Bison v 3.8.1

Versiuanea septembrie 2021

Introducere

Bison este un generator de parser-e care convertește o gramatică independentă de context adnotată într-o tabelă de analiză sintactică de tip LR(1) sau LALR(1).
Bison permite dezvoltarea unei mari varietăți de parser-e pentru diferite limbaje, de la cele utilizate pentru calculatoarele simple până la limbajele de programare.
Bison este în mare măsură compatibil cu Yacc (Yet another compiler compiler, dezvoltat pentru Unix)
Utilizatorul de bison trebuie să fie familiarizat cu programarea în C, C++, D sau Java.

Bison a fost conceput de către Robert Corbett. Richard Stallman l-a făcut compatibil cu Yacc.
Wilfred Hansen de la Univ Carnegie Mellon a adăugat mai multe caracteristici..

CONCEPTE DE BAZĂ ÎN *bison*

- Gramatici și limbaje independente de context
- Formatul *bison: fișierul de intrare* bison
- Valori semantice
- Acțiuni semantice
- Scrierea parser-elor de tip GLR (Generalised LR parsers)
- Localizarea (textuală)
- Ce produce bison: fișierul pentru implementarea parser-ului
- Stadii în utlilizarea bison
- Aspectul general al unei gramatici bison

Fișiere ce conțin gramatici în format bison

- Bison primește la intrare specificația unei gramatici independente de context și produce o funcție C care recunoaște instanțele (șirurile) corecte (din punct de vedere sintactic) produse de gramatică.
- Fișierul de intrare Bison ce conține regulile gramaticii are numele urmat de extensia ‘.y’.

Structura unei gramatici în format bison

```
%{
  Prologue
%}

Bison declarations

%%
Grammar rules
%%

Epilogue
```

Prologul

Secțiunea Prolog conține macro-definiții și declarații de funcții și variabile care sunt utilizate în acțiunile ce apar în regulile gramaticii. Toate acestea sunt copiate la începutul fișierului ce conține implementarea parser-ului astfel ca să preceadă definiția funcției yyparse.

Se poate utiliza ‘#include’ pentru a prelua declarații dintr-un fișier header.

Dacă nu aveți nevoie de astfel de declarații, puteți omite delimitatorii ‘%{’ și ‘%}’ care marchează această secțiune.

Secțiunea Prolog este încheiată la prima apariție a ‘%}’ care este în afara unui comentariu, literal de tip șir sau constantă caracter.

Pot exista mai multe secțiuni Prolog, ce se pot amesteca cu declarațiile Bison. Aceasta permite să aveți declarații C și Bison care se referă unele la altele. De exemplu, declarația %union poate utliliza tipuri definite într-un fișier header, sau poate doriți ca funcțiile prototip să aibă argumente de tip YYSTYPE. Aceasta se poate realiza cu ajutorul a două blocuri Prolog, unul înainte și altul după declarația %union.

Prologul

```
%{
  #define _GNU_SOURCE
  #include <stdio.h>
  #include "ptypes.h"
%}


%union {
  long n;
  tree t;  /* tree is defined in ptypes.h. */
}


%{
  static void print_token (yytoken_kind_t token, YYSTYPE val);
%}


…
```

Declararea structurii union

Declarația %union specifică o multime de tipuri de date ce pot fi folosite pentru valorile semantice. Cuvântul cheie %union este urmat de cod între acolade, la fel ca în C. De exemplu:

```
%union {
  double val;
  symrec *tptr;
}
```

Aceasta arată că cele două tipuri alternative sunt double și symrec * cu numele *val* și *tptr*. Aceste nume vor fi utlilizate în declaratorii %token, %nterm și %type pentru a aloca un tip valorilor semantice asociate terminalilor și neterminalilor. Poate fi asociat un tag pentru %union, ca în exemplul:

```
%union value {
  double val;
  symrec *tptr;
}
```

Prologul - Bison are o directivă **%code** cu un câmp calificator, care indică locațiile unde Bison trebuie să îl genereze. Pentru C/C++, calificatorul poate fi omis pentru locația predefinită sau poate fi **requires**, **provides**, **top**.

```
%code top {
   #define _GNU_SOURCE
   #include <stdio.h>

   /* WARNING: The following code really belongs
    * in a '%code requires'; see below. */

   #include "ptypes.h"
   #define YYLTYPE YYLTYPE
   typedef struct YYLTYPE
   {
      int first_line;
      int first_column;
      int last_line;
      int last_column;
      char *filename;
   } YYLTYPE;
}

%union {
   long n;
   tree t;   /* tree is defined in ptypes.h. */
}

%code {
   static void print_token (yytoken_kind_t token, YYSTYPE val);
   static void trace_token (yytoken_kind_t token, YYLTYPE loc);
}
```

Secțiunea de declarații *bison*

Secțiunea de declarații definește simbolurile utilizate de gramatica *bison* (terminali, neterminali) și tipurile de date ale valorilor semantice.

Numele tuturor tipurilor de tokeni (exceptând tokenii de un caracter, ca de exemplu'+' și '*') trebuie declarate. Neterminalii trebuie declarați dacă veți folosi valori semantice asociate.

Prima regulă (producție) din gramatică specifică și simbolul de start, în mod implicit. Dacă doriți ca un alt neterminal să fie simbol de start, acesta trebuie declarat explicit.

Declararea tokenilor

- Cel mai simplu mod de a declara un token este:

%token *name*

- *Bison* va converti această declarație în parser, astfel că dacă funcția `yylex` este în fișier, va folosi *name* ca token.
- Alternativ, se pot utiliza directivele `%left`, `%right`, `%precedence` sau `%nonassoc` în loc de `%token`, dacă tokenul reprezintă un operator căruia îi puteți astfel specifica precedența sau asociativitatea. Nu utilizați acești declaratori pentru nume de șiruri, tipuri semantice etc.
- Unui tip de token i se poate asocia un literal de tip string :

%token ARROW "=>"

Declararea tokenilor

```
%token
     OR      "||"
     LPAREN  "("
     RPAREN  ")"
     '\n'    _("end of line")
   <double>
     NUM     _("number")
```

Tokenilor li se pot asocia valori, de exemplu constantelor numerice valoarea acestora. Dacă stiva care păstrează valori asociate simbolurilor gramaticii este declarată ca o structură union, atunci la declaratorul unui token putem adăuga un tip:

```
%union {                /* define stack type */
  double val;
  symrec *tptr;
}
%token <val> NUM        /* define token NUM and its type */
```

Declararea tokenilor – specificarea asociativității și a precedenței

```
%left '<'
%left '-'
%left '*'
```

Toți tokenii declarați pe aceeași linie (la același nivel) cu %left sau %right au același nivel de precedență. Când 2 tokeni sunt declarați pe nivele diferite (sub incidența a doi declaratori diferiți), cel declarat mai târziu (mai jos) are precedența cea mai mare.

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

Declararea neterminalilor

Când este folosită directiva %union pentru a declara diferite tipuri pentru valorile (semantice) asociate regulilor gramaticii, trebuie declarat tipul pentru fiecare neterminal care utilizează astfel de valori. Aceasta se face cu ajutorul declarației %type:

```
%type <type> nonterminal…
```

Aici *nonterminal* este numele unui neterminal, iar *type* este numele tipului valorii asociate, declarat în %union. Puteși aloca orice *număr* mai multor simboluri neterminale ce apar în declarativa %tipe, separate prin spații.
În Yacc este permisă %type doar pentru neterminali. În Bison poate fi utilizată și pentru tokeni. Pentru a o folosi exclusiv pentru neterminali, se folosește directiva %nterm.

```
%nterm <type> nonterminal…
```

Pentru simbolul de start Bison consideră implicit că este primul neterminal specificat în gramatică. Programatorul poate folosi directiva %start pentru a declara un alt neterminal ca simbol de start.

```
%start symbol
```

Sintaxa declarațiilor simbolurilor (terminali sau neterminali ai gramaticii)

Sintaxa diferitelor declarații de simboluri ale gramaticii (terminali –tokeni, neterminali)

```
%token tag? ( id number? string? )+ ( tag ( id number? string? )+ )*
%left  tag? ( id number?)+ ( tag ( id number? )+ )*
%type  tag? ( id | char | string )+ ( tag ( id | char | string )+ )*
%nterm tag? id+ ( tag id+ )*
```

unde *tag* desemnează un tip, *id* desemnează un identificator ce reprezintă numele simbolului (token sau neterminal), *number* un număr în zecimal sau hexazecimal asociat simbolului respectiv (bisun asociază automat asemenea numere, care identifică în mod unic simbolurile), *char* este un literal ca '+', iar *string* este un literal care desemnează un șir asociat simbolului respectiv. Simbolurile '?', '*', '+' au aceeași semnificație ca în expresiile regulate: '?' pentru 0 sau o apariție, '*' pentru 0 sau mai multe apariții ,'+' pentru cel puțin o apariție.

Directivele %precedence, %right și %nonassoc au aceeași sintaxă ca %left.

Regulile unei gramatici în format *Bison*

O gramatică *Bison* constă dintr-un set de reguli.

- Sintaxa regulilor gramaticii Bison
- Regulile vide
- Reguli recursive

Sintaxa regulilor gramaticii Bison

O regulă a gramaticii *Bison* are forme generală:

```
result: components…;
```

unde *result* este neterminal iar *components* reprezintă diverși terminali și neterminali

De exemplu:

```
exp: exp '+' exp;
```

ne spune că două expresii *exp* cu ‘+’ între ele pot fi grupate împreună într-o grupare mai mare de tip *exp.*
Spațiile sunt semnificative în sensul că separă simboluri. Puteți folosi câte spații doriți.

În interiorul membrului drept al regulilor pot să apară componente numite *acțiuni* care desemnează reguli semantice. O acțiune arată astfel:

```
{C statements}
```

{*C statements*} de mai sus reprezintă *cod* C între acolade, ce poate să conțină orice secvență de tokeni C, oricât de lung, până când se balansează acoladele. Bison nu verifică corectitudinea codului C, care va fi copiat ca atare în fișierul de ieșire ce conține parserul; compilatorul de C va face această verificare.

În interiorul unui cod C între acolade, acoladele de început și sfârșit nu sunt afectate de acoladele ce apar în comentarii, în literali de tip string sau în constante de tip char, dar sunt afectate de ‘<%’ and ‘%>’ care reprezintă acolade, deci codul trebuie să fie încheiat cu ‘}’

De regulă, există doar o acțiune pentru o regulă.

Reguli multiple pentru același neterminal denumit *result* pot fi introduse separat sau cu ajutorul caracterului ‘|’ ca în exemplul:

```
result:
   rule1-components…
|  rule2-components…
…
;
```

Acestea sunt considerate reguli distincte.

Reguli vide (λ-producții)

O regulă este *vidă* (*empty)* dacă partea sa dreaptă (componentele) este vidă. Aceasta înseamnă că neterminalul din membrul stâng se potrivește șirului vid. De exemplu, definirea unui simbol *punct și virgulă* opțional:

```
semicolon.opt: | ";";
```

Uneori regula vidă nu este ușor de observat, în special când folosim |. Directiva `%empty` ne permite să declarăm în mod explicit că o regulă este vidă:

```
semicolon.opt:
  %empty
| ";"
;
```

Semnalarea unei reguli nevide cu `%empty` este eroare.

Directiva `%empty` este o extensie Bison, nu există în Yacc. Pentru a rămâne compatibilă cu POSIX Yacc, de obicei se folosește un comentariu: '`/* empty */`' pentru fiecare regulă fără componente:

```
semicolon.opt:
  /* empty */
| ";"
;
```

Epilogul

Epilogul este copiat ca atare (verbatim) în fișierul care conține implementarea parser-ului, exact așa cum *prologul* este copiat la început. Acesta este locul cel mai potrivit de a plasa codul pe care doriți să îl conțină parserul, dar care trebuie să apară după definirea lui `yyparse`. De examplu, definițiile pentru `yylex` și `yyerror` apar adesea în *epilog*. Deoarece C cere ca funcțiile să fie declarate înainte de utilizare, adesea este necesară declararea funcțiilor ca `yylex` și `yyerror` în *Prolog*, chiar dacă le definiți în *Epilog*.

Dacă ultima secțiune este vidă (*Epilogul*), caracterele *'%%'* care separă regulile de *Epilog* pot fi omise.

Parserul pentru Bison conține el însși multe macro-uri și identificatori ale căror nume încep cu 'yy' sau 'YY', deci cel mai bine este să evitați astfel de nume (cu excepția celor documentate în acest manual) în epilogul gramaticii din fișierul de intrare.