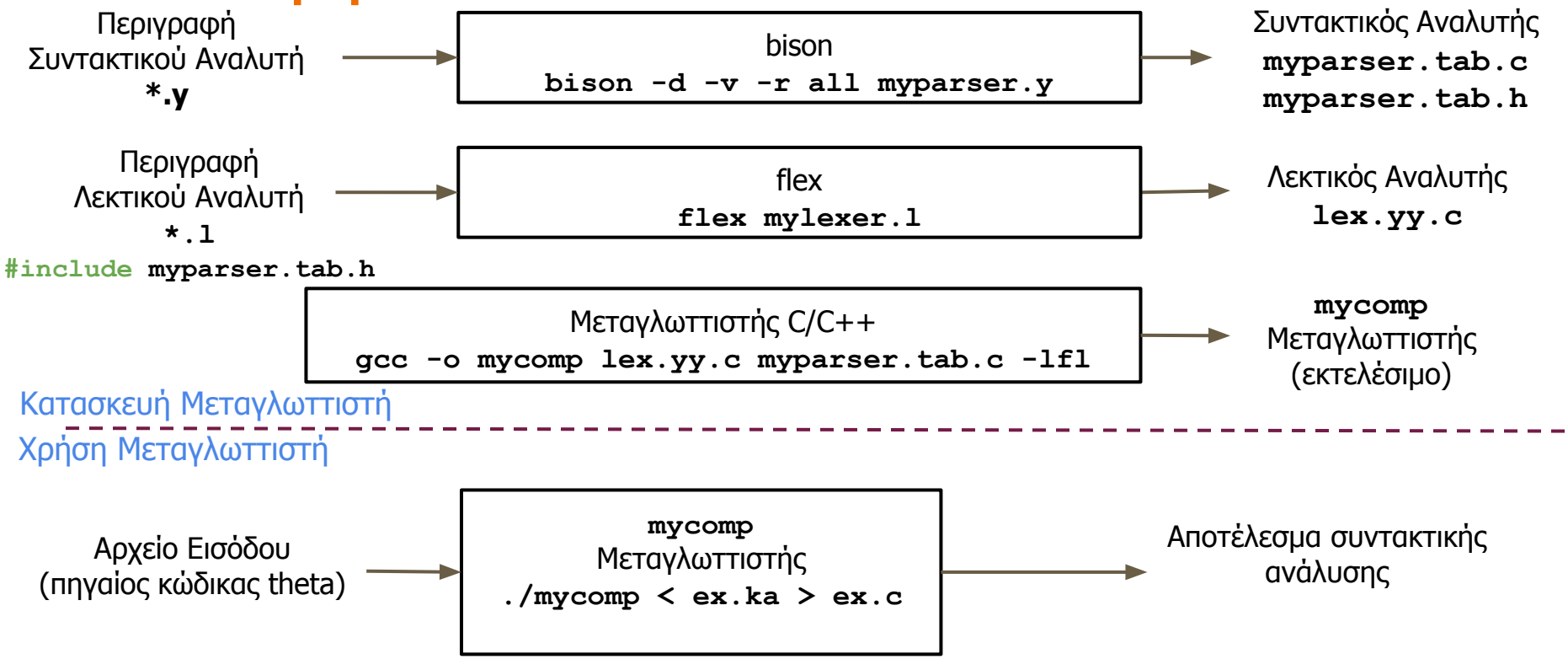
Θεωρία Υπολογισμού

Το Εργαλείο BISON

Το Εργαλείο BISON

- Εργαλείο για την ανάλυση (parsing) γλωσσών περιγεγραμμένων από γραμματικές χωρίς συμφραζόμενα (context-free grammars)
- Αυτοματοποιημένη ***παραγωγή*** Συντακτικών Αναλυτών (parser generator)
  - Βελτιστοποιημένος για LR(1) γραμματικές
  - Υποστηρίζει και την ανάλυση πιο γενικών γραμματικών μέσω GLR (Generalized LR)
- Απόγονος του Unix εργαλείου yacc (yet another compiler compiler)
- Συμβατό με το εργαλείο flex για Λεκτική Ανάλυση
- Ευκολία χρήσης, υψηλή απόδοση, ευελιξία και εκφραστικότητα
- Τρέχουσα έκδοση 3.0.4 (https://www.gnu.org/software/bison)

Λειτουργία BISON



Μορφή Αρχείου Εισόδου bison

Κάθε αρχείο bison αποτελείται από τέσσερις ενότητες χωρισμένες μεταξύ τους από κατάλληλους διαχωριστές:

```
%{
Prologue
%}
Bison declarations
%%
Grammar rules
%%
Epilogue
```

Σε οποιαδήποτε ενότητα μπορεί να εμφανίζονται σχόλια ανάμεσα σε `/*` και `*/` ή σχόλια γραμμής `//`

Ενότητα Προλόγου (Prologue)

- Περικλείεται σε `%{` και `%}`
- Περιέχει:
  - Ορισμούς μακροεντολών (macro definitions)
  - Δηλώσεις (declarations) συναρτήσεων και μεταβλητών οι οποίες θα χρησιμοποιηθούν στις υπόλοιπες ενότητες
  - Οδηγίες `#include`
- Τα περιεχόμενα μεταφέρονται αυτολεξεί στην αρχή του παραγόμενου αρχείου κώδικα

Ενότητα Προλόγου - Παράδειγμα

```
%{
   #include <stdio.h>
   #include <math.h>
   int line_num = 1;
%}
```

Ενότητα Δηλώσεων (declarations)

- Περιέχει τις δηλώσεις των **συμβόλων** που θα χρησιμοποιηθούν για το σχηματισμό των γραμματικών κανόνων και τους τύπους των σημασιολογικών τους τιμών
  - Πρέπει να δηλωθούν όλα τα ***τερματικά σύμβολα (token type names)***

```
%token KW_INTEGER
%token KW_REAL
%token KW_MOD
```

  - Δεν χρειάζεται να δηλωθούν τα τερματικά σύμβολα ενός χαρακτήρα (single-character literal tokens) όπως `'+'` και `'*'`

Ενότητα Δηλώσεων (2)

- Τα ***μη-τερματικά σύμβολα*** πρέπει να δηλωθούν μόνο εάν θέλουμε να δηλώσουμε τον τύπο της σημασιολογικής τους τιμής

- Ο πρώτος γραμματικός κανόνας θεωρείται ότι ορίζει το αρχικό σύμβολο
  - Μπορεί να αλλάξει με τη ρητή δήλωση του ***αρχικού συμβόλου***

```
%start program
```

- Δηλώσεις προσεταιριστικότητας και προτεραιότητας

```
%left '-' '+'
%left '*' '/' TK_MOD
```

Ενότητα Γραμματικών Κανόνων

- Περιέχει έναν ή περισσότερους ***γραμματικούς κανόνες (grammar rules)***
- Πρέπει να υπάρχει ένας τουλάχιστον γραμματικός κανόνας
- Το πρώτο %% δεν μπορεί ποτέ να παραλειφθεί ακόμα και εάν είναι το πρώτο στοιχείο του αρχείου bison

Ενότητα Επιλόγου (Epilogue)

- Προαιρετική ενότητα
- Περιέχει βοηθητικό κώδικα C/C++
- Μεταφέρεται αυτολεξεί στο παραγόμενο αρχείο και ενσωματώνεται ως έχει
- Ορισμός (definition) βοηθητικών συναρτήσεων οι οποίες καλούνται από τις ενέργειες των κανόνων
- Περιέχει τη **`main`** για αυτόνομο Συντακτικό Αναλυτή

Ενότητα Επιλόγου (2)

- Παράδειγμα

```
%%
int main() {
    if ( yyparse() == 0 )
        printf("Accepted!\n");
    else
        printf("Rejected!\n");
}
```

Σύμβολα, Τερματικά και Μη-Τερματικά

- Τα **σύμβολα** (symbols) διαχωρίζονται σε:
  - ***τερματικά σύμβολα*** (terminal symbols also known as *token types*)
  - ***μη-τερματικά*** (nonterminals or groupings)
- Τα ονόματα των συμβόλων μπορεί να περιέχουν γράμματα, χαρακτήρες υπογράμμισης, τελείες, αριθμούς και παύλες

Τερματικά Σύμβολα (1/5)

- Ένα ***τερματικό σύμβολο*** αναπαριστά μια κλάση/κατηγορία συντακτικά ισοδύναμων λεκτικών μονάδων (tokens)
- Χρησιμοποιείται για την περιγραφή των γραμματικών κανόνων
- Δεν μπορεί να αναλυθεί περισσότερο σε άλλα σύμβολα
- Ένα σύμβολο αναπαρίσταται εσωτερικά στον bison με έναν αριθμητικό κωδικό
- Η συνάρτηση **`yylex()`** επιστρέφει τον κωδικό ως ένδειξη της λεκτικής μονάδας που διάβασε στην είσοδο

Τερματικά Σύμβολα (2/5)

- Τρόποι γραφής των τερματικών συμβόλων:
  - *Ονοματισμένου τύπου (named token type)*: Δηλώνεται ως:
    - `%token name`
    - Παραδείγματα (από σύμβαση χρησιμοποιούμε κεφαλαία γράμματα):
      - `%token KW_INTEGER`
      - `%token KW_REAL`
  - Ο bison θα μετατρέψει αυτή τη δήλωση εσωτερικά σε `#define`, έτσι ώστε η `yylex()` να μπορεί να χρησιμοποιήσει το δηλωθέν όνομα `name` κατά τη Λεκτική Ανάλυση
  - `%left`, `%right`, `%nonassoc`, `%precedence`, αντί για `%token` για τον καθορισμό προσεταιριστικότητας (associativity) και προτεραιότητας (precedence)

Τερματικά Σύμβολα (3/5)

- Προτεραιότητα τελεστών - προσεταιριστικότητα. Πως εφαρμόζεται το: x op y op z
  - `%left op` ερμηνεύεται ως (x op y) op z
  - `%right op` ερμηνεύεται ως x op (y op z)
  - `%nonassoc op` το x op y op z θεωρείται συντακτικό λάθος
- Προτεραιότητα μόνο
  - `%precedence`
    - Ορίζει μόνο προτεραιότητα συμβόλων (όχι προσεταιριστικότητα)
- Η σχετική προτεραιότητα καθορίζεται από τη σειρά δήλωσης στο αρχείο του bison
  - Η πρώτη δήλωση στο αρχείο έχει χαμηλότερη από την επόμενη, κοκ.

Τερματικά Σύμβολα (4/5)

- Τρόποι γραφής των τερματικών συμβόλων:
  - *Τύπου χαρακτήρα* (single-character literal token type): γράφεται στη γραμματική όπως οι χαρακτήρες στη γλώσσα C
    - Για παράδειγμα το `'+'` είναι λεκτική μονάδα τύπου χαρακτήρα
    - Δεν χρειάζεται να δηλωθεί ξεχωριστά εκτός εάν θέλουμε να καθορίσουμε την προτεραιότητα ή την προσεταιριστικότητά του
    - Συνήθως χρησιμοποιείται για την αναπαράσταση λεκτικής μονάδας η οποία αποτελείται μόνο από ένα χαρακτήρα

Τερματικά Σύμβολα (5/5)

- Τρόποι γραφής των τερματικών συμβόλων:
  - *Τύπου αλφαριθμητικού* (literal string token type): γράφεται στη γραμματική όπως τα αλφαριθμητικά στη γλώσσα C
    - Για παράδειγμα το `"<="` είναι λεκτική μονάδα τύπου αλφαριθμητικού
    - Δεν χρειάζεται να δηλωθεί ξεχωριστά εκτός εάν θέλουμε να καθορίσουμε τη σημασιολογική του τιμή, την προτεραιότητα ή την προσεταιριστικότητά του
    - Μπορεί να δηλωθεί χρησιμοποιώντας τη δήλωση `%token`

Μη-Τερματικά Σύμβολα

- Ένα ***μη-τερματικό σύμβολο*** αναπαριστά μια κλάση συντακτικά ισοδύναμων ομαδοποιήσεων (groupings)
- Ορίζεται ομαδοποιώντας άλλα σύμβολα στη βάση γραμματικών κανόνων
  - Πρέπει να υπάρχει κανόνας που να περιγράφει πως δημιουργείται από άλλα τερματικά ή/και μη-τερματικά σύμβολα
- Χρησιμοποιείται για τον ορισμό γραμματικών κανόνων
- Από σύμβαση για το όνομά του χρησιμοποιούνται πεζά γράμματα
- Μπορεί να δηλωθεί ως: `%type <type> nonterminal`
  (θα εξηγηθεί παρακάτω)

Σημασιολογικές τιμές (semantic values)

- Κάθε τερματικό σύμβολο εκτός από τον τύπο του έχει και μία **σημασιολογική τιμή**
  - Παράδειγμα:
    - Λεκτική Μονάδα - Τερματικό : **`43`**
      - τύπος: **`KW_INTEGER`**
      - σημασιολογική τιμή: **`43`**
- Κάθε μη-τερματικό σύμβολο εκτός από το όνομά του έχει μία σημασιολογική τιμή
  - Παράδειγμα
    - Σε μια αριθμομηχανή (calculator) μια έκφραση (expression) συνήθως έχει ως σημασιολογική τιμή έναν αριθμό, το αποτέλεσμα του υπολογισμού της
    - Σε ένα μεταγλωττιστή μια έκφραση συνήθως έχει ως σημασιολογική τιμή μια δενδρική δομή η οποία περιγράφει το νόημα της έκφρασης

Σημασιολογικές τιμές (2/4)

- Η σημασιολογική τιμή των συμβόλων έχει τύπο. Ο bison θεωρεί ότι ο προκαθορισμένος τύπος της σημασιολογικής τιμής όλων των συμβόλων είναι `int` (default type)
- Για να ορίσουμε ένα διαφορετικό τύπο χρησιμοποιούμε την οδηγία `%define`
  - `%define api.value.type {double}`
- Εναλλακτικά μπορεί να χρησιμοποιηθεί, στην ενότητα prologue του bison αρχείου, το `macro YYSTYPE`
  - `#define YYSTYPE double`

Σημασιολογικές τιμές (3/4)

- Ορισμός διαφορετικών τύπων
  - Συνήθως χρειαζόμαστε διαφορετικούς τύπους σημασιολογικών τιμών για διαφορετικά τερματικά και μη-τερματικά σύμβολα
  - Παράδειγμα:
    - αριθμητική σταθερά, τύπος: `int` ή `long int`
    - σταθερή συμβολοσειρά, τύπος: `char*`
- Ορισμός συλλογής τύπων χρησιμοποιώντας τη δήλωση `union` (όπως στη C)

```
%union {
    char* string;
    double val;
}
```

Σημασιολογικές τιμές (4/4)

- Επιλέγουμε έναν από τους τύπους της συλλογής `%union` για κάθε σύμβολο (τερματικό ή μη-τερματικό) για το οποίο χρησιμοποιείται η σημασιολογική του τιμή. Παράδειγμα:
  - `%token <string> IDENTIFIER`
- Το `%type` χρησιμοποιείται για τη δήλωση μη-τερματικών συμβόλων όπως το `%token` χρησιμοποιείται για τη δήλωση τερματικών
  - `%type <type> nonterminal`
    - `nonterminal`: το όνομα του μη-τερματικού συμβόλου
    - `<type>`: το όνομα του τύπου όπως δηλώνεται στο `%union`
    - `%type <val> expr`

Γραμματικοί Κανόνες

- Η γραμματική του bison είμαι μια λίστα από κανόνες
- Ένας γραμματικός κανόνας έχει την ακόλουθη γενική μορφή
  - ***`result: components...;`***

  Όπου:
  - ***`result`*** *(αριστερή πλευρά):* είναι το μη-τερματικό σύμβολο το οποίο περιγράφεται από τον κανόνα
  - ***`components`*** *(δεξιά πλευρά):* διάφορα τερματικά και μη-τερματικά σύμβολα (μηδέν ή περισσότερα)
- Παράδειγμα
  - **`exp: exp '+' exp;`**

Γραμματικοί Κανόνες (2/5)

- Πολλοί κανόνες για το ίδιο result μπορούν να γραφούν είτε ξεχωριστά είτε χρησιμοποιώντας το χαρακτήρα ‘|’ ως εξής:
  - ```
    result:
      rule1-components...
    | rule2-components...
    ...
    ;
    ```

```
%token <val> NUM
%type <val> exp
%%
exp:
  NUM
| '(' exp ')'
| exp '+' exp
| exp '-' exp
| exp '/' exp
| exp '*' exp
;
```
```

Γραμματικοί Κανόνες (3/5)

- Αναδρομικοί Κανόνες (recursive rules)
  - Ένας κανόνας καλείται αναδρομικός όταν το μη-τερματικό σύμβολο της αριστερής πλευράς (result) εμφανίζεται επίσης στη δεξιά του πλευρά
    Παράδειγμα:
    - ```
      identifier_list:
        IDENTIFIER
      | identifier_list ',' IDENTIFIER
      ;
      ```
  - *Η αναδρομή είναι ο μόνος τρόπος για να δηλώσουμε μια ακολουθία πραγμάτων οποιουδήποτε μεγέθους*
      ```

Γραμματικοί Κανόνες (4/5)

- *left recursion*

```
identifier_list:
  IDENTIFIER
| identifier_list ',' IDENTIFIER
;
```

- *right recursion*

```
identifier_list:
  IDENTIFIER
| IDENTIFIER ','identifier_list
;
```

- *indirect or mutual recursion*

```
expr:
  primary
| primary '+' primary
;

primary:
  constant
| '(' expr ')'
;
```

Γραμματικοί Κανόνες (5/5)

- Κενοί Κανόνες
  - Ένας κανόνας καλείται κενός όταν η δεξιά του πλευρά (components) είναι κενή
  - Σημαίνει ότι ο κανόνας ταιριάζει την κενή συμβολοσειρά
    - Παράδειγμα: `semicolon: | ';';`
    - Πρόβλημα: Δεν είναι εύκολο να δούμε τον κενό χαρακτήρα όταν χρησιμοποιείται το |
    - Λύση: χρήση της οδηγίας `%empty` στη δήλωση κενών κανόνων ή σχολίου `/* empty */`
    - Παράδειγμα:

```
semicolon: %empty | ';';
```

ή

```
semicolon:
  /* empty */
| ';'
;
```

Γραμματικοί Κανόνες - Ενέργειες (actions)

- Ένας γραμματικός κανόνας μπορεί να συνοδεύεται από μία ενέργεια
- Η ενέργεια αποτελείται από ένα σύνολο εντολών της C περικλειόμενο σε άγκιστρα { και }
- Η ενέργεια εκτελείται κάθε φορά που ο Συντακτικός Αναλυτής ταιριάζει το συγκεκριμένο κανόνα
- Ο σκοπός της ενέργειας είναι ο υπολογισμός της σημασιολογικής τιμής του μη-τερματικού συμβόλου του (αριστερή πλευρά) από τις τιμές των συστατικών του συμβόλων, τερματικών και μη-τερματικών, (δεξιά πλευρά του κανόνα)

Γραμματικοί Κανόνες - Ενέργειες (2/4)

- Ο C κώδικας μιας ενέργειας μπορεί να προσπελάσει τη σημασιολογική τιμή των συστατικών του συμβόλων (δεξιά πλευρά κανόνα) ως:
  - `$n`
  - Όπου `n` αντιστοιχεί στο n-οστό συστατικό (component)
- Η σημασιολογική τιμή του result (αριστερή πλευρά κανόνα) μπορεί να προσπελαστεί με το συμβολισμό `$$`
- Κάθε φορά που χρησιμοποιείται το `$$` ή το `$n` ο τύπος του καθορίζεται από το σύμβολο στο οποίο αναφέρεται

Γραμματικοί Κανόνες - Ενέργειες (3/4)

- Παράδειγμα

```
%token <val> NUM
%type <val> exp
%%
exp:
  NUM             {$$ = $1}
| '(' exp ')'     {$$ = $2}
| exp '+' exp     {$$ = $1 + $3}
| exp '-' exp     {$$ = $1 - $3}
| exp '/' exp     {$$ = $1 / $3}
| exp '*' exp     {$$ = $1 * $3}
;
```

Γραμματικοί Κανόνες - Ενέργειες (4/4)

- Αν δεν καθοριστεί κάποια ενέργεια για ένα κανόνα εκτελείται η προκαθορισμένη ενέργεια:
  - `$$ = $1`
  - Η ενέργεια αυτή είναι έγκυρη μόνο όταν οι τύποι των δύο σημασιολογικών τιμών ταιριάζουν

Μεταβλητές και Συναρτήσεις (1/2)

Σύμβολο	Περιγραφή
`int yyparse()`	• Η κύρια συνάρτηση του *Συντακτικού Αναλυτή* η οποία εκτελεί τη Συντακτική Ανάλυση στη βάση των δοθέντων γραμματικών κανόνων. Επιστρέφει 0 αν η συντακτική ανάλυση τελειώσει χωρίς σφάλματα, 1 αν βρέθηκαν συντακτικά σφάλματα
`int yylex()`	• Η κύρια συνάρτηση του *Λεκτικού Αναλυτή* η οποία σαρώνει το αρχείο εισόδου για την αναγνώριση Λεκτικών Μονάδων οι οποίες δίνονται στο Συντακτικό Αναλυτή • Συνήθως παρέχεται από τον flex και καλείται από την **`yyparse()`** • Επιστρέφει έναν ακέραιο αριθμό, που συνήθως αντιστοιχεί σε μια Λεκτική Μονάδα (Τερματικό Σύμβολο) ή 0 για **`EOF`**

Μεταβλητές και Συναρτήσεις (2/2)

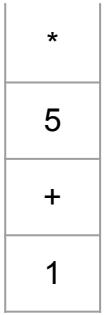
Σύμβολο	Περιγραφή
`void yyerror(s) const char *s`	• Η συνάρτηση χειρισμού σφαλμάτων που πρέπει να δίνεται από το χρήστη. Η συμβολοσειρά **`s`** περιέχει κάποιο μήνυμα που μπορεί να τυπωθεί για το σφάλμα που παρουσιάστηκε • Ο Συντακτικός Αναλυτής εντοπίζει κάποιο λάθος όταν διαβάζει κάποιο σύμβολο και δεν μπορεί να ταιριάξει κανένα γραμματικό κανόνα • Καλείται από την **`yyparse()`** όταν βρεθεί κάποιο λάθος
`YYSTYPE`	Ο τύπος της σημασιολογικής τιμής των συμβόλων
`YYSTYPE yylval`	Η καθολική μεταβλητή όπου η συνάρτηση **`yylex()`** πρέπει να τοποθετεί τις σημασιολογικές τιμές των τερματικών συμβόλων

Ο αλγόριθμος του bison (1/11)

- Καθώς ο bison αναλυτής διαβάζει Λεκτικές Μονάδες, τις εισάγει (push) σε μια στοίβα μαζί με τις σημασιολογικές τους τιμές
- Η στοίβα ονομάζεται ***στοίβα συντακτικού αναλυτή (parser stack)***
- Η εισαγωγή μιας Λεκτικής Μονάδας στη στοίβα παραδοσιακά ονομάζεται ***μετακίνηση (shifting)***

Ο αλγόριθμος του bison (2/11)

- Παράδειγμα (υποθέτουμε ότι έχει δηλωθεί κατάλληλα η προσεταιριστικότητα)
  - Έστω ότι έχει διαβαστεί το `'1 + 5 *'` και ακολουθεί το `'3'`
  - Η στοίβα θα έχει τέσσερα στοιχεία, ένα για κάθε Λεκτική Μονάδα
- ... αλλά δεν εισάγονται συνεχώς στοιχεία στη στοίβα



Ο αλγόριθμος του bison (3/11)

- Όταν τα τελευταία *ν* σύμβολα που έχουν μετακινηθεί στη στοίβα ταιριάξουν με κάποιο γραμματικό κανόνα τότε:
  - Βγαίνουν από τη στοίβα
  - Εισάγεται στη στοίβα το σύμβολο του αριστερού τμήματος του κανόνα που ταίριαξε
- Η παραπάνω διαδικασία ονομάζεται ***αναγωγή (reduction)*** (**χωρίς lookahead)
- Οι ενέργειες ενός κανόνα εκτελούνται κατά τη διαδικασία της αναγωγής διότι τότε υπολογίζεται η σημασιολογική τιμή της αριστερής πλευράς του κανόνα

Ο αλγόριθμος του bison (4/11)

- Παράδειγμα:
  - Υπολογισμός αριθμητικών εκφράσεων
  - Αν η στοίβα περιέχει `1 + 5 * 3`
  - και η επόμενη Λεκτική Μονάδα είναι ο χαρακτήρας νέας γραμμής τότε:
    - Τα τελευταία 3 στοιχεία ανάγονται στο 15 μέσω του κανόνα `expr:expr '*' expr;`
    - Η στοίβα περιέχει `1 + 15`
    - Εκτελείται δεύτερη αναγωγή μέσω του κανόνα: `expr: expr '+' expr;`
    - Η στοίβα περιέχει `16`
    - Μετακινείται ο χαρακτήρας νέας γραμμής στη στοίβα
    - Εκτελείται μια ακόμα αναγωγή μέσω του κανόνα: `expr: NUM;`
    - Ολοκληρώνεται επιτυχώς η ανάλυση

Ο αλγόριθμος του bison (5/11)

- Ο Αναλυτής προσπαθεί, μέσω διαδοχικών μετακινήσεων και αναγωγών, να ανάγει τη συνολική είσοδο στο *αρχικό σύμβολο* (start symbol) της γραμματικής
- Αυτού του είδους ο αναλυτής ονομάζεται ***από κάτω προς τα πάνω (bottom-up)***

Ο αλγόριθμος του bison (6/11)

- Πρόβλεψη ενός συμβόλου (lookahead)
  - Ο bison αναλυτής δεν εκτελεί αμέσως μια αναγωγή με το ταίριασμα *ν* συμβόλων με κάποιο κανόνα αλλά:
    - Όταν μια αναγωγή είναι εφικτή τότε ο αναλυτής, κάποιες φορές, “κοιτάζει μπροστά” (“looks ahead”) το επόμενο διαθέσιμο σύμβολο προκειμένου να αποφασίσει τι θα κάνει:
      - Αναγωγή ή Μετακίνηση
    - Όταν διαβάζεται ένα σύμβολο, δεν μετακινείται αμέσως στη στοίβα, αλλά αρχικά γίνεται το *lookahed token*
    - Στη συνέχεια ο αναλυτής, και ανάλογα με το lookahed token, αποφασίζει τι θα κάνει

Ο αλγόριθμος του bison (7/11)

- Παράδειγμα (παραγοντικό !):

```
expr:
   term '+' expr
| term
;
term:
   '(' expr ')'
| term '!'
| NUM
;
```

- Έστω στη στοίβα `'1+2'`
- Αν επόμενο σύμβολο (lookahead):
  - `')'`
    - reduce `'1+2'`
      - If shift, then: term ‘)’ (no rule allows it)
  - `'!'`
    - shift `'!'`
      - If reduce, then: expr ‘!’ (no rule)
    - reduce `'2!'`

Ο αλγόριθμος του bison (8/11)

- Συγκρούσεις μετακίνησης/αναγωγής (shift/reduce conflicts)

```
if_stmt:
  "if" expr "then" stmt
| "if" expr "then" stmt "else" stmt
;
```

- **`"if"`**, **`"then"`**, **`"else"`** τερματικά σύμβολα για τις αντίστοιχες λέξεις κλειδιά
- Τι πρέπει να γίνει όταν το **`"else"`** γίνεται lookahead token;
  - αναγωγή με βάση τον πρώτο κανόνα; ή
  - μετακίνηση **`"else"`** στη στοίβα η οποία θα οδηγήσει τελικά σε αναγωγή με βάση το δεύτερο κανόνα;
  - Η κατάσταση αυτή ονομάζεται σύγκρουση ***μετακίνησης/αναγωγής***

Ο αλγόριθμος του bison (9/11)

- Συγκρούσεις μετακίνησης/αναγωγής
  - Ο bison για την επίλυση αυτών των συγκρούσεων επιλέγει τη μετακίνηση (shift)
  - *Εκτός* εάν έχει οριστεί διαφορετικά μέσω κάποιας δήλωσης προτεραιότητας
- Προτεραιότητα τελεστών - προσεταιριστικότητα
  - `%left op` ερμηνεύεται ως (x op y) op z
  - `%right op` ερμηνεύεται ως x op (y op z)
  - `%nonassoc op` το x op y op z θεωρείται συντακτικό λάθος
  - Παράδειγμα:

```
%left '<' '>' '=' "!=" "<=" ">="
%left '+' '-'
%left '*' '/'
```

Ο αλγόριθμος του bison (10/11)

- Προτεραιότητα (precedence) μη τελεστών, δηλαδή συμβόλων
  - `%precedence THEN`
  - `%precedence ELSE`
- Η σχετική προτεραιότητα καθορίζεται από τη σειρά δήλωσης στο αρχείο του bison
  - Η πρώτη δήλωση στο αρχείο έχει χαμηλότερη από την επόμενη, κοκ.
- Προτεραιότητα, πως δουλεύει
  - Οι δηλώσεις προτεραιότητας αποδίδουν επίπεδα προτεραιότητας σε τερματικά σύμβολα
  - Επιπλέον, αποδίδεται εμμέσως προτεραιότητα σε κανόνες
    - *Κάθε κανόνας παίρνει την προτεραιότητα από το τελευταίο τερματικό του σύμβολο*

Ο αλγόριθμος του bison (11/11)

- Επίλυση συγκρούσεων
  - Βασίζεται στη σύγκριση της προτεραιότητας του εξεταζόμενου κανόνα σε σχέση με την προτεραιότητα του lookahead token:
  - Αν προτεραιότητα lookahead token > προτεραιότητα κανόνα τότε
    - Μετακίνηση
  - Διαφορετικά
    - Αναγωγή
  - Αν η προτεραιότητα είναι ίδια τότε, επιλογή βάση προσεταιριστικότητας
- Δεν έχουν όλοι οι κανόνες ούτε όλα τα τερματικά σύμβολα προτεραιότητα
  - Αν δεν έχει ούτε ο κανόνας ούτε το lookahead token τότε: μετακίνηση (default shift)

Αποσφαλμάτωση (debugging)

- Απαιτεί γνώση του αλγορίθμου του bison
- Είναι δύσκολο
- Αξιοποίηση του αρχείου που παράγεται με τη χρήση των flags -v -r
  - **`bison -d -v -r all myparser.y`**
  - Παραγόμενα αρχεία:
    - **`myparser.tab.h, myparser.tab.c, myparser.output`**
  - Το αρχείο **`myparser.output`** περιέχει πληροφορίες σχετικά με:
    - Τερματικά, μη-τερματικά σύμβολα και κανόνες που δεν χρησιμοποιούνται
    - Το αυτόματο που δημιουργείται (καταστάσεις, μεταβάσεις)
    - Αναγωγές, Μετακινήσεις που πραγματοποιούνται
    - Συγκρούσεις (conflicts)

Παράδειγμα 1

- Λίστα από αριθμητικές εκφράσεις της μορφής
  - 10+100*12/(5-7),
  - A+B*C/(E-F);

- Γραμματική για λίστα από αριθμητικές εκφράσεις
  - listOfExprs -> expr | listOfExprs ‘,’ expr
  - expr -> expr ‘+’ expr | expr ‘-’ expr | expr ‘*’ expr | expr ‘/’ expr | (expr) | NUM | ID

Παράδειγμα 2

- Μετατροπή εκφράσεων από infix σε postfix
  - Η σημειογραφία postfix για μία έκφραση E μπορεί να οριστεί επαγωγικά ως εξής:
    - Αν η E είναι μία μεταβλητή ή μία σταθερά, τότε η postfix σημειογραφία για την E είναι η ίδια η E
    - Αν η E είναι μια έκφραση της μορφής E1 **op** E2, όπου **op** είναι ένας δυαδικός τελεστής, τότε η postfix σημειογραφία για την E είναι E1’ E2’ **op**, όπου E1’ και E2’ είναι οι postfix σημειογραφίες για τις E1 και E2 αντίστοιχα
    - Αν η E είναι μία έκφραση σε παρανθέσεις της φορφής (E1), τότε η postfix σημειογραφία για την E είναι ίδια με την postfix σημειογραφία της E1
  - Παράδειγμα:
    - Infix: (9-5)+2, 9-(5+2), A+B*C/(E-F)
    - Postfix: 95-2+, 952+-, ABC*EF-/+

Αναφορές

- Εγχειρίδιο χρήσης (manual) του bison, έκδοση 3.0.4
- Διαφάνειες Φροντιστηρίου Θεωρίας Υπολογισμού 2014