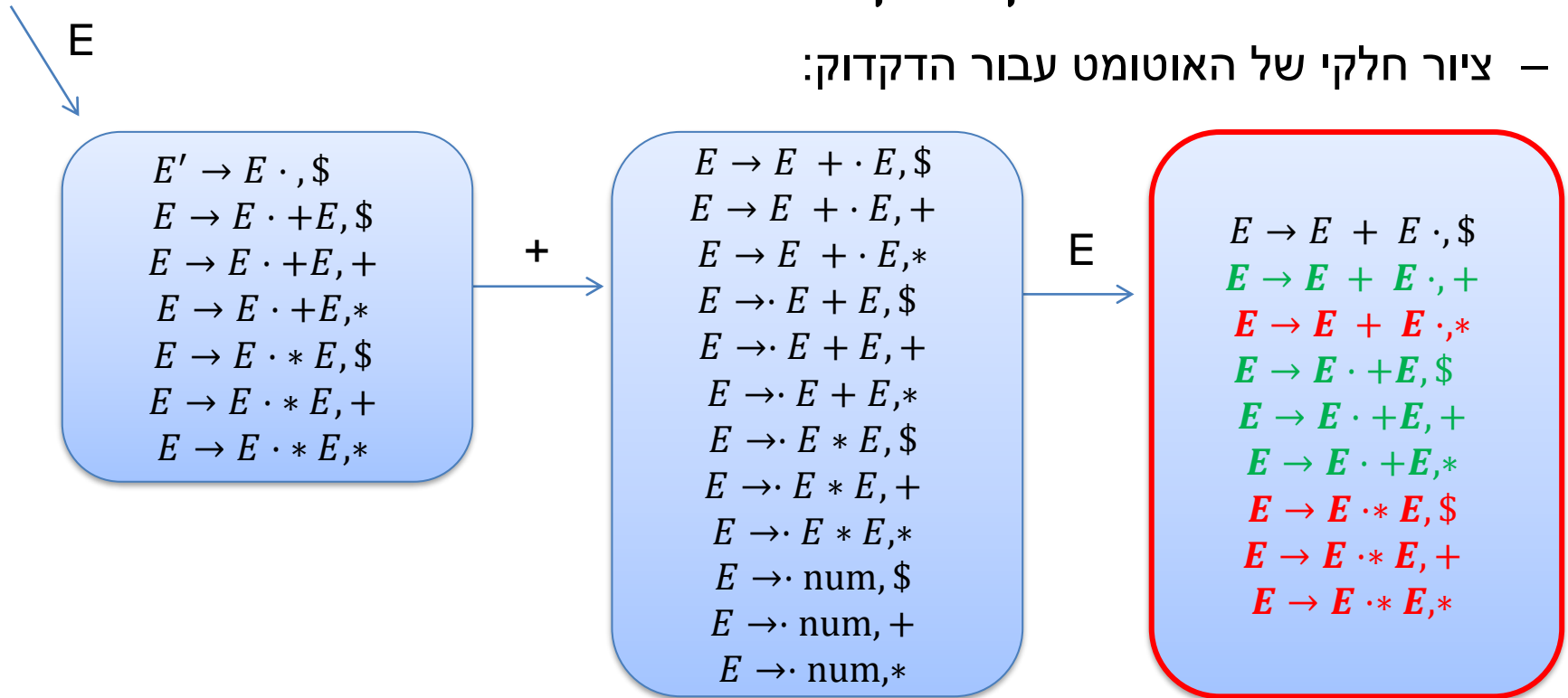
- מתאים לפירוש $1+(2*3)$

2. בצע reduce ע"פ הכלל הראשון.

- מתאים לפירוש $(1+2)*3$

טיפול בדקדוקים רב משמעיים

– ציור חלקי של האוטומט עבור הדקדוק:



– במצב המסומן ישנו קונפליקט עבור הטרמינל $*$ המסומן באדום.

– בנוסף, ישנו קונפליקט גם עבור הטרמינל $+$ המסומן בירוק.

• מתאים לשני הפירושים של המחרוזת "1+2+3":

– $(1+2)+3$ או $1+(2+3)$. (עבור חיבור זה לא משנה, אבל עבור חיסור כן).

קדימות אופרטורים

- פתרון לקונפליקט בין כפל לחיבור: $1+(2*3)$ לעומת $3*(1+2)$:

עדיפות לאסימונים

- ניתן לציין לbison שלאסימון '*' יש עדיפות גדולה מאשר לאסימון '+'.
 - מבוצע ע"י סדר ההכרזה על אסימונים בקובץ ההגדרות.
- האסימון בעל העדיפות הגבוהה ביותר, יוכרז אחרון.

עדיפות לכללי גזירה.

- ברירת המחדל היא עדיפות האסימון האחרון המופיע בכלל.
- ניתן לציין עדיפות באופן מפורש ע"י שימוש ב

"%prec t"

כאשר t הוא האסימון שאת עדיפותו רוצים לתת לכלל
(ניתן להוסיף אסימון חדש למטרה זו).

%token '+'

%token '*'

%%

E: E '+' E

{...}

| E '*' E

{...};

קדימות אופרטורים

- פתרון לקונפליקט בין כפל לחיבור: $1+(2*3)$ לעומת $(1+2)*3$:

– כאשר bison מוצא קונפליקט shift/reduce: shift עבור טרמינל t אל מול reduce עם כלל $A \rightarrow \alpha$,

- יבוצע shift אם לטרמינל t עדיפות גדולה מכלל $A \rightarrow \alpha$.

- יבוצע reduce אם לכלל עדיפות גבוהה מאשר לטרמינל.

- אחרת הקונפליקט לא נפתר.

– בדוגמא, במצב הבעייתי, קונפליקט זה ייפתר לטובת הshift מאחר ולאסימון $*$ עדיפות גדולה מאשר לכלל $E \rightarrow E+E$.

- התקבלה הפרשנות הראשונה.

%token '+'

%token '*'

%%

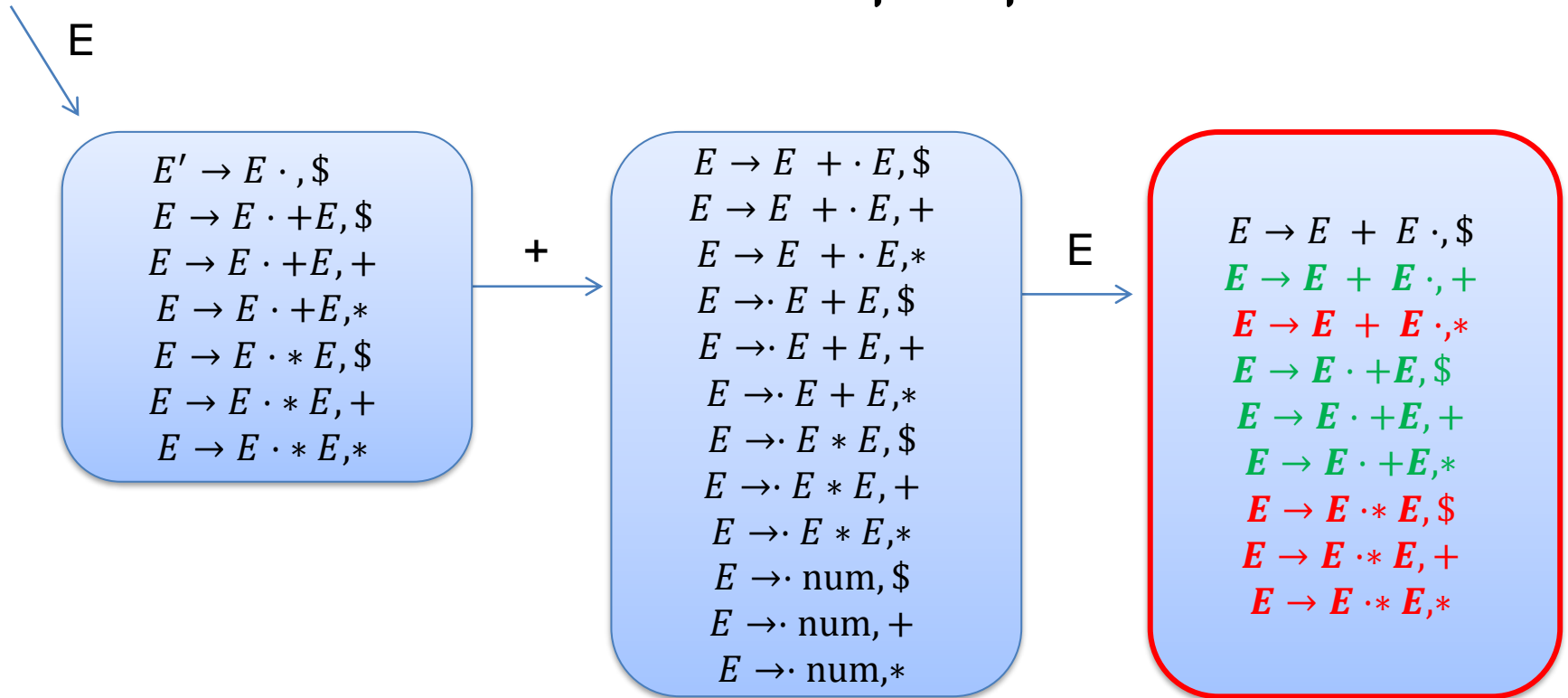
E: E '+' E {...}

| E '*' E {...} ;

למעשה בדוגמא זו צריך היה להשתמש ב
%nonassoc במקום ב %token כדי לקבל
קדימות.

נדבר על כך בהמשך.

טיפול בדקדוקים רב משמעיים



– נראה כעת כיצד מתגברים על הקונפליקט עבור הטרמינל $+$.

אסוציאטיביות

- פתרון לקונפליקט עבור הטרמינל $+$:

$1+(2+3)$ לעומת $(1+2)+3$:

– לא ניתן להכריע בעזרת עדיפות מאחר ולאסימון $'+'$ אותה העדיפות כמו לכלל $E \rightarrow E+E$.

– ניתן להגדיר את האסימון $'+'$ כבעל אסוציאטיביות שמאלית.

- יוכרז עם `%left` במקום עם `%token`

```
%left '+'
```

```
%token '*'
```

```
%%
```

```
E: E '+' E {...}
```

```
  | E '*' E {...} ;
```

אסוציאטיביות

- פתרון לקונפליקט עבור הטרמינל + :

: $1+(2+3)$ לעומת $(1+2)+3$

– כאשר קונפליקט S/R אינו נפתר בעזרת קדימויות, bison יכריע באופן הבא:

- אם לאסימון אסוציאטיביות שמאלית יועדף ה reduce.

- אם לאסימון אסוציאטיביות ימנית יועדף ה shift.

– במקרה שלנו יועדף ה reduce ונקבל את הפרשנות השנייה.

```
%left '+'
```

```
%token '*'
```

```
%%
```

```
E: E '+' E {...}
```

```
  | E '*' E {...} ;
```

אסוציאטיביות

- הערה: עדיפות מוגדרת רק עבור אסימונים שהוכרזו עם אסוציאטיביות, לא עבור אסימונים שהוגדרו ע"י %token.
 - ניתן להשתמש ב %nonassoc כדי להגדיר אסימון עם עדיפות שאין לו אסוציאטיביות.
- שאלה: לאילו אופרטורים מתמטיים נרצה אסוציאטיביות ימנית?
 - למשל- אופרטור החזקה: a^b^c
 - למשל תנאי טרינארי: $b ? a : b$

טיפול בדקדוקים רב משמעיים

- סיכום קונפליקט shift/reduce:

– אם לכלל הגזירה עדיפות r ולטרמינל הקלט עדיפות t :

- אם $r > t$ יבוצע reduce.

- אם $r < t$ יבוצע shift.

- אם $r = t$ הפעולה תיקבע ע"פ האסוציאטיביות של הכלל (זהה לאסוציאטיביות הטרמינל):

– אסוציאטיביות שמאלית יבוצע reduce.

– אסוציאטיביות ימנית יבוצע shift.

– אסוציאטיביות אינה מוגדרת שגיאה.

- קונפליקט reduce/reduce:

– התנהגות ברירת המחדל של bison היא לבחור בכלל הגזירה שהופיע קודם בדקדוק.

– מומלץ לשנות את הדקדוק כך שיתמודד עם קונפליקט זה.

עבודה עם bison

- `bison -d source.ypp` מייצר קבצי `*.tab.h`, `*.tab.cpp`
- `flex source.lex` מייצר קובץ `lex.yy.c`
- `g++ source.tab.cpp lex.yy.c` מייצר קובץ הרצה

-
- `bison -v source.ypp` מייצר קובץ `source.output` עם מצבי האוטומט וטבלת המעברים.
ניתן להשתמש באפשרות זאת על מנת לראות את הקונפליקטים, במידה וקיימים.

ישנו חומר עזר נוסף באתר:

`bison07_tips_bison.pdf`

Lex and Bison Examples and Guides(Course Material)

שאלות

ממבחנים

שאלה 1

נתון $G=(V, T, P, S)$ השייך ל $LR(1)$. עבור כל אחד מהדקדוקים הבאים, רשמו האם הוא **בהכרח** ב $LR(1)$:

א. $G_1 = (V \cup \{X\}, T, P \cup \{X \rightarrow t\}, S)$ כאשר $t \in T, X \notin V$

ב. $G_2 = (V, T, P \cup \{S \rightarrow \varepsilon\}, S)$

שאלה 1

כאשר $t \in T, X \notin V$

$G_1 = (V \cup \{X\}, T, P \cup \{X \rightarrow t\}, S)$.א

• הדקדוק בהכרח ב LR(1):

– המשתנה X הוא משתנה חדש שאינו נגזר באגף ימין של אף כלל.

– בזמן ביצוע closure במצבי האוטומט, הכלל $X \rightarrow t$ לעולם לא ייכנס.

– האוטומט יהיה זהה לאוטומט של G , שהוא חסר קונפליקטים.

X משתנה לא ישיג !

שאלה 1

$$\text{ב. } G_2 = (V, T, P \cup \{S \rightarrow \varepsilon\}, S)$$

• אינו בהכרח ב LR(1).

– הכלל החדש יכול להפוך את הדקדוק לדו משמעי.
לדוגמא:

$$G = \{S \rightarrow A, A \rightarrow \varepsilon\}$$

$$G_2 = \{S \rightarrow A, A \rightarrow \varepsilon, S \rightarrow \varepsilon\}$$

דקדוק דו משמעי: יש שתי גזירות ימניות ביותר בדקדוק לאותה מילה.

שאלה - אסוציאטיביות

```
%{
#include <stdio.h>
#include <iostream>

using namespace std;

int yylex();
void yyerror(const char*);
}%

% left 'a'
% right 'b'
%%

P : S '\n' ;
S : S 'a' S {cout << '1';}
  | S 'b' S {cout << '2';}
  | 'c' {cout << '3';}
  | 'd' {cout << '4';}
  | /*epsilon*/ ;
%%
```

נתונה תוכנית ה bison הבאה:

```
int yylex(){
    return getchar();
}

void yyerror(const char*){
    cout << '5';
}

int main(){
    return yyparse();
}
```

כתבו מה יהיה פלט התוכנית עבור כל אחד מהקלטים הבאים (כל שורה מסתיימת בסימן ירידת שורה '\n'):

1. cbdbc
2. cadbc