Μεταγλωττιστές 2020-21

Συντακτική ανάλυση top-down LL(1)

Συντακτική ανάλυση top-down LL(1)

- Για μια σημαντική υποκατηγορία context-free γραμματικών με πολυπλοκότητα O(n)
  - Σάρωση εισόδου από αριστερά προς τοα δεξιά (left-to-right)
  - Αριστερότερες ακολουθίες παραγωγής (leftmost derivations)
  - Συμβουλευόμενοι ένα (το επόμενο) token εισόδου σε κάθε βήμα

Προϋποθέσεις για αριστερότερες παραγωγές

- Δεν πρέπει οι κανόνες να έχουν αριστερή αναδρομή

  `Expr → Expr + Term`

  - Ο συντακτικός αναλυτής θα πέσει σε άπειρη αναδρομή
  - Απαλείφεται με αλγοριθμικό τρόπο

    `A → A α | β`

    γίνεται

    `A → β A'`

    `A'→ α A' | ε`

  - Προσοχή: πρέπει να γίνει και σε περιπτώσεις έμμεσης αναδρομής (αλυσιδωτά, μετά από περισσότερα από ένα βήματα)

Προϋποθέσεις για αριστερότερες παραγωγές

- Δεν πρέπει να υπάρχει κοινός παράγοντας στην αρχή π.χ.

```
Factor → id | id [ ExprList ] | id ( ExprList )
```

- Μετασχηματισμός με προσθήκη ενδιάμεσου κανόνα

```
Factor → id Args

Arg    → ε | [ ExprList ] | ( ExprList )
```

Υλοποίηση συντακτικού αναλυτή LL(1)

- Από τη Θεωρία Υπολογισμού έχουμε ένα θεωρητικό εργαλείο
  - Pushdown Automaton (PDA)
  - Αυτόματο ενισχυμένο με μια στοίβα (stack)
  - Για κάθε μετάβαση το αυτόματο συμβουλεύεται το σύμβολο εισόδου και την κορυφή της στοίβας
    - για να αποφασίσει ποια θα είναι η επόμενη κατάσταση
    - και αν/τι θα μπει στη στοίβα
  - Υπάρχει πάντα ένα αυτόματο PDA ισοδύναμο με μια γραμματική χωρίς συμφραζόμενα (CFG)
  - Για τις γραμματικές LL(1) το αυτόματο PDA είναι αιτιοκρατικό (ντετερμινιστικό)
    - είναι δυνατή το πολύ μια μετάβαση

Πώς χρησιμοποιείται το αυτόματο PDA στη συντακτική ανάλυση

- Αρχικά στη στοίβα μπαίνει το αρχικό μη τερματικό σύμβολο της γραμματικής
- Το αυτόματο επαναλαμβάνει
  - Αν στην κορυφή της στοίβας βρίσκεται μη τερματικό σύμβολο Α (λειτουργία predict), τότε το Α αντικαθίσταται στη στοίβα από το δεξιό μέρος ενός κατάλληλου κανόνα της γραμματικής, σύμφωνα και με το επόμενο σύμβολο στην είσοδο
  - Αν στην κορυφή της στοίβας βρίσκεται τερματικό σύμβολο a (λειτουργία match), τότε, αν το επόμενο σύμβολο εισόδου είναι ίδιο με το a, το a αφαιρείται από τη στοίβα
  - Σε κάθε άλλη περίπτωση παράγεται σφάλμα

Απλή (αλλά μη ρεαλιστική) περίπτωση LL(1)

- Έστω η γραμματική

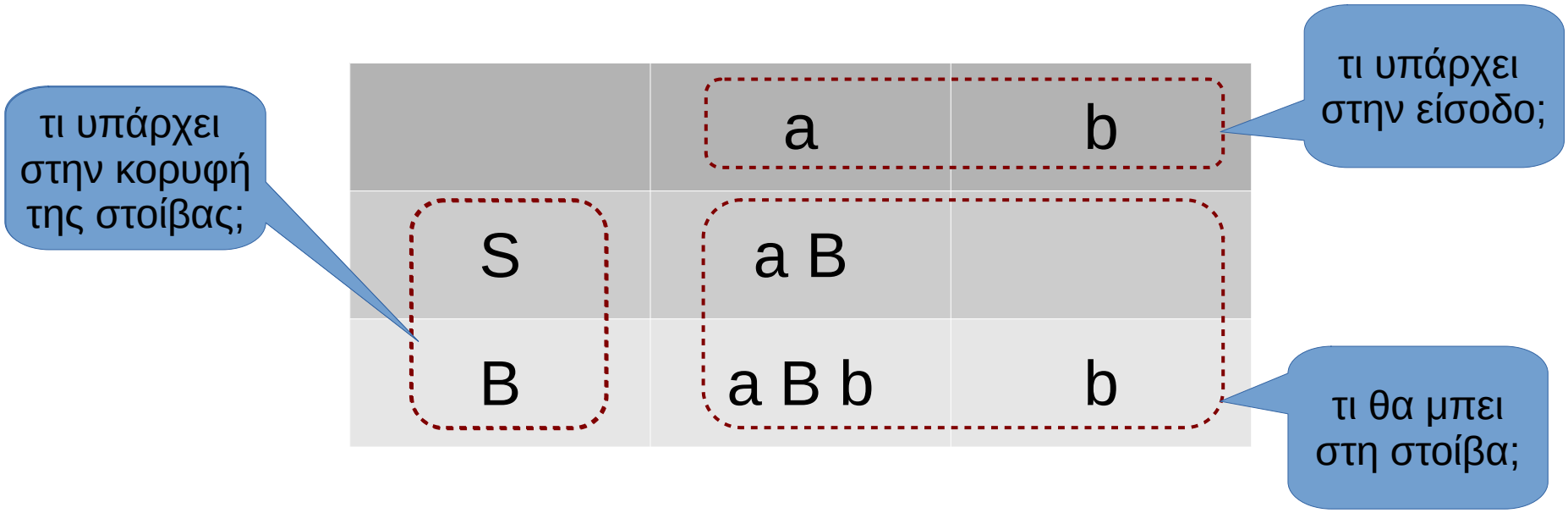
  S → a B

  B → b | a B b

- Όλα τα δεξιά μέρη των κανόνων ξεκινούν με τερματικό σύμβολο, διαφορετικό για τις εναλλακτικές παραγωγές του ίδιου μη τερματικού συμβόλου

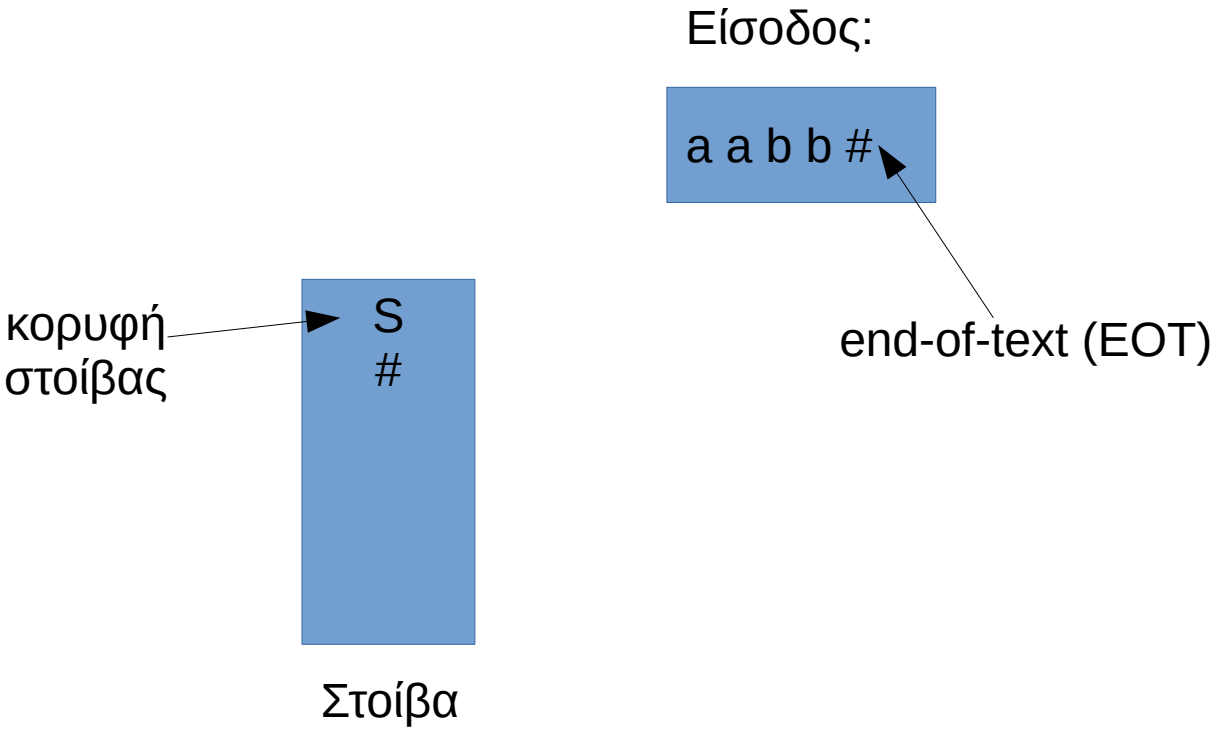
Πίνακας συντακτικής ανάλυσης

- Θεωρητικό εργαλείο, μέρος του αυτομάτου PDA, αλλά μπορεί να χρησιμοποιηθεί και σε έναν πραγματικό συντακτικό αναλυτή
  - επιλέγει τι θα μπει στη στοίβα στο predict

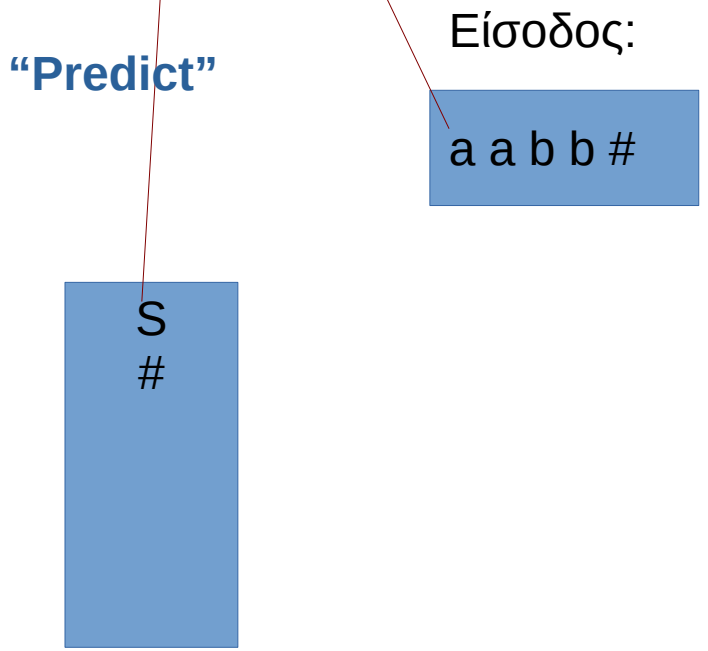


Π.χ. αν στην κορυφή της στοίβας είναι το S και το επόμενο σύμβολο στην είσοδο είναι το a, το S θα αντικατασταθεί στη στοίβα από το a B

	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

“Predict”

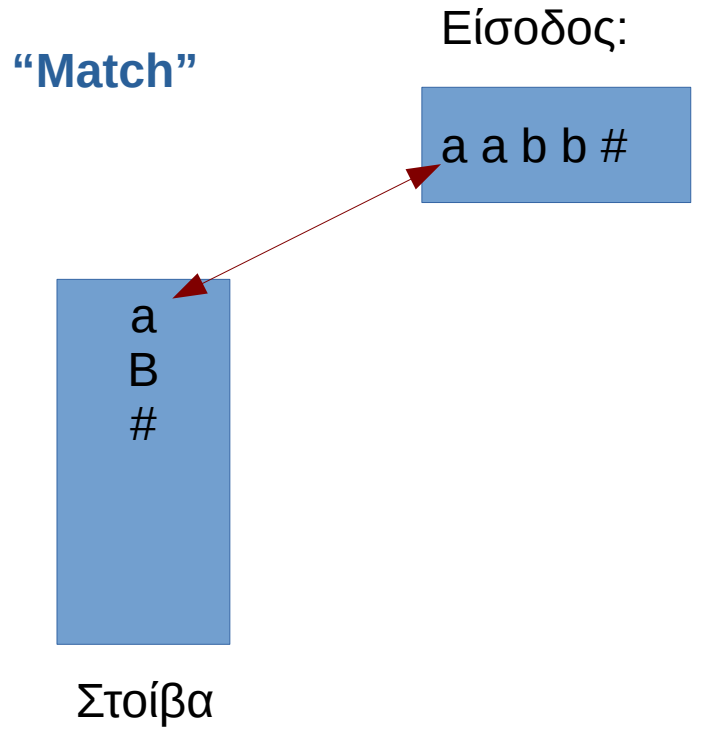
Είσοδος:

a a b b #

a
B
#

Στοίβα

	a	b
S	a B	
B	a B b	b

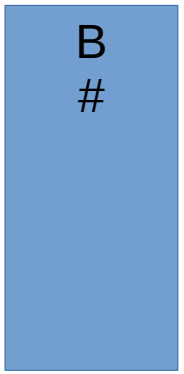


	a	b
S	a B	
B	a B b	b

Είσοδος:

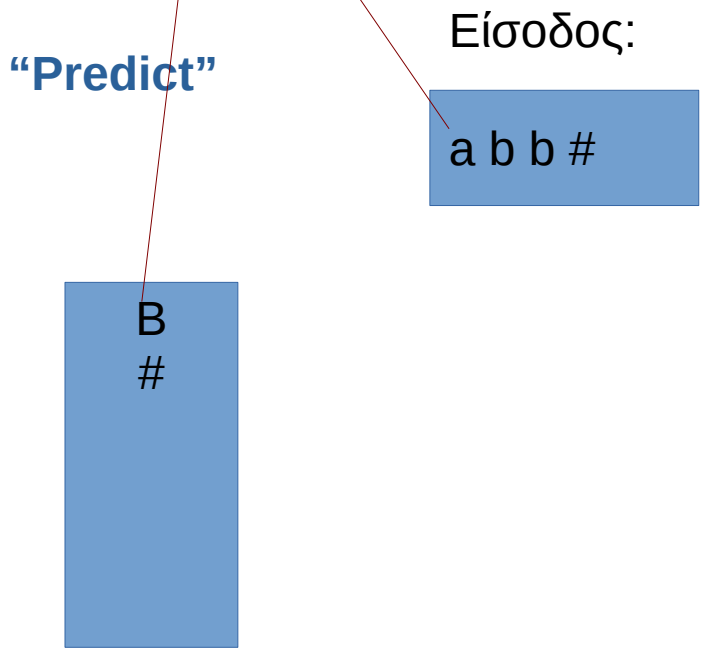
a b b #

“Match”



Στοίβα

	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

“Predict”

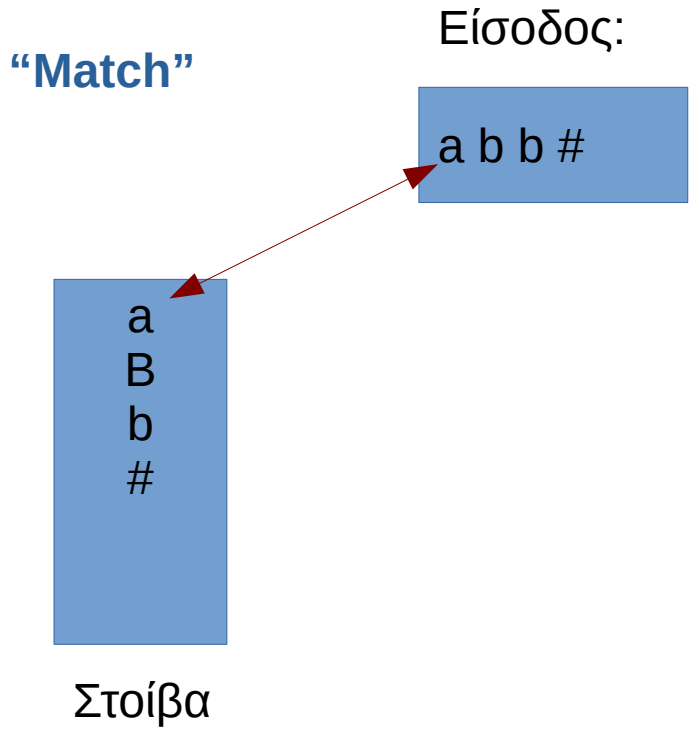
Είσοδος:

a b b #

a
B
b
#

Στοίβα

	a	b
S	a B	
B	a B b	b

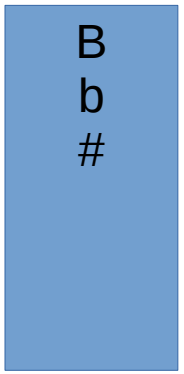


	a	b
S	a B	
B	a B b	b

Είσοδος:

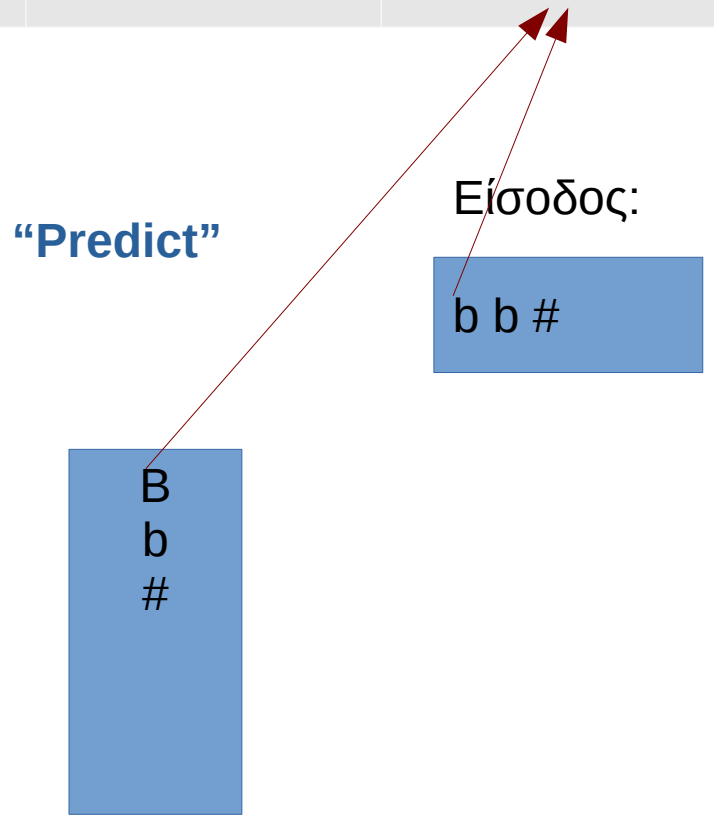
b b #

“Match”



Στοίβα

	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

“Predict”

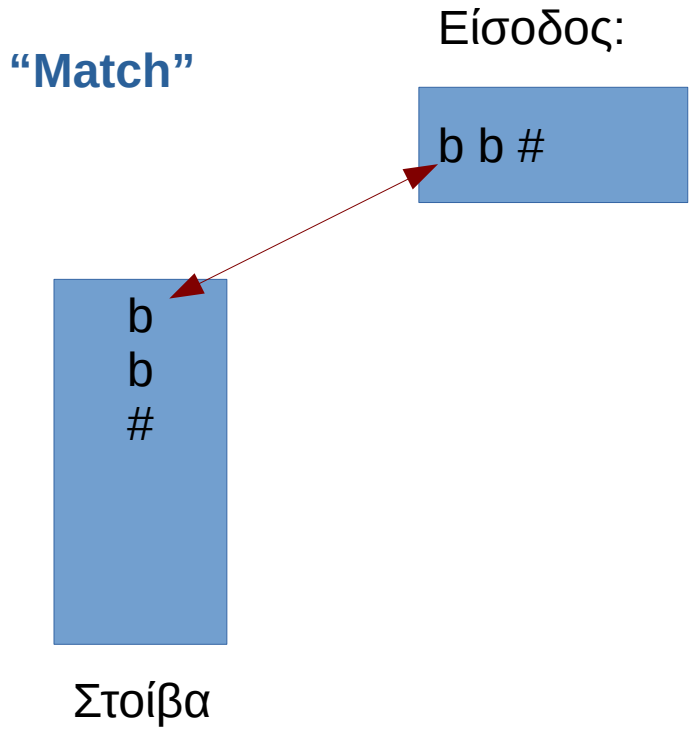
Είσοδος:

b b #

b
b
#

Στοίβα

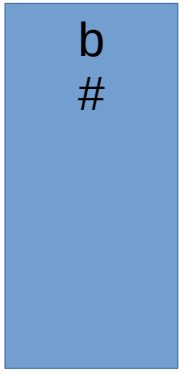
	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

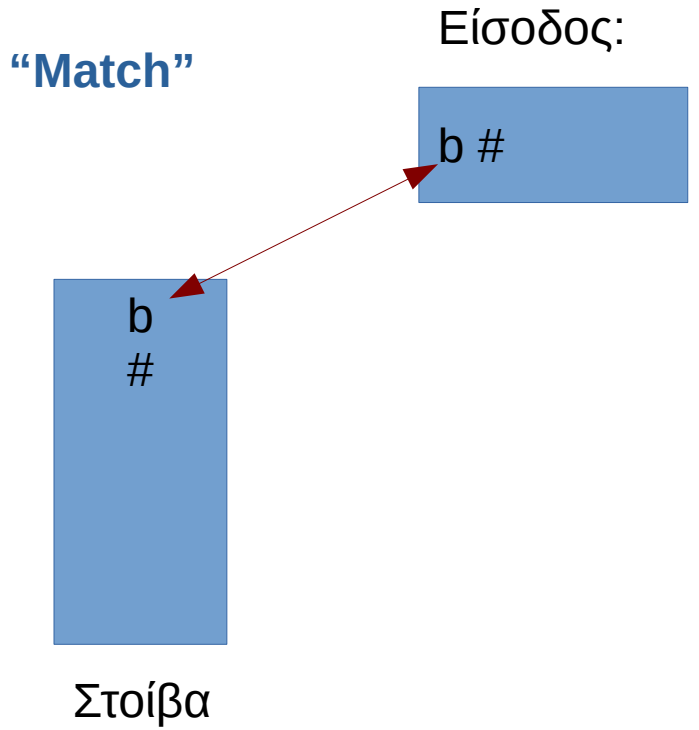
Είσοδος:

“Match”



Στοίβα

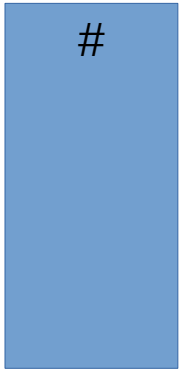
	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

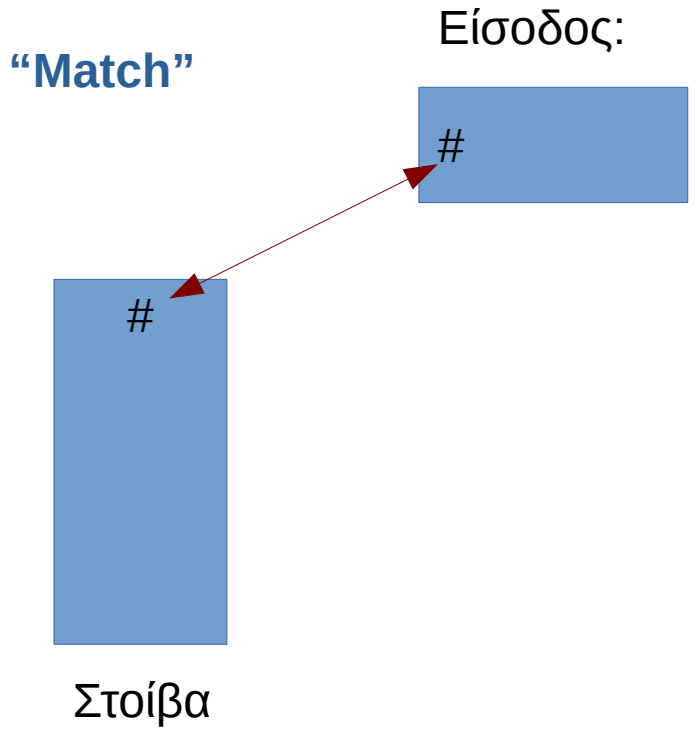
“Match”

Είσοδος:



Στοίβα

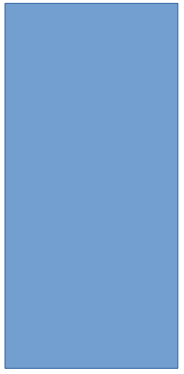
	a	b
S	a B	
B	a B b	b



	a	b
S	a B	
B	a B b	b

Είσοδος:

Τερματισμός



Στοίβα

Κατασκευή συντακτικού αναλυτή LL(1)

- Θα μπορούσαμε να φτιάξουμε ένα πρόγραμμα που να μιμείται τη λειτουργία ενός PDA
- Είναι όμως πιο βολικό να υλοποιήσουμε με συστηματικό τρόπο τον συντακτικό αναλυτή ως σύνολο συναρτήσεων που καλούν η μία την άλλη (ενδεχομένως και αναδρομικά)
  - «Stack»: υλοποιείται από τη στοίβα κλήσης των συναρτήσεων
  - Μπορούμε να κάνουμε τη σημασιολογική ανάλυση μέσα στις συναρτήσεις αυτές
- Συντακτικός αναλυτής αναδρομικής κατάβασης (recursive descent parser)

Μέθοδος αναδρομικής κατάβασης

- Για κάθε μη τερματικό σύμβολο φτιάχνουμε μια συνάρτηση
  - Καλείται πρώτη η συνάρτηση του αρχικού μη τερματικού συμβόλου της γραμματικής
- Κάθε συνάρτηση υλοποιεί τους κανόνες που έχουν στο αριστερό μέρος το αντίστοιχο μη τερματικό σύμβολο
  - Ένας κλάδος if για κάθε κανόνα
    - Πώς διαλέγω; στην απλή γραμματική του παραδείγματος, αρκεί να ελέγχω το επόμενο σύμβολο εισόδου – δεν αρκεί φυσικά για πιο σύνθετες γραμματικές!
- Υλοποίηση κανόνων
  - Υλοποιώ το δεξιό μέρος του κανόνα
    - Όπου υπάρχει τερματικό, καλώ συνάρτηση match()
      - Η match(), αν η είσοδος ταιριάζει με το αναμενόμενο τερματικό, προχωρά στο επόμενο σύμβολο εισόδου
    - Όπου υπάρχει μη τερματικό, καλώ την αντίστοιχη συνάρτηση

Παράδειγμα

```
def B():

    if next_token=='B_TOKEN':
        # B -> b
        match('B_TOKEN')

    elif next_token=='A_TOKEN':
        # B -> a B b
        match('A_TOKEN')
        B()
        match('B_TOKEN')

    else:
        raise ParseError("...error msg...")
```

Γραμματική LL(1) αριθμητικών εκφράσεων

```
Stmt_list   → Stmt Stmt_list | ε
Stmt        → id = Expr | print Expr
Expr        → Term Term_tail
Term_tail   → Addop Term Term_tail | ε
Term        → Factor Factor_tail
Factor_tail → Multop Factor Factor_tail | ε
Factor      → (Expr) | id | number
Addop       → + | -
Multop      → * | /
```

- Πιο σύνθετη από το προηγούμενο παράδειγμα
  - Υπάρχουν κανόνες που το δεξιό μέρος ξεκινά με μη τερματικό
  - Υπάρχουν κενές παραγωγές (με ε στο δεξιό μέρος των κανόνων)
- Ποιους κανόνες μπορείτε να υλοποιήσετε με όσα ξέρετε μέχρι τώρα;

Διατύπωση προβλήματος

- Αν το δεξιό μέρος ενός κανόνα ξεκινά με τερματικό, ξέρω πώς θα τον επιλέξω για υλοποίηση

```
    def B():

        if next_token=='B_TOKEN':
            # B -> b
            match('B_TOKEN')
```

- Αν ξεκινά με μη τερματικό, πώς γίνεται η επιλογή;

```
    def Expr():

        if next_token==??????:
            # Expr -> Term Term_tail
            Term()
            Term_tail()
```

- Κι αν ήξερα όλα τα πιθανά τερματικά με τα οποία ξεκινούν οι προτάσεις που παράγει το Term;

Σύνολα FIRST

- FIRST(x) είναι το σύνολο των τερματικών από τα οποία ξεκινά κάθε πρόταση που παράγεται από το x σε 0 ή περισσότερα βήματα
  - Όπου x οποιαδήποτε ακολουθία τερματικών και μη τερματικών συμβόλων
  - Αν x ξεκινά με τερματικό σύμβολο a, το FIRST(x) περιέχει το ίδιο το a
  - Αν x ξεκινά με μη τερματικό σύμβολο A, το FIRST(x) ισούται με το FIRST(A)

Εύρεση συνόλων FIRST γραμματικής (όταν δεν υπάρχουν κενές παραγωγές με ε)

- Ξεκινάμε με άδεια σύνολα FIRST για κάθε μη τερματικό σύμβολο της γραμματικής
- Για κάθε κανόνα της γραμματικής:
  - Αν το δεξιό μέρος του κανόνα ξεκινά με τερματικό a, προσθέτουμε το a στο σύνολο FIRST του μη τερματικού που βρίσκεται στο αριστερό μέρος του κανόνα
  - Αν το δεξιό μέρος του κανόνα ξεκινά με μη τερματικό A, προσθέτουμε τα σύμβολα του FIRST(A) που ξέρουμε εκείνη τη στιγμή στο σύνολο FIRST του μη τερματικού που βρίσκεται στο αριστερό μέρος του κανόνα
- Επαναλαμβάνουμε τη διαδικασία από την αρχή έως ότου να μην μπορεί να προστεθεί άλλο σύμβολο σε κάποιο σύνολο FIRST
  - Ο αλγόριθμος τερματίζει εγγυημένα: εξαρτάται από τον αριθμό των τερματικών και των κανόνων, που είναι πεπερασμένος.

Παράδειγμα

Σύνολα FIRST	Κανόνες

	`Session -> Fact Session`
	`| Question`
	`| ( Session ) Session`

	`Fact -> ! string`
	`Question -> ? string`

Παράδειγμα

Σύνολα FIRST	Κανόνες
((	Session -> Fact Session | Question | (Session) Session
!	Fact -> ! string
?	Question -> ? string

Παράδειγμα

Σύνολα FIRST	**Κανόνες**

!
?
(
	`Session -> Fact Session`
	`           | Question`
	`           | ( Session ) Session`

!	`Fact -> ! string`
?	`Question -> ? string`

Χρήση συνόλων FIRST (γραμματικές χωρίς ε)

- Επιλέγουμε την υλοποίηση ενός κανόνα A → x όταν εμφανιστεί token που ανήκει στο FIRST(x)

```
def Expr():

    if next_token=='(' or next_token=='id' or next_token=='number':
        # Expr -> Term Term_tail
        # FIRST(Term Term_tail) = FIRST(Term) = { (, id, number }
        Term()
        Term_tail()
```

- ΄Η εναλλακτικά

```
def Expr():

    if next_token in ('(','id','number'):
       # Expr -> Term Term_tail
       # FIRST(Term Term_tail) = FIRST(Term) = { (, id, number }
       Term()
       Term_tail()
```

- Προσοχή: για κάθε εναλλακτικό κανόνα του ίδιου μη τερματικού συμβόλου θα πρέπει τα σύνολα FIRST των δεξιών μερών να μην έχουν κοινά στοιχεία αλλιώς η γραμματική δεν είναι LL(1)!

Γραμματική LL(1) αριθμητικών εκφράσεων

```
Stmt_list   → Stmt Stmt_list | ε
Stmt        → id = Expr | print Expr
Expr        → Term Term_tail
Term_tail   → Addop Term Term_tail | ε
Term        → Factor Factor_tail
Factor_tail → Multop Factor Factor_tail | ε
Factor      → (Expr) | id | number
Addop       → + | -
Multop      → * | /
```

- Βρείτε τα σύνολα FIRST και υλοποιήστε τους αντίστοιχους κανόνες
  - Παραμένει η εκκρεμότητα με τις κενές παραγωγές (με ε στο δεξιό μέρος των κανόνων)