Bottom Up Parsing

אותו דבר כמו Top Down Parsing רק הפוך!
בונים את עץ הגזירה מלמטה למעלה

הדקדוק חסר ההקשר של שפת C

- שימו לב לגודלו העצום של הדקדוק חסר ההקשר של שפת C:
- http://www.lysator.liu.se/c/ANSI-C-grammar-y.html, שימו לב לרקורסיה שמאלית (כמעט בכל כלל..)

```
multiplicative_expression
        : cast_expression
        | multiplicative_expression '*' cast_expression
        | multiplicative_expression '/' cast_expression
        | multiplicative_expression '%' cast_expression
        ;

additive_expression
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
        ;
```

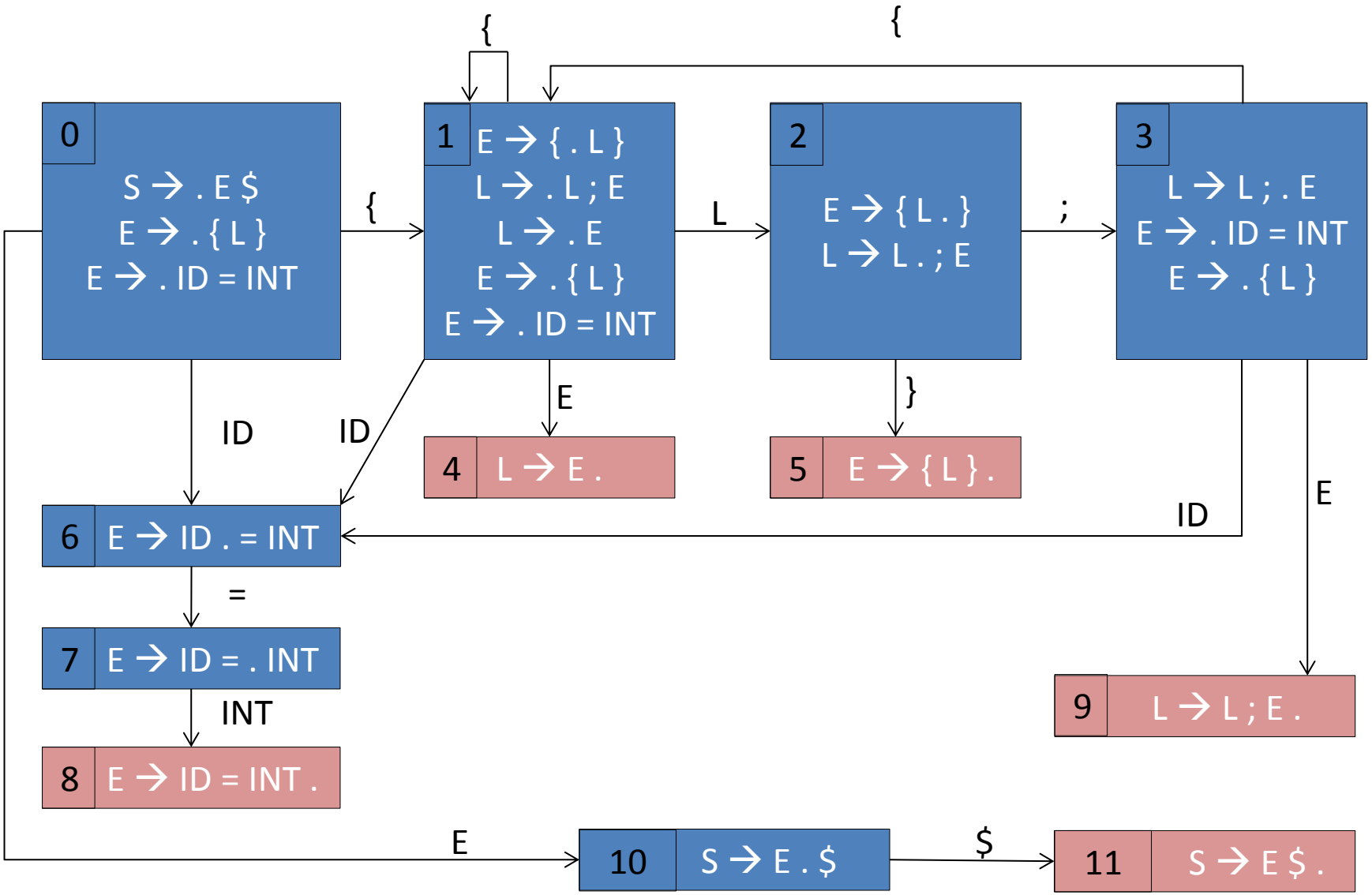
Bottom Up Parsing

- ראינו שדקדוקים שמכילים רקורסיה שמאלית, אי אפשר לבנות עבורם predictive parser. אבל בדקדוק של שפת C שזה עתה ראינו יש רקורסיה שמאלית.
- האם אפשר לבנות parser שידע להתגבר על רקורסיה שמאלית? כן.
- נבנה את עץ הגזירה מלמטה למעלה. אם בשלב מסויים לא יודעים איזה כלל הופעל, <u>נדחה את ההחלטה</u> ונמשיך לקרוא את הקלט.

Bottom Up Parsing – a simple example

- נסתכל על הדקדוק הבא שמכיל רקורסיה שמאלית:
- S → E$ E → { L } E → ID = INT
- L → E L → L ; E
- ונסתכל על המילה הבאה בשפה: { x=8 ; y=7 }
- בקריאת התו הראשון, סוגריים מסולסלים שמאליים, אני יודע איזה כלל הופעל. אבל אחר כך, כשאני קורא x כלומר ID, אני לא יודע איזה כלל גזירה הופעל:
- אפשרות אחת: L → E ואז E → ID = INT
- אפשרות שניה: L → L ; E ואז L → E ואז E → ID = INT
- כלומר, <u>אי אפשר לנבא מראש איזה כלל הופעל</u>. צריך <u>לדחות את ההחלטה למועד מאוחר יותר</u>. מה המועד? קריאת התו ; מהקלט.
- נדחוף את התו למחסנית ונמשיך לקרוא

Bottom Up LR(0) Parsing – example

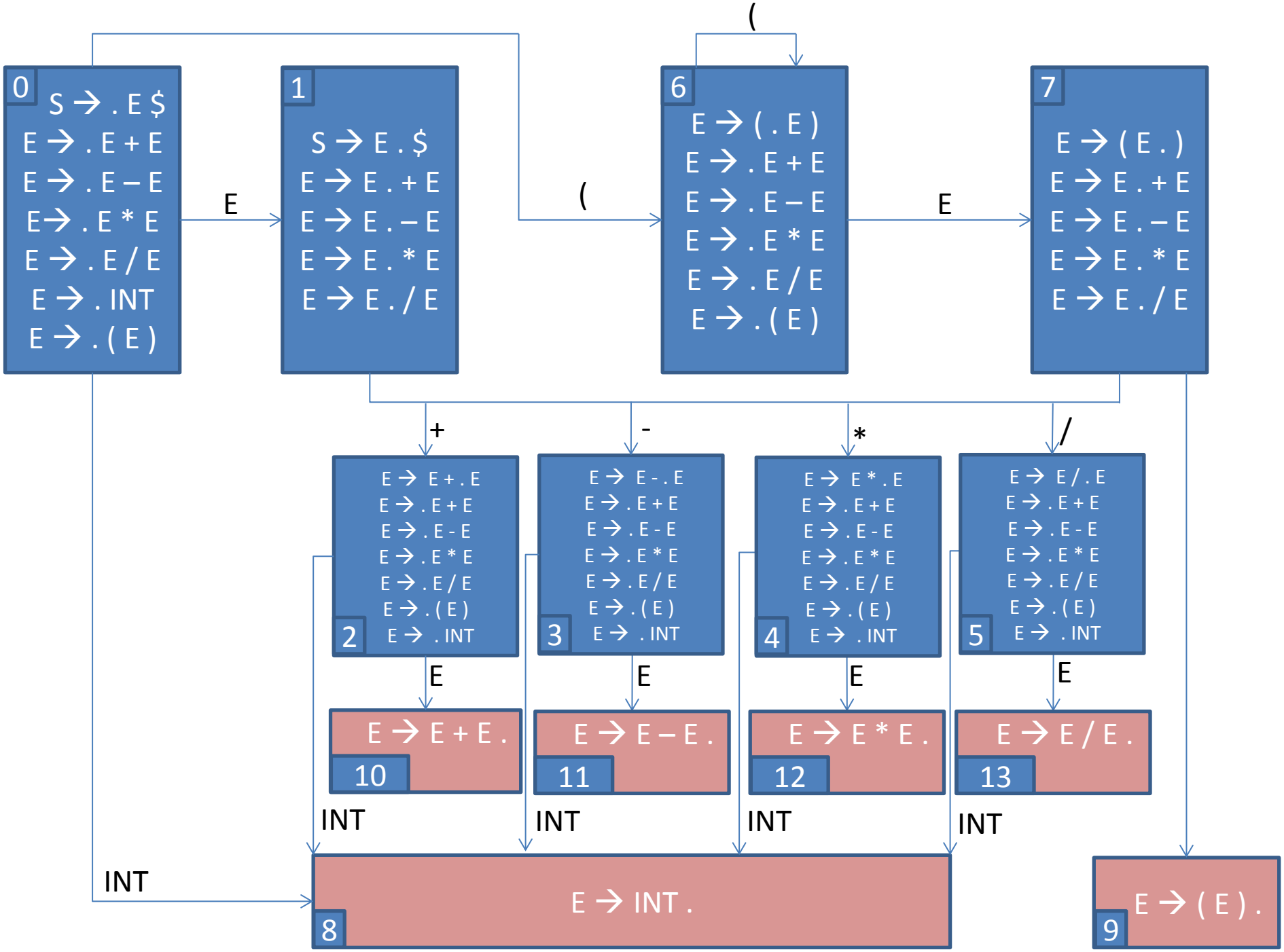


Building a Bottom Up, LR(0) Parser

- יש ארבע אפשרויות, והן תלויות בלבד במצב בו אני נמצא
- Shift: קרא את ה token הבא בקלט, דחוף אותו למחסנית, ועבור למצב הבא לפי הדיאגרמה
- Reduce: בראש המחסנית נמצא צד ימין של כלל גזירה. החלף אותו בצד שמאל של כלל הגזירה. עבור למצב המתאים
- Accept: סיים את הריצה ודווח: הכל תקין.
- Error: סיים את הריצה ודווח: המילה אינה בשפה

Bottom Up SLR(0) Parsing – example

- ניזכר בדקדוק של שפת המחשבון:
- E → E + E E→INT
- E → E * E E → (E)
- E → E / E
- E → E – E
- האם ניתן לבנות עבורו parser כמו לדקדוק הקודם?
- מהו עץ הגזירה של הביטוי 3+8*4?


0
S → . E $
E → . E + E
E → . E – E
E → . E * E
E → . E / E
E → . INT
E → . (E)
1
S → E . $
E → E . + E
E → E . – E
E → E . * E
E → E . / E
6
E → (. E)
E → . E + E
E → . E – E
E → . E * E
E → . E / E
E → . (E)
7
E → (E .)
E → E . + E
E → E . – E
E → E . * E
E → E . / E
2
E → E + . E
E → . E + E
E → . E - E
E → . E * E
E → . E / E
E → . (E)
E → . INT
3
E → E - . E
E → . E + E
E → . E - E
E → . E * E
E → . E / E
E → . (E)
E → . INT
4
E → E * . E
E → . E + E
E → . E - E
E → . E * E
E → . E / E
E → . (E)
E → . INT
5
E → E / . E
E → . E + E
E → . E - E
E → . E * E
E → . E / E
E → . (E)
E → . INT
10
E → E + E .
11
E → E – E .
12
E → E * E .
13
E → E / E .
8
E → INT .
9
E → (E) .
E
(
+
-
*
/
INT

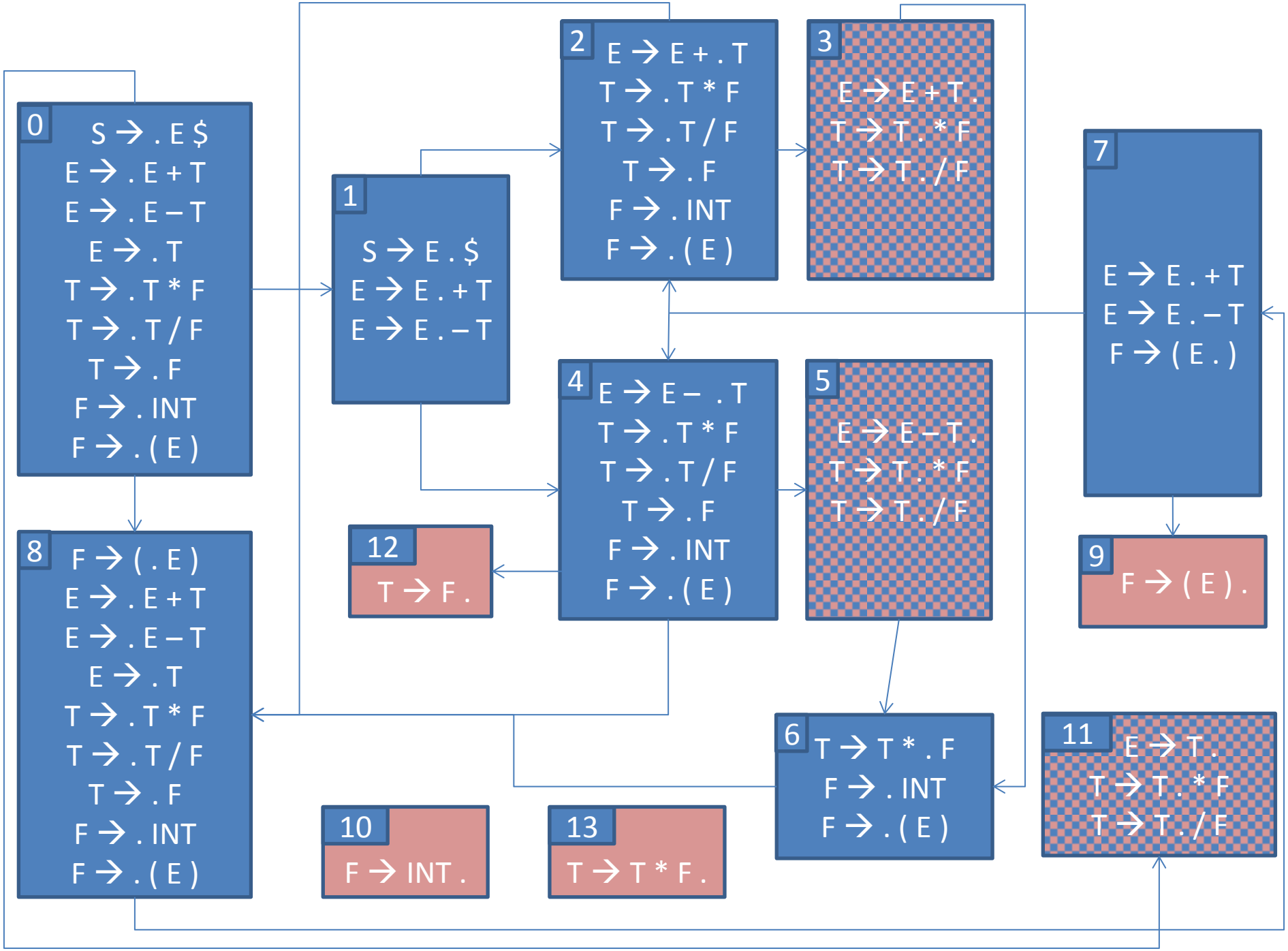
הרצה על הקלט 1 + 2 * (500-3)

C:\Windows\system32\cmd.exe

```
[0]
[0] INT [8]
[0]   E [1]
[0]   E [1]   + [2]
[0]   E [1]   + [2] INT [8]
[0]   E [1]   + [2]   E [10]
[0]   E [1]
[0]   E [1]   * [4]
[0]   E [1]   * [4]   ( [6]
[0]   E [1]   * [4]   ( [6] INT [8]
[0]   E [1]   * [4]   ( [6]   E [7]
[0]   E [1]   * [4]   ( [6]   E [7]   - [3]
[0]   E [1]   * [4]   ( [6]   E [7]   - [3] INT [8]
[0]   E [1]   * [4]   ( [6]   E [7]   - [3]   E [11]
[0]   E [1]   * [4]   ( [6]   E [7]
[0]   E [1]   * [4]   ( [6]   E [7]   ) [9]
[0]   E [1]   * [4]   E [12]
[0]   E [1]
[0]   E [1]   $ [14]

Legal Expression

Press any key to continue . . .
```

Building a Bottom Up SLR(1) Parsing – example

- נסתכל על בניה דומה עבור הדקדוק של שפת המחשבון עם קדימויות אופרטורים:
- $E \rightarrow E + T$ $T \rightarrow T * F$ $F \rightarrow INT$
- $E \rightarrow E - T$ $T \rightarrow T / F$ $F \rightarrow (E)$
- $E \rightarrow T$ $T \rightarrow F$
- מה ניתן לומר על מצב 5 בדיאגרמת המצבים בעמוד הבא?


0
S → . E $
E → . E + T
E → . E – T
E → . T
T → . T * F
T → . T / F
T → . F
F → . INT
F → . (E)
1
S → E . $
E → E . + T
E → E . – T
2
E → E + . T
T → . T * F
T → . T / F
T → . F
F → . INT
F → . (E)
3
E → E + T .
T → T . * F
T → T . / F
4
E → E – . T
T → . T * F
T → . T / F
T → . F
F → . INT
F → . (E)
5
E → E – T .
T → T . * F
T → T . / F
6
T → T * . F
F → . INT
F → . (E)
7
E → E . + T
E → E . – T
F → (E .)
8
F → (. E)
E → . E + T
E → . E – T
E → . T
T → . T * F
T → . T / F
T → . F
F → . INT
F → . (E)
9
F → (E) .
10
F → INT .
11
E → T .
T → T . * F
T → T . / F
12
T → F .
13
T → T * F .

Building a Bottom Up SLR(1) Parser!

- ההבדל בין SLR(0) לבין SLR(1) הוא בכך שב SLR(1) מותר לקרוא את התו הבא בקלט, והוא יכול להשפיע אם עושים shift או reduce. ב SLR(0) ההחלטה אם לעשות shift או reduce מושפעת <u>אך ורק מהמצב הנוכחי</u> (זה שבקצה המחסנית)

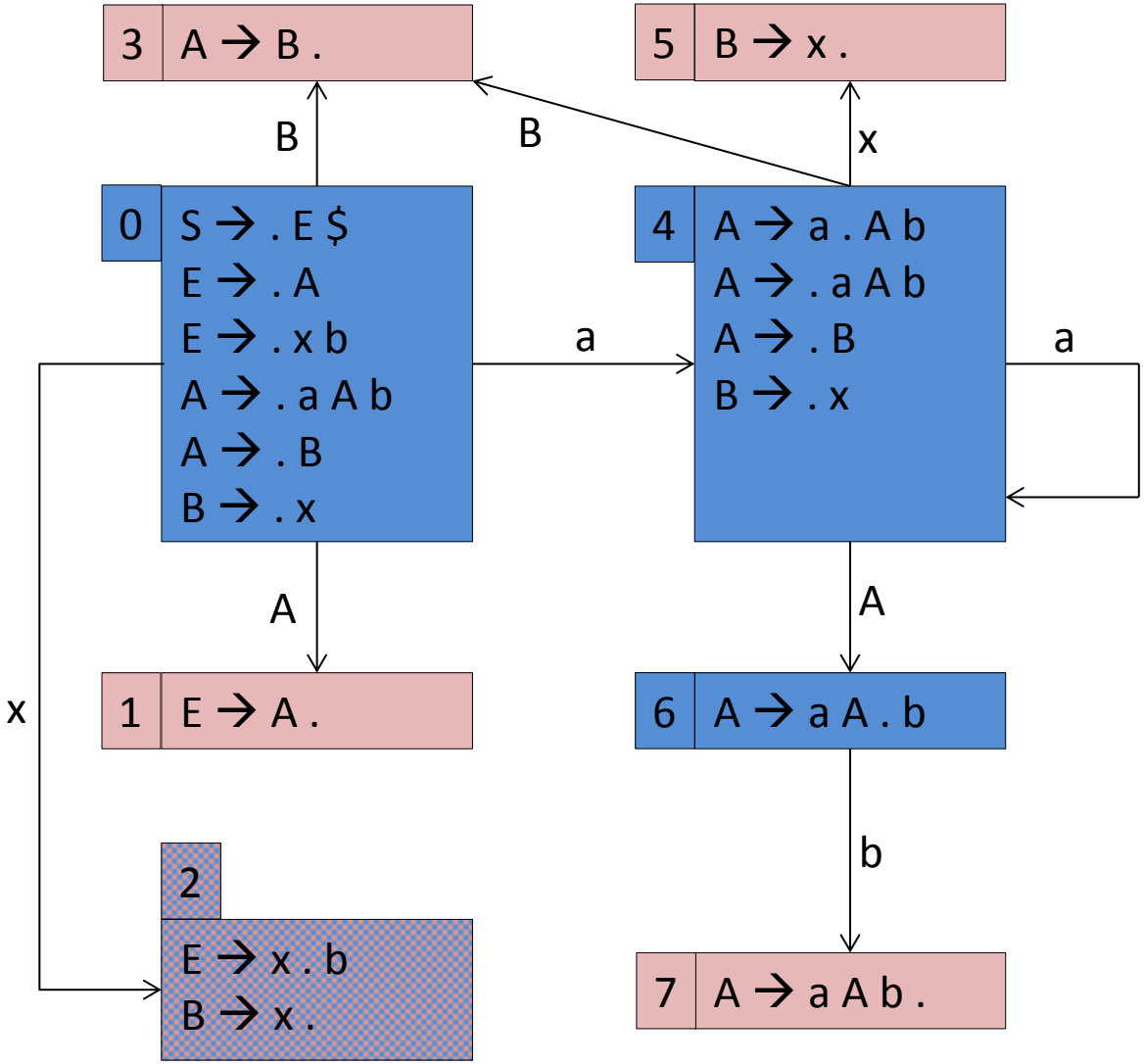
שימו לב איך קדימות האופרטורים באה לידי ביטוי:

אם ראיתי E + T ואחריו כפל, אני לא עושה reduce אלא shift

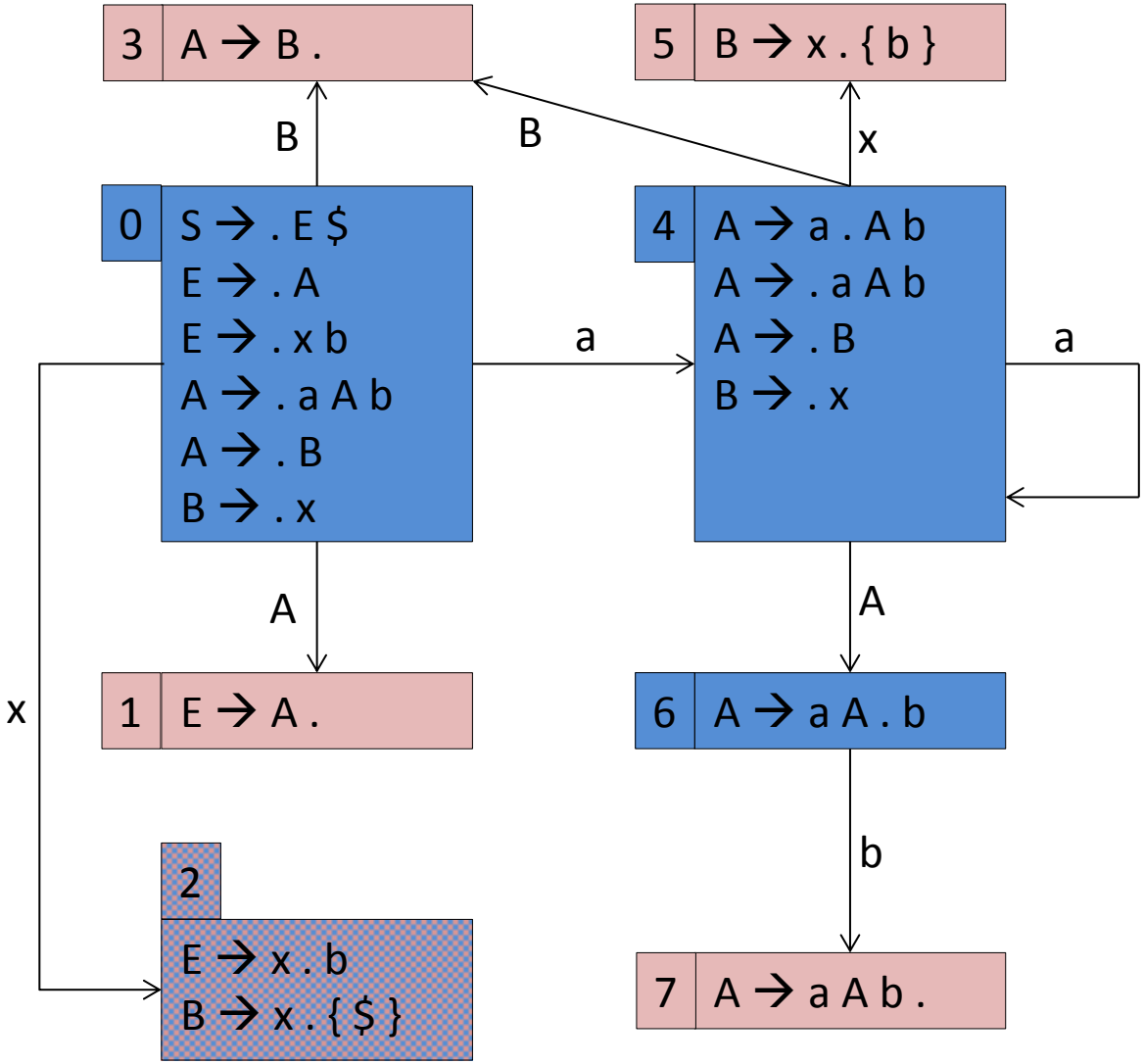
C:\Windows\system32\cmd.exe

```
[ 0]
[ 0] INT [10]
[ 0]   F [12]
[ 0]   T [11]
[ 0]   E [ 1]
[ 0]   E [ 1]   - [ 4]
[ 0]   E [ 1]   - [ 4] INT [10]
[ 0]   E [ 1]   - [ 4]   F [12]
[ 0]   E [ 1]   - [ 4]   T [ 5]
[ 0]   E [ 1]
[ 0]   E [ 1]   + [ 2]
[ 0]   E [ 1]   + [ 2] INT [10]
[ 0]   E [ 1]   + [ 2]   F [12]
[ 0]   E [ 1]   + [ 2]   T [ 3]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8] INT [10]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   F [12]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   T [11]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]   - [ 4]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]   - [ 4] INT [10]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]   - [ 4]   F [12]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]   - [ 4]   T [ 5]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   < [ 8]   E [ 7]   > [ 9]
[ 0]   E [ 1]   + [ 2]   T [ 3]   * [ 6]   F [13]
[ 0]   E [ 1]   + [ 2]   T [ 3]
[ 0]   E [ 1]


Legal Expression

Press any key to continue . . .
```

LR(1), Non SLR(1) Parsing – example

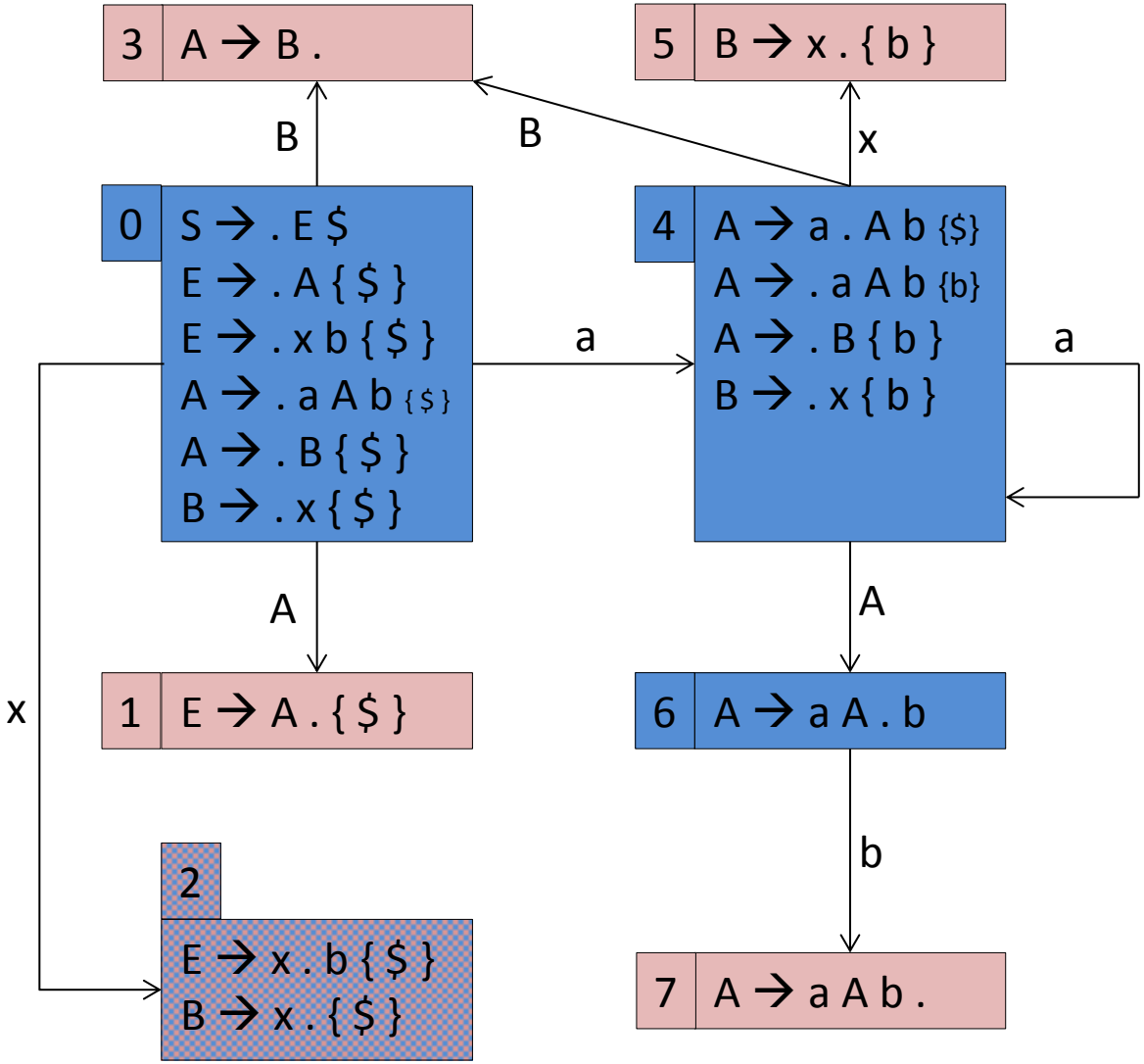


LR(1), Non SLR(1) Parsing – example



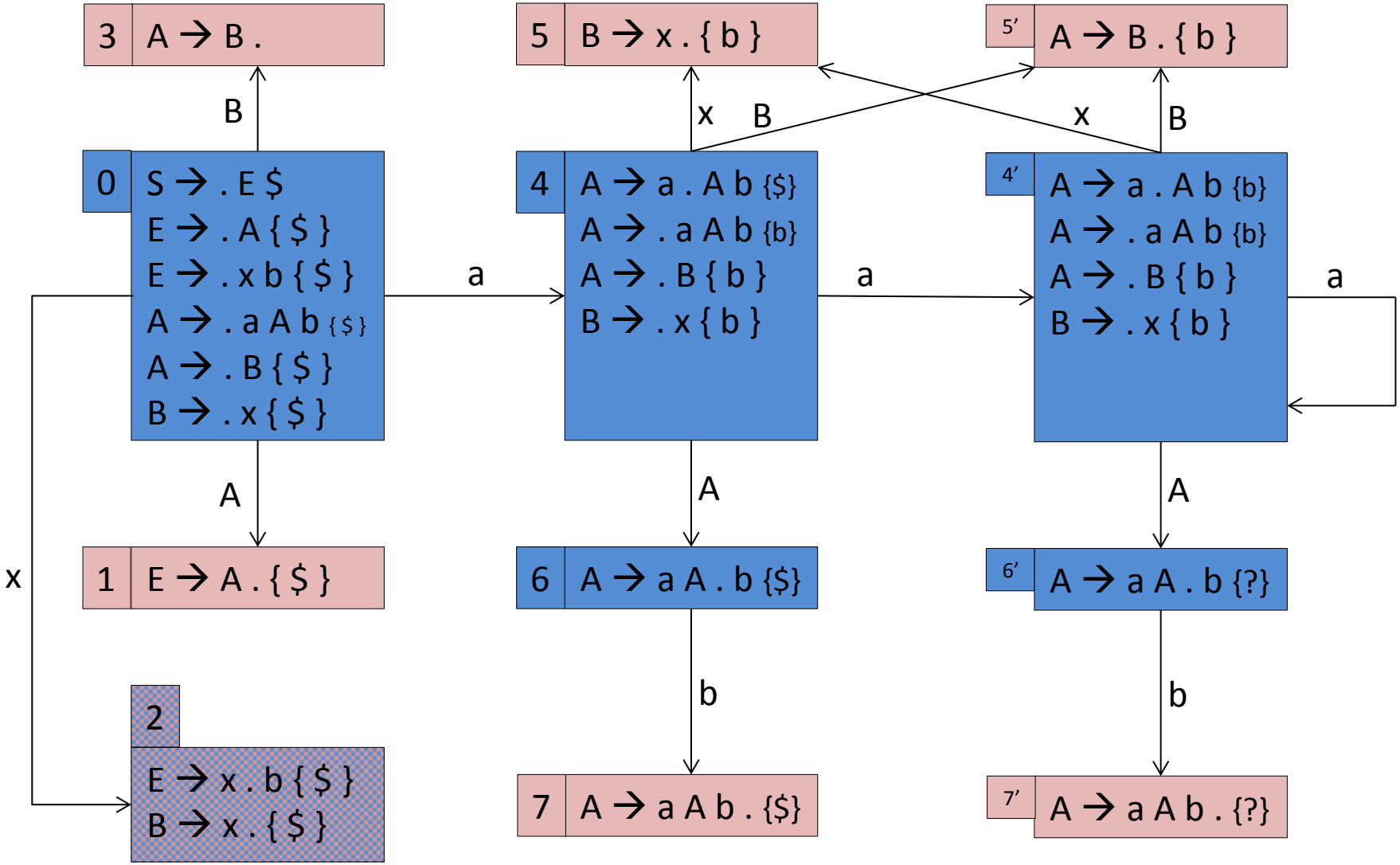
- נסתכל על הכלל B →x במצבים 2, 5
- נשאל איזה token יכול לבוא אחרי B?
- אפשר לשים לב שהתשובה על כך תלויה במצב!
- כלומר, ניתן להחליט בצורה יותר מדויקת איזה token-ים יכולים לבוא אחרי B, אם ידוע לנו המצב.
- זה בדיוק ההבדל בין SLR(1) לבין LR(1)
- בפרט, זה מאפשר לפתור את הקונפליקט במצב 2.
- המידע הזה נשמר עבור כל שורה בכל מצב

LR(1), Non SLR(1) Parsing – example



- אופס יש כאן בעיה! נסתכל על מצב 3, ונשאל מי יכול לבוא אחרי המשתנה A?
- אם הגענו ממצב 0, אז רק $ יכול לבוא אחריו
- אם הגענו ממצב 4, אז רק b יכול לבוא אחריו
- כלומר, מצב 3, בעצם צריך להתפצל לשני מצבים!
- ובאותו אופן, הלולאה ממצב 4, כבר לא יכולה להיות לולאה! צריך ליצור מצב חדש שישכפל כמעט בדיוק את מצב 4, למעט העובדה שבשורה הראשונה יהיה לנו: _____
- (התשובה בשקף הבא ...)

LR(1), Non SLR(1) Parsing – example



Parsing Questions From Tests

- עבור כל אחד מהדקדוקים הבאים, קבעו האם הוא ניתן ל predictive parsing כמו שהוא, האם הוא ב LR(0), האם הוא ב SLR(1) והאם הוא ב LR(1)
- S →A$ A → A int int | int
- S →A$ A → int A int | int
- S →A$ A → int int A | int