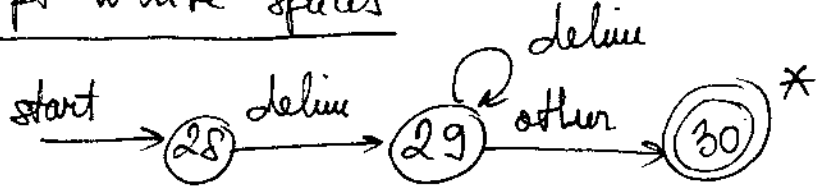
D.T. pt white-spaces



Implementarea unei D.T. (diagr. tranziții)

Fiec. stare → asoc. un segment de cod.

Dc ∃ arce care pleacă din starea resp. → citit urm. caracter și selectată calea.

nextchar() → 1) citește un caracter din buff.
2) avansează fwd. pointerul
3) întoarce caract. citit

Pt. a întoarce un at. lexical → var. globală → lex_value

Clasa at. lexical este întoarsă de ființa principală a an. lexical → urm Atom()

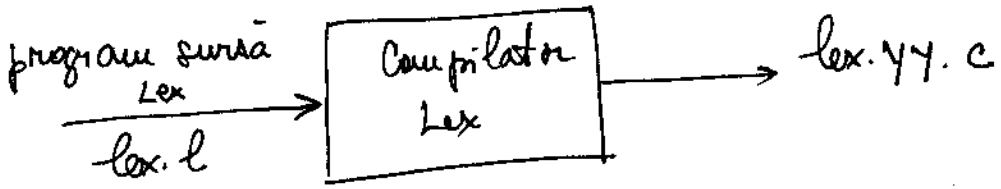
```
int store = 0, start = 0;
int lex_val;

int fail() {
    fwd_ptr = beg_ptr;
    switch (start) {
      case 0 : start = 9; break;
      case 9 : start = 12; break;
      case 12 : start = 20; break;
      case 20 : start = 25; break;
      case 25 : recover(); break;
      default: /* err. comp. */
    }
    return start;
}
```

```
Token runAtom() {
    while (1) {
        switch (state) {
            case 0: c = nextchar();
                    if ( c == blank || c == tab || c == newline ) {
                        state = 0;
                        lexeme_beg++;
                    }
                    else if (c == '<') state = 1;
                        else if ( c == '=') state = 5;
                            else if ( c == '>') state = 6;
                                else state = fail();
                    break;
```

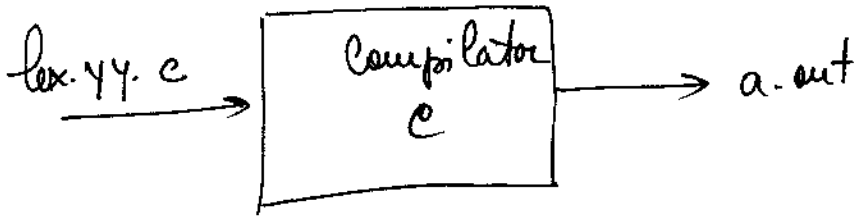
```
/* casuvile 1-8 */
case 9 : c = nextchar();
         if (isletter(c)) state = 10;
         else state = fail();
         break;
case 10: c = nextchar();
         if (isletter(c) || isdigit(c)) state = 10;
         else state = 11;
         break;
case 11: retract(1); instal_id();
         return getAt();
 /* casuvile 12-24 */
case 25 : c = nextchar();
          if (isdigit(c)) state = 26;
          else state = fail();
          break;
```

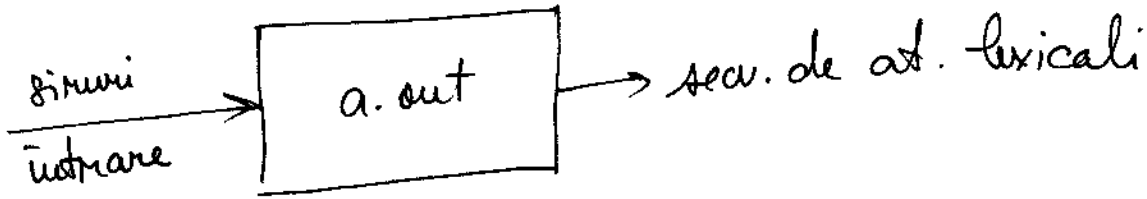
```
case 26: c = nextchar();
         if (isdigit(c)) state = 26;
         else state = 27;
         break;
case 27: retract(1); install_num();
         return NUM;

    }
  }
}
```

Limbaj pentru specificarea an. lexicale LEX



o reprezentare tabelară a diagr. de tranziții construite din expresiile reg. din lex.l + fișier standard





Specificare lex.

Un program lex → 3 părți:

declarații

%%

reguli de translatare

%%

fcții auxiliare

Declarații → def. variabile, constante, def. regulate

Reguli de translatare:

p1 { actiune 1 }

p2 { actiune 2 }

⋮

pn { actiune n }

unde p_i → exp. regulată

acțiunea i → fragment de cod care descrie ce acțiune trebuie să execute an. lexical când este găsită o potrivire cu p_i

- Fcții auxiliare → fcții apelate în acțiuni

Obs:

An. lexical întoarce doar at. lexical. Pt a întoarce o valoare atribut împreună cu info despre lexemă => var. globală yylval.

Spec lex.

```
%{ /* def. constante
      LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUM, OPREL */
%}
/* def. regulate */
```

```
delim   [ \t\n]
ws      {delim}+
lit     [A-Za-z]
dig     [0-9]
id      {lit} ({lit} | {dig})*
num     {dig}+ (\.{dig}+)? (E [+\-]? {dig}+)?

%%
{ws}    { /* nicio actiune */ }
if      { return IF; }
then    { return THEN; }
else    { return ELSE; }
{id}    { yylval = instal_id(); return ID; }
{num}   { yylval = instal_num(); return NUM; }
"<"     { yylval = LT ; return OPREL; }
```

"<=" { yylval = LE ; return OPREL; }

;

%%

instal_id() { /* fcție care pune lexema (șir de caract.) al cărei prim caracter este indicat de yytext și a cărei lungime este yyleng, în tab. de simboluri și întoarce intr. din tabelă */ }

instal_num() { /* similar pt. a adauga o const. numerică în T.S. */ }

Obs:

→ Tot ceea ce este între %{ și %} este copiat direct în lex.yy.c și nu este tratat ca parte a def. regulate din regulile de translat.

→ ac. lucru ⇒ fcții / proc. def. în a 3a secțiune

→ \. ⇒ pct, altfel reprez. clasa tuturor caracterelor except newline

→ \- => -, altfel simbol de interval [A-Z]

yylval → def. în lex.yy.c
yytext → pointer către primul caracter din lexemă (leg-ptr)
yyleng → lungimea lexemei (am folosit fwd-ptr).

Limbaje Independente de Context

Gramatici Indep. de Context

L. regulate < acceptori → A.F.
generatori → E.R.

$a(a^* \cup b^*)b$

Ex. S → nou simbol interpretat ca șir în limbaj
M → simbol corresp. $(a^* \cup b^*)$

$S \to aMb \overset{!}{\Rightarrow}$ regulă

$M \to A$
$M \to B$
$A \to aA \mid e$
$B \to bB \mid e$

Pornesc cu S, găsesc un simbol ~~care~~ în șirul generat de S, care apare la stg. într-una dintre reguli. Înloc simb cu șirul care apare în dr. regulii, etc

$S \to aAb \to aAb \to aaAb \to aaaAb \to aaab$

Def. autoioară → gramatică (G)

? G. indep. de context

(aa) Ab)

✓ contextul lui A

$A \to aA \Rightarrow$ A se poate înlocui prin șirul aA indiferent de contextul lui A.

Def.

O gramatică indep. de context, G, este un tuplu (V, Σ, R, S), unde:

$V \to$ alfabet

$\Sigma \subseteq V \to$ mulțimea simb. terminale

$R \subseteq (V-\Sigma) \times V^* \to$ mulțimea regulilor

↑ submulțime finită

$S \in V-\Sigma \to$ simbolul de start

Obs:

$V-\Sigma \to$ mult. simb. neterminale (neterminali)