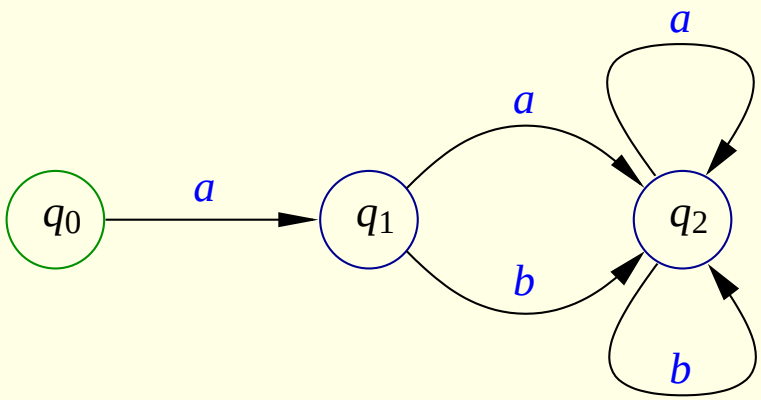
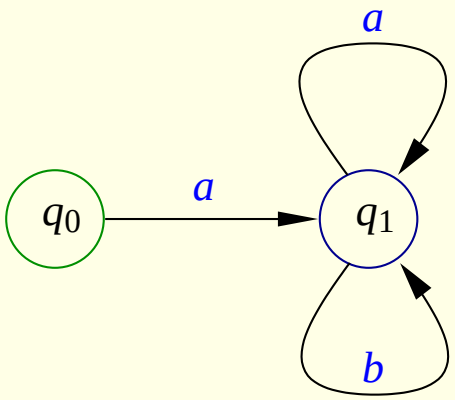
Determineeritud lõpliku automaadi minimiseerimine

- Regulaaravaldisest $a(a \mid b)^\star$ konstrueeritud determineeritud lõplik automaat:



- Temaga ekvivalentne, vähema olekute arvuga, automaat:



Determineeritud lõpliku automaadi minimiseerimine

- Determineeritud lõplik automaat on *minimaalne*, kui ei leidu temaga ekvivalentset, vähemate olekute arvuga, determineeritud lõplikku automaati.
- Iga determineeritud lõpliku automaadi $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ korral leidub (unikaalne) temaga ekvivalentne minimaalne determineeritud lõplik automaat $A' = \langle Q', \Sigma, \delta', q_0', F' \rangle$.
- Idee: tükeldame olekute hulga ekvivalentsiklassideks.
  - Olekud $p, q \in Q$ on *ekvivalentsed* ehk *eristamatud*, kui iga sõna $w \in \Sigma^\star$ korral automaat, alustades neist olekutest, mõlemal juhul kas õnnestub või ebaõnnestub.
  - Iga tähega üleminek viib ekvivalentsed olekud ekvivalentseteks olekuteks.

Determineeritud lõpliku automaadi minimiseerimine

Minimiseerimise algoritm:

- Eemadame kõik algolekust q_0 kättesaamatud olekud.
- Järelejäänud olekute hulgal leiame suurima tükelduse Π ekvivalentsiklassideks.
- Konstrueerime uue automaadi $A' = \langle Q', \Sigma, \delta', q_0', F' \rangle$, kus
  - olekutehulk $Q' = \Pi$;
  - algolek $q_0' = P_0$, kus $P_0 \in \Pi$ ja $q_0 \in P_0$;
  - lõppolekute hulk $F' = \{ P \in \Pi \mid P \cap F \neq \emptyset \}$;
  - üleminekufunktsioon $\delta' = \{ (P_i, a) \mapsto P_j \mid P_j \in move(P_i, a) \}$.

Determineeritud lõpliku automaadi minimiseerimine

Naiivne algoritm tükelduse leidmiseks:

$P := \{F,\ Q/F\};$
do $\Pi := P;\ P := \emptyset;$
  foreach $S \in \Pi$ **do**
    foreach $a \in \Sigma$ **do**
      $U := \{T \in \Pi \mid T \cap move(S,a) \neq \emptyset\};$
      $V := \{S \cap move_a^{-1}(T) \mid T \in U\};$
      $P := P \cup V;$
    end
  end
until $\Pi = P;$

Determineeritud lõpliku automaadi minimiseerimine

- Toodud algoritm proovib igas iteratsioonis ”peenendada” kõiki tükke.
  - Halvimal juhul ruutkeerukusega.
  - Piisab, kui vaadelda ainult neid tükke, millest on võimalik ”liikuda” mõnda ”lõhenenud” tükki.
- Hopfcroft’i algoritm tükelduse leidmiseks:
  - kasutab ”töölisti” veel läbi vaatamata ”peenenenud” tükkide hoidmiseks;
  - kui mõni väljaspool ”töölisti” olev tükk lõheneb, siis paigutatakse ainult üks (väiksem) alamtükk ”töölisti”.

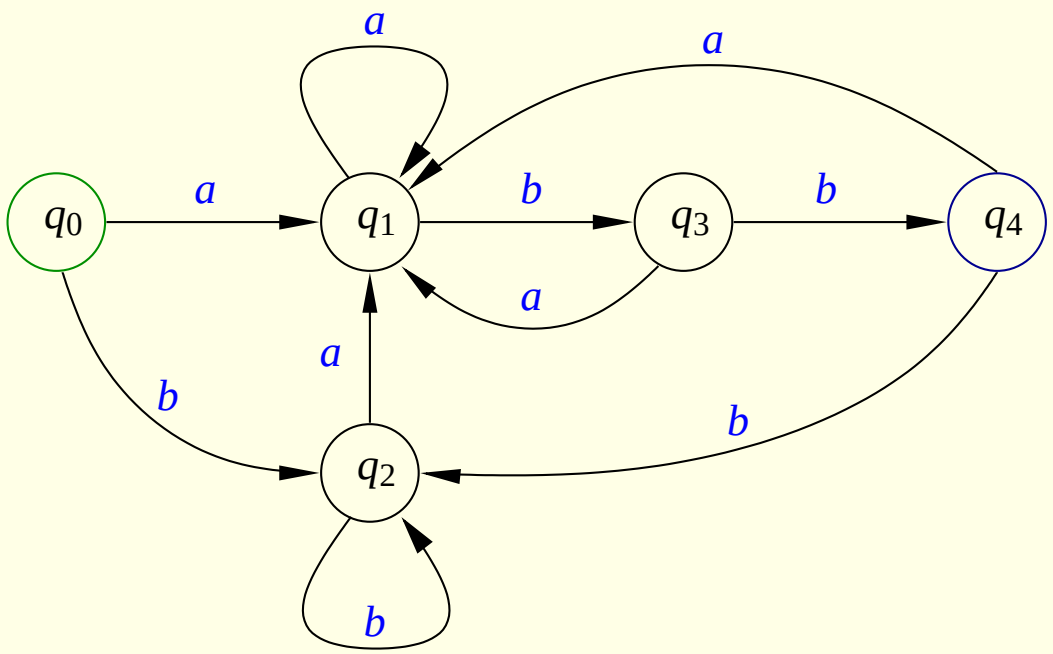
Determineeritud lõpliku automaadi minimiseerimine

Hopcroft'i algoritm:

$\Pi := \{F,\ Q/F\}$; $W := \Pi$;
while $\exists S \in W$ **do**
  $W := W/S$;
  foreach $a \in \Sigma$ **do**
    $P := move_a^{-1}(S)$;
    foreach $R \in \{T \in \Pi \mid T \cap P \neq \emptyset,\ T \not\subseteq P\}$ **do**
      $R_1 := R \cap P$; $R_2 := R/R_1$;
      $\Pi := (\Pi/R) \cup \{R_1,\ R_2\}$;
      if $R \in W$ **then** $W := (W/R) \cup \{R_1,\ R_2\}$;
      else if $|R_1| \leq |R_2|$ **then** $W := W \cup \{R_1\}$;
      else $W := W \cup \{R_2\}$;
    end
  end
end

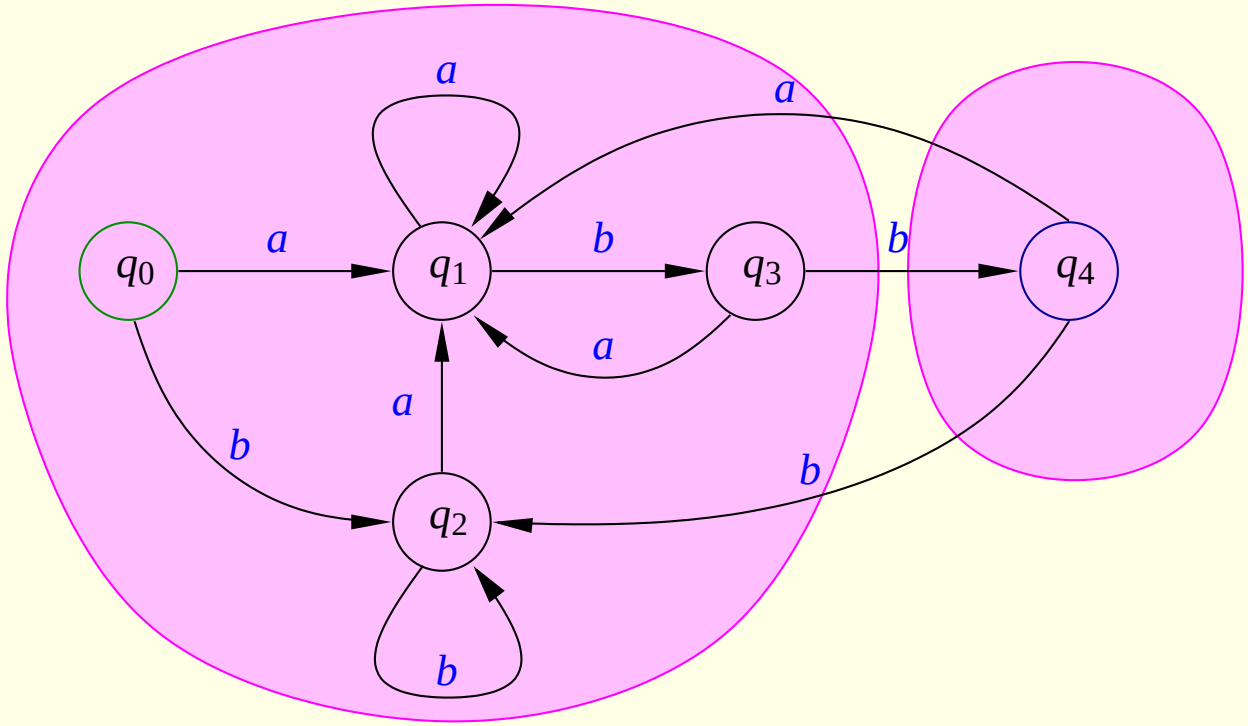
Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:



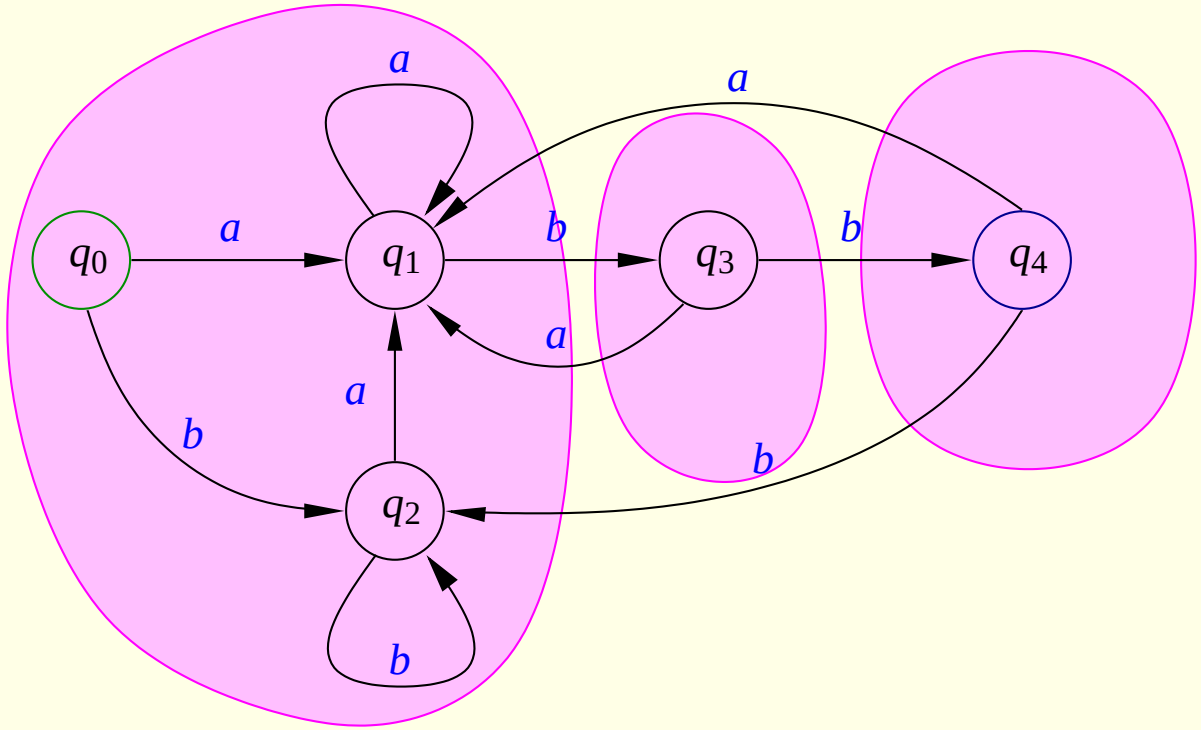
Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:



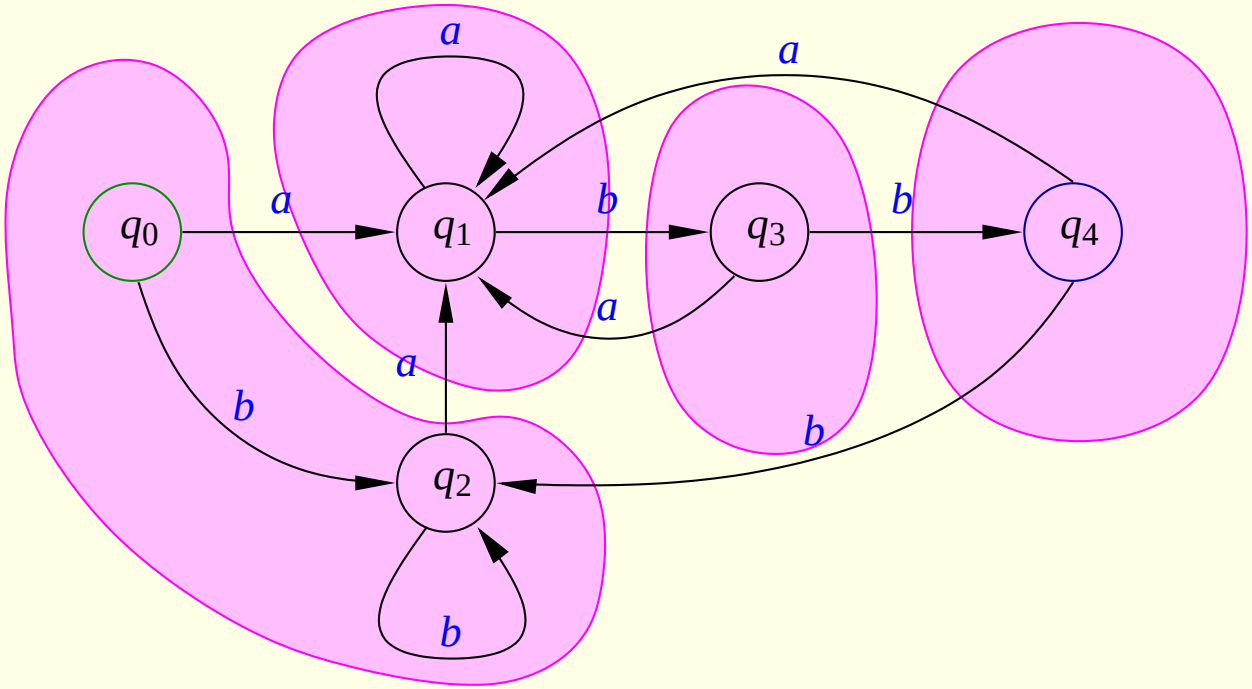
Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:



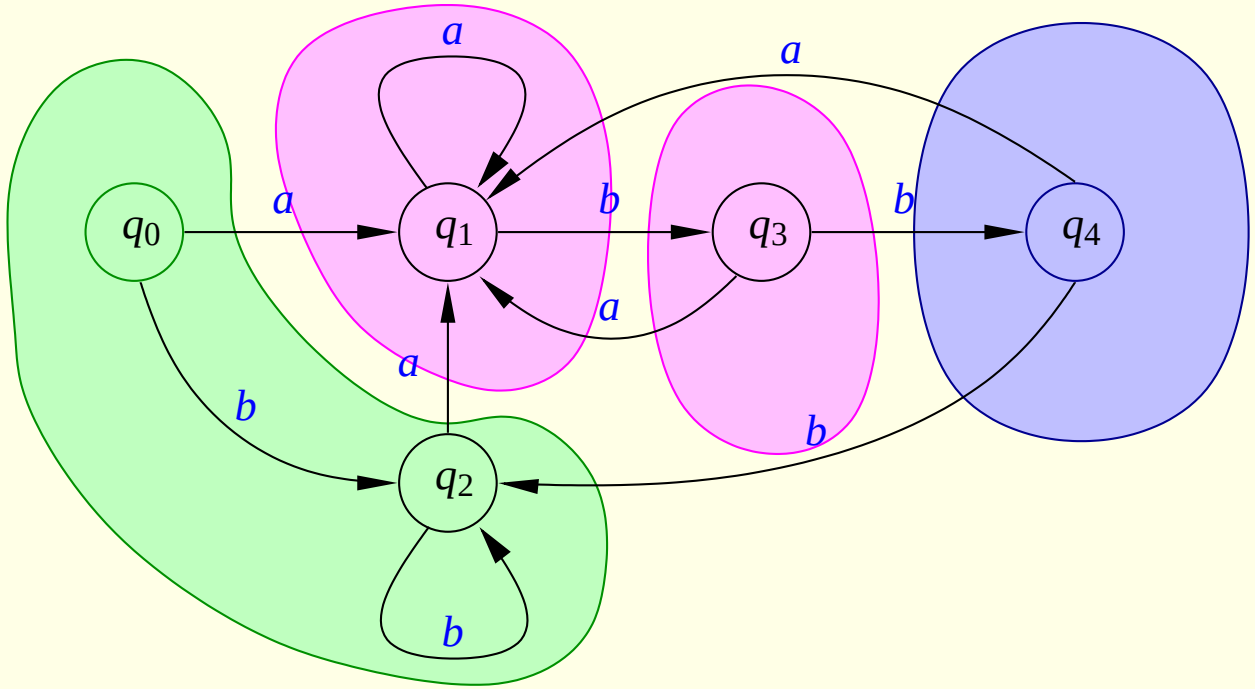
Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:



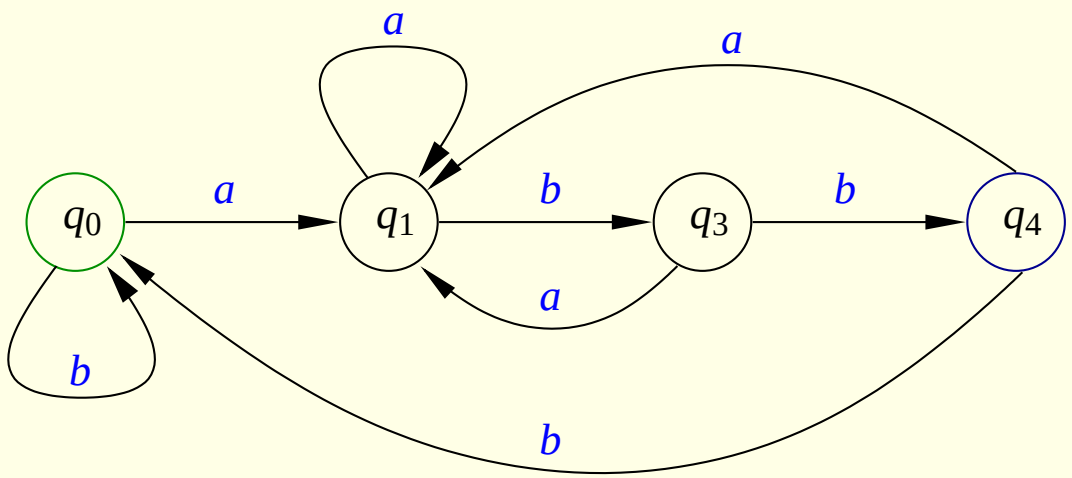
Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:

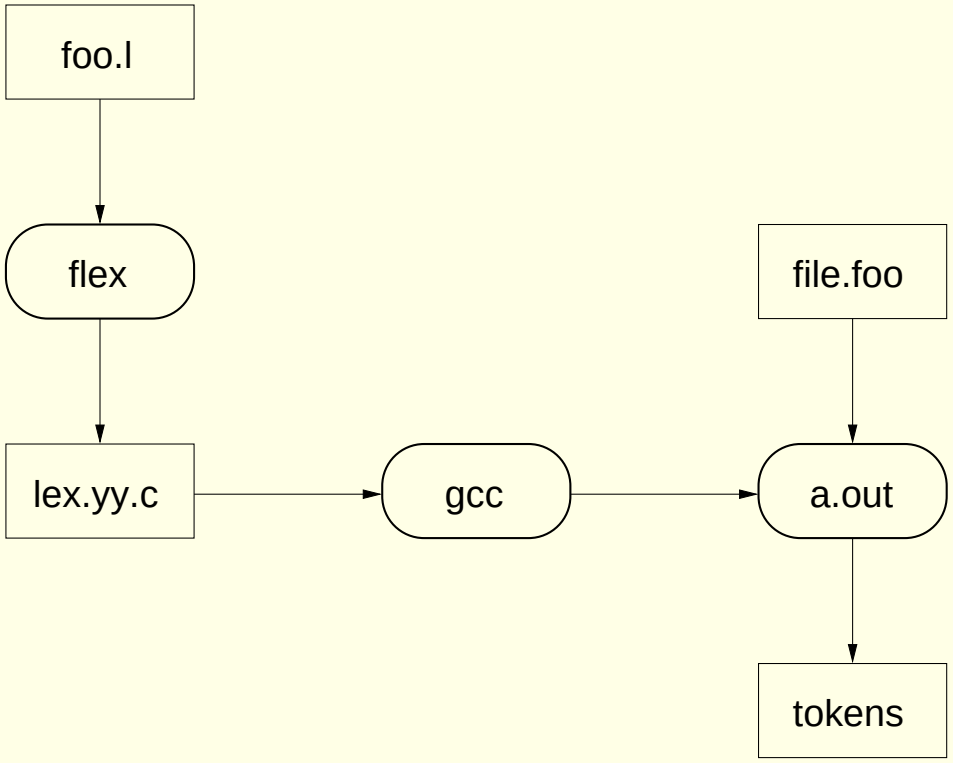


Determineeritud lõpliku automaadi minimiseerimine

Näide — regulaaravaldisele $(a \mid b)^\star abb$ vastava DFA minimiseerimine:



Skannerite generaator Flex



Skannerite generaator Flex

Sisendfaili formaat:

- Flex-i sisendfail koosneb kolmest osast:

```
definitions
%%
rules
%%
user code
```

- Definitsioonide osa koosneb:
  - C kood (kaasatavad päisfailid ja globaalsete muutujate definitsioonid);
  - regulaarsed kirjeldused;
  - algtingimuste definitsioonid.

Skannerite generaator Flex

- Reeglite osa koosneb paaride jadast kujul:

  `pattern  action`

  kus näidis peab algama ilma taandeta ning lõpeb esimese tühisümboliga; aktsioon peab algama näidisega samalt realt.

- Näidis on (laiendatud) regulaaravaldis; aktsioon on suvaline C lause.
  - Kui aktsioon on tühi, siis näidisele vastav sisend eemaldatakse.
  - Kui sisend ei sobi ühegi näidisega, siis ta kopeeritakse.

- Sisendfaili kolmas osa koosneb C koodist, mis kopeeritakse loodavasse faili `lex.yy.c` ilma ühegi muutuseta.
  - Võib puududa, millisel juhul võib ka teise eraldusrea ära jätta.

Skannerite generaator Flex

Liides parseriga suhtlemiseks:

`int yylex(void)`	peafunktsioon; väljastab leitud sõna klassi; kui faililõpp, siis 0
`char *yytext`	viit viimati skaneeritud sõnale
`int yyleng`	viimati skaneeritud sõna pikkus
`FILE *yyin`	vaikimisi loetav sisendfail
`FILE *yyout`	vaikimisi kasutatav väljundfail
`int yywrap(void)`	peaks olema defineeritud kolmandas osas; kui ei ole, siis linkimisel kasutada '`-lfl`'; reeglina väljastab lihtsalt 1
`YYSTYPE yylval`	sümboli väärtust sisaldav struktuur; defineeritud parseris (kaasata päisfail `parser.tab.h`)

Skannerite generaator Flex

Näide:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parse.h"
#include "parse.tab.h"
%}
digit      [0-9]
ident      [a-zA-Z][a-zA-Z0-9]*
whitespace [ \t]+
newline    \n
%%
"("                    {return '(';}
")"                    {return ')';}
"+"                    {return '+';}
"-"                    {return '-';}
"*"                    {return '*';}
```

Skannerite generaator Flex

Näide (järg):

```
"/"                     {return '/';}
"^"                     {return '^';}
"="                     {return '=';}
{digit}+                {yylval.val = atof(yytext);
                         return FLOAT;}
{digit}+"."{digit}*     {yylval.val = atof(yytext);
                         return FLOAT;}
{digit}*"."{digit}+     {yylval.val = atof(yytext);
                         return FLOAT;}
{ident}                 {yylval.str = (char *) malloc(strlen(yytext)+1);
                         strcpy(yylval.str, yytext);
                         return ID;}
{whitespace}            {}
{newline}               {return '\n';}
.                       {fprintf(stderr,
                           "Scanning error!  Unexpected character \"%s\".\n",
                         exit(1);}
```