

# Μελέτη και βελτίωση της επίδοσης του συντακτικού αναλυτή Packrat

Νίκος Μαυρογεώργης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Εθνικό Μετσόβειο Πολυτεχνείο

Παρουσίαση Διπλωματικής

Ιούλιος 2020

Επιβλέπων Καθηγητής: Νίκος Παπασπύρου

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Συντακτική Ανάλυση

- Πρακτικά όλες οι γλώσσες, είτε φυσικές είτε γλώσσες μηχανής, βασίζονται στην έκφραση της πληροφορίας με γραμμικό τρόπο
- Συνήθως η αναπαράσταση γίνεται με τη μορφή μίας *συμβολοσειράς*, που είναι μια ακολουθία χαρακτήρων από ένα τυποποιημένο σύνολο
- Οποιαδήποτε εφαρμογή επεξεργασίας γλώσσας πρέπει να μετατρέψει τις συμβολοσειρές σε πιο αφηρημένες δομές όπως λέξεις, φράσεις, προτάσεις, εκφράσεις ή εντολές

# Συντακτική Ανάλυση

- Πρακτικά όλες οι γλώσσες, είτε φυσικές είτε γλώσσες μηχανής, βασίζονται στην έκφραση της πληροφορίας με γραμμικό τρόπο
- Συνήθως η αναπαράσταση γίνεται με τη μορφή μίας *συμβολοσειράς*, που είναι μια ακολουθία χαρακτήρων από ένα τυποποιημένο σύνολο
- Οποιαδήποτε εφαρμογή επεξεργασίας γλώσσας πρέπει να μετατρέψει τις συμβολοσειρές σε πιο αφηρημένες δομές όπως λέξεις, φράσεις, προτάσεις, εκφράσεις ή εντολές

## Ορισμός

*Συντακτική ανάλυση (parsing)* είναι η διαδικασία που εξάγει χρήσιμη δομημένη πληροφορία από γραμμικό κείμενο.

# Πόσο κοστίζει η συντακτική ανάλυση?

# Πόσο κοστίζει η συντακτική ανάλυση?

- Αποτελεί σημαντικό κομμάτι της εκτέλεσης προγραμμάτων, ειδικά στις διερμηνευόμενες γλώσσες όπου οι εντολές δεν μετατρέπονται σε ένα εκτελέσιμο, αλλά εκτελούνται διαρκώς εκ νέου:
  - Γλώσσες Σεναρίων: Python, Javascript
  - Γλώσσες Σήμανσης: HTML, CSS, Postscript
  - Γλώσσες ανταλλαγής δεδομένων: XML, JSON

## Πόσο κοστίζει η συντακτική ανάλυση?

- Αποτελεί σημαντικό κομμάτι της εκτέλεσης προγραμμάτων, ειδικά στις διερμηνευόμενες γλώσσες όπου οι εντολές δεν μετατρέπονται σε ένα εκτελέσιμο, αλλά εκτελούνται διαρκώς εκ νέου:
  - Γλώσσες Σεναρίων: Python, Javascript
  - Γλώσσες Σήμανσης: HTML, CSS, Postscript
  - Γλώσσες ανταλλαγής δεδομένων: XML, JSON
- Κατά το rendering ιστοσελίδων, η συντακτική ανάλυση των HTML, CSS και Javascript καταναλώνει έως και το 40% της διαδικασίας.



# Πόσο κοστίζει η συντακτική ανάλυση?

- Αποτελεί σημαντικό κομμάτι της εκτέλεσης προγραμμάτων, ειδικά στις διερμηνευόμενες γλώσσες όπου οι εντολές δεν μετατρέπονται σε ένα εκτελέσιμο, αλλά εκτελούνται διαρκώς εκ νέου:
  - Γλώσσες Σεναρίων: Python, Javascript
  - Γλώσσες Σήμανσης: HTML, CSS, Postscript
  - Γλώσσες ανταλλαγής δεδομένων: XML, JSON
- Κατά το rendering ιστοσελίδων, η συντακτική ανάλυση των HTML, CSS και Javascript καταναλώνει έως και το 40% της διαδικασίας.

## Συμπέρασμα

Θα άξιζε να μειώναμε το χρόνο εκτέλεσής της, ιδιαίτερα αν αξιοποιούσαμε και τα πολυπύρρηνα συστήματα που είναι σχεδόν πάντα διαθέσιμα.

Σε ποιες γραμματικές απευθύνεται το packrat?

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Parsing Expression Grammars - Κίνητρο

- Οι δύο πιο συνηθισμένες μέθοδοι για να περιγραφεί η σύνταξη μίας γλώσσας: οι κανονικές εκφράσεις και οι γραμματικές χωρίς συμφραζόμενα (CFGs)

# Parsing Expression Grammars - Κίνητρο

- Οι δύο πιο συνηθισμένες μέθοδοι για να περιγραφεί η σύνταξη μίας γλώσσας: οι κανονικές εκφράσεις και οι γραμματικές χωρίς συμφραζόμενα (CFGs)
- Ένα ακόμη χρήσιμο πρότυπο περιγραφής της σύνταξης είναι οι *Parsing Expression Grammars* (PEGs)
- Μοιάζουν με τις γραμματικές χωρίς συμφραζόμενα, αλλά έχουν και ορισμένες θεμελιώδεις διαφορές

# Parsing Expression Grammars - Κίνητρο

- Οι δύο πιο συνηθισμένες μέθοδοι για να περιγραφεί η σύνταξη μίας γλώσσας: οι κανονικές εκφράσεις και οι γραμματικές χωρίς συμφραζόμενα (CFGs)
- Ένα ακόμη χρήσιμο πρότυπο περιγραφής της σύνταξης είναι οι *Parsing Expression Grammars (PEGs)*
- Μοιάζουν με τις γραμματικές χωρίς συμφραζόμενα, αλλά έχουν και ορισμένες θεμελιώδεις διαφορές
- Δαισθητικά μια CFG μας περιγράφει το πώς κατασκευάζεται μία συμβολοσειρά που ανήκει σε κάποια γλώσσα, ενώ οι PEGs το πώς αναλύεται η συμβολοσειρά ώστε να προκύψει δομική πληροφορία για αυτή

# Parsing Expression Grammars - Κίνητρο

## Example

Γλώσσα από τη συνένωση ζευγών **a**

- Παραγωγικός ορισμός:  $\{s \in a^* | s = (aa)^n\}$  δηλαδή μια γλώσσα με ένα μόνο γράμμα στο λεξιλόγιό της της οποίας οι συμβολοσειρές κατασκευάζονται συνενώνοντας ζεύγη από **a**
- Αναγνωριστικός ορισμός:  $\{s \in a^* | (|s| \bmod 2 = 0)\}$  δηλαδή μία συμβολοσειρά από **a**'s γίνεται αποδεκτή μόνο αν το μήκος της είναι άρτιο

# Parsing Expression Grammars - Κίνητρο

## Example

Γλώσσα από τη συνένωση ζευγών **a**

- Παραγωγικός ορισμός:  $\{s \in a^* | s = (aa)^n\}$  δηλαδή μια γλώσσα με ένα μόνο γράμμα στο λεξιλόγιό της της οποίας οι συμβολοσειρές κατασκευάζονται συνενώνοντας ζεύγη από **a**
- Αναγνωριστικός ορισμός:  $\{s \in a^* | (|s| \bmod 2 = 0)\}$  δηλαδή μία συμβολοσειρά από **a**'s γίνεται αποδεκτή μόνο αν το μήκος της είναι άρτιο

Ο σχεδιαστής της γραμματικής είναι ευκολότερο να σκέφτεται πώς αναλύεται μία δοσμένη συμβολοσειρά στα συστατικά της, παρά πώς θα γεννηθεί (generated) η συμβολοσειρά μέσα από τους κανόνες της γραμματικής.



# Parsing Expression Grammars - Ορισμοί

- Κανόνες της μορφής  $n \leftarrow e$ , όπου  $n$  μη τερματικό και  $e$  έκφραση ("για να αναγνωρίσεις το  $n$ , αναγνώρισε πρώτα το  $e$ ")

# Parsing Expression Grammars - Ορισμοί

- Κανόνες της μορφής  $n \leftarrow e$ , όπου  $n$  μη τερματικό και  $e$  έκφραση ("για να αναγνωρίσεις το  $n$ , αναγνώρισε πρώτα το  $e$ ")
- Αριστερό βέλος αντί για δεξί: διασθητική διαφορά στην "ροή της πληροφορίας"

# Parsing Expression Grammars - Ορισμοί

- Κανόνες της μορφής  $n \leftarrow e$ , όπου  $n$  μη τερματικό και  $e$  έκφραση ("για να αναγνωρίσεις το  $n$ , αναγνώρισε πρώτα το  $e$ ")
- Αριστερό βέλος αντί για δεξί: διασθητική διαφορά στην "ροή της πληροφορίας"
- Οι κανόνες των CFGs εκφράζουν "παραγωγές" από μη τερματικά στις αντίστοιχες εκφράσεις τους ενώ των PEGs αναπαριστούν "αφαιρέσεις" από τις εκφράσεις στους αντίστοιχους κανόνες

# Parsing Expression Grammars - Εκφράσεις

Κενή συμβολοσειρά `()` : "Μην προσπαθήσεις να διαβάσεις τίποτα:  
απλά επίστρεψε επιτυχώς χωρίς να καταναλώσεις τίποτα  
από την είσοδο."

# Parsing Expression Grammars - Εκφράσεις

**Κενή συμβολοσειρά  $()$** : "Μην προσπαθήσεις να διαβάσεις τίποτα:  
απλά επέστρεψε επιτυχώς χωρίς να καταναλώσεις τίποτα  
από την είσοδο."

**Τερματικό  $\alpha$** : "Αν το επόμενο τερματικό στην είσοδο είναι  $\alpha$  τότε  
κατανάλωσε ένα τερματικό και επέστρεψε επιτυχώς.  
Αλλιώς, απέτυχε και μην καταναλώσεις τίποτα."

# Parsing Expression Grammars - Εκφράσεις

**Κενή συμβολοσειρά  $()$** : "Μην προσπαθήσεις να διαβάσεις τίποτα:  
απλά επέστρεψε επιτυχώς χωρίς να καταναλώσεις τίποτα  
από την είσοδο."

**Τερματικό  $\alpha$** : "Αν το επόμενο τερματικό στην είσοδο είναι  $\alpha$  τότε  
κατανάλωσε ένα τερματικό και επέστρεψε επιτυχώς.  
Αλλιώς, απότυχε και μην καταναλώσεις τίποτα."

**Μη Τερματικό  $A$** : "Προσπάθησε να διαβάσεις την είσοδο με βάση  
τον κανόνα που αντιστοιχεί στο  $A$  και επέστρεψε επιτυχώς  
ή απότυχε αντίστοιχα."

# Parsing Expression Grammars - Εκφράσεις

Ακολουθία  $(e_1 e_2 \dots e_n)$ : "Προσπάθησε να διαβάσεις μία συμβολοσειρά ώστε να επιτύχει η  $e_1$ . Αν η  $e_1$  επιτύχει, κάνε το ίδιο με την  $e_2$ , ξεκινώντας από το σημείο της εισόδου που δεν κατανάλωσε η  $e_1$  κ.ό.κ. Αν και οι  $n$  εκφράσεις αναγνωριστούν επίστρεψε επιτυχώς και κατανάλωσε τα αντίστοιχα κομμάτια της εισόδου. Αν οποιαδήποτε υποέκφραση αποτύχει, απότυχε χωρίς να καταναλώσεις τίποτα."

# Parsing Expression Grammars - Εκφράσεις

Διατεταγμένη Επιλογή  $(e_1/e_2/\dots/e_n)$ : "Προσπάθησε να διαβάσεις μία συμβολοσειρά ώστε να επιτύχει η  $e_1$ . Αν επιτύχει τότε η επιλογή επιστρέφει επιτυχώς καταναλώνοντας το αντίστοιχο κομμάτι της εισόδου. Αλλιώς, προσπάθησε με την  $e_2$  και την αρχική είσοδο κ.ό.κ, μέχρις ότου να επιτύχει κάποια από τις υποεκφράσεις. Αν καμία από τις  $n$  εναλλακτικές δεν πετύχουν, τότε απότυχε χωρίς να καταναλώσεις τίποτα."



# Parsing Expression Grammars - Εκφράσεις

Διατεταγμένη Επιλογή  $(e_1/e_2/\dots/e_n)$ : "Προσπάθησε να διαβάσεις μία συμβολοσειρά ώστε να επιτύχει η  $e_1$ . Αν επιτύχει τότε η επιλογή επιστρέφει επιτυχώς καταναλώνοντας το αντίστοιχο κομμάτι της εισόδου. Αλλιώς, προσπάθησε με την  $e_2$  και την αρχική είσοδο κ.ό.κ, μέχρις ότου να επιτύχει κάποια από τις υποεκφράσεις. Αν καμία από τις  $n$  εναλλακτικές δεν πετύχουν, τότε απέτυχε χωρίς να καταναλώσει τίποτα."

## Example

Έστω ο κανόνας  $Number \leftarrow Digit\ Number / Digit$ . Η σειρά έχει σημασία, διότι αν ήταν ανάποδα και θέλαμε να αναλύσουμε τον αριθμό 12, θα πηγαίναμε πρώτα στην εναλλακτική  $Digit$ , θα αναγνωρίζαμε το 1 και θα επιστρέφαμε χωρίς να πάμε στο 2.

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing**
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Ορισμοί

- Ο απλούστερος και διαισθητικά προφανής τρόπος να σχεδιάσουμε έναν συντακτικό αναλυτή είναι η από πάνω προς τα κάτω ανάλυση ή ανάλυση αναδρομικής κατάβασης.
- *Προβλέποντες (predictive) συντακτικοί αναλυτές*: επιχειρούν να προβλέψουν ποιο στοιχείο της γλώσσας ακολουθεί, βλέποντας ορισμένα από τα προπορευόμενα σύμβολα στην είσοδο.
- *Συντακτικοί αναλυτές με οπισθαναχώρηση (backtracking)*: παίρνουν αποφάσεις υποθετικά (speculatively) και δοκιμάζουν διαδοχικά διάφορες εναλλακτικές. Αν μία αποτύχει, τότε ο αναλυτής οπισθαναχωρεί στη θέση της εισόδου που ήταν προτού δοκιμάσει την εναλλακτική και μετά εξετάζει την επόμενη εναλλακτική.

# Ορισμοί

- Ο απλούστερος και διαισθητικά προφανής τρόπος να σχεδιάσουμε έναν συντακτικό αναλυτή είναι η από πάνω προς τα κάτω ανάλυση ή ανάλυση αναδρομικής κατάβασης.
- *Προβλέποντες (predictive) συντακτικοί αναλυτές*: επιχειρούν να προβλέψουν ποιο στοιχείο της γλώσσας ακολουθεί, βλέποντας ορισμένα από τα προπορευόμενα σύμβολα στην είσοδο.
- *Συντακτικοί αναλυτές με οπισθαναχώρηση (backtracking)*: παίρνουν αποφάσεις υποθετικά (speculatively) και δοκιμάζουν διαδοχικά διάφορες εναλλακτικές. Αν μία αποτύχει, τότε ο αναλυτής οπισθαναχωρεί στη θέση της εισόδου που ήταν προτού δοκιμάσει την εναλλακτική και μετά εξετάζει την επόμενη εναλλακτική.

Ποιο από τα δύο θα επιλέξουμε?

# Ορισμοί

- Ο απλούστερος και διαισθητικά προφανής τρόπος να σχεδιάσουμε έναν συντακτικό αναλυτή είναι η από πάνω προς τα κάτω ανάλυση ή ανάλυση αναδρομικής κατάβασης.
- *Προβλέποντες (predictive) συντακτικοί αναλυτές*: επιχειρούν να προβλέψουν ποιο στοιχείο της γλώσσας ακολουθεί, βλέποντας ορισμένα από τα προπορευόμενα σύμβολα στην είσοδο.
- *Συντακτικοί αναλυτές με οπισθαναχώρηση (backtracking)*: παίρνουν αποφάσεις υποθετικά (speculatively) και δοκιμάζουν διαδοχικά διάφορες εναλλακτικές. Αν μία αποτύχει, τότε ο αναλυτής οπισθαναχωρεί στη θέση της εισόδου που ήταν προτού δοκιμάσει την εναλλακτική και μετά εξετάζει την επόμενη εναλλακτική.

Ποιο από τα δύο θα επιλέξουμε? Και τα δύο!

# Θεμέλια

- Αξιοποιούμε τη λογική της αναδρομικής κατάβασης (απλότητα), αποθηκεύοντας τα ενδιάμεσα αποτελέσματα σε έναν πίνακα, ώστε να μη χρειάζεται backtracking

# Θεμέλια

- Αξιοποιούμε τη λογική της αναδρομικής κατάβασης (απλότητα), αποθηκεύοντας τα ενδιάμεσα αποτελέσματα σε έναν πίνακα, ώστε να μη χρειάζεται backtracking

## Πίνακας Packrat

Θεωρούμε έναν πίνακα όπου το κελί  $(i, j)$  περιέχει το αποτέλεσμα της συντακτικής ανάλυσης του μη τερματικού  $i$ , ξεκινώντας από τη θέση  $j$ .

# Παράδειγμα

*Additive*  $\leftarrow$  *Multitive* '+' *Additive* | *Multitive*  
*Multitive*  $\leftarrow$  *Primary* '\*' *Multitive* | *Primary*  
*Primary*  $\leftarrow$  '(' *Additive* ')' | *Decimal*  
*Decimal*  $\leftarrow$  '0' | ... | '9'

(1)



# Παράδειγμα

$$\begin{aligned}
 \textit{Additive} &\leftarrow \textit{Multitive} \text{ '+' } \textit{Additive} \mid \textit{Multitive} \\
 \textit{Multitive} &\leftarrow \textit{Primary} \text{ '*' } \textit{Multitive} \mid \textit{Primary} \\
 \textit{Primary} &\leftarrow \text{'(' } \textit{Additive} \text{'')} \mid \textit{Decimal} \\
 \textit{Decimal} &\leftarrow \text{'0'} \mid \dots \mid \text{'9'}
 \end{aligned}
 \tag{1}$$

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

*Additive*  $\leftarrow$  *Multitive* '+' *Additive* | *Multitive*  
*Multitive*  $\leftarrow$  *Primary* '\*' *Multitive* | *Primary*  
*Primary*  $\leftarrow$  '(' *Additive* ')' | *Decimal*  
*Decimal*  $\leftarrow$  '0' | ... | '9'

(1)

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

$$\begin{aligned}
 \textit{Additive} &\leftarrow \textit{Multitive} \text{ '+' } \textit{Additive} \mid \textit{Multitive} \\
 \textit{Multitive} &\leftarrow \textit{Primary} \text{ '*' } \textit{Multitive} \mid \textit{Primary} \\
 \textit{Primary} &\leftarrow \text{'(' } \textit{Additive} \text{' )' } \mid \textit{Decimal} \\
 \textit{Decimal} &\leftarrow \text{'0' } \mid \dots \mid \text{'9' }
 \end{aligned}
 \tag{1}$$

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

$$\begin{aligned}
 \textit{Additive} &\leftarrow \textit{Multitive} \text{ '+' } \textit{Additive} \mid \textit{Multitive} \\
 \textit{Multitive} &\leftarrow \textit{Primary} \text{ '*' } \textit{Multitive} \mid \textit{Primary} \\
 \textit{Primary} &\leftarrow \text{'(' } \textit{Additive} \text{'')} \mid \textit{Decimal} \\
 \textit{Decimal} &\leftarrow \text{'0'} \mid \dots \mid \text{'9'}
 \end{aligned}
 \tag{1}$$

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

$$\begin{aligned}
 \text{Additive} &\leftarrow \text{Multitive } '+' \text{ Additive} \mid \text{Multitive} \\
 \text{Multitive} &\leftarrow \text{Primary } '*' \text{ Multitive} \mid \text{Primary} \\
 \text{Primary} &\leftarrow '(' \text{ Additive } ')' \mid \text{Decimal} \\
 \text{Decimal} &\leftarrow '0' \mid \dots \mid '9'
 \end{aligned}
 \tag{1}$$

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

$Additive \leftarrow Multitive \text{ '+' } Additive \mid Multitive$   
 $Multitive \leftarrow Primary \text{ '*' } Multitive \mid Primary$   
 $Primary \leftarrow \text{'(' } Additive \text{ ')'} \mid Decimal$   
 $Decimal \leftarrow \text{'0'} \mid \dots \mid \text{'9'}$ 
(1)

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

*Additive*  $\leftarrow$  *Multitive* '+' *Additive* | *Multitive*  
*Multitive*  $\leftarrow$  *Primary* '\*' *Multitive* | *Primary*  
*Primary*  $\leftarrow$  '(' *Additive* ')' | *Decimal*  
*Decimal*  $\leftarrow$  '0' | ... | '9'

(1)

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

$$\begin{aligned}
 \text{Additive} &\leftarrow \text{Multitive } '+' \text{ Additive} \mid \text{Multitive} \\
 \text{Multitive} &\leftarrow \text{Primary } '*' \text{ Multitive} \mid \text{Primary} \\
 \text{Primary} &\leftarrow '(' \text{ Additive } ')' \mid \text{Decimal} \\
 \text{Decimal} &\leftarrow '0' \mid \dots \mid '9'
 \end{aligned}
 \tag{1}$$

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF



# Παράδειγμα

*Additive*  $\leftarrow$  *Multitive* '+' *Additive* | *Multitive*  
*Multitive*  $\leftarrow$  *Primary* '\*' *Multitive* | *Primary*  
*Primary*  $\leftarrow$  '(' *Additive* ')' | *Decimal*  
*Decimal*  $\leftarrow$  '0' | ... | '9'

(1)

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Παράδειγμα

*Additive*  $\leftarrow$  *Multitive* '+' *Additive* | *Multitive*  
*Multitive*  $\leftarrow$  *Primary* '\*' *Multitive* | *Primary*  
*Primary*  $\leftarrow$  '(' *Additive* ')' | *Decimal*  
*Decimal*  $\leftarrow$  '0' | ... | '9'

(1)

column	C1	C2	C3	C4	C5	C6	C7	C8
Additive			↑	(7,C7)	X	(4,C7)	X	X
Multitive			⋮	(3,C5)	X	(4,C7)	X	X
Primary		← ...	(?)	(3,C5)	X	(4,C7)	X	X
Decimal			X	(3,C5)	X	(4,C7)	X	X
Input String	'2'	'*'	'('	'3'	'+'	'4'	')'	EOF

# Υπομνηματισμός

- Δεν χρειάζεται να υπολογίσουμε όλα τα ενδιάμεσα κελιά
- Ξεκινάμε με αναδρομή από το αρχικό σύμβολο στην αρχή της εισόδου και αποθηκεύουμε τα ενδιάμεσα αποτελέσματα που υπολογίζουμε (memoisation)

# Υπομνηματισμός

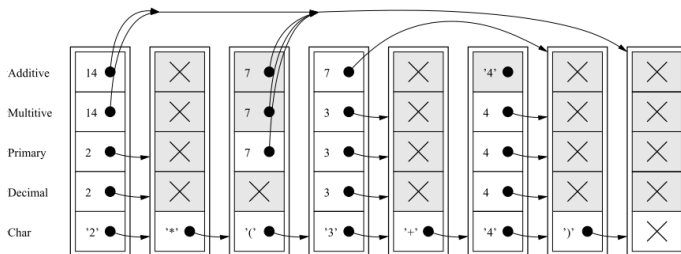
- Δεν χρειάζεται να υπολογίσουμε όλα τα ενδιάμεσα κελιά
- Ξεκινάμε με αναδρομή από το αρχικό σύμβολο στην αρχή της εισόδου και αποθηκεύουμε τα ενδιάμεσα αποτελέσματα που υπολογίζουμε (memoisation)
- Ακόμα και backtracking να γίνει κανένα κελί δεν θα χρειαστεί να ξαναυπολογιστεί

# Υπομνηματισμός

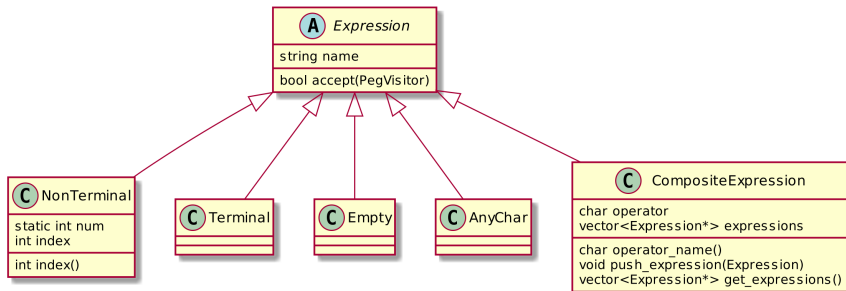
- Δεν χρειάζεται να υπολογίσουμε όλα τα ενδιάμεσα κελιά
- Ξεκινάμε με αναδρομή από το αρχικό σύμβολο στην αρχή της εισόδου και αποθηκεύουμε τα ενδιάμεσα αποτελέσματα που υπολογίζουμε (memoisation)
- Ακόμα και backtracking να γίνει κανένα κελί δεν θα χρειαστεί να ξαναυπολογιστεί → γραμμικός αλγόριθμος!

# Υπομνηματισμός

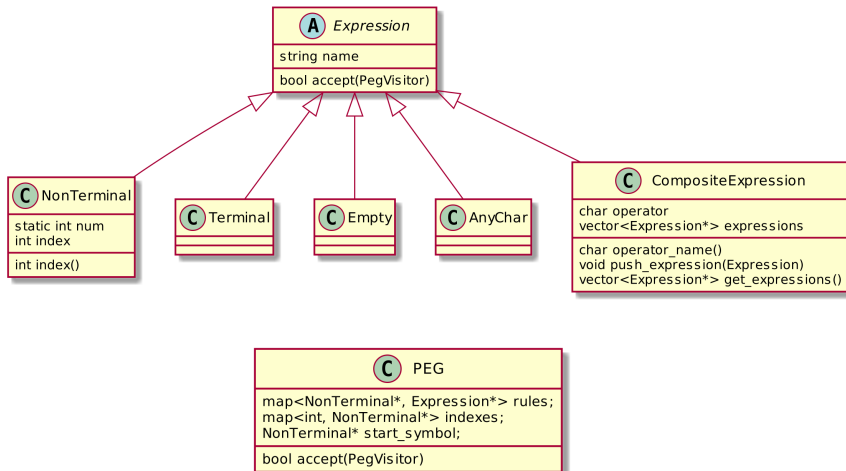
- Δεν χρειάζεται να υπολογίσουμε όλα τα ενδιάμεσα κελιά
- Ξεκινάμε με αναδρομή από το αρχικό σύμβολο στην αρχή της εισόδου και αποθηκεύουμε τα ενδιάμεσα αποτελέσματα που υπολογίζουμε (memoisation)
- Ακόμα και backtracking να γίνει κανένα κελί δεν θα χρειαστεί να ξαναυπολογιστεί → γραμμικός αλγόριθμος!



# Υλοποίηση - PEG



# Υλοποίηση - PEG





# Υλοποίηση - Packrat



Packrat

```
string in;  
int pos;  
PEG peg;  
Cell** cells;
```

```
bool visit(NonTerminal& nt);  
bool visit(Terminal& t);  
bool visit(CompositeExpression& ce);  
bool visit(Empty& e);  
bool visit(AnyChar& ac);  
bool visit(PEG& peg);
```

# Υλοποίηση - Visit Terminal

```
1 bool SerialPackrat::visit(Terminal& t)
2 {
3     int terminal_char = t.name()[0];
4     ...
5     if (pos < in.size() && terminal_char == this->cur_tok()) {
6         pos++;
7         return true;
8     }
9     return false;
10 }
```

# Υλοποίηση - Visit Non Terminal

```

1  bool SerialPackrat::visit(NonTerminal& nt)
2  {
3      int row = nt.index();
4      Cell* cur_cell = &cells[row][pos];
5      Result cur_res = cur_cell->res();
6
7      switch (cur_res) {
8
9          case Result::success:
10             {
11                 pos = cur_cell->pos();
12                 return true;
13             }
14          case Result::fail:
15             {
16                 return false;
17             }

```

```

18  case Result::unknown:
19      {
20          Expression* e = peg.get_expr(&nt);
21          auto res = e->accept(*this);
22
23          if (res) {
24              cur_cell->set_res(Result::success);
25              cur_cell->set_pos(pos); // pos has changed
26              return true;
27          } else {
28              cur_cell->set_res(Result::fail);
29              return false;
30          }
31      }
32  }
33  return false;
34  }

```

# Υλοποίηση - Visit Composite Expression

```
1  bool SerialPackrat::visit(CompositeExpression& ce)
2  {
3      char op = ce.op_name();
4      std::vector<Expression*> exprs = ce.expr_list();
5      int orig_pos = pos;
6
7      switch (op) {
8
9          case '\b': // sequence
10             {
11                 for (auto expr : exprs)
12                     if (!expr->accept(*this)) {
13                         pos = orig_pos;
14                         return false;
15                     }
16                 return true;
17             }
18             case '||': // ordered choice
19             {
20                 for (auto expr : exprs) {
21                     pos = orig_pos;
22                     if (expr->accept(*this))
23                         return true;
24                 }
25                 pos = orig_pos;
26                 return false;
27             }
28             ...
29         }
30 }
```

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Κίνητρο

- Ο αλγόριθμος packrat καταναλώνει χώρο  $O(NT * n)$ ,  $NT$  ο αριθμός των μη τερματικών,  $n$  το μήκος της εισόδου

# Κίνητρο

- Ο αλγόριθμος packrat καταναλώνει χώρο  $O(NT * n)$ ,  $NT$  ο αριθμός των μη τερματικών,  $n$  το μήκος της εισόδου
- Δεν ενδείκνυται για μεγάλα αρχεία εισόδου, καθώς ο πίνακας ενδιάμεσων αποτελεσμάτων μπορεί να γίνει απαγορευτικά μεγάλος

# Κίνητρο

- Ο αλγόριθμος packrat καταναλώνει χώρο  $O(NT * n)$ ,  $NT$  ο αριθμός των μη τερματικών,  $n$  το μήκος της εισόδου
- Δεν ενδείκνυται για μεγάλα αρχεία εισόδου, καθώς ο πίνακας ενδιάμεσων αποτελεσμάτων μπορεί να γίνει απαγορευτικά μεγάλος
- Χρειάζεται να τα κρατάμε όλα τα ενδιάμεσα αποτελέσματα?



# Κίνητρο

- Ο αλγόριθμος packrat καταναλώνει χώρο  $O(NT * n)$ ,  $NT$  ο αριθμός των μη τερματικών,  $n$  το μήκος της εισόδου
- Δεν ενδείκνυται για μεγάλα αρχεία εισόδου, καθώς ο πίνακας ενδιάμεσων αποτελεσμάτων μπορεί να γίνει απαγορευτικά μεγάλος
- Χρειάζεται να τα κρατάμε όλα τα ενδιάμεσα αποτελέσματα?
- Θα μπορούσαμε να μειώσουμε τις στήλες του πίνακα?

# Κίνητρο

- Ο αλγόριθμος packrat καταναλώνει χώρο  $O(NT * n)$ ,  $NT$  ο αριθμός των μη τερματικών,  $n$  το μήκος της εισόδου
- Δεν ενδείκνυται για μεγάλα αρχεία εισόδου, καθώς ο πίνακας ενδιάμεσων αποτελεσμάτων μπορεί να γίνει απαγορευτικά μεγάλος
- Χρειάζεται να τα κρατάμε όλα τα ενδιάμεσα αποτελέσματα?
- Θα μπορούσαμε να μειώσουμε τις στήλες του πίνακα?
- Τις γραμμές?

## Μείωση των στηλών - Κυλιόμενο Παράθυρο

- Το πόσες στήλες πραγματικά χρειαζόμαστε εξαρτάται από το μήκος της μέγιστης οπισθαναχώρησης (έστω  $W$ )
- Αν το γνωρίζαμε από πριν θα μπορούσαμε να έχουμε ένα παράθυρο που να "κυλάει" προς τα δεξιά ως προς την είσοδο

## Μείωση των στηλών - Κυλιόμενο Παράθυρο

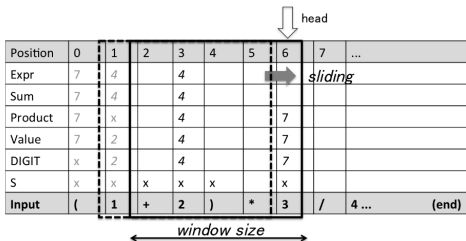
- Το πόσες στήλες πραγματικά χρειαζόμαστε εξαρτάται από το μήκος της μέγιστης οπισθαναχώρησης (έστω  $w$ )
- Αν το γνωρίζαμε από πριν θα μπορούσαμε να έχουμε ένα παράθυρο που να "κυλάει" προς τα δεξιά ως προς την είσοδο
- $w = 1$ : αλγόριθμος με backtracking,  $w = n$ : αλγόριθμος packrat

# Μείωση των στηλών - Κυλιόμενο Παράθυρο

- Το πόσες στήλες πραγματικά χρειαζόμαστε εξαρτάται από το μήκος της μέγιστης οπισθαναχώρησης (έστω  $w$ )
- Αν το γνωρίζαμε από πριν θα μπορούσαμε να έχουμε ένα παράθυρο που να "κυλάει" προς τα δεξιά ως προς την είσοδο
- $w = 1$ : αλγόριθμος με backtracking,  $w = n$ : αλγόριθμος packrat
- Στην πράξη δεν ξέρουμε πόσο είναι, οπότε πρέπει να πειραματιστούμε

# Μείωση των στηλών - Κυλιόμενο Παράθυρο

- Το πόσες στήλες πραγματικά χρειαζόμαστε εξαρτάται από το μήκος της μέγιστης οπισθαναχώρησης (έστω  $w$ )
- Αν το γνωρίζαμε από πριν θα μπορούσαμε να έχουμε ένα παράθυρο που να "κυλάει" προς τα δεξιά ως προς την είσοδο
- $w = 1$ : αλγόριθμος με backtracking,  $w = n$ : αλγόριθμος packrat
- Στην πράξη δεν ξέρουμε πόσο είναι, οπότε πρέπει να πειραματιστούμε



Position	0	1	2	3	4	5	6	7	...
Expr	7	4		4			→	sliding	
Sum	7	4		4					
Product	7	x		4			7		
Value	7	2		4			7		
DIGIT	x	2		4			7		
S	x	x	x	x	x		x		
Input	(	1	+	2	)	*	3	/	4 ... (end)

window size

# Μείωση των γραμμών - Απενεργοποίηση μη τερματικών

- Το πόσες γραμμές πραγματικά χρειαζόμαστε εξαρτάται από το πόσο αξιοποιούνται τα αποτελέσματα των μη τερματικών
- Θέτουμε ένα κατώφλι απενεργοποίησης *limit*
- Αν μετά τις πρώτες *limit* προσπελάσεις κελιών που αντιστοιχούν σε ένα μη τερματικό δεν αξιοποιήσουμε ούτε ένα αποτέλεσμα, σταματάμε να κρατάμε το μη τερματικό στον πίνακα

---

```
1 if (!Int_activated[row]) {  
2     Expression* e = peg.get_expr(&nt);  
3     return e->accept(*this);  
4 }
```

---

## Συνδυασμός ιδεών: Elastic Packrat

- Χρησιμοποιούμε έναν μονοδιάστατο πίνακα  $W * N$ , όπου  $W$  είναι το πλάτος του παραθύρου και  $N$  ο αριθμός των ενεργών μη τερματικών.
- Ακολουθώς, χρησιμοποιούμε έναν δείκτη κατακερματισμού (hashing-based index) για να εντοπίσουμε σε ποιο σημείο του μονοδιάστατου πίνακα θα αποθηκευτεί το ενδιαμέσο αποτέλεσμα

```
1  long int key = (pos << shift) | row;
2  unsigned int index = hash(key) % (w * n);
3  ...
4  ElasticCell* cur_cell = &elastic_cells[index];    // πάρε το αντίστοιχο κελί
5  cur_cell->set_key(key);                          // θέσε το κλειδί στο αντίστοιχο κελί
```



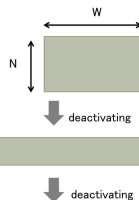
# Συνδυασμός ιδεών: Elastic Packrat

- Χρησιμοποιούμε έναν μονοδιάστατο πίνακα  $W * N$ , όπου  $W$  είναι το πλάτος του παραθύρου και  $N$  ο αριθμός των ενεργών μη τερματικών.
- Ακολουθώς, χρησιμοποιούμε έναν δείκτη κατακερματισμού (hashing-based index) για να εντοπίσουμε σε ποιο σημείο του μονοδιάστατου πίνακα θα αποθηκευτεί το ενδιαμέσο αποτέλεσμα
- Σταθερή μνήμη:  $O(NT * W)$ !

```
1  long int key = (pos << shift) | row;
2  unsigned int index = hash(key) % (w * n);
3  ...
4  ElasticCell* cur_cell = &elastic_cells[index];    // πάρε το αντίστοιχο κελί
5  cur_cell->set_key(key);                          // θέσε το κλειδί στο αντίστοιχο κελί
```

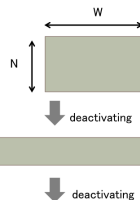
# Elastic Packrat

Όταν απενεργοποιούμε ένα μη  
τερματικό ο χώρος του στον πίνακα  
μπορεί να αξιοποιηθεί από άλλα  
διευρύνοντας ουσιαστικά το παράθυρο



# Elastic Packrat

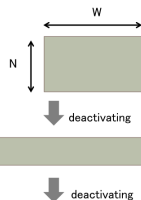
Όταν απενεργοποιούμε ένα μη  
τερματικό ο χώρος του στον πίνακα  
μπορεί να αξιοποιηθεί από άλλα  
διευρύνοντας ουσιαστικά το παράθυρο



- Μειονέκτημα του hashing-based index: πιθανές συγκρούσεις (collisions) μεταξύ διαφορετικών κλειδιών, που όμως αντιστοιχίζονται στο ίδιο index

# Elastic Packrat

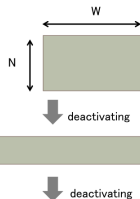
Όταν απενεργοποιούμε ένα μη  
τερματικό ο χώρος του στον πίνακα  
μπορεί να αξιοποιηθεί από άλλα  
διευρύνοντας ουσιαστικά το παράθυρο



- Μειονέκτημα του hashing-based index: πιθανές συγκρούσεις (collisions) μεταξύ διαφορετικών κλειδιών, που όμως αντιστοιχίζονται στο ίδιο index
- Χάνεται η εγγύηση γραμμικού χρόνου

# Elastic Packrat

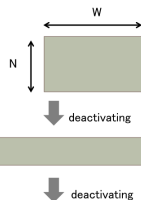
Όταν απενεργοποιούμε ένα μη  
τερματικό ο χώρος του στον πίνακα  
μπορεί να αξιοποιηθεί από άλλα  
διευρύνοντας ουσιαστικά το παράθυρο



- Μειονέκτημα του hashing-based index: πιθανές συγκρούσεις (collisions) μεταξύ διαφορετικών κλειδιών, που όμως αντιστοιχίζονται στο ίδιο index
- Χάνεται η εγγύηση γραμμικού χρόνου
- Στην πράξη δουλεύει καλά και οδηγεί σε απλή υλοποίηση

# Elastic Packrat

Όταν απενεργοποιούμε ένα μη  
τερματικό ο χώρος του στον πίνακα  
μπορεί να αξιοποιηθεί από άλλα  
διευρύνοντας ουσιαστικά το παράθυρο



- Μειονέκτημα του hashing-based index: πιθανές συγκρούσεις (collisions) μεταξύ διαφορετικών κλειδιών, που όμως αντιστοιχίζονται στο ίδιο index
- Χάνεται η εγγύηση γραμμικού χρόνου
- Στην πράξη δουλεύει καλά και οδηγεί σε απλή υλοποίηση
- Όσο προχωράμε στην είσοδο, τα νέα κελιά "εκτοπίζουν" τα παλιά που προσομοιώνοντας απλά την "κύλιση" του παραθύρου

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing**
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα

# Ιδέα

- Θέλουμε να βρούμε κάποιο κομμάτι του αλγορίθμου που μπορεί να εκτελεστεί από πολλά νήματα ταυτόχρονα, ώστε να μοιραστεί ο φόρτος εργασίας



# Ιδέα

- Θέλουμε να βρούμε κάποιο κομμάτι του αλγορίθμου που μπορεί να εκτελεστεί από πολλά νήματα ταυτόχρονα, ώστε να μοιραστεί ο φόρτος εργασίας
- Στη διατεταγμένη επιλογή  $E \leftarrow E_1/E_2/E_3$  οι υποεκφράσεις αναλύονται ξεκινώντας από το ίδιο σημείο της εισόδου

# Ιδέα

- Θέλουμε να βρούμε κάποιο κομμάτι του αλγορίθμου που μπορεί να εκτελεστεί από πολλά νήματα ταυτόχρονα, ώστε να μοιραστεί ο φόρτος εργασίας
- Στη διατεταγμένη επιλογή  $E \leftarrow E_1/E_2/E_3$  οι υποεκφράσεις αναλύονται ξεκινώντας από το ίδιο σημείο της εισόδου
- Αν αποτύχει η  $E_1$ , τότε ξαναπροσπαθεί από το ίδιο σημείο η  $E_2$  κλπ

# Ιδέα

- Θέλουμε να βρούμε κάποιο κομμάτι του αλγορίθμου που μπορεί να εκτελεστεί από πολλά νήματα ταυτόχρονα, ώστε να μοιραστεί ο φόρτος εργασίας
- Στη διατεταγμένη επιλογή  $E \leftarrow E_1/E_2/E_3$  οι υποεκφράσεις αναλύονται ξεκινώντας από το ίδιο σημείο της εισόδου
- Αν αποτύχει η  $E_1$ , τότε ξαναπροσπαθεί από το ίδιο σημείο η  $E_2$  κλπ
- Θεωρητικά θα μπορούσαμε να τις αναθέσουμε σε ξεχωριστά νήματα

# Ιδέα

- Θέλουμε να βρούμε κάποιο κομμάτι του αλγορίθμου που μπορεί να εκτελεστεί από πολλά νήματα ταυτόχρονα, ώστε να μοιραστεί ο φόρτος εργασίας
- Στη διατεταγμένη επιλογή  $E \leftarrow E_1/E_2/E_3$  οι υποεκφράσεις αναλύονται ξεκινώντας από το ίδιο σημείο της εισόδου
- Αν αποτύχει η  $E_1$ , τότε ξαναπροσπαθεί από το ίδιο σημείο η  $E_2$  κλπ
- Θεωρητικά θα μπορούσαμε να τις αναθέσουμε σε ξεχωριστά νήματα
- Αναδρομική κλήση νημάτων: Αν κάποιο νήμα που κληθεί κατά την διατεταγμένη επιλογή πάει να αναλύσει με τη σειρά του διατεταγμένη επιλογή, καλεί και δικά του νήματα

# Λεπτά σημεία

- Τα κελιά με τα ενδιάμεσα αποτελέσματα πρέπει να προστατεύονται από ταυτόχρονη χρήση

# Λεπτά σημεία

- Τα κελιά με τα ενδιάμεσα αποτελέσματα πρέπει να προστατεύονται από ταυτόχρονη χρήση
- Αν το νήμα που ανέλυσε την  $E_2$ , επιστρέψει επιτυχώς, πρέπει πάλι να περιμένουμε το νήμα που αναλύει την  $E_1$ , καθώς η υποέκφραση αυτή έχει προτεραιότητα

# Λεπτά σημεία

- Τα κελιά με τα ενδιάμεσα αποτελέσματα πρέπει να προστατεύονται από ταυτόχρονη χρήση
- Αν το νήμα που ανέλυσε την  $E_2$ , επιστρέψει επιτυχώς, πρέπει πάλι να περιμένουμε το νήμα που αναλύει την  $E_1$ , καθώς η υποέκφραση αυτή έχει προτεραιότητα
- Αν το νήμα που ανέλυσε την  $E_i$ , επιστρέψει επιτυχώς, πρέπει τερματίσουμε τα νήματα που ανέλαβαν τις  $E_{i+1}, \dots, E_n$

# Λεπτά σημεία

- Τα κελιά με τα ενδιάμεσα αποτελέσματα πρέπει να προστατεύονται από ταυτόχρονη χρήση
- Αν το νήμα που ανέλυσε την  $E_2$ , επιστρέψει επιτυχώς, πρέπει πάλι να περιμένουμε το νήμα που αναλύει την  $E_1$ , καθώς η υποέκφραση αυτή έχει προτεραιότητα
- Αν το νήμα που ανέλυσε την  $E_i$ , επιστρέψει επιτυχώς, πρέπει τερματίσουμε τα νήματα που ανέλαβαν τις  $E_{i+1}, \dots, E_n$
- Αν υπάρχουν πολλές εναλλακτικές στην επιλογή, τότε ίσως να μην αξίζει να καλέσουμε πολλά νήματα, καθώς το overhead του fork-join θα είναι μεγάλο



# Λεπτά σημεία

- Τα κελιά με τα ενδιάμεσα αποτελέσματα πρέπει να προστατεύονται από ταυτόχρονη χρήση
- Αν το νήμα που ανέλυσε την  $E_2$ , επιστρέψει επιτυχώς, πρέπει πάλι να περιμένουμε το νήμα που αναλύει την  $E_1$ , καθώς η υποέκφραση αυτή έχει προτεραιότητα
- Αν το νήμα που ανέλυσε την  $E_i$ , επιστρέψει επιτυχώς, πρέπει τερματίσουμε τα νήματα που ανέλαβαν τις  $E_{i+1}, \dots, E_n$
- Αν υπάρχουν πολλές εναλλακτικές στην επιλογή, τότε ίσως να μην αξίζει να καλέσουμε πολλά νήματα, καθώς το overhead του fork-join θα είναι μεγάλο
- Αν το βάθος του fork-join tree είναι ήδη μεγάλο, τότε ίσως πάλι να μην αξίζει να καλέσουμε νέα νήματα

## Λεπτά σημεία - Κλειδώματα

- Για να αποφευχθούν συνθήκες ανταγωνισμού για τα κελιά πρέπει να προστατεύονται με κλειδώματα
- Ένας απλός και αποτελεσματικός τρόπος είναι κάθε κελί να έχει το κλειδί του οπότε για να το υπολογίσει κάποιος θα πρέπει να το κλειδώσει

## Λεπτά σημεία - Κλειδώματα

- Για να αποφευχθούν συνθήκες ανταγωνισμού για τα κελιά πρέπει να προστατεύονται με κλειδώματα
- Ένας απλός και αποτελεσματικός τρόπος είναι κάθε κελί να έχει το κλειδί του οπότε για να το υπολογίσει κάποιος θα πρέπει να το κλειδώσει

---

```
1 cur_cell->lock();  
2 cur_cell->set_res(Result::pending);  
3 cur_cell->unlock();
```

---

## Λεπτά σημεία - Κλειδώματα

- Για να αποφευχθούν συνθήκες ανταγωνισμού για τα κελιά πρέπει να προστατεύονται με κλειδώματα
- Ένας απλός και αποτελεσματικός τρόπος είναι κάθε κελί να έχει το κλειδί του οπότε για να το υπολογίσει κάποιος θα πρέπει να το κλειδώσει

---

```
1 cur_cell->lock();  
2 cur_cell->set_res(Result::pending);  
3 cur_cell->unlock();
```

---

- Όσο το κελί είναι pending (δηλαδή το υπολογίζει κάποιος) οι υπόλοιποι κάνουν busy-wait

# Λεπτά σημεία - Ιεραρχία

- Κάθε νήμα που καλείται λαμβάνει έναν αριθμό *rank* από το γονέα
- Υψηλότερο *rank* είναι το 0

## Λεπτά σημεία - Ιεραρχία

- Κάθε νήμα που καλείται λαμβάνει έναν αριθμό *rank* από το γονέα
- Υψηλότερο *rank* είναι το 0

```
1  for (auto j = 0; j < max_rank; ++j) {  
2      threads[j].join();  
3      if (results[j]) {  
4          finished_rank.store(j);  
5          pos = positions[j];  
6          for (auto k = j + 1; k < max_rank; ++k) {  
7              threads[k].join();  
8          }  
9          return true;  
10     }  
11 }  
12 pos = orig_pos;  
13 return false;
```

# Λεπτά σημεία - Τερματισμός

- Περιοδικός έλεγχος για πρόωρο τερματισμό από τον γονέα

# Λεπτά σημεία - Τερματισμός

- Περιοδικός έλεγχος για πρόωρο τερματισμό από τον γονέα

```
1  auto fr = parent_finished_rank->load();  
2  if (fr >= 0 && fr < rank) {  
3      return false;  
4  }
```



# Όλα μαζί

```
1 finished_rank.store(-1);
2
3 int results[exprs.size()];
4 int positions[exprs.size()];
5 std::vector<std::thread> threads;
6
7 auto i = 0;
8 for (auto& expr : exprs) {
9     threads.emplace_back([&, expr, i]()
10     {
11         SimpleWorker sw{in, peg, cells, pos, expr_limit,
12                         cur_tree_depth + 1, max_tree_depth, i, &finished_rank};
13         results[i] = expr->accept(sw);
14         positions[i] = sw.cur_pos();
15     });
16     i++;
17 }
```

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα**
- 7 Συμπεράσματα

# Δοκιμαστικά προγράμματα

- 3 αρχεία από τον πηγαίο κώδικα της Java
  - *Arrays.java* - 116K
  - *BigDecimal.java* - 140K
  - *Throwable.java* - 28K
- Μέτριο προς μεγάλο μέγεθος ώστε να φανούν οι διαφορές στους χρόνους εκτέλεσης των αλγορίθμων
- 12 νήματα διαθέσιμα
- *std::chrono::high\_resolution\_clock* της STL

# Ρύθμιση των παραμέτρων

- Τόσο για το elastic όσο και για το παράλληλο packrat εκτελούμε πειράματα για να βελτιώσουμε τους συνδυασμούς των παραμέτρων

# Ρύθμιση των παραμέτρων

- Τόσο για το elastic όσο και για το παράλληλο packrat εκτελούμε πειράματα για να βελτιώσουμε τους συνδυασμούς των παραμέτρων
- Για το elastic δοκιμάζουμε παράθυρα μήκους 256, 512, 1024 και κατώφλια απενεργοποίησης 0, 16, 32, 48

# Ρύθμιση των παραμέτρων

- Τόσο για το elastic όσο και για το παράλληλο packrat εκτελούμε πειράματα για να βελτιώσουμε τους συνδυασμούς των παραμέτρων
- Για το elastic δοκιμάζουμε παράθυρα μήκους 256, 512, 1024 και κατώφλια απενεργοποίησης 0, 16, 32, 48

## Καλύτερος Συνδυασμός

$w = 256$ , ***thres*** = 32

## Ρύθμιση των παραμέτρων

- Τόσο για το elastic όσο και για το παράλληλο packrat εκτελούμε πειράματα για να βελτιώσουμε τους συνδυασμούς των παραμέτρων
- Για το elastic δοκιμάζουμε παράθυρα μήκους 256, 512, 1024 και κατώφλια απενεργοποίησης 0, 16, 32, 48

### Καλύτερος Συνδυασμός

$w = 256$ , ***thres*** = 32

- Για τον παράλληλο δοκιμάζουμε όρια μήκους διατεταγμένης επιλογής 2, 4, 6, 8 και και μέγιστο βάθος δέντρου 1, 2

## Ρύθμιση των παραμέτρων

- Τόσο για το elastic όσο και για το παράλληλο packrat εκτελούμε πειράματα για να βελτιώσουμε τους συνδυασμούς των παραμέτρων
- Για το elastic δοκιμάζουμε παράθυρα μήκους 256, 512, 1024 και κατώφλια απενεργοποίησης 0, 16, 32, 48

### Καλύτερος Συνδυασμός

$w = 256$ , ***thres*** = 32

- Για τον παράλληλο δοκιμάζουμε όρια μήκους διατεταγμένης επιλογής 2, 4, 6, 8 και και μέγιστο βάθος δέντρου 1, 2

### Καλύτερος Συνδυασμός

***expr\_limit*** = 4, ***max\_depth*** = 1. Δηλαδή 5 ενεργά νήματα κατά το μέγιστο.



# Τελική Σύγκριση

Χρόνοι εκτέλεσης (ms)			
Αλγόριθμος Packrat	Arrays 116K	BigDecimal 140K	Throwable 28K
Κλασικός	404	350	46
Elastic (256, 32)	380	329	51
Παράλληλος (1, 4)	432	369	62

**Πίνακας:** Τελικά αποτελέσματα για τους τρεις αλγορίθμους

# Πίνακας Περιεχομένων

- 1 Εισαγωγή
- 2 Parsing Expression Grammars
- 3 Packrat Parsing
- 4 Packrat Parsing με ελαστικό κυλιόμενο παράθυρο
- 5 Παράλληλο Packrat Parsing
- 6 Πειραματικά Αποτελέσματα
- 7 Συμπεράσματα**

# Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat

# Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη

# Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη
- Στον παράλληλο δεν παρατηρείται επιτάχυνση (speedup) σε σχέση με τη σειριακή περίπτωση

## Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη
- Στον παράλληλο δεν παρατηρείται επιτάχυνση (speedup) σε σχέση με τη σειριακή περίπτωση
- Η πράξη της διατεταγμένης επιλογής, παρόλο που θεωρητικά αφήνει χώρο για παράλληλη εκτέλεση, δεν συμπεριφέρεται τόσο καλά στην πράξη

## Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη
- Στον παράλληλο δεν παρατηρείται επιτάχυνση (speedup) σε σχέση με τη σειριακή περίπτωση
- Η πράξη της διατεταγμένης επιλογής, παρόλο που θεωρητικά αφήνει χώρο για παράλληλη εκτέλεση, δεν συμπεριφέρεται τόσο καλά στην πράξη
- Overhead από fork-join

# Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη
- Στον παράλληλο δεν παρατηρείται επιτάχυνση (speedup) σε σχέση με τη σειριακή περίπτωση
- Η πράξη της διατεταγμένης επιλογής, παρόλο που θεωρητικά αφήνει χώρο για παράλληλη εκτέλεση, δεν συμπεριφέρεται τόσο καλά στην πράξη
- Overhead από fork-join
- Ίσως πετυχαίνουν συχνά στη γραμματική μας οι πρώτες (αν όχι η πρώτη) επιλογή, οπότε μετά υπάρχει και ο επιπλέον χρόνος αναμονής νημάτων



# Συμπεράσματα

- Αδιαφιλονίκητος νικητής: elastic packrat
- Καλύτερος χρόνος στα μεγάλα προγράμματα + σταθερή μνήμη
- Στον παράλληλο δεν παρατηρείται επιτάχυνση (speedup) σε σχέση με τη σειριακή περίπτωση
- Η πράξη της διατεταγμένης επιλογής, παρόλο που θεωρητικά αφήνει χώρο για παράλληλη εκτέλεση, δεν συμπεριφέρεται τόσο καλά στην πράξη
- Overhead από fork-join
- Ίσως πετυχαίνουν συχνά στη γραμματική μας οι πρώτες (αν όχι η πρώτη) επιλογή, οπότε μετά υπάρχει και ο επιπλέον χρόνος αναμονής νημάτων
- Ο περιοδικός έλεγχος για τερματισμό αυξάνει το χρόνο εκτέλεσης

# Απορίες

# Απορίες

Ευχαριστώ για την προσοχή σας!<sup>1</sup>

---

<sup>1</sup>Hope you slept well!