قسمت دوم – تحلیل لغوی

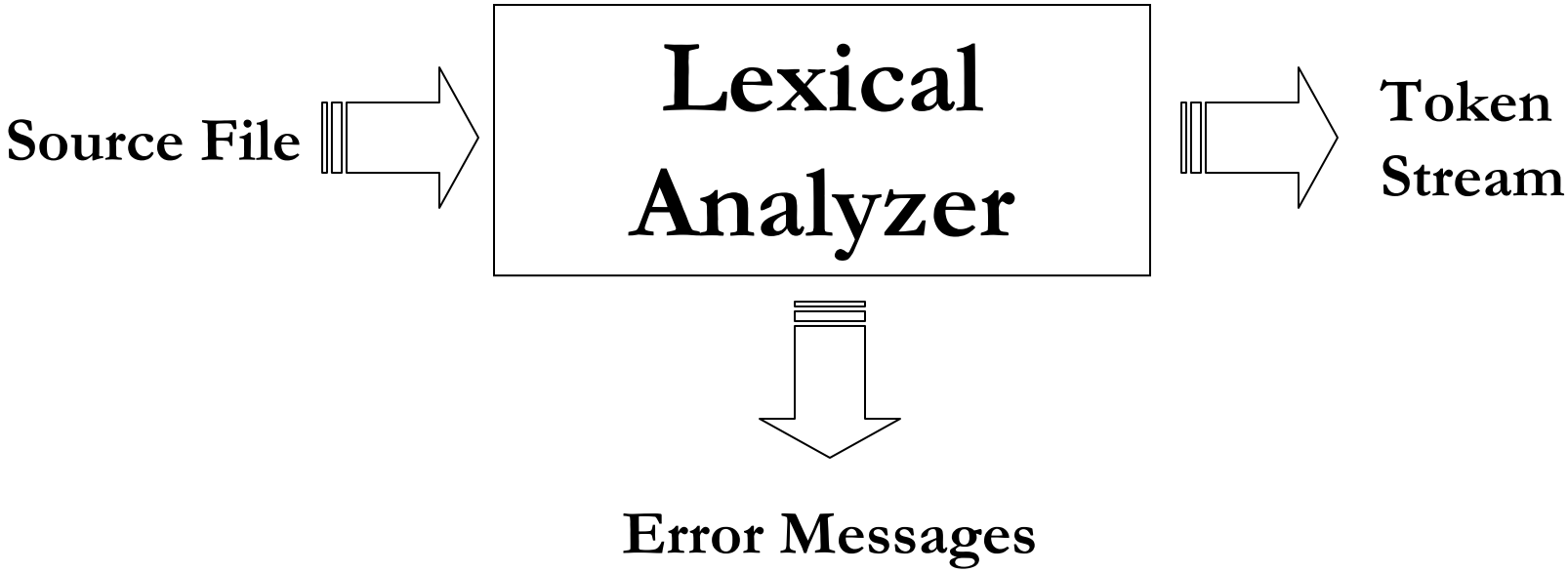
احمد خادم زاده (khademzadeh@mshdiau.ac.ir)
دانشگاه آزاد مشهد

اصول طراحی کامپایلر

مفهوم کلی تحلیل لغوی (۱)

- scanning – lexical analysis
- نشانه ها در یک زبان (tokens)
  - Constants
    - Integer ، Double، Char و String و ...
  - عملگرها
    - محاسباتی، رابطه ای و منطقی
  - نمادهای دستوری و نقطه گذاری
  - کلمات کلیدی
- کلمه (lexeme)

مفهوم کلی تحلیل لغوی (۲)



مفهوم کلی تحلیل لغوی (۳) – مثال

```
float match0(char *s) /* find a zero */ {
    if (!strncmp(s, "0.0", 3)) return 0.;
}
```

- نشانه های موجود در این قطعه برنامه

CHAR	LPAREN	ID(match0)	FLOAT
LBRACE	RPAREN	ID(s)	STAR
ID(strncmp)	BANG	LPAREN	IF
STRING(0.0)	COMMA	ID(s)	LPAREN
RPAREN	RPAREN	NUM(3)	COMMA
RBRACE	SEMI	REAL(0.0)	RETURN
			EOF

مفهوم کلی تحلیل لغوی (٤)

- خطاهای قابل شناخت توسط تحلیلگر لغوی
  - تعداد کم
  - نماد غیر مجاز یا ناشناخته
  - وجود خطا در یک ثابت رشته ای
  - وجود کاراکتر غیر مجاز
- مثال

fload { xy if swithch name[3] == ;

- وظایف کلی دیگر آن نادیده گرفتن کاراکترهای فضای خالی و توضیحات و گاهی جایگزینی ماکرو ها می باشد.
  - جایگزینی ماکروها اغلب وظیفهٔ بخشی با نام پیش پردازنده (preprocessor) است.

شیوهٔ غیر رسمی برای ساخت تحلیل گر لغوی

- در این روش از حلقهٔ while و switch استفاده می کنیم.
- این کار را برای یک زبان نمونه انجام می دهیم که در این زبان کلمات کلیدی بصورت کامل با حروف بزرگ و شناسه ها با حروف کوچک نوشته می شوند.
- در تحلیل گر لغوی گاهی نیاز به کاراکتر پیش بینی (look ahead) داریم.
- بجز در مواردی که طول نشانه مشخص باشد، برای فهمیدن رسیدن به انتهای نشانه، باید یک کاراکتر بیشتر بخوانیم.

شیوهٔ غیر رسمی برای ساخت تحلیل گر لغوی

```
#define T_SEMICOLON ';'
#define T_LPAREN '('
#define T_RPAREN ')'
#define T_ASSIGN '='
#define T_DIVIDE '/'
...
#define T_WHILE 257 // reserved words
#define T_IF 258
#define T_RETURN 259
...
#define T_IDENTIFIER 268 // identifiers, constants, etc.
#define T_INTEGER 269
#define T_DOUBLE 270
#define T_STRING 271
#define T_END 349 // end of file
#define T_UNKNOWN 350 // unrecognized token
```

```
struct token_t {
    int type;
    union {
        char stringValue[256];
        int intValue;
        double doubleValue;
    } val;
};
int main(int argc, char *argv[]) {
    struct token_t token;
    InitScanner();
    while (ScanOneToken(stdin, &token) != T_END)
        ; // process each token
    return 0;
}
{ static void InitScanner()
    create_reserved_table();
    insert_reserved("WHILE", T_WHILE)
    insert_reserved("IF", T_IF)
    insert_reserved("RETURN", T_RETURN)
    ....
}
```

```
static int ScanOneToken(FILE *fp, struct token_t *token) {
                                        int i, ch, nextch;
                                           ch = getc(fp);
                        while (isspace(ch)) ch = getc(fp);
                                              switch(ch) {
                       case '/': nextch = getc(fp);
          if (nextch == '/' || nextch == '*')
            ; // skip over the comment
                     else ungetc(nextch, fp);
 case ';':case ',':case '=':// ... Single char  tokens
                          token->type = ch;
                                 return ch;
               case 'A': case 'B': case 'C': // ...
              token->val.stringValue[0] = ch;
     for (i = 1; isupper(ch = getc(fp)); i++)
       token->val.stringValue[i] = ch;
                             ungetc(ch, fp);
           token->val.stringValue[i] = '\0';
               token->type = lookup_reserved(
      token->val.stringValue);
                        return token->type;
```

```
                    case 'a': case 'b': case 'c': // ...
                       token->type = T_IDENTIFIER;
                  token->val.stringValue[0] = ch;
          for (i = 1; islower(ch = getc(fp)); i++)
                 token->val.stringValue[i] = ch;
                                ungetc(ch, fp);
               token->val.stringValue[i] = '\0';
if (lookup_symtab(token->val.stringValue) == NULL)
      add_symtab(token->val.stringValue);
                            return T_IDENTIFIER;
          case '0': case '1': case '2': case '3': //....
                        token->type = T_INTEGER;
                  token->val.intValue = ch - '0';
                   while (isdigit(ch = getc(fp)))
token->val.intValue=token->val.intValue*10+ch-'0';
                                ungetc(ch, fp);
                              return T_INTEGER;
                               case EOF: return T_END;
                     default: token->val.intValue = ch;
                        token->type = T_UNKNOWN;
                              return T_UNKNOWN;
                                                  }
                                                    }
```

شیوهٔ دوم ساخت تحلیل گر لغوی

- استفاده از عبارات منظم و FA
- مروری بر عبارات منظم
  - نماد (symbol)
  - الفبا
  - رشته
  - رشته تهی
  - زبان رسمی (formal language) – Σ^*
  - عبارت منظم

عبارات منظم

- عبارات منظم با کمک قواعد بازگشتی زیر تعریف می گردند:
  - هر نماد عضو الفبا یک عبارت منظم است
  - رشته تهی هم یک عبارت منظم است
  - اگر a و b عبارت منظم باشند، آنگاه (a)، ab، a|b و a* هم عبارت منظم هستند.
  - هیچ چیز دیگری عبارت منظم نیست.

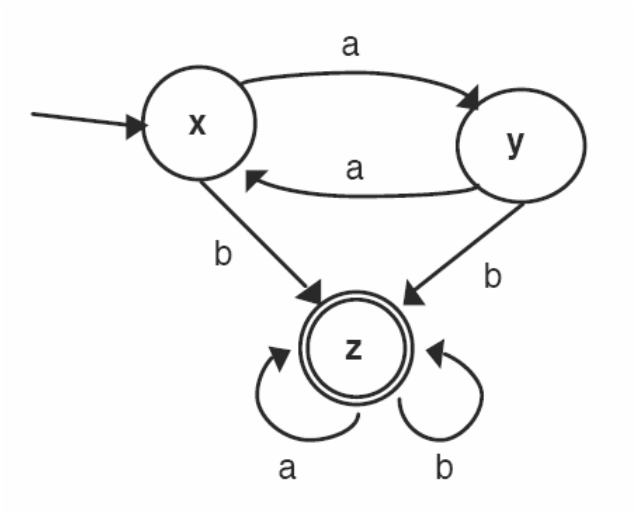
چند سوال

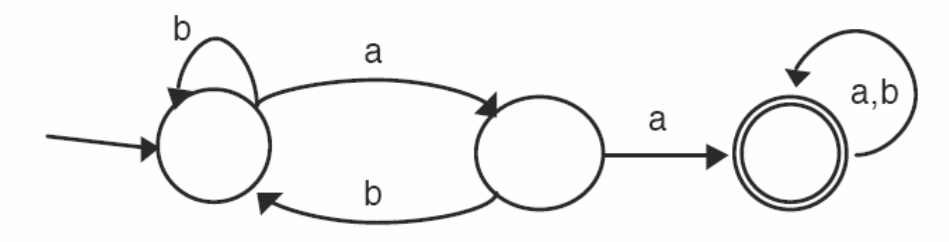
- روی الفبای {a,b} عبارت منظم مربوط به زبان های زیر چیست ؟
  - رشته هائی که با a شروع و با آن خاتمه می یابند.
  - رشته هائی که تعداد فردی a دارند.
  - رشته هائی که در آنها دو کاراکتر a در مجاورت هم قرار ندارند.
- عبارت منظمی که زبان تولید اعداد صحیح مبنای را تعریف می کند چیست ؟

مروری بر ماشین های خودکار محدود

- پس از تعریف عبارت منظم مربوط به تمام نشانه های زبان FA مربوطه را جهت شناخت نشانه ها می سازیم. یک FA
  - تعداد محدودی حالت دارد که یکی از حالات حالت شروع می باشد و تعدادی (شاید هیچ) از حالات، حالت پایانی می باشند.
  - یک الفبا از نمادهائی که ممکن است به عنوان ورودی مورد استفاده قرار گیرند.
  - تعداد محدودی تغییر حالت، که در هر یک، با مشاهدهٔ یک نماد ورودی از یک حالت به حالت دیگری منتقل می شویم.

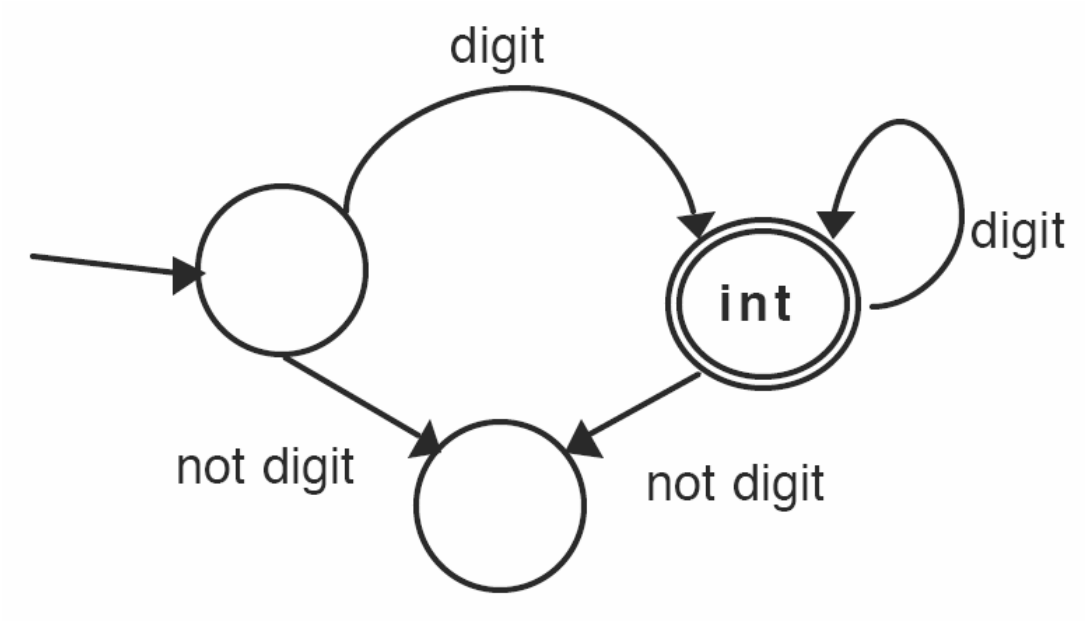
مثالهائی از ماشین خودکار محدود



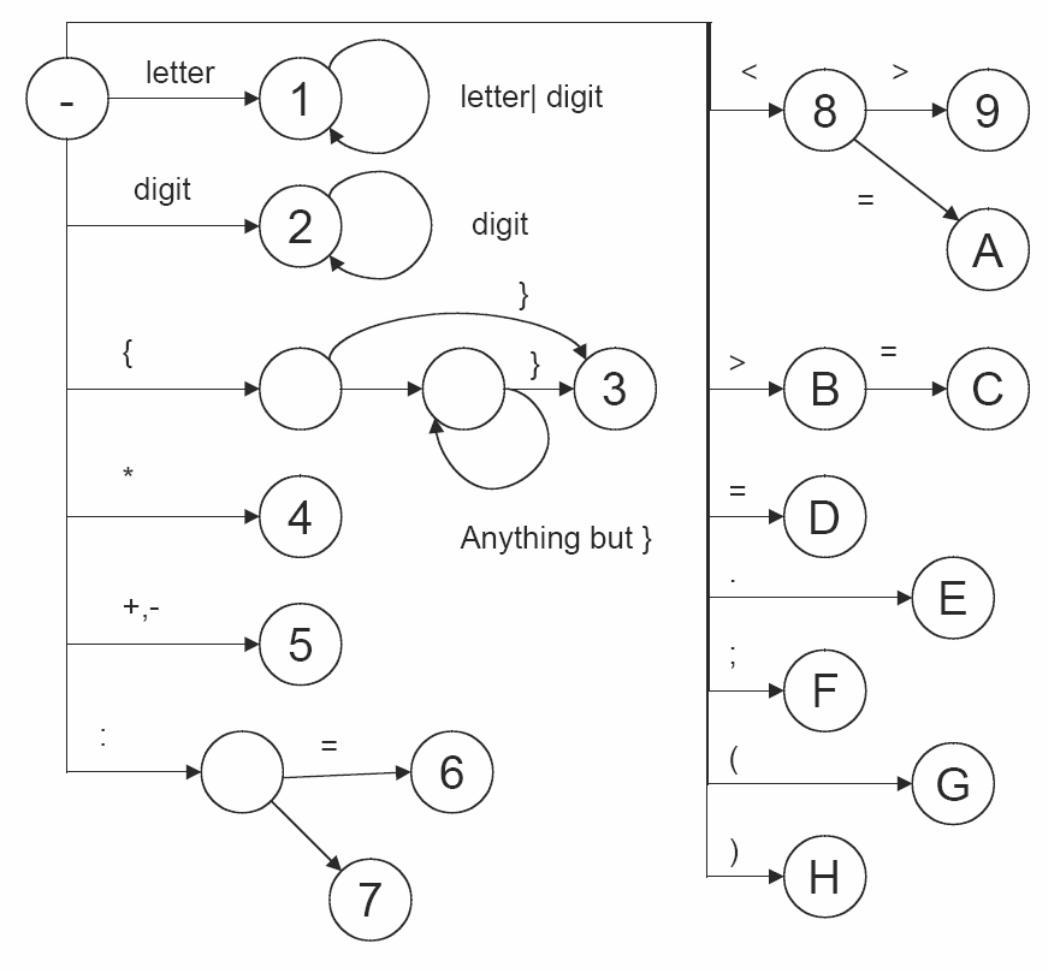


- یک FA تعریف کنید که زبان شامل تمام رشته هائی که با b ختم شده و شامل زیر رشتهٔ aa نیستند را می پذیرد. عبارت منظم متناظر با آن چیست؟

مثال دیگری از یک FA



مثالی از یک FA که زیر مجموعه ای از نشانه های زبان پاسکال را می پذیرد



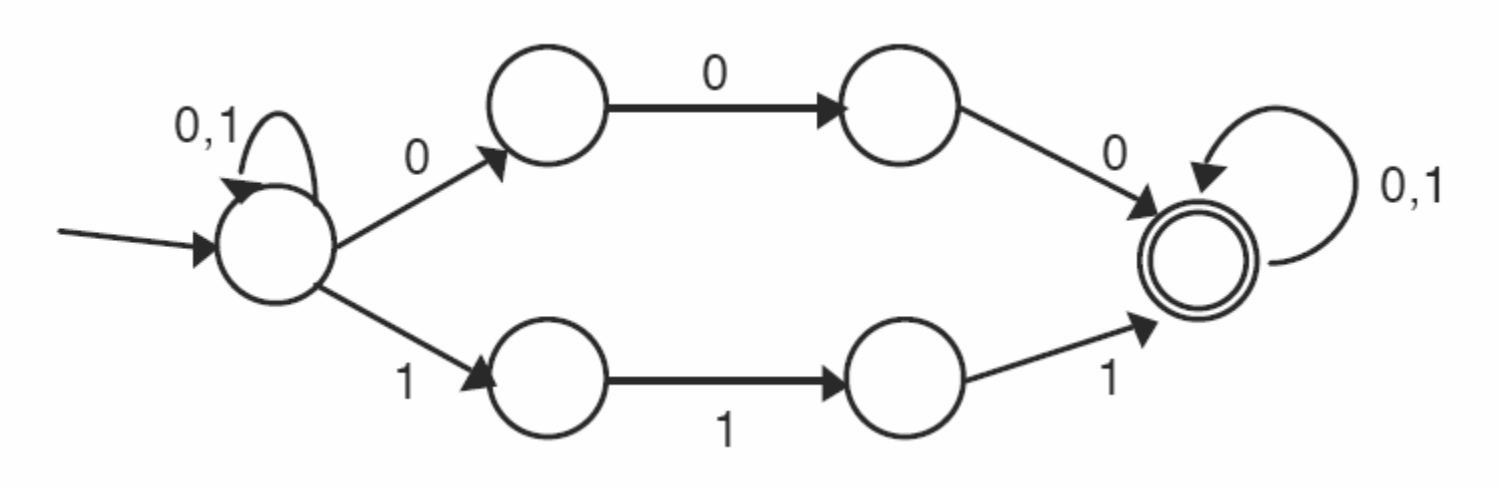
چگونگی عمل در تحلیل گر لغوی مبتنی بر FA

- ابتدا تحلیل گر در حالت شروع قرار دارد.
- ورودی بصورت کاراکتر به کاراکتر خوانده می شود.
- بازای هر ورودی و با توجه به حالت فعلی حالت تغییر می کند.
- اگر در حالت نهائی قرار گرفتیم، عمل متناظر با آن حالت نهائی را انجام می دهیم.
- پس از انجام قدم قبلی، کاراکتر اضافی خوانده شده را به ورودی بر می گردانیم. و حالت FA را، حالت شروع قرار می دهیم.
- در صورتی که در حالت نهائی قرار نگیریم یا به کاراکتری که در یک حالت، انتظار آن را نداریم، برخورد کردیم، خطا گزارش می کنیم.

تبدیل RE به FA

- یک NFA دارای
  - تعداد محدودی حالت و یک حالت شروع و تعدادی (شاید هیچ) حالت نهائی
  - الفبائی از نمادهای ورودی ممکن
  - تعداد محدودی انتقال حالت که با الفبا برچسب گذاری شده اند (به جز نماد تهی)
  - امکان انتقال حالت از یک حالت خاص، با یک نماد از الفبا به چندین حالت ، امکان پذیر می باشد.

مثالی از NFA



*(0 | 1) (000 | 111) * (1 | 0)

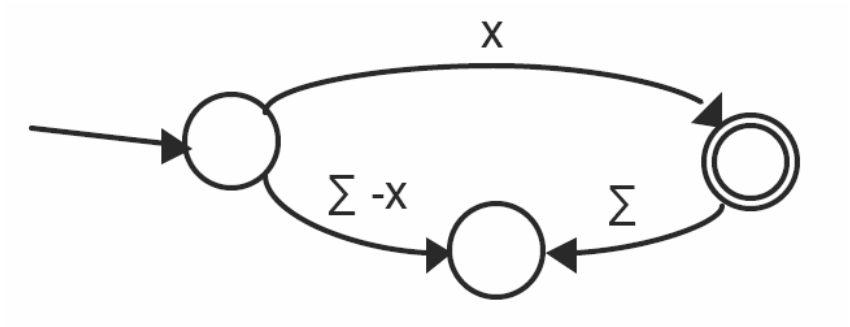
- چگونگی پذیرش رشتهٔ 111 توسط این ماشین

ε-NFA

- در این نوع ماشین، لبه های انتقال می توانند با ε نیز برچسب گذاری شده باشند.
- Kleene دریک قضیه نشان داده است که قدرت DFA، NFA و ε-NFA ها برای تولید زبانهای مربوط به عبارات منظم با هم برابر است.
- تعبیر دیگر قضیه Kleene : اگر R یک عبارت منظم و L زبان متناظر با آن باشد، یک FA وجود دارد که رشته های مربوط به این زبان را شناسائی کند و بالعکس.

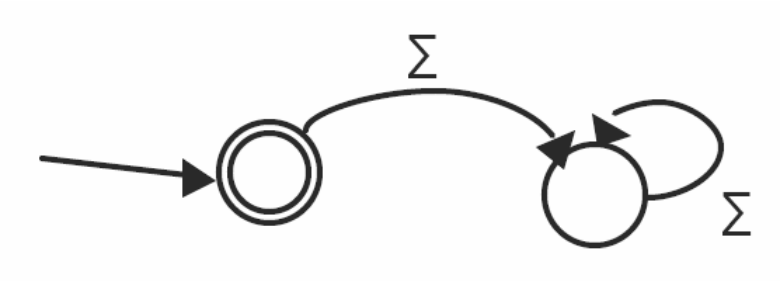
تبدیل RE به ε-NFA

- به این منظور از قواعد Thompson استفاده می کنیم.
- قاعدهٔ شمارهٔ یک : برای پذیرش هر یک از نمادهای الفبای ورودی یک NFA وجود دارد.



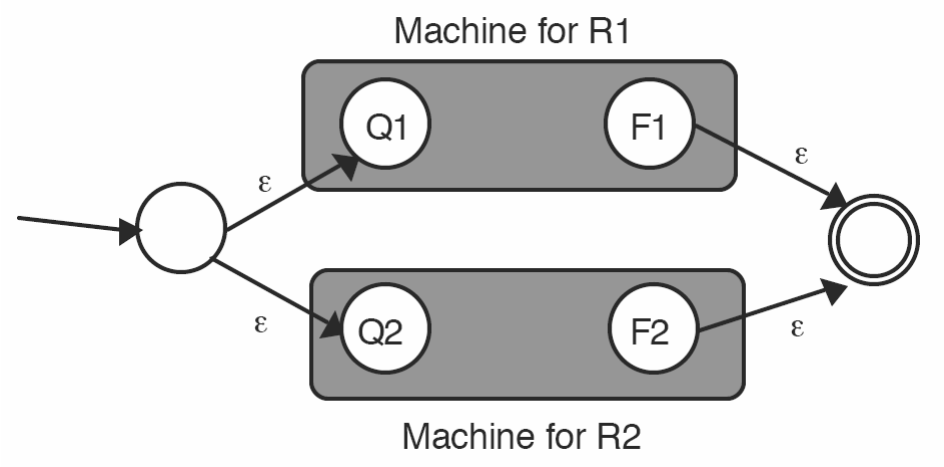
قاعدهٔ (۲) تامپسون

- NFA ای وجود دارد که فقط ε را می پذیرد.



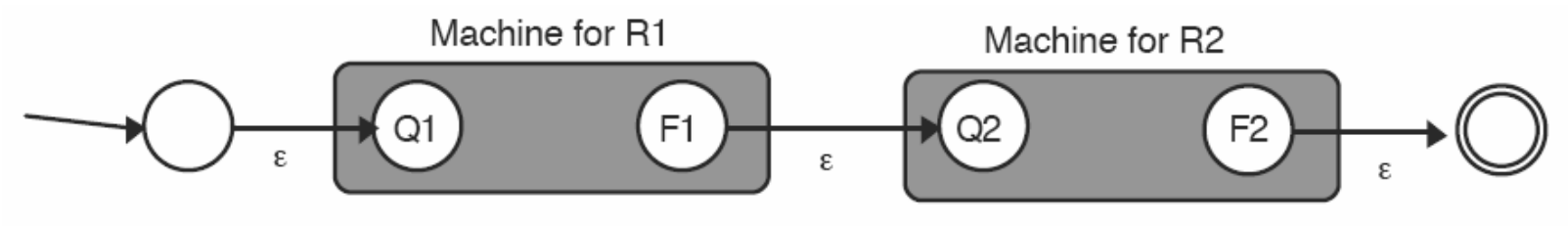
قاعدهٔ (۳) تامپسون

- اگر NFA های مربوط به R1 و R2 را داشته باشیم، NFA ای وجود دارد که R1 | R2 را بپذیرد.



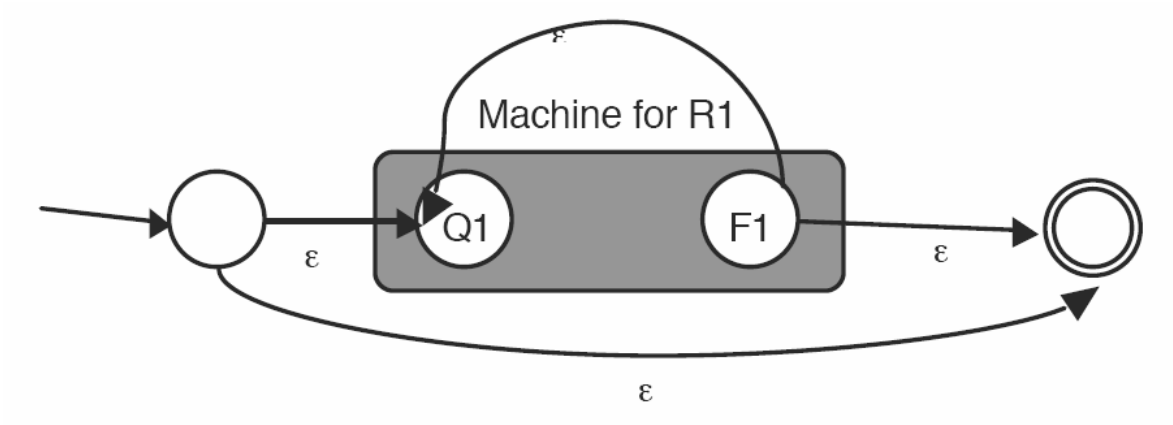
قاعدهٔ (٤) تامپسون

- NFA ای برای پذیرش R1R2 وجود دارد.



قاعدهٔ (۵) تامپسون

- NFA ای برای پذیرش R1* وجود دارد.

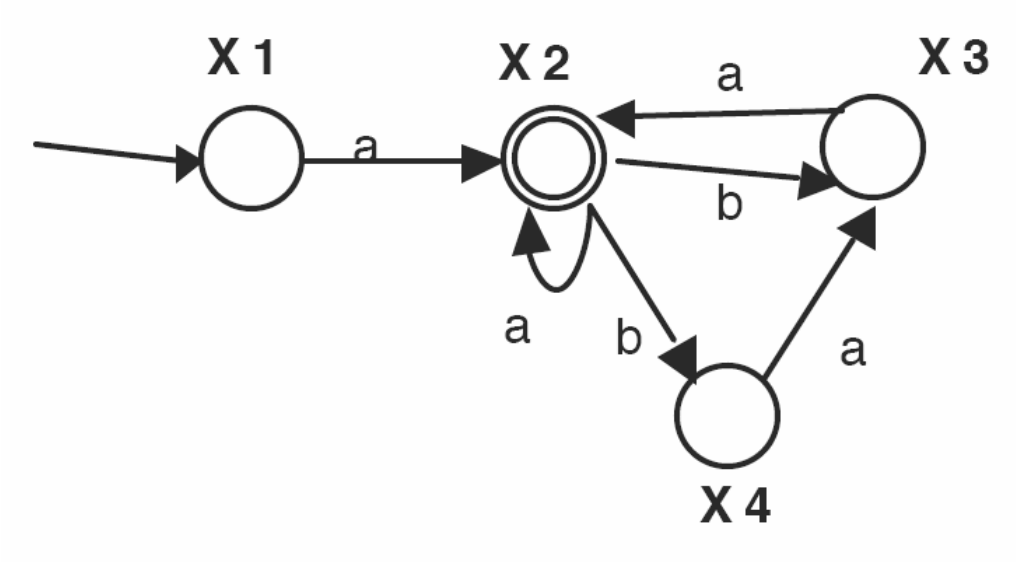


تبدیل یک NFA به DFA با ایجاد زیرمجموعه ها

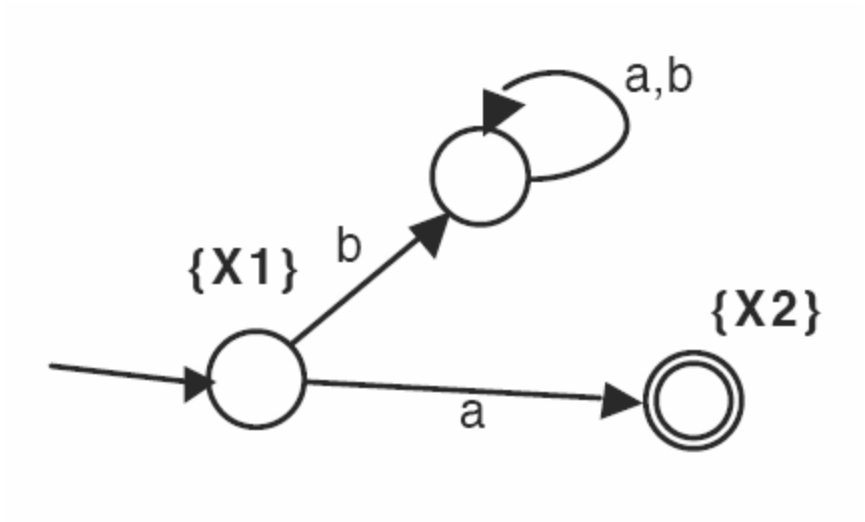
- ساخت یک DFA که معادل با یک NFA باشد. یعنی تمام رشته هائی که توسط NFA پذیرفته می شوند را، بپذیرد.
- هر یک از حالات DFA ساخته شده، شامل چندین (شاید یک) حالت NFA اولیه می باشد.
- حالت شروع DFA، همان حالت شروع NFA می باشد.
- حالت پایان DFA
- حداکثر تعداد حالات ممکن DFA

مثال (۱)

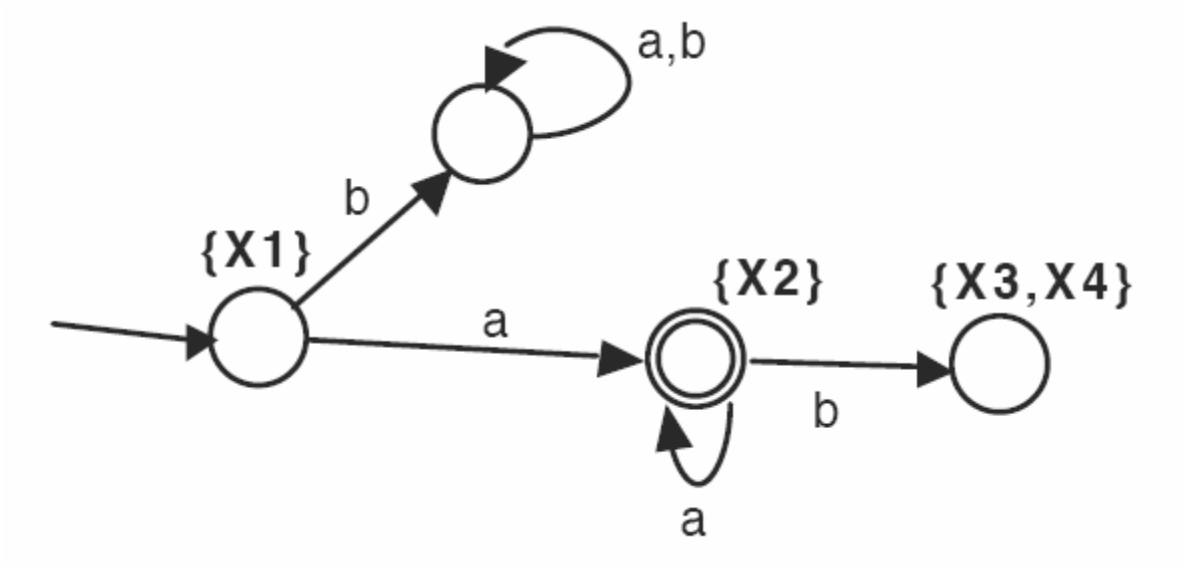
- ماشین زیر را قطعی می کنیم:



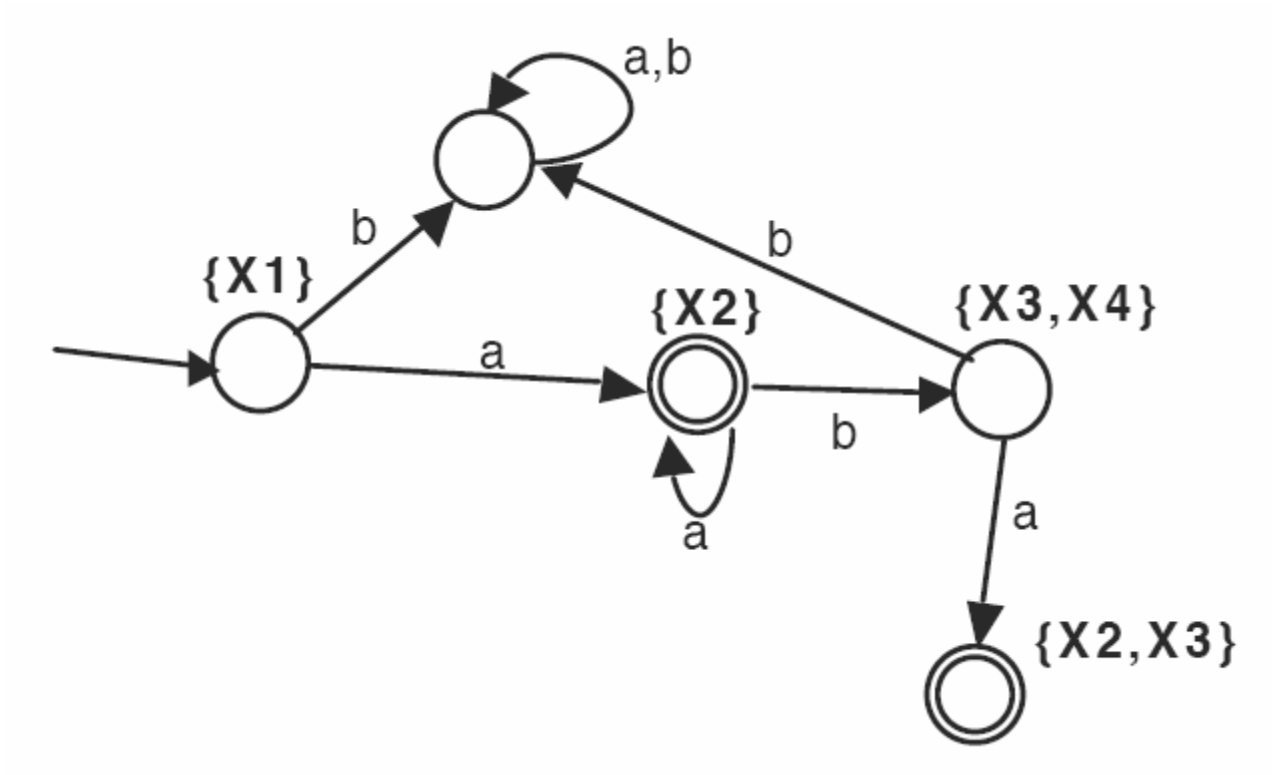
مثال (۲)



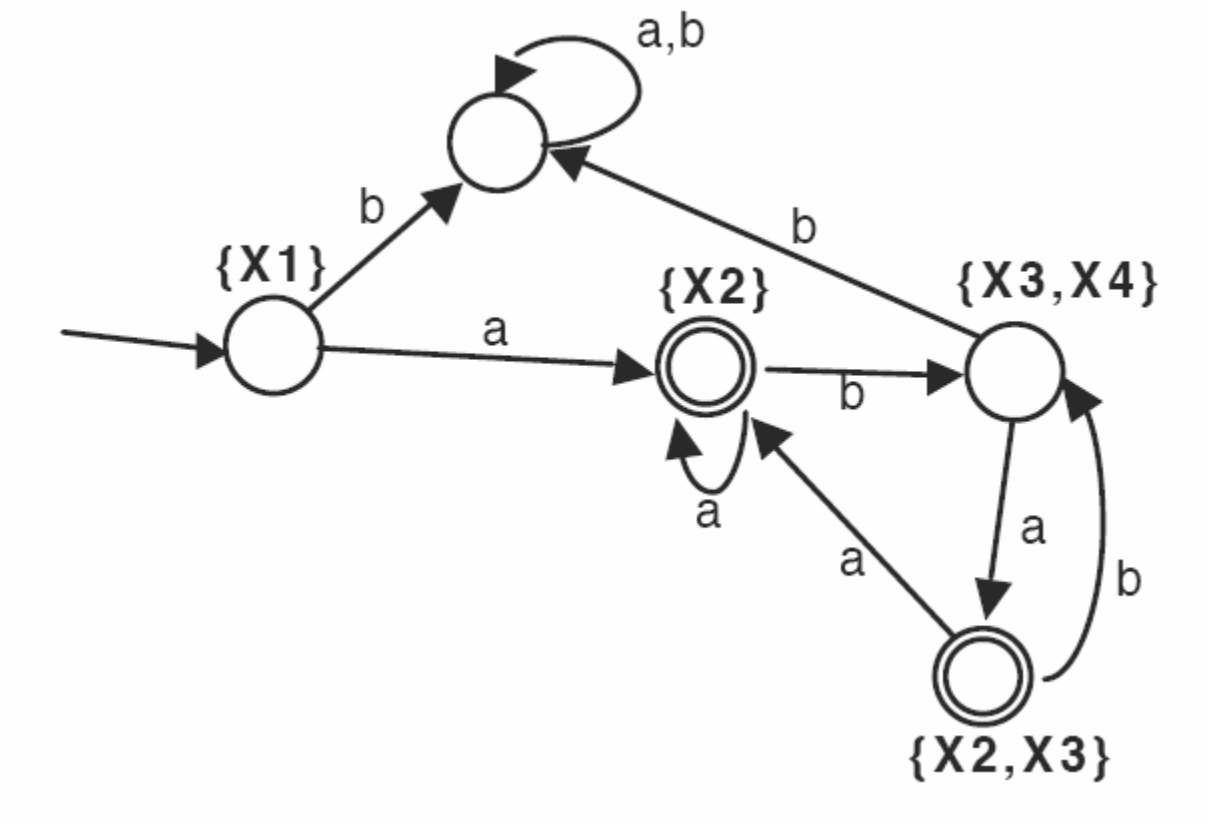
مثال (۳)



مثال (٤)



مثال (۵)



- بهینه سازی DFA
- نوشتن عبارت منظم معادل با یک FA