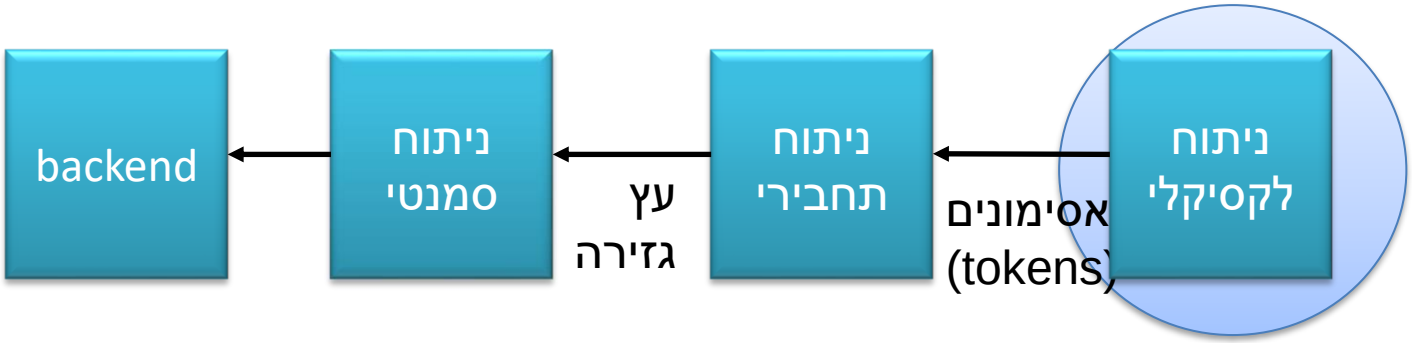
ניתוח סמנטי

תזכורת מהתרגולים הקודמים

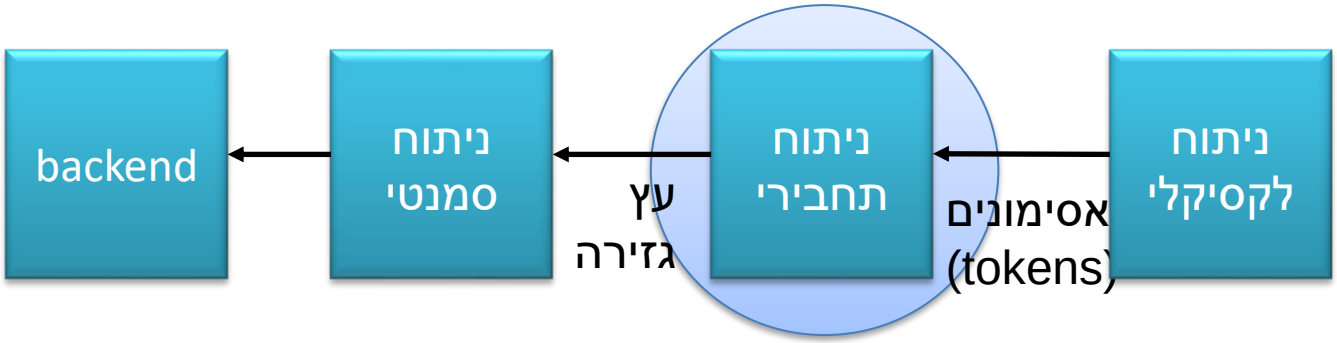
- מבנה סכמתי של קומפיילר



- ניתוח לקסיקלי:
  - אסימונים ולקסמות.
  - מנתח לקסיקלי.
  - כלי Lex.

תזכורת מהתרגולים הקודמים

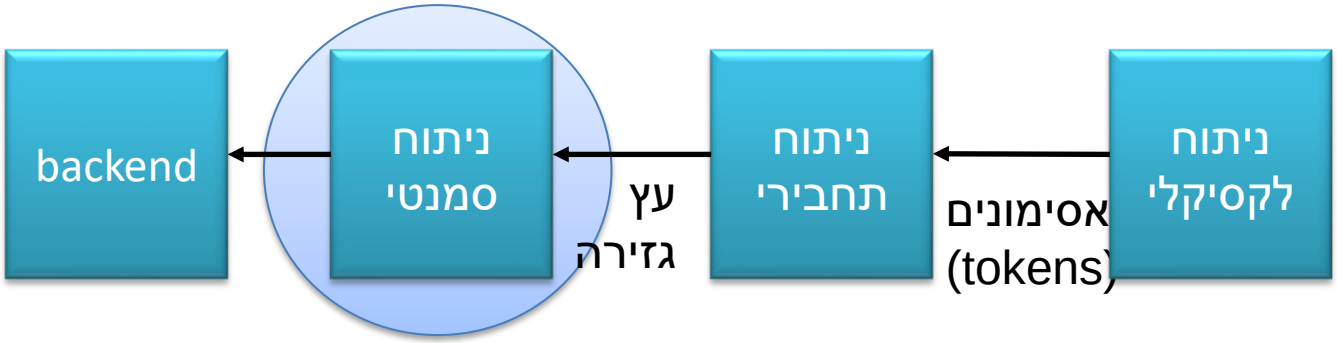
- מבנה סכמתי של קומפיילר



- ניתוח תחבירי:
  - **Top Down:** RD LL(1)
  - **Bottom up**:
    - LR(0)
    - SLR
    - CLR \LR(1)
    - LALR

תזכורת מהתרגולים הקודמים

- מבנה סכמתי של קומפיילר



הצורך בניתוח סמנטי-דוגמא 1

התאמה בין תג פותח וסוגר ב XML

Tag →S_TAG TXT E_TAG

TXT	[a-z]*
S_TAG	< [a-z]+ >
E_TAG	<\ [a-z]+ >

<teacher> eran <\teacher>

<student> roni <\student>

<teacher> bony <\student>

למרות שתחבירית התוצאה נכונה, נרצה שתג ההתחלה יהיה זהה לתג הסוף

Attribute Grammar

התאמה בין תג פותח וסוגר ב XML

עבור כלל הגזירה

Tag →S_TAG TXT E_TAG

נצמיד ל- S_TAG ול- E_TAG תכונה סמנטיות שנקראת name.

ונגדיר פעולה סמנטית:

```
If (S_TAG.name != E_TAG.name)
        error();
```

הצורך-דוגמא 2

בדיקת טיפוסים

statement → id ASSIGN id

a	=	b
id	ASSIGN	id

יש צורך לבדוק האם המשתנה **a** הוא אכן **מאותו טיפוס** של **b**.

Attribute Grammar

בדיקת טיפוסים

statement → id ASSIGN id

נוסיף תכונות סמנטיות:

1. $id.name$
2. $symbolTable.Type: \quad name \mapsto Type$

נדרוש פעולה סמנטית:

```
If (symbolTable.Type(id1.name) != symbolTable.Type(id2.name))
        error();
```

הגדרה מונחית תחביר
syntax directed definition

- פורמליזם המשלב ניתוח תחבירי וסמנטי
- מאפשר לבצע **בדיקות ופעולות סמנטיות בזמן הניתוח התחבירי**
- **תכונה סמנטית:** טיפוס בעל שם המוצמד למשתנה או לטרמינל
- **מופע של תכונה**: תכונה של מופע כלשהו של משתנה או טרמינל בעץ גזירה נתון
- **כלל סמנטי**: פקודה המותאמת לכלל גזירה מסוים ומגדירה ערך מופע תכונה ע"י ערכי מופעים אחרים של התכונה או של תכונות אחרות

הגדרה מונחית תחביר -- דוגמה

<u>המטרה</u>: חישוב (והדפסת) ערך של ביטוי אריטמטי

<u>תכונות</u>: נגדיר תכונה מטיפוס int בשם VAL
למשתנה גזירה E ולטרמינל num.

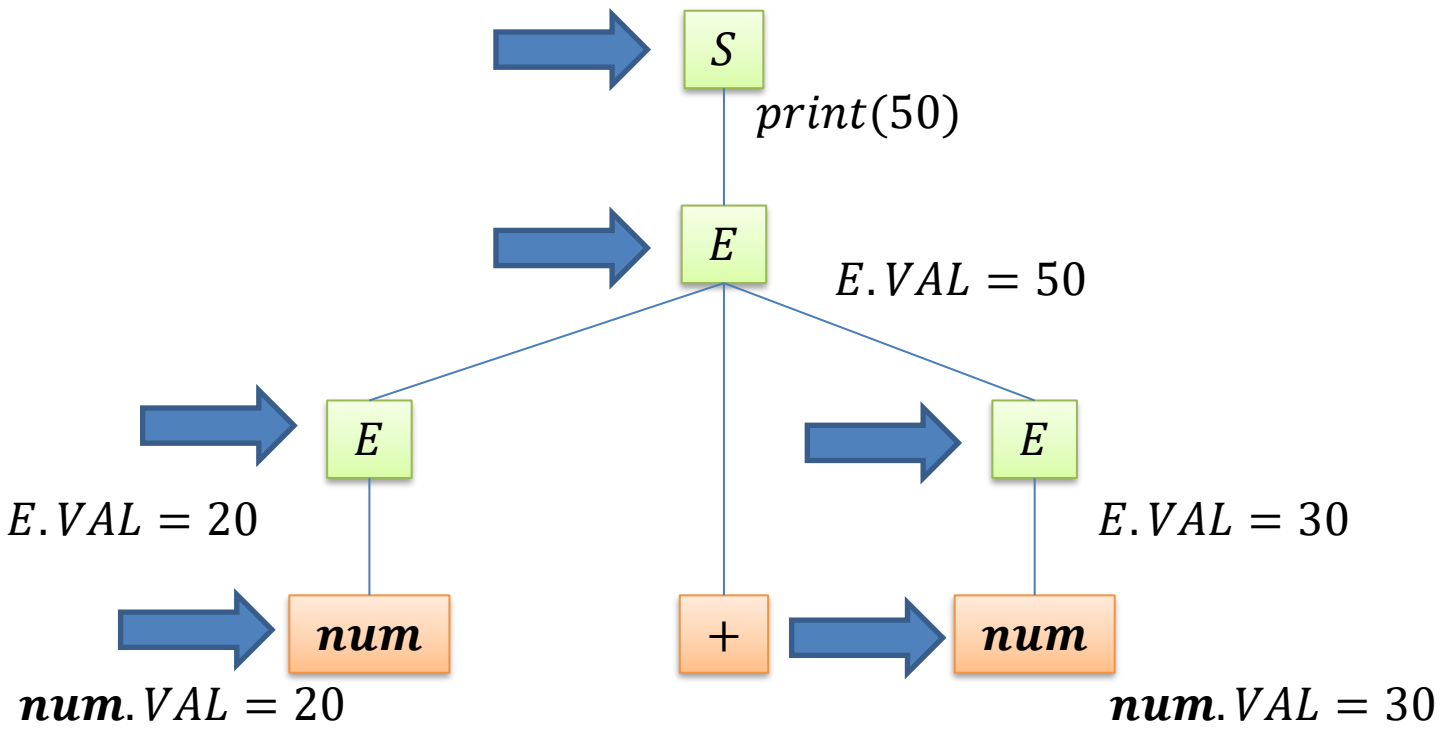
המנתח הלקסיקלי יספק את ערך VAL עבור כל מופע של הטרמינל num.
נבצע את חישוב התכונות בטיול על העץ

<u>הגדרות עבור התחביר</u>:

	כלל הגזירה	כלל סמנטי
(1)	$S \rightarrow E$	$print(E.VAL)$
(2)	$E \rightarrow E_1 + E_2$	$E.VAL = E_1.VAL + E_2.VAL$
(3)	$E \rightarrow \boldsymbol{num}$	$E.VAL = num.VAL$

עץ סינטקס עם תכונות (Annotated)

<u>העץ המורחב המתקבל עבור w='20+30'</u>:



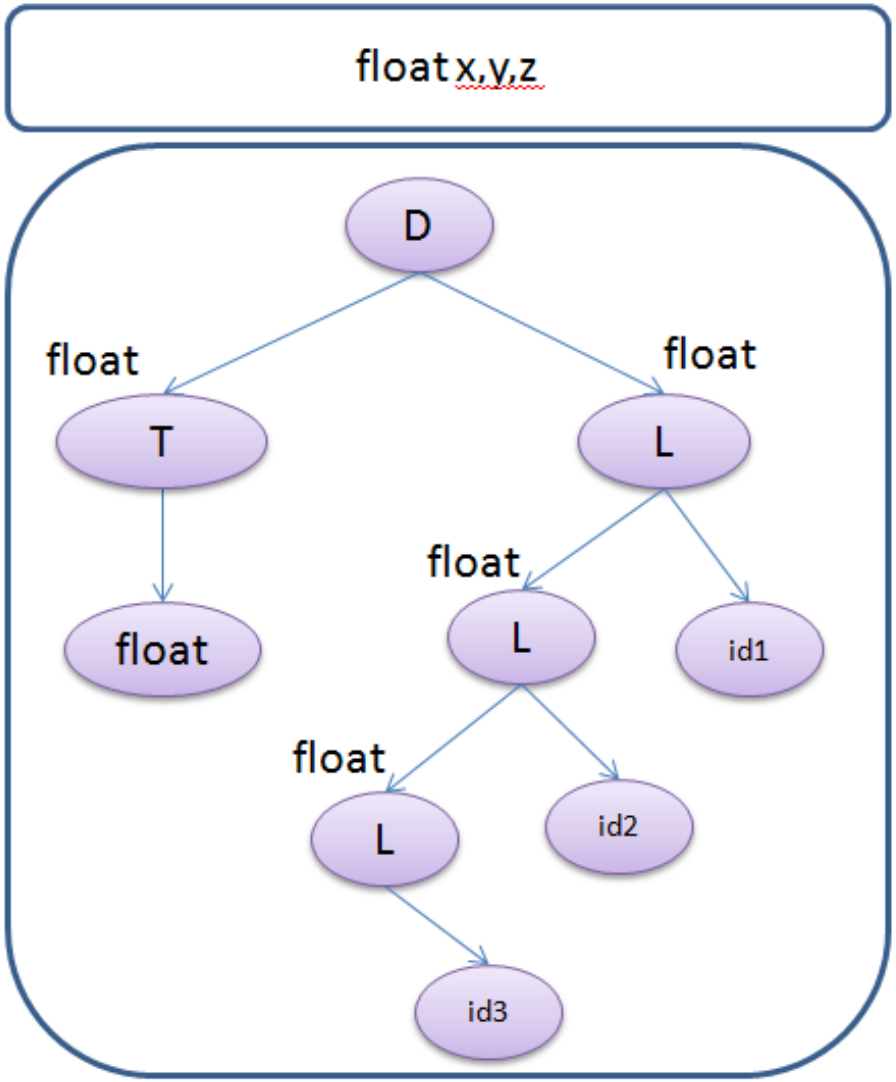
דוגמה נוספת (תזכורת מההרצאה)

- לדוגמא רוצים לגזור: '*float x,y,z*'
- להלן דקדוק המייצר הגדרה של משתנים מטיפוס מסוים:

$D \rightarrow T\ L$
$T \rightarrow$ **int**
$T \rightarrow$ **float**
$L \rightarrow L_1$, **id**
$L \rightarrow$ **id**

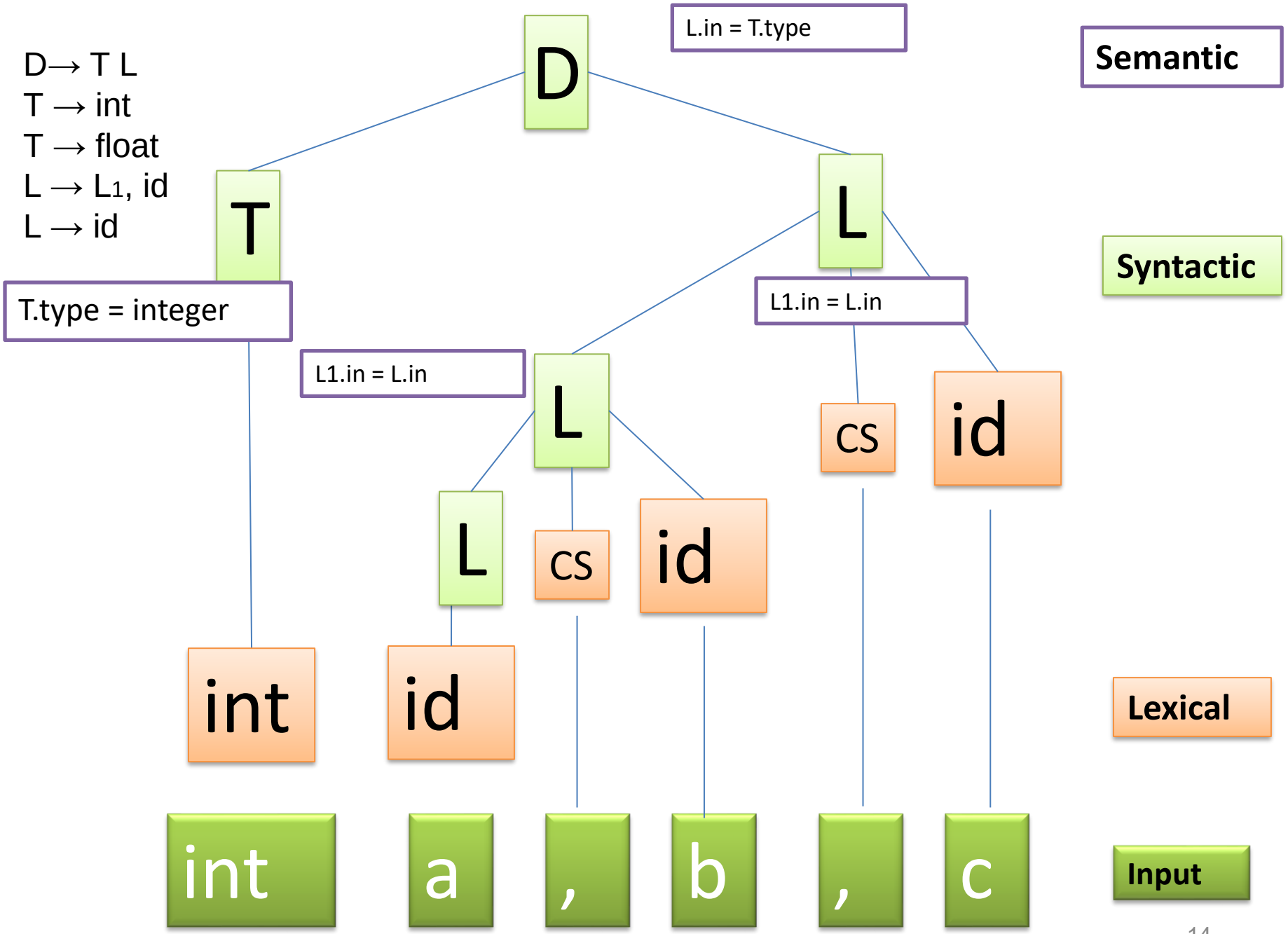
- הגדירו תכונות סמנטיות וכללים סמנטיים כדי לעדכן את טבלת הסמלים עם הטיפוסים של המשתנים בהגדרה
- הערה: יש כאן רקורסיה שמאלית

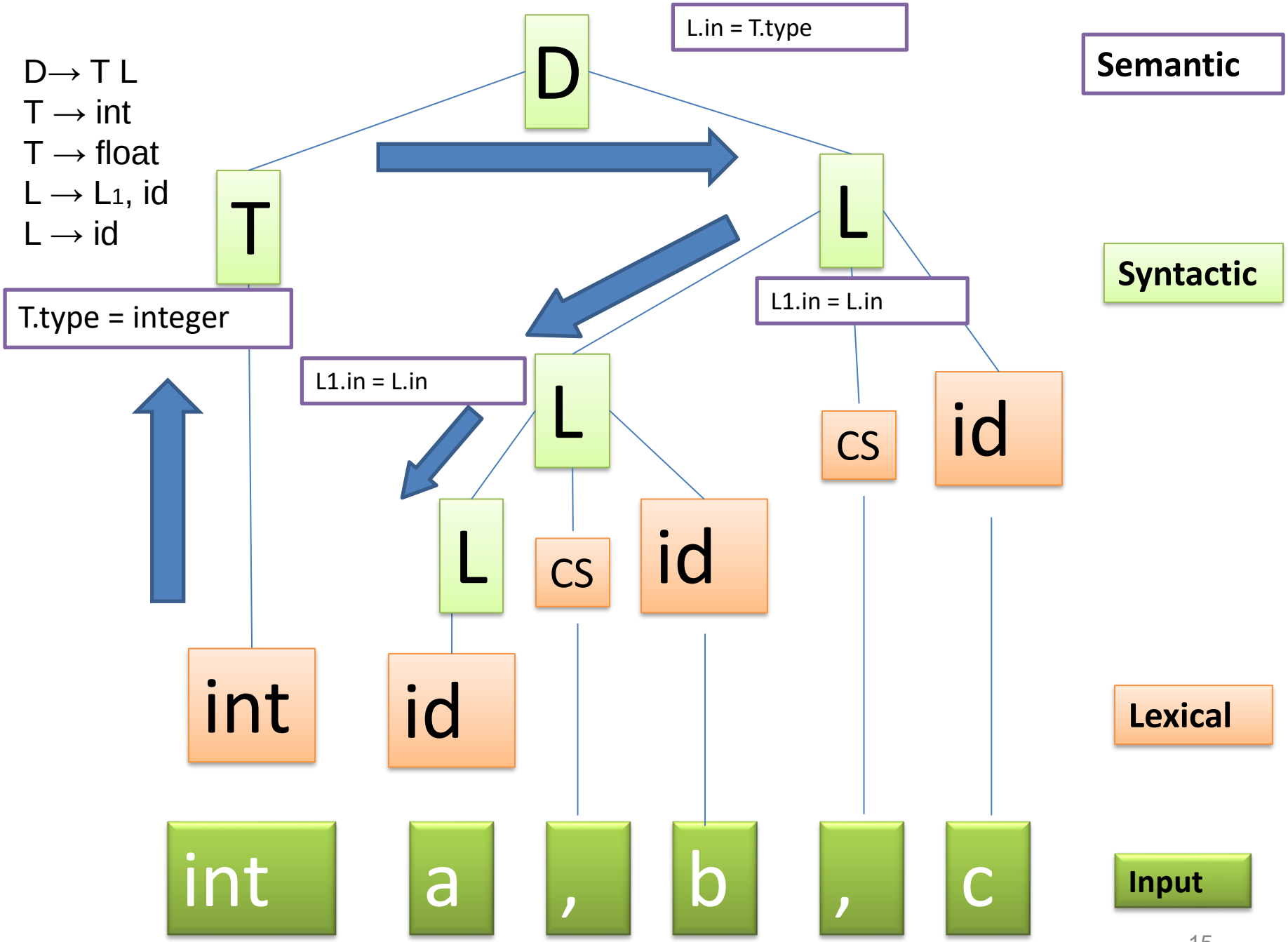
פתרון (תזכורת מההרצאה)



Production	Semantic Rule
D → T L	L.in = T.type
T → int	T.type = integer
T → float	T.type = float
L → L1, id	L1.in = L.in addType(id.entry,L.in)
L → id	addType(id.entry,L.in)

$D \rightarrow T\ L$
$T \rightarrow int$
$T \rightarrow float$
$L \rightarrow L_1, id$
$L \rightarrow id$



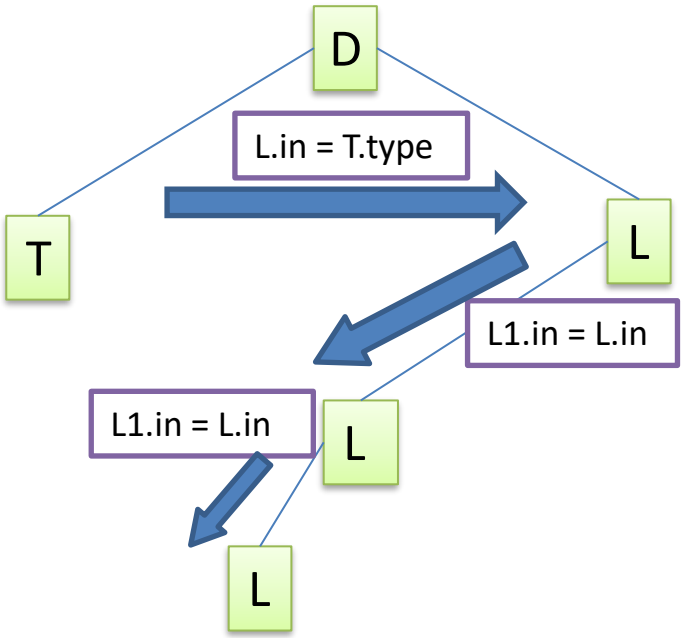

D→ T L
T → int
T → float
L → L1, id
L → id
D
L.in = T.type
T
T.type = integer
L
L1.in = L.in
L
L1.in = L.in
L
cs
id
cs
id
int
id
int
a
,
b
,
c
Semantic
Syntactic
Lexical
Input

סוגי תכונות סמנטיות

- **תכונות נוצרות (Synthesized Attribute)**
  - התכונה של משתנה V תלויה אך ורק בערכי התכונות של בנים של V בעץ הגזירה הנוצר

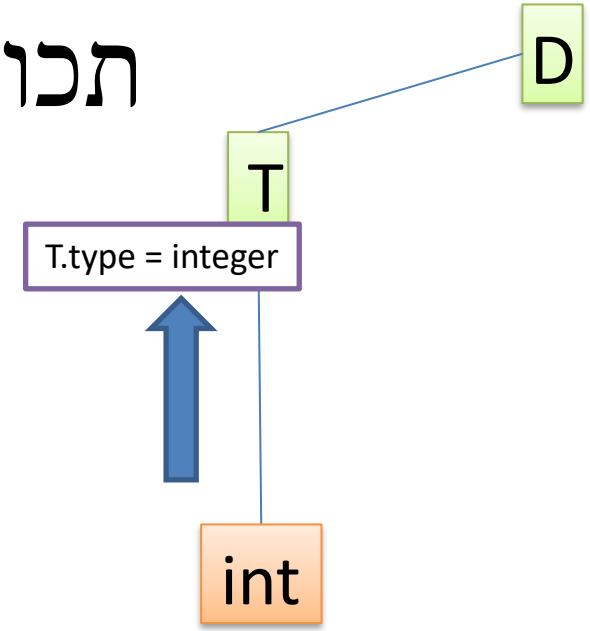
- **תכונות נורשות(Inherited Attribute)**
  - התכונה של משתנה V תלויה אך ורק בערכי התכונות של האחים או האב של V בעץ הגזירה הנוצר

תכונות נורשות - דוגמה



D→T L	{L.type = T.type}
T → int	{T.type = integer}
T → float	{T.type = float}
L → L1, id	{L1. type = L. type; addType(id.entry, L. type)}

תכונות נוצרות - דוגמה



D→T L	{L.in = T.type}
T → int	{T.type = integer}
T → float	{T.type = float}
L → L1, id	{L1.type = L.type ; addType(id.entry, L.type)}
L → id	{addType(id.entry, L.type)}

מה עדיף? תכונות נוצרות או נורשות?

שאלה ממבחן

- להלן **דקדוק המייצר ביטויים שיכולים להכיל השמות**:

$S \rightarrow E$
$E \rightarrow E := E$
$E \rightarrow E + E$
$E \rightarrow (E)$
$E \rightarrow id$
$E \rightarrow num$
$E \rightarrow * E$

- הסמנטיקה של הביטויים היא כמו בשפת C
- למשל "**b := c**" הוא ביטוי ששם את הערך של c לתוך b.
  ערך הביטוי כולו הוא כמו הערך של c.
- הביטוי "**a := (b := c)**" שם את הערך של c לתוך b ואח"כ ל a.

- **בנו הגדרה מונחית תחביר** שתבדוק כי ביטויים הממוקמים בצד שמאל של השמה הם כולם **l-values**, כלומר **ניתן לבצע השמה אליהם.**

- השתמשו בתכונה **נורשת E.target** שתקבע אם הביטוי E הוא יעד של השמה או לא.

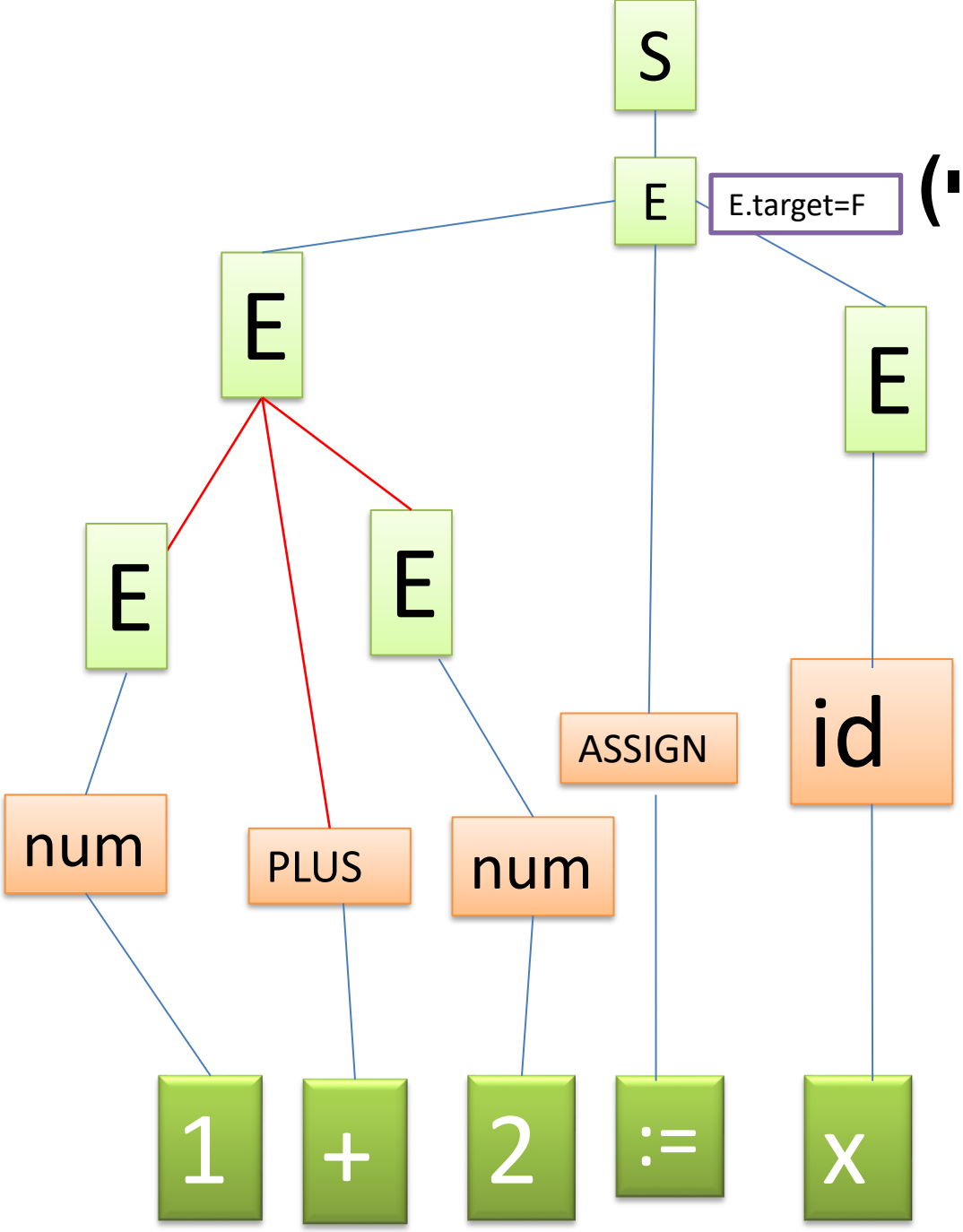
- דווחו על שגיאה במקרה שמבוצעת השמה לביטוי שאינו l-value.

פתרון

- נשתמש באוסף כללים סמנטיים שיקבעו את E.target להיות TRUE אם E הוא יעד של השמה, ויזהו מתי צד שמאל של השמה אינו יכול להיות l-value

S→ E	E.target = **FALSE**;
E → E1 := E2	E1.target = **TRUE**; E2.target = **FALSE**; **if** E.target **then error**
E → E1 + E2	E1.target = **FALSE**; E2.target = **FALSE**; **if** E.target **then error**;
E → (E1)	E1.target = E.target;
E → id	
E → num	**if** E.target **then error**
E → * E1	E1.target = **FALSE**;

דוגמה 1
(ביטוי שגוי)



S→ E

E.target = FALSE;

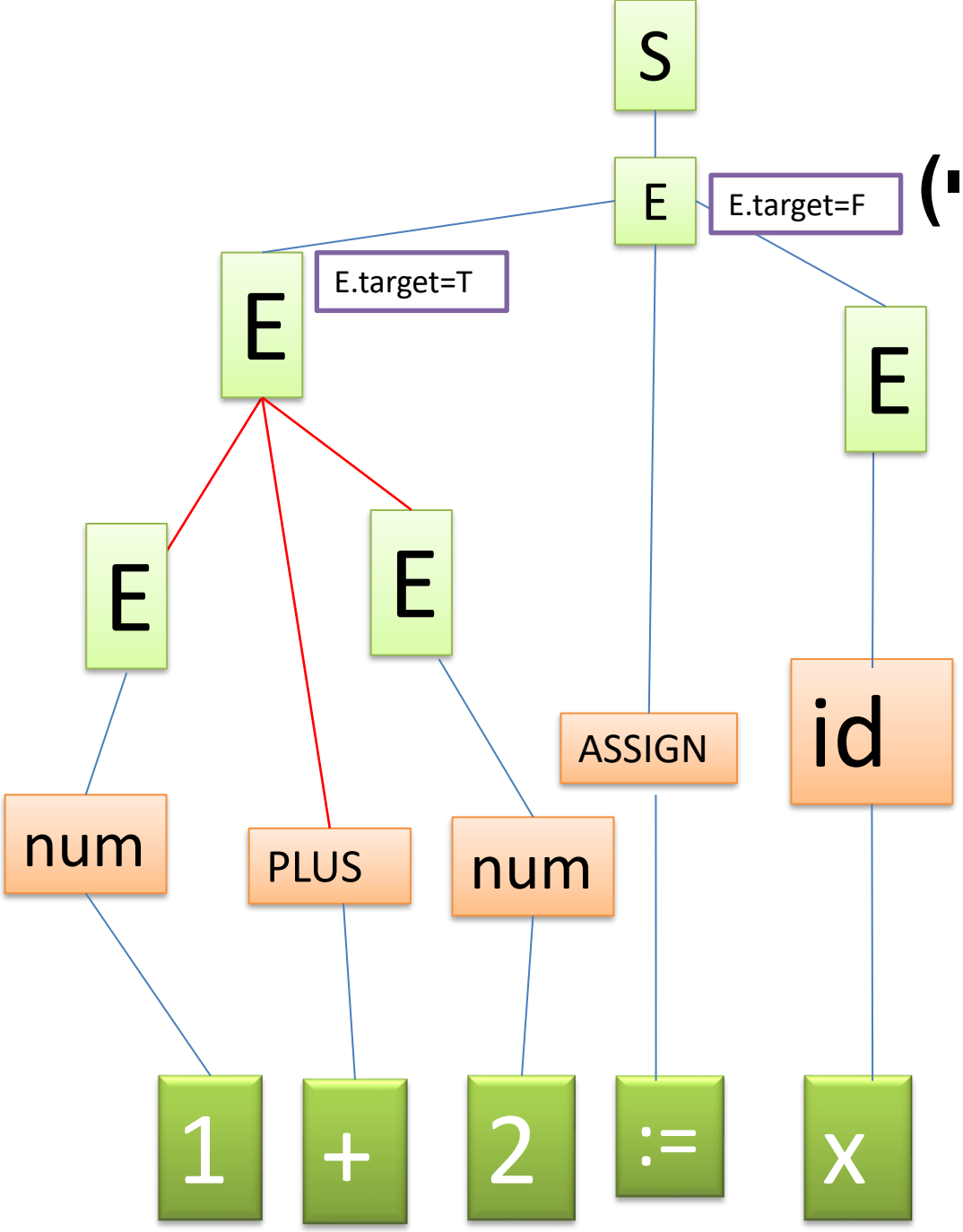
Semantic

Syntactic

Lexical

Input

דוגמה 1
(ביטוי שגוי)



Semantic

Syntactic

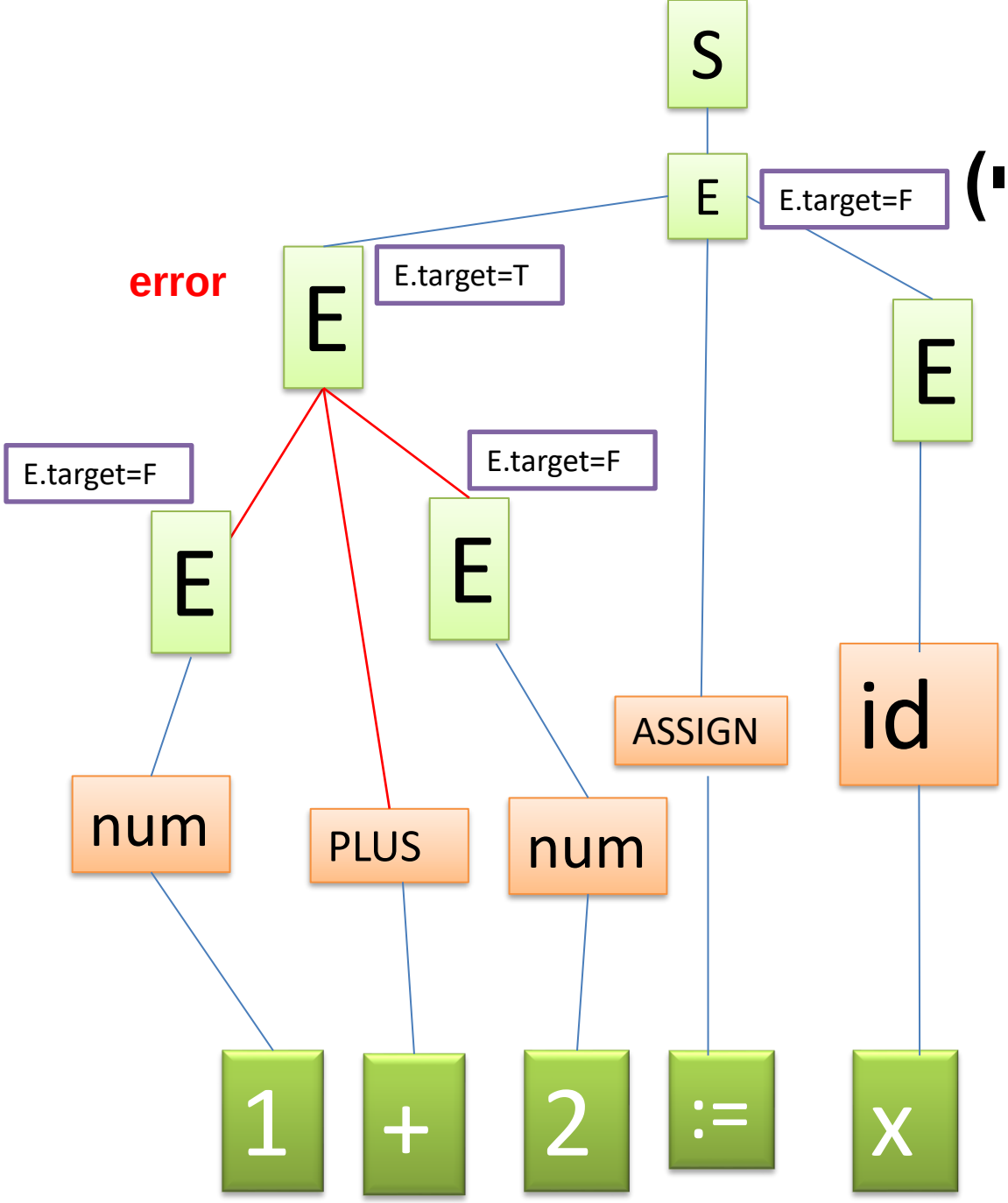
Lexical

Input

$E \rightarrow E1 := E2$

E1.target = TRUE; E2.target = FALSE;
if E.target then error

דוגמה 1 (ביטוי שגוי)



Semantic

Syntactic

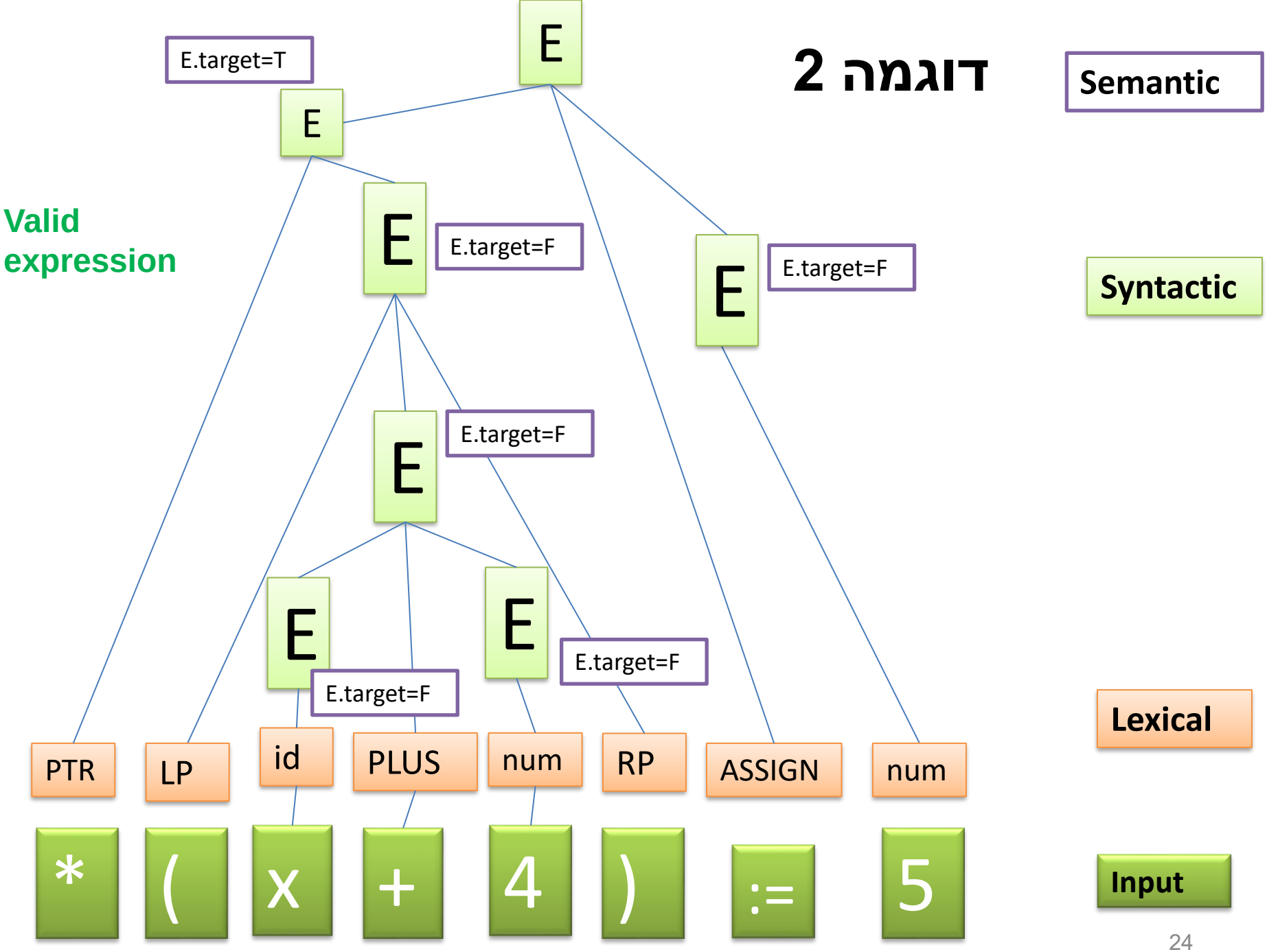
Lexical

Input

$E \rightarrow E1 + E2$

E1.target = FALSE; E2.target = FALSE;
if E.target then error;

דוגמה 2


Semantic
Syntactic
Lexical
Input
Valid expression
E
E.target=T
E
E.target=F
E
E.target=F
E
E.target=F
E
E.target=F
E
E.target=F
PTR
LP
id
PLUS
num
RP
ASSIGN
num
*
(
x
+
4
)
:=
5

המשך השאלה

- האם יש דרך אחרת לאכוף את הבדיקה הזו?
(כי ביטויים הממוקמים בצד שמאל של השמה הם כולם l-values)

פתרון

- ניתן על ידי **שינוי הדקדוק**:
- <u>הרעיון</u>: נשתמש במשתנה גזירה חדש עבור ביטויים שנמצאים ב LHS(צד שמאלי) של השמה
- משתנה הגזירה החדש יגזור את מה ש E גוזר פרט לגזירות שמייצרות ביטויים שאינם l-value

המשך הפתרון

- הדקדוק שנקבל:

S -> E
E -> **LV** := E
E -> E + E
E -> (E)
E -> id
E -> num
E -> * E
LV -> (LV)
LV -> id
LV -> * E

דקדוקי S - S-attributed Grammars

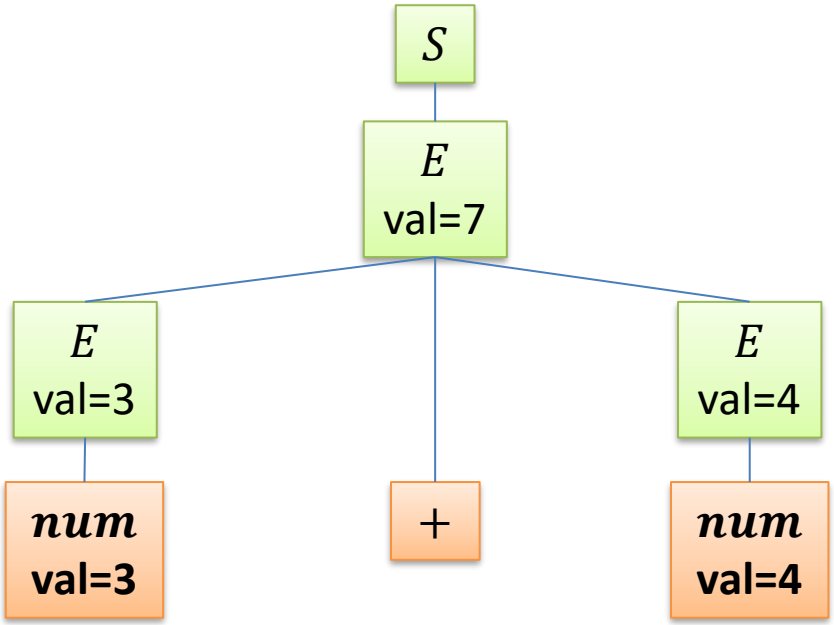
- הגדרה:
  - משתמש רק בתכונות נוצרות (Synthesized).

- יתרונות:
  - הניתוח הסמנטי מבוצע תוך כדי בניית עץ הגזירה במנתחי bottom-up.
  - אין צורך במעבר נוסף על עץ הגזירה

- מבצעים את הפעולה הסמנטית <u>בסוף הכלל</u>, מיד לאחר הגזירה

Bottom-up עם או בלי עץ

- דקדוק S מתאים לעבודה עם מנתח bottom-up
- אפשר לבנות את העץ ו"לקשט אותו" בתכונות סמנטיות תוך כדי:



- ואפשר גם לעבוד בלי עץ בכלל.

ניתוח LR ובייזון: תזכורת

- Bison בונה מנתח LALR
  - reduce מוציא מהמחסנית את הילדים
  - ומכניס למחסנית את המשתנה הנגזר
- מבצעים את החישוב בהתבסס על הילדים מהמחסנית
  - S-attributed!

בייזון: תזכורת

```
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 - (n>0) הסימן ה-n באגף ימין של החוק

A→ B C NUM D E

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

שאלה (ממבחן)

- נתון הדקדוק הנל' שמאפשר הכרזה על משתנים

$$DEC \rightarrow type\ L$$
$$L \rightarrow id|id, L$$

הדקדוק גוזר למשל את המשפט int x,y,z.

כתבו דקדוק S ב-Bison עבור הדקדוק, כך שבסיום הגזירה לDEC כל המשתנים יהיו בטבלת הסמלים.

- ניתן להניח כי קיימת פונקציה insert(type,id) שמכניסה משתנה בשם id מטיפוס type לטבלת הסמלים.

פתרון, שלב 1: YYSTYPE

- גם האסימונים וגם הצמתים יהיו מטיפוס YYSTYPE
- מה צריך לאכסן?
  - לאסימון type יש **value** שהוא שם הטיפוס.
  - לאסימון id יש **name** שהוא שם המשתנה.
- כשגזרנו את L, יהיו כבר כמה name מכמה id

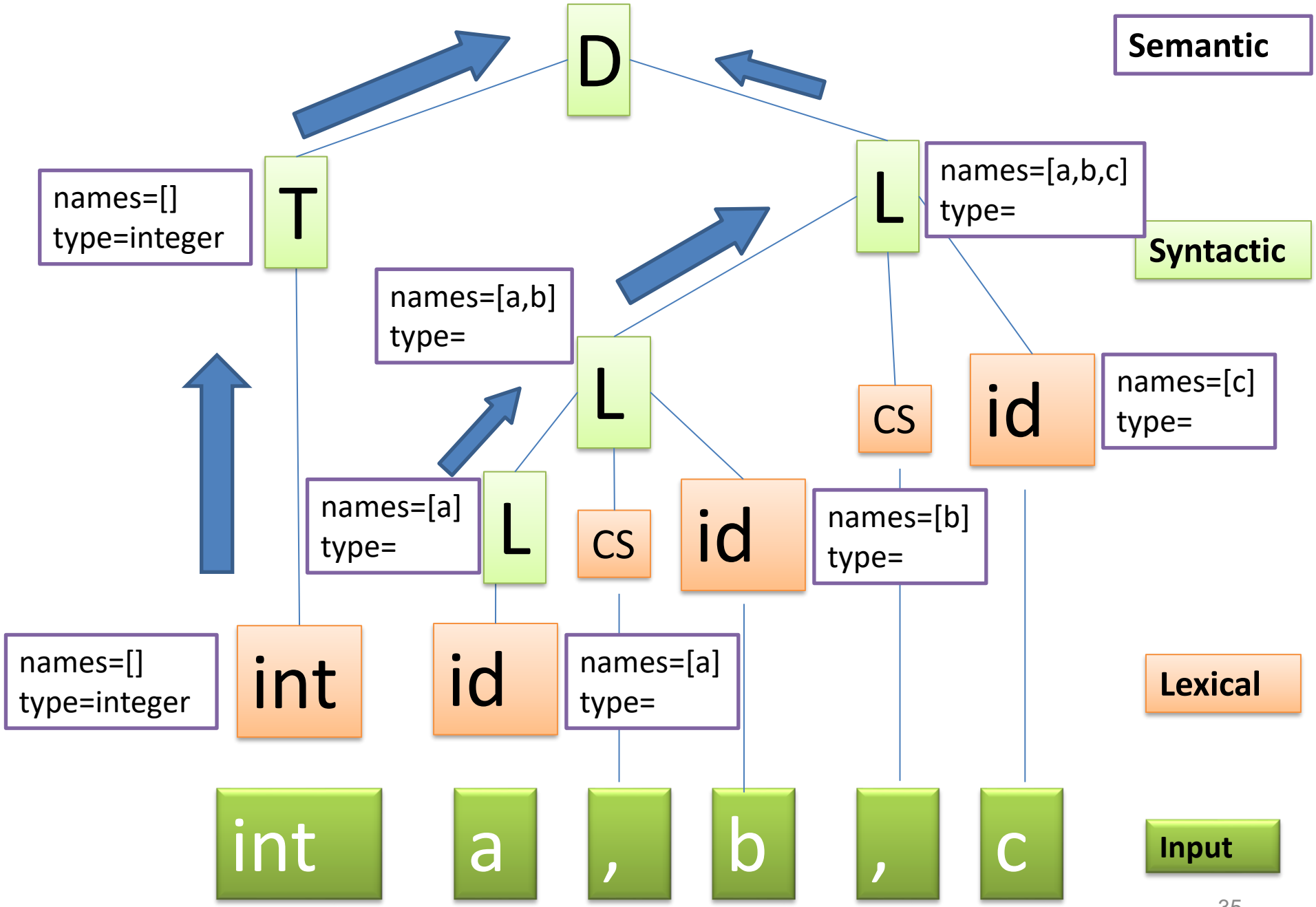
```
struct NameTypeInfo {
        std::vector<std::string> names;
        std::string type;
};
```

לשם הקצרנות אין טיפוס נפרד לכל נונטרמינל. אבל כשיש יותר אינפורמציה, חייבים!

פתרון, שלב 2: כללים סמנטיים

- להוסיף את האינפורמציה בשלב הלקסינג (גם flex עובד עם YYSTYPE)
- כללי גזירה:

```
DEC : type L {for (int i = 0; i < $2.names.size(); ++i)
                    insert($1.type, $2.names[i]);};
L : id , L  {$$.names = $3.names ;
             $$.names.push_back($1.names[0]); };
  | id      {$$.names = $1.names;};
```


Semantic
D
T
names=[]
type=integer
L
names=[a,b,c]
type=
Syntactic
L
names=[a,b]
type=
L
cs
id
names=[c]
type=
names=[a]
type=
L
cs
id
names=[b]
type=
names=[]
type=integer
int
id
names=[a]
type=
Lexical
int
a
,
b
,
c
Input

דקדוקי S- המשך

- מגבלות:
  - מבצעים את הפעולה הסמנטית רק <u>בסוף הכלל</u>.

S→AB	{print (“a & b”)}
A→a	{print (“a”)}
B→b	{print (“b”)}

Output: a b a&b

  - מה קורה אם נרצה לבצע פעולה <u>באמצע הכלל</u>?
    - כלומר, רוצים שתתבצע פעולה באמצע הגזירה S→A_B

Output: a a&b b

הפתרון: מרקרים

S→AMB { }

A→a {print (“a”)}

B→b {print (“b”)}

M→ε {print (“a & b”)}

Output: a a&b b

- אבל... שינויים בדקדוק עלולים לגרום לדקדוק שהיה ב-LR, כבר לא להיות ב-LR (עקב **יצירת קונפליקט**):

S→abc | abd → S→abc | aMbd

M→ε {print ...)}

בבייזון:Mid-Rule Actions

- שורה כזו בבייזון:

A : B {printf(“b”);} C {printf(“c”);};

- תתורגם כ

A : B M C {printf(“c”);};

M: {printf(“b”);};

גישה למחסנית

- בגרסאות חדשות של בייזון ניתן לגשת למחסנית מאמצע כלל:

```
A : B {printf("%d",$1);} C {printf("%d",$3);};
```

- איך כזה דבר מתורגם?

```
A : B M C {printf("%d",$3);};
M : { YYSTYPE b = stack.peek(0); //top
      printf("%d",b); };
```

- שימו לב! $$ באמצע כלל הוא $$ של המרקר!

קונפליקטים

- מה קורה כשיש כמה מרקרים?

```
A : B { print("a"); } C
  | B { print("a"); } D;
```

יהפוך ל:

```
A : B M1 C
  | B M2 D;
```

קונפליקט! איך נפתור זאת? (פתרון על הלוח)