CHƯƠNG TRÌNH DỊCH

Bài 4: Phân tích từ vựng bằng DFA

Nội dung

1. Bộ phân tích từ vựng cho ngôn ngữ A
2. Automat hữu hạn (FA)
   1. Đồ thị chuyển (TD)
   2. Automat hữu hạn không đơn định (NFA)
   3. Automat hữu hạn đơn định (DFA)
3. Chuyển đổi biểu thức chính quy sang DFA
   1. Chuyển đổi từ biểu thức chính quy sang NFA
   2. Chuyển đổi từ NFA sang DFA
   3. DFA tối ưu cho phân tích từ vựng
4. Bộ phân tích từ vựng dựa trên DFA
5. Bài tập

Phần 1

Bộ phân tích từ vựng cho ngôn ngữ A

Ngôn ngữ A

Ngôn ngữ lập trình A chuyên thực hiện các phép toán số

1. Mỗi lệnh viết trên 1 dòng
2. Lệnh bao giờ cũng có dạng <biến> = <biểu thức>
3. <biến> là một tên riêng, không cần khai báo trước, giống quy cách tên biến thông dụng, biến luôn là số
4. <biểu thức> được viết theo quy cách biểu thức số học, có thể gồm:
   - Số nguyên, số thực, biến
   - Lời gọi hàm toán học thông dụng: sqrt, log, exp, power,...
   - Các phép toán thông dụng + - * / %
   - Các cặp ngoặc tròn

Bộ PTTV đơn giản (mã giả C#)

```
// chứa thông tin về một từ tố
class Word {
    public int wordType;          // chứa từ loại của từ
    public string wordContent;    // chứa nội dung của từ
}
// bộ phân tích từ vựng
class PTTV {
    // phân tích chuỗi S thành dãy các từ tố
    public List<Word> process(string S) { … }
    // lấy ra từ tố tiếp theo
    Word getNextWord() { … }
}
```

Bộ PTTV cho ngôn ngữ A

```
using System;
using System.Collections.Generic;

// định nghĩa các từ loại của bộ PTTV
enum WordType {
    TYPE_EOF,         // loại kết thúc đầu vào
    TYPE_ERROR,       // loại đầu vào lỗi
    TYPE_SPACE,       // dấu trống, tab,...
    TYPE_VAR,         // tên biến
    TYPE_INTEGER,     // số nguyên
    TYPE_OPERATOR,    // phép toán
    TYPE_PARETHESIS   // ngoặc
}
```

Bộ PTTV cho ngôn ngữ A

```
// lớp chứa thông tin của 1 từ
class Word {
    public WordType wordType;                    // chứa từ loại của từ
    public string wordContent;                   // chứa nội dung của từ
    public Word(WordType t, string c) {
        wordType = t; wordContent = c;
    }
}
// lớp automata thực hiện PTTV
class PTTV {
    string input;
    int pos;
```

Bộ PTTV cho ngôn ngữ A

```
public List<Word> process(string a) {
    List<Word> list = new List<Word>();
    pos = 0;
    do {
        Word x = getNextWord();
        list.Add(x);
        if (x.wordType == WordType.TYPE_EOF) break;
        if (x.wordType == WordType.TYPE_ERROR) break;
        pos += x.wordContent.Length;
    } while (true);
    return list;
}
```

Bộ PTTV cho ngôn ngữ A

```
// lấy ra từ tiếp theo
Word getNextWord() {
    Word x = new Word(WordType.TYPE_EOF, "");
    if (pos >= input.Length) return x;
    x = nextIsSpace();
    if (x != null) return x;
    x = nextIsOperator();
    if (x != null) return x;
    ...
    x = new Word(WordType.TYPE_ERROR, "");
    return x;
}
```

Bộ PTTV cho ngôn ngữ A

```
    // từ tiếp theo có phải kí hiệu trống, tab,...?
    Word nextIsSpace() {
        if (input[pos]==' ') return new Word(WordType.TYPE_SPACE, " ");
        return null;
    }
    // từ tiếp theo có phải kí hiệu phép toán?
    Word nextIsOperator() {
        if ((input[pos]=='+') || (input[pos]=='-') ||
            (input[pos]=='*') || (input[pos]=='/') || (input[pos]=='%'))
            return new Word(WordType.TYPE_OPERATOR, "" + input[pos]);
        return null;
    }
}
```

Bộ PTTV cho ngôn ngữ A

```
// hàm chính thử nghiệm bộ PTTV
class MyApp {
    public static void Main() {
        PTTV scanner = new PTTV();
        List<Word> x = scanner.process("=+1");
        foreach (Word w in x)
            Console.WriteLine("{0}: {1}", w.wordType, w.wordContent);
    }
}
```

Bộ PTTV cho ngôn ngữ A

- Đoạn mã trên minh họa cách thức một bộ phân tích từ vựng làm việc
- Được xây dựng hoàn toàn "bằng tay"
  - Chạy chậm
  - Không uyển chuyển
  - Dễ hiểu và mở rộng thêm từ loại
  - Xử lý nhập nhằng rất phức tạp
  - Có tính tham khảo hơn là tính ứng dụng thực tế

Phần 2

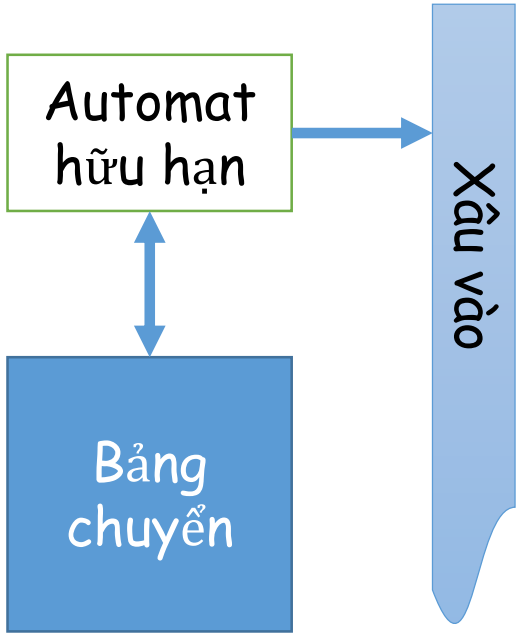
Automat hữu hạn (FA)

Automat hữu hạn (FA)

- Nhận xét về bộ PTTV đơn giản ở phần trước:
  - Cấu trúc chương trình đơn giản, dễ hiểu
  - Dễ mở rộng nếu bổ sung các từ loại mới
  - Hoạt động chậm, mỗi từ loại được thử đoán nhận một lần; trường hợp tệ nhất (có lỗi) có độ phức tạp cao vì phải thử tất cả các từ loại
  - Khó xử lý nhập nhằng
- Trong phần tiếp theo, chúng ta sẽ thảo luận một thiết kế mới khắc phục vấn đề tốc độ dựa trên ý tưởng xây dựng bộ đoán nhận chỉ với một lần thử duy nhất

Automat hữu hạn (FA)

- Automat hữu hạn (finite-state automaton) dùng để đoán nhận lớp ngôn ngữ chính quy (regular expression)
- Cấu trúc cơ học của FA gồm:
  - Bảng chuyển
  - Đầu đọc
  - Xâu vào
- Quá trình hoạt động:
  - Bắt đầu từ trạng thái xuất phát
  - Đọc từ kí tự từ xâu vào
  - Quan sát bảng chuyển để biết sẽ chuyển sang trạng thái nào
  - Dừng khi kết thúc xâu vào và trả về trạng thái đoán nhận



Automat hữu hạn (FA)

- Hoạt động của automat hữu hạn rất đơn giản:
  - Mỗi bước đọc một kí tự từ đầu vào
  - Từ trạng thái bắt đầu, dựa trên kí tự đầu vào biến đổi trạng thái, quá trình này kết thúc khi đến trạng thái dừng
  - Trạng thái dừng sẽ quyết định từ loại mà FA đoán nhận được (bao gồm cả lỗi)
- Dễ thấy độ phức tạp tính toán của thuật toán đoán nhận là tuyến tính theo độ dài của dữ liệu đầu vào (vì mỗi bước chuyển nhận một kí tự đầu vào)
- Vấn đề chính của automat hữu hạn: làm thế nào xây dựng được bảng chuyển hiệu quả?

Automat hữu hạn (FA)

- Automat hữu hạn được chia làm 2 loại:
  - Automat hữu hạn đơn định (deterministic finite automata – DFA)
    - Với một kí hiệu đầu vào, chỉ có thể chuyển sang tối đa một trạng thái thái tiếp theo (hoặc dừng và báo lỗi)
    - Không chấp nhận kí hiệu đầu vào là ε
  - Automat hữu hạn không đơn định (non-deterministic finite automata – NFA)
    - Chấp nhận kí hiệu đầu vào là ε
    - Với một kí hiệu đầu vào, có thể chuyển sang nhiều trạng thái tiếp theo
  - Hai loại automat này tương đương về khả năng đoán nhận ngôn ngữ và có thể chuyển đổi qua lại lẫn nhau

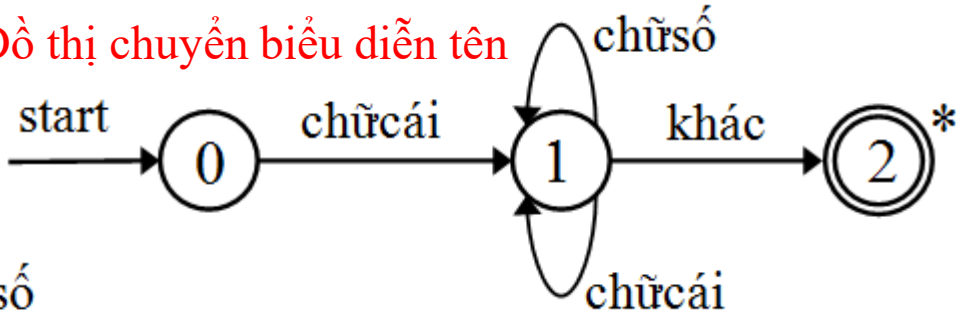
Phần 2.1

Đồ thị chuyển (TD - transition diagram)

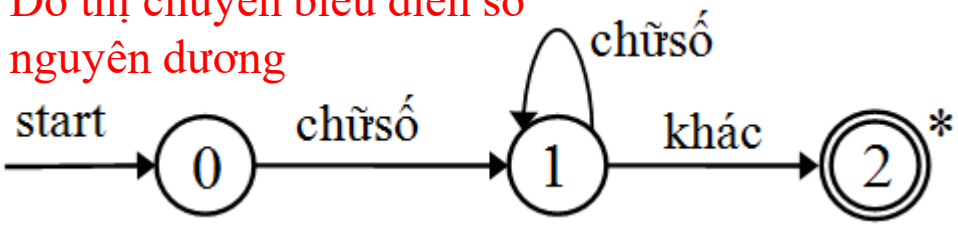
Đồ thị chuyển

Đồ thị chuyển là phương pháp thường sử dụng để mô tả một cách trực quan sơ đồ hoạt động của các automat hữu hạn

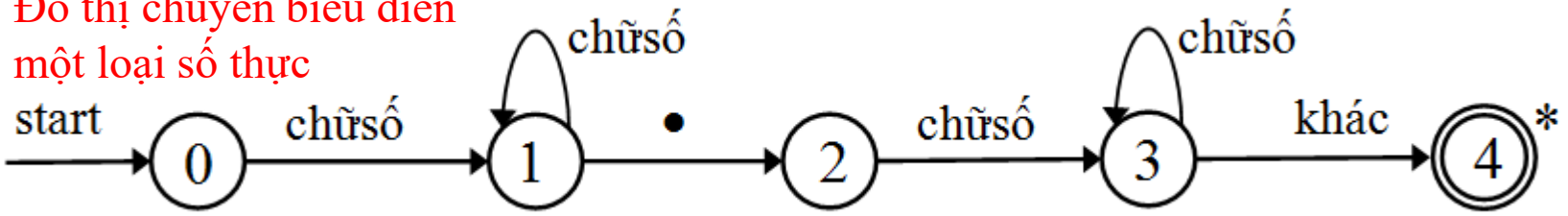
Đồ thị chuyển biểu diễn tên



Đồ thị chuyển biểu diễn số nguyên dương



Đồ thị chuyển biểu diễn một loại số thực



Các kí hiệu của đồ thị chuyển

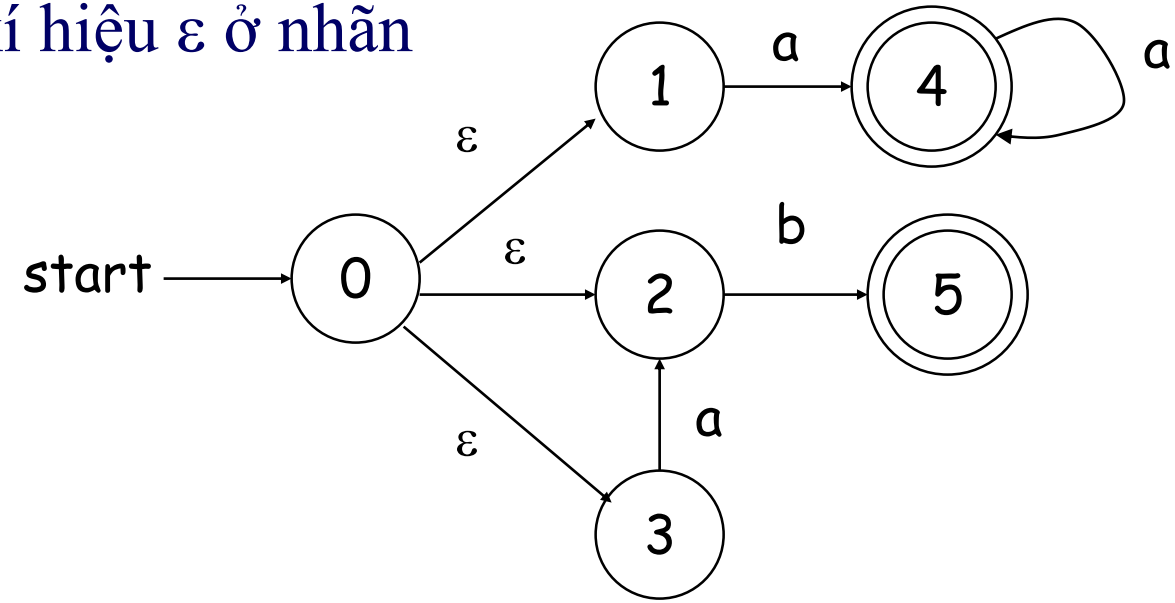
- Trạng thái: vẽ bởi vòng tròn, kí hiệu ghi bên trong là "tên" (số hiệu) của trạng thái đó
  - Trạng thái kết thúc: vòng tròn kép
  - Trạng thái kết thúc có đánh dấu sao (*): ký tự cuối cùng không thuộc vào từ tố được đoán nhận
- Bước chuyển: vẽ bởi mũi tên nối tới trạng thái sẽ chuyển đến, kí hiệu ghi bên cạnh là "nhãn" của bước chuyển
  - Nhãn ghi các kí tự hoặc loại kí tự cho phép thực hiện bước chuyển
  - Nhãn "start": xác định trạng thái bắt đầu của automat

Đồ thị chuyển của một NFA

Xét ngôn ngữ chính quy L = aa* | b | ab

Ta có thể xây dựng đồ thị chuyển nhận biết L có các đặc trưng của NFA:

- Từ một trạng thái có thể có nhiều bước chuyển tương tự
- Chứa kí hiệu ε ở nhãn

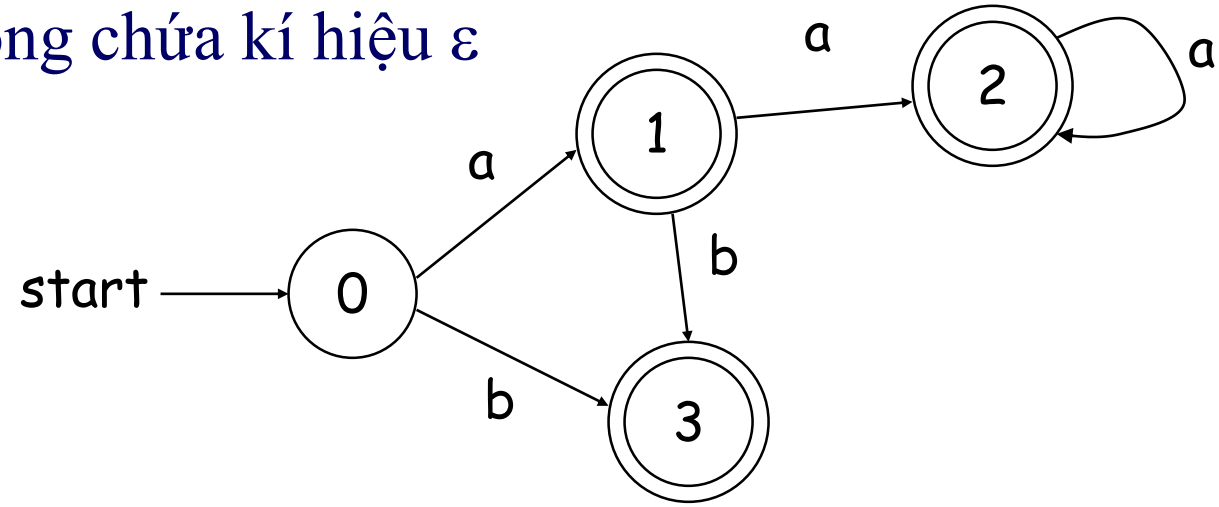


Đồ thị chuyển của một DFA

Cũng vẫn với ngôn ngữ L = aa* | b | ab

Ta có thể xây dựng đồ thị chuyển nhận biết L có các đặc trưng của DFA:

- Từ một trạng thái không thể có các bước chuyển tương tự nhau (nhãn giống nhau)
- Nhãn không chứa kí hiệu ε



Phần 2.2

Automat hữu hạn không đơn định (NFA)

Mô hình toán học của NFA

- Một automat hữu hạn không đơn định (NFA) là mô hình toán học gồm:
  1. Một *tập trạng thái* $\boldsymbol{S}$
  2. Một *tập ký hiệu vào* $\boldsymbol{\Sigma}$ (bảng ký hiệu vào)
  3. Một *hàm chuyển* ***move*** ánh xạ cặp (trạng thái, ký hiệu) tới tập các trạng thái
  4. Một trạng thái $\boldsymbol{s_0}$ đặc biệt gọi là *trạng thái bắt đầu* (hoặc *trạng thái khởi tạo*)
  5. Một tập các trạng thái $\boldsymbol{F}$ đặc biệt gọi là *các trạng thái chấp thuận* (hay *các trạng thái kết thúc*)
- NFA không có ràng buộc nào về các thành phần

Cài đặt NFA trên máy tính

- Có nhiều cách mã hóa NFA trên máy tính, phương pháp được sử dụng nhiều nhất là dùng bảng chuyển
- Bảng chuyển là một ma trận 2 chiều:
  - Các dòng thể hiện trạng thái của NFA
  - Các cột thể hiện kí hiệu đầu vào
  - Bảng ghi các trạng thái chuyển tới

State	Input Symbol	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

- Hai cản trở lớn khi mã hóa NFA:
  - Xử lý kí hiệu ε thế nào?
  - Xử lý như thế nào khi có nhiều phương án dịch chuyển ứng với một kí hiệu đầu vào?

Phần 2.3

Automat hữu hạn đơn định (DFA)

Automat hữu hạn đơn định

- Lớp automat hữu hạn đơn định (DFA) là lớp các NFA thỏa mãn các ràng buộc sau:
  - Không có trạng thái nào có dịch chuyển ε
  - Với mỗi trạng thái s và ký hiệu đầu vào a, có nhiều nhất một cạnh nhãn a rời khỏi s
    - Nói cách khác, hàm move(s,a) là hàm đơn trị, đây chính là ý nghĩa của chữ "đơn định" trong DFA
- Như vậy ta thấy DFA là NFA nhưng đã loại bỏ đi những chi tiết khó lập trình nhất
  - Một điều khá đặc biệt, khả năng đoán nhận của DFA và NFA là tương đương

Mô phỏng hoạt động của DFA

```
// đầu vào: chuỗi x kết thúc bởi kí hiệu eof
// đầu ra: yes nếu chấp thuận x, no nếu ngược lại
s := s0;
c := nextchar(x);
while c ≠ eof do
    s := move(s, c);
    c := nextchar(x);
end;
if s ∈ F then return “yes”;
else return “no”;
```

Phần 3

Chuyển đổi biểu thức chính quy sang DFA

Chuyển đổi BTCQ sang DFA

- Như vậy nếu có một DFA phù hợp, ta có thể xây dựng bộ PTTV chỉ với độ phức tạp tuyến tính một cách dễ dàng
- Hầu hết các định nghĩa từ vựng đều viết dưới dạng biểu thức chính quy (RE – regular expression), vậy làm thế nào có được DFA từ các RE đã có?
- Trong phần sau ta sẽ xem xét các bước chuyển đổi từ các RE thành DFA thông qua việc xây dựng NFA tương đương và tối ưu trạng thái

Phần 3.1

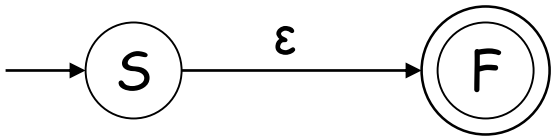
Chuyển đổi từ biểu thức chính quy sang NFA

Thuật toán Thompson

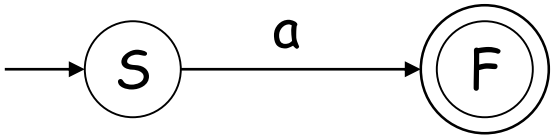
- Kenneth "Ken" Thompson (đồng tác giả của hệ điều hành Unix, ngôn ngữ lập trình Go), năm 1968, đã phát triển một thuật toán (Thompson's construction algorithm) để chuyển đổi từ biểu thức chính quy sang NFA, thuật toán gồm 3 bước:
  1. Phân rã biểu thức chính quy thành các thành phần cơ bản (loại bỏ các yếu tố khó xây dựng NFA)
  2. Xây dựng NFA cho các thành phần cơ bản
  3. Ghép các NFA trong bước 2 thành một NFA lớn
- Ngược lại, thuật toán Kleene (Kleene's algorithm) chuyển từ NFA (DFA) thành biểu thức chính quy

Thuật toán Thompson

- Bước 1: đơn giản hóa biểu thức chính quy
  - M+ được chuyển đổi thành M M*
  - M? được chuyển đổi thành M | ε
  - Như vậy kết thúc bước này biểu thức chính quy chỉ gồm các kí hiệu, phép chọn (|), phép nối (viết liên tiếp) và phép lặp (*)
- Bước 2: xây dựng NFA cho các kí hiệu cơ bản
  - NFA cho kí hiệu rỗng

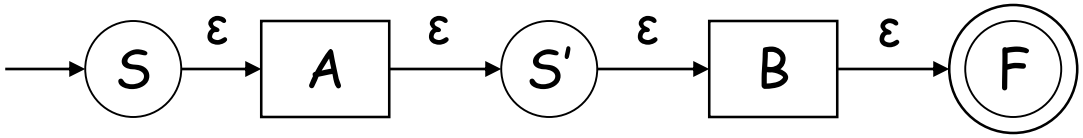


  - NFA cho kí hiệu thường

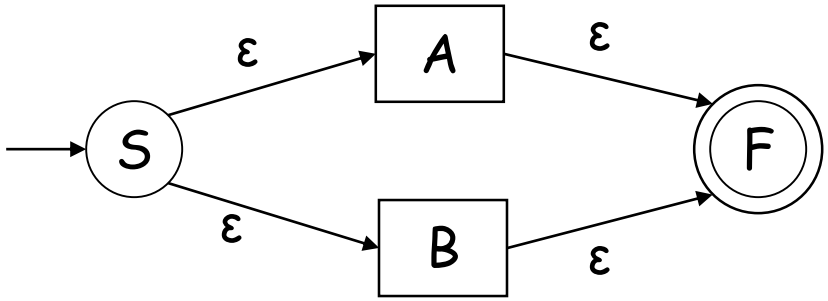


Thuật toán Thompson

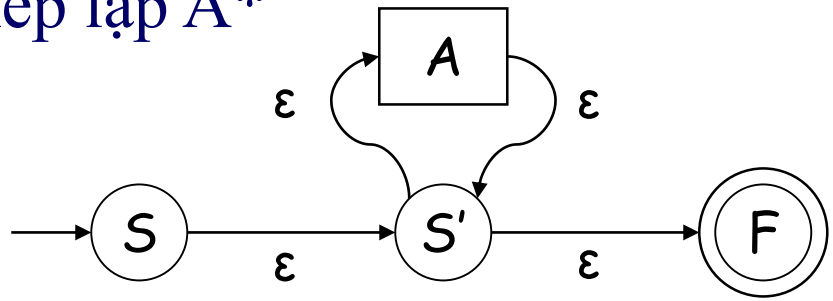
- Bước 3: ghép các NFA theo quy tắc chuyển đổi phép toán sau đây
  - Phép nối AB



  - Phép chọn A | B

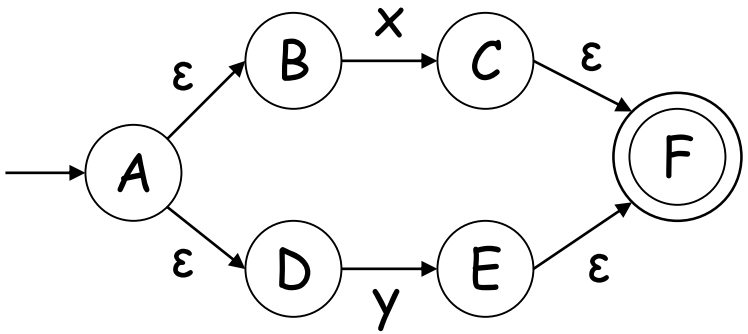


  - Phép lặp A*

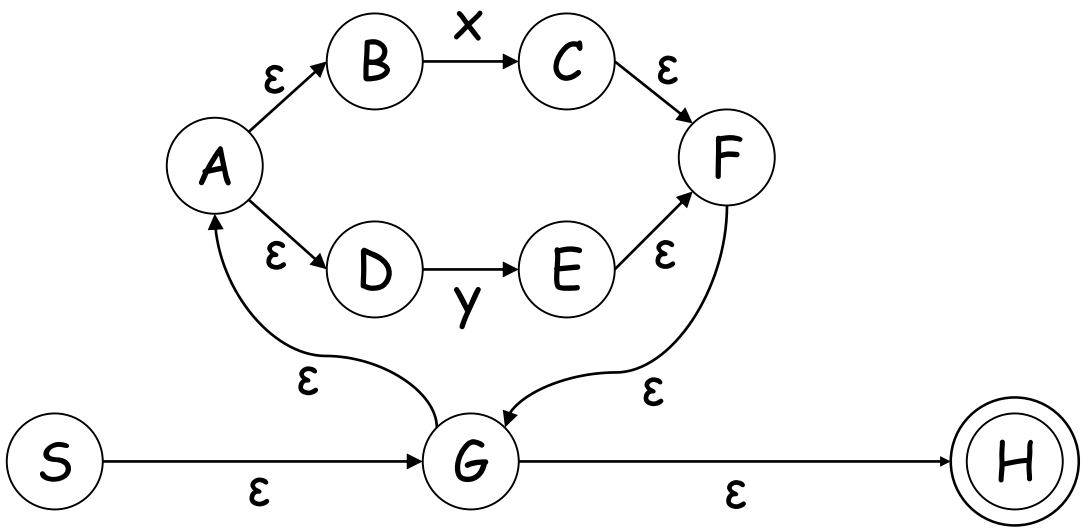


Ví dụ: dựng NFA cho (x | y)*

- Tạo NFA cho (x | y)



- Đặt NFA trên vào phép lặp



Phần 3.2

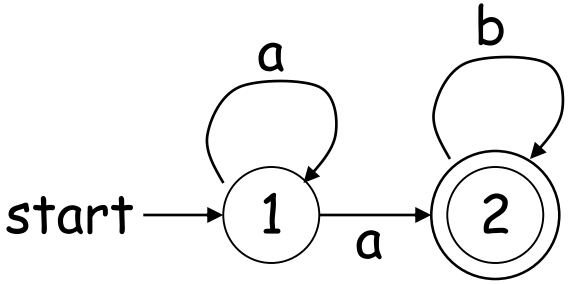
Chuyển đổi từ NFA sang DFA

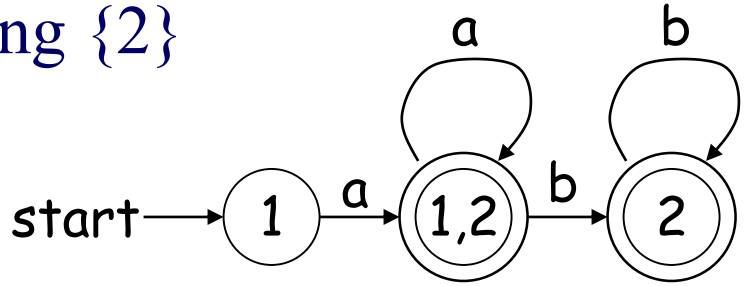
Chuyển đổi từ NFA sang DFA

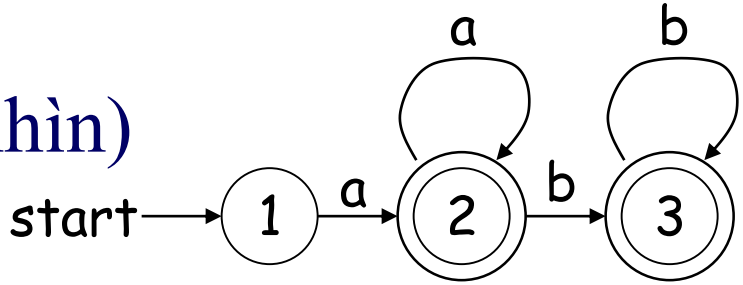
- Chuyển đổi từ NFA sang DFA gồm hai bài toán:
  1. Loại bỏ các bước chuyển chấp nhận kí hiệu đầu vào ε
  2. Loại bỏ các trạng thái đa định
- Input: một NFA (gọi là N)
- Output: một DFA (gọi là D) đoán nhận cùng ngôn ngữ với N. Xây dựng D, gồm 2 bước:
  - Xây dựng tập trạng thái của D
  - Xây dựng các hàm chuyển move(s,a) đơn trị
- Ý tưởng: quan sát hoạt động của N, một trạng thái của D là tập các trạng thái của N, một bước chuyển của D là một bước chuyển của tập trạng thái của N

Ví dụ 1

- Xét NFA đoán nhận a+b*
- Quan sát hoạt động của NFA
  - Trạng thái bắt đầu chuyển sang {1}
  - {1} nhận kí hiệu a chuyển sang {1,2}
  - {1,2} nhận kí hiệu a chuyển sang {1,2}
  - {1,2} nhận kí hiệu b chuyển sang {2}
  - {2} nhận b chuyển sang {2}
- Ta được DFA tương đương:
- Đổi tên trạng thái (cho dễ nhìn)

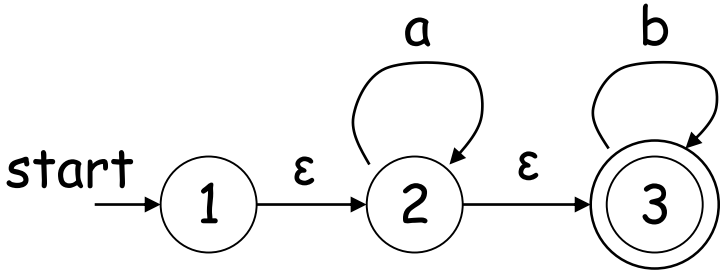


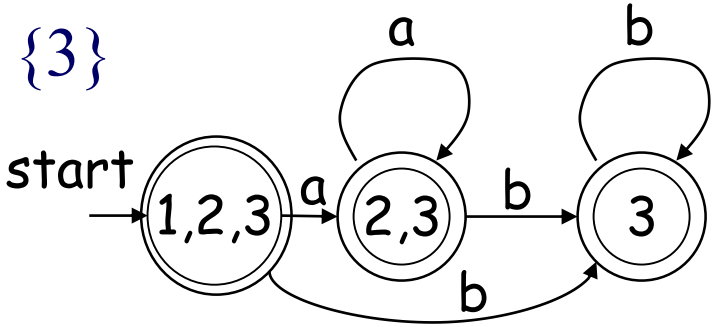




Ví dụ 2

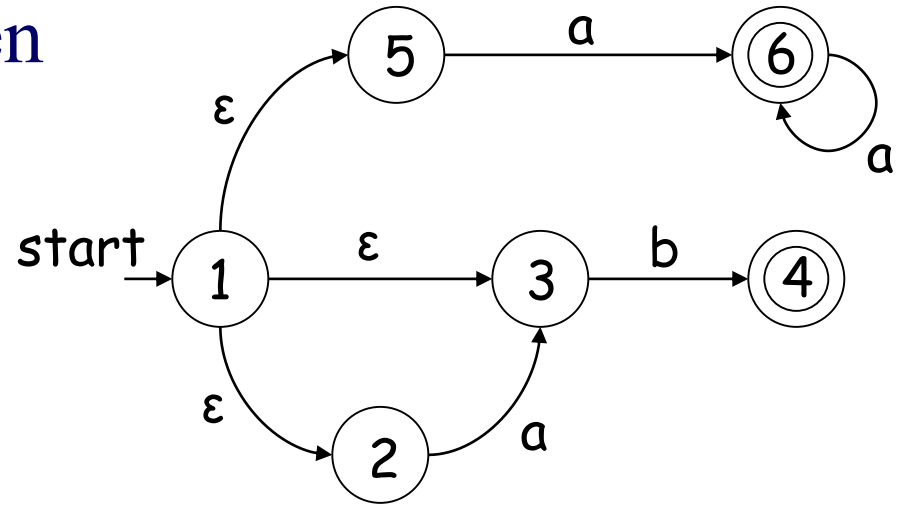
- Xét NFA đoán nhận a*b*
- Quan sát hoạt động của NFA
  - Trạng thái bắt đầu chuyển sang {1,2,3}
  - {1,2,3} nhận kí hiệu a chuyển sang {2,3}
  - {1,2,3} nhận kí hiệu b chuyển sang {3}
  - {2,3} nhận kí hiệu a chuyển sang {2,3}
  - {2,3} nhận kí hiệu b chuyển sang {3}
  - {3} nhận kí hiệu b chuyển sang {3}
- Ta được DFA tương đương:

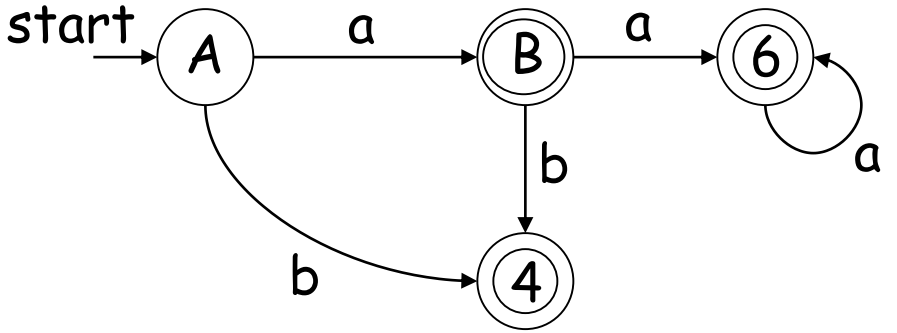




Ví dụ 3

- Trạng thái bắt đầu chuyển sang {1,2,3,5}, đặt tên trạng thái này là A
- move(A, a) = {3,6}, đặt tên trạng thái này là B
- move(A, b) = {4}
- move(B, a) = {6}
- move(B, b) = {4}
- move({6}, a) = {6}





Phần 3.3

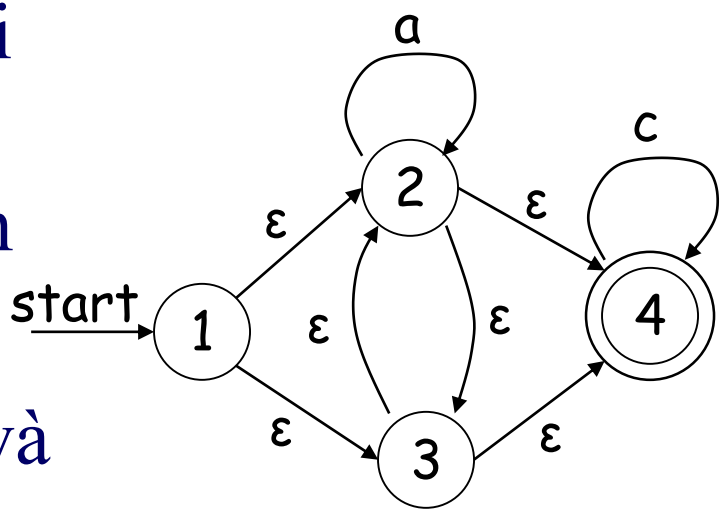
DFA tối ưu cho phân tích từ vựng

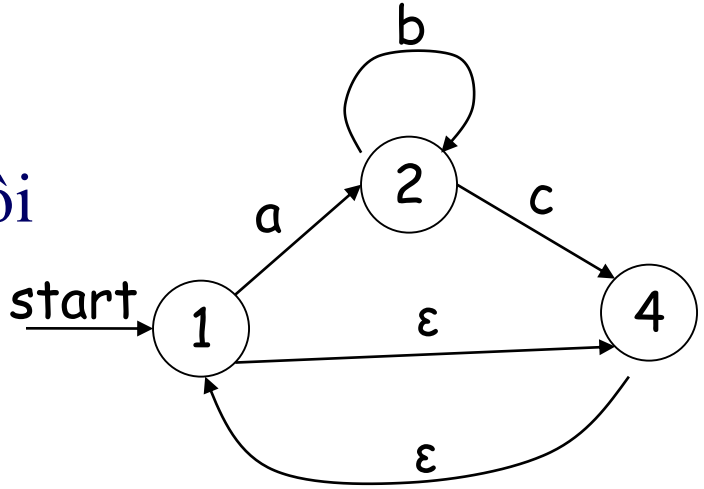
Số lượng trạng thái của DFA

- DFA đơn giản hơn NFA trong lập trình, nhưng lại đối mặt với vấn đề khác, đó là số lượng trạng thái của DFA có thể nhiều hơn NFA một cách đáng kể
  - Một NFA có n trạng thái có thể chuyển đổi thành một DFA có tới 2^n trạng thái (trường hợp tệ nhất)
- Kích thước bảng chuyển (hàm move) có liên quan chặt chẽ tới số lượng trạng thái, vì thế việc giảm số trạng thái của DFA là quan trọng trong thực tế
- Một cách logic (?) thì nếu NFA có ít trạng thái thì sẽ sinh DFA ít trạng thái hơn, vì thế ta có thể bắt đầu tối ưu ngay từ NFA

Tối ưu NFA

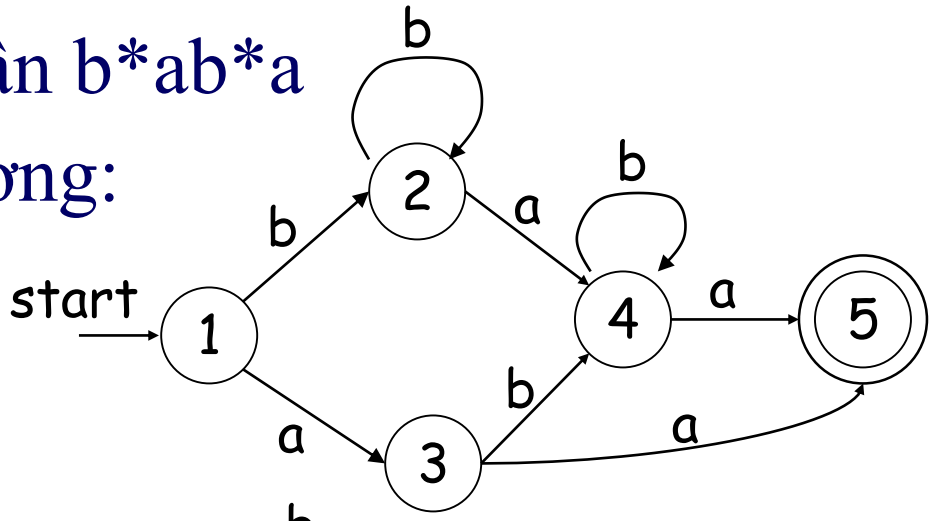
- Không có nhiều cơ hội cho tối ưu NFA, ý tưởng dễ thấy nhất là hợp các trạng thái cùng trên một chu trình ε
- Trong NFA trên: trạng thái 2 và 3 có thể ghép đôi
- Trong NFA dưới:
  - Trạng thái 1 và 4 có thể ghép đôi
  - Sửa đổi hàm move(2, c) = 4 thành move(2, c) = 1

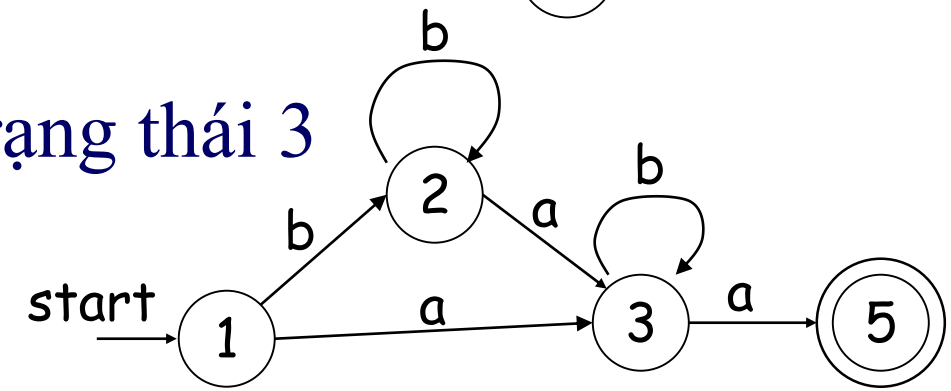




Tối ưu DFA

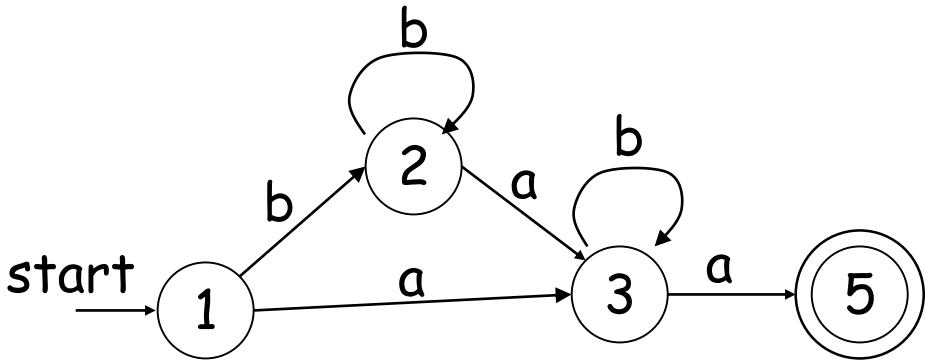
- Ý tưởng: ghép các trạng thái tương đương (hàm move giống nhau)
- Ví dụ: xét DFA đoán nhận b*ab*a
- Ta thấy 3 và 4 tương đương:
  - move(3, a) = 5
  - move(3, b) = 4
  - move(4, a) = 5
  - move(4, b) = 4
- Ghép 3 và 4 thành trạng thái 3



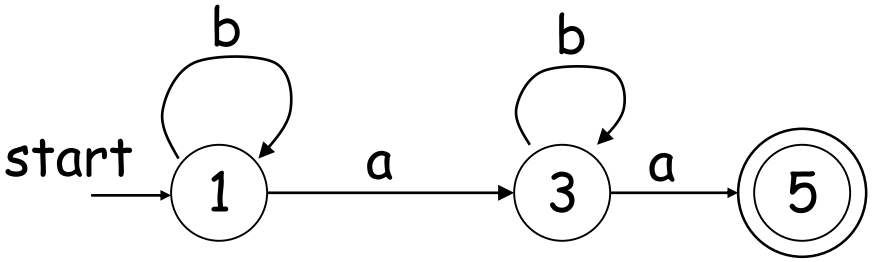


Tối ưu DFA

- Với DFA mới, ta thấy 1 và 2 tương đương:
  - move(1, a) = 3
  - move(1, b) = 2
  - move(2, a) = 3
  - move(2, b) = 2



- Ghép trạng thái 1 và 2 thành trạng thái 1, ta được trạng thái tối ưu sau



- Chú ý: chưa có thuật giải tối ưu cho bài toán này

Tối ưu bảng chuyển

- Tổ chức bảng chuyển thường sử dụng ma trận
  - Ưu điểm: đơn giản, dễ hiểu, tốc độ cao
  - Nhược điểm: kích thước lớn, dễ nhầm lẫn khi mã hóa
- Có một số chiến thuật tối ưu bảng chuyển, chủ yếu dựa trên ý tưởng nén các trạng thái giống nhau

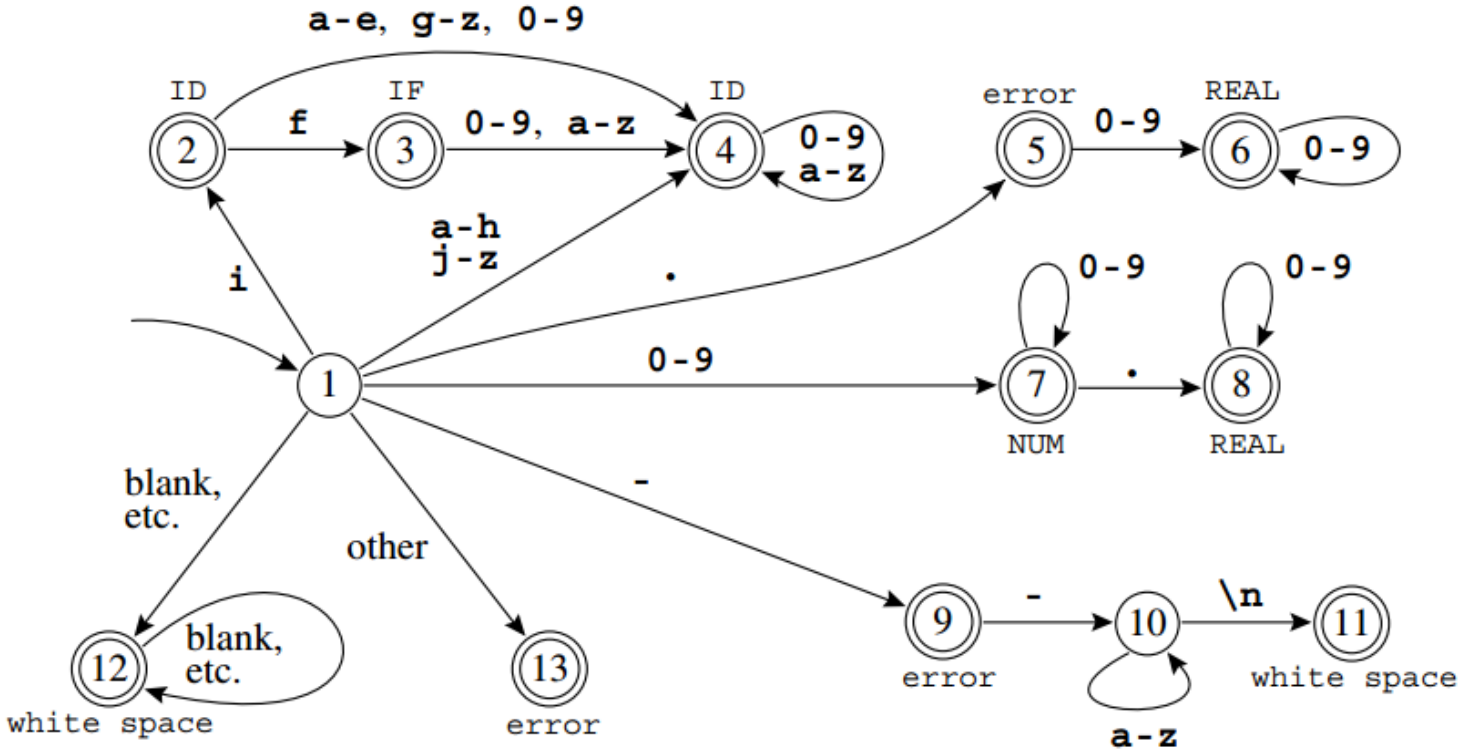
```
int edges[][] = {  /* ···0 1 2···-···e f g h i j··· */
/* state 0 */     {0,0,···0,0,0···0···0,0,0,0,0,0···},
/* state 1 */     {0,0,···7,7,7···9···4,4,4,4,2,4···},
/* state 2 */     {0,0,···4,4,4···0···4,3,4,4,4,4···},
/* state 3 */     {0,0,···4,4,4···0···4,4,4,4,4,4···},
/* state 4 */     {0,0,···4,4,4···0···4,4,4,4,4,4···},
/* state 5 */     {0,0,···6,6,6···0···0,0,0,0,0,0···},
/* state 6 */     {0,0,···6,6,6···0···0,0,0,0,0,0···},
/* state 7 */     {0,0,···7,7,7···0···0,0,0,0,0,0···},
/* state 8 */     {0,0,···8,8,8···0···0,0,0,0,0,0···},
```

Phần 4

Bộ phân tích từ vựng dựa trên DFA

DFA trong thực tế

DFA trong thực tế là việc ghép từ rất nhiều các DFA con, hãy xem DFA dưới đây và chỉ ra những từ loại mà nó đoán nhận



Bộ PTTV dựa trên DFA

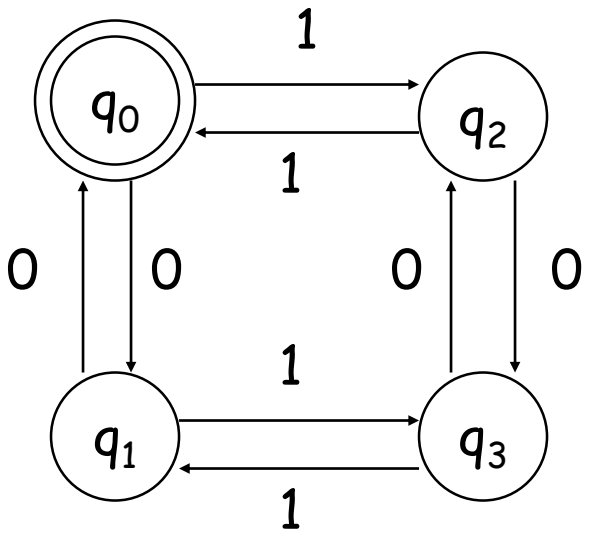
```
// đầu vào: chuỗi x kết thúc bởi kí hiệu EOF
// đầu ra: trạng thái chấp nhận hoặc lỗi (ERROR)
s := START;
while (s != ERROR) {
    c := nextInput(x);
    if (c == EOF) break;
    s := move(s, c);
}
if (isAcceptState(s)) return acceptState(s);
else return ERROR;
```

Phần 5

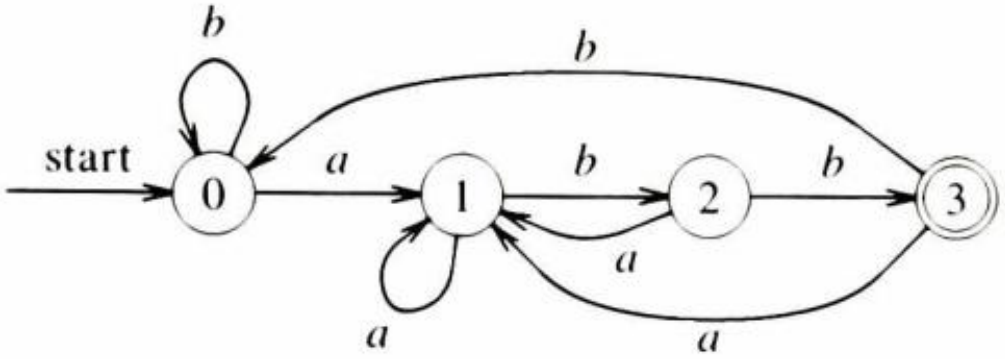
Bài tập

Bài tập

1. Hình bên thể hiện đồ thị chuyển của một DFA (bắt đầu từ q_o). Hãy cho biết DFA đó sau đoán nhận ngôn ngữ nào? (viết dạng biểu thức chính quy)

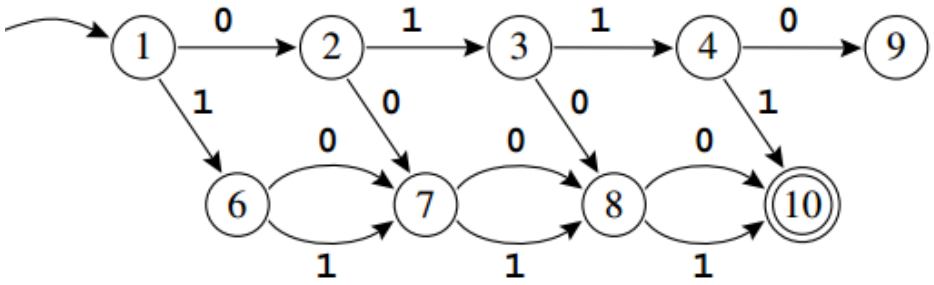


2. DFA dưới đoán nhận biểu thức nào?

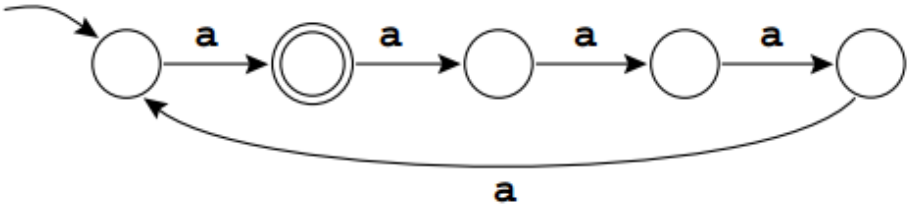


Bài tập

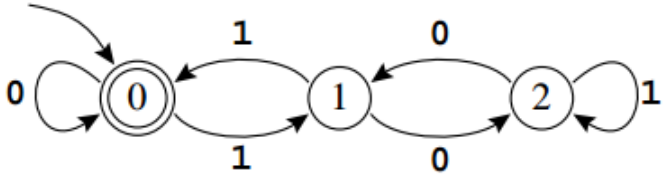
3. DFA dưới đây đoán nhận biểu thức chính quy nào?



4. DFA dưới đây đoán nhận biểu thức chính quy nào?



5. DFA dưới đây đoán nhận biểu thức chính quy nào?

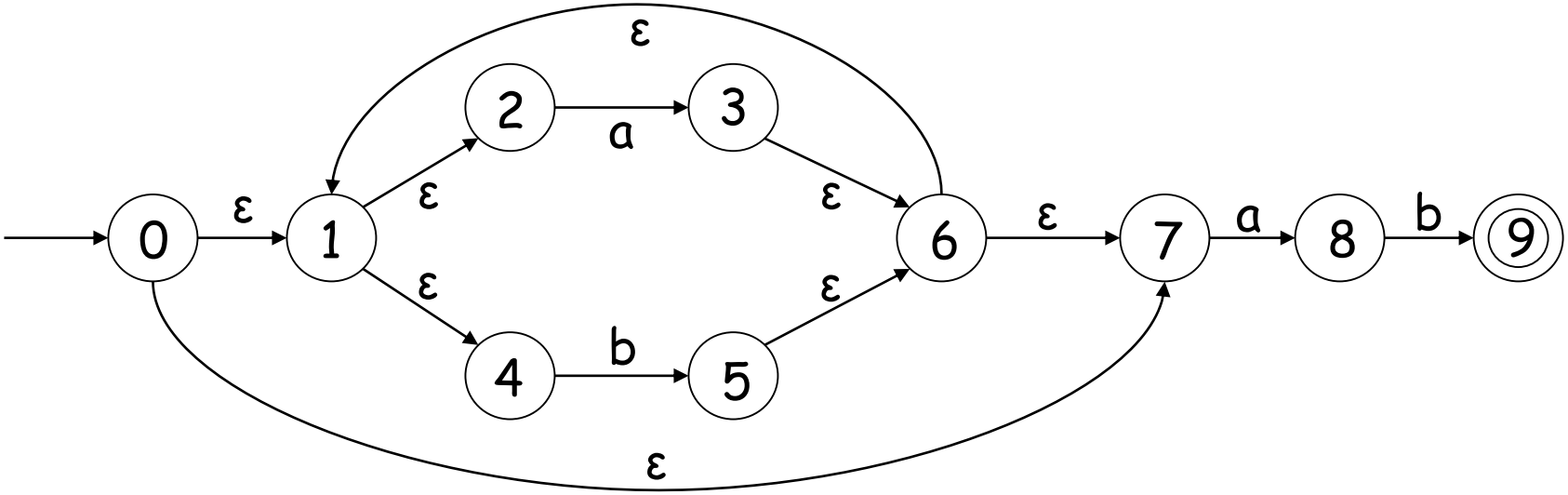


Bài tập

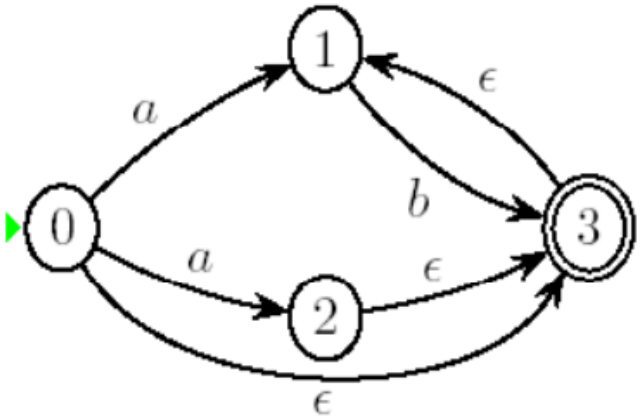
6. Xây dựng NFA đoán nhận biểu thức (\+? | -?) d+
7. Xây dựng NFA đoán nhận các biểu thức dưới đây:
   1. (a* | b*)*
   2. ((ε | a) b)*
   3. (a | b)*abb(a | b)*
   4. (if | then | else)
   5. a((b|a∗c)x)∗|x∗a
   6. ab* (a|b)+ a
   7. (a|ε)b*ab
8. Xây dựng DFA đoán nhận (0|(1(01*(00)*0)*1)*)*

Bài tập

9. Chuyển đổi NFA sau thành DFA

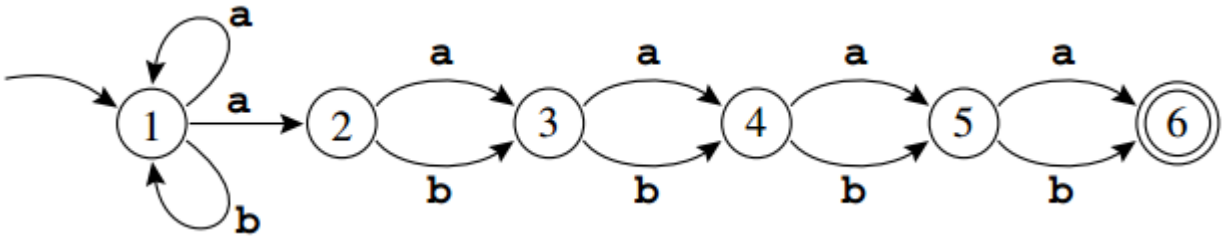


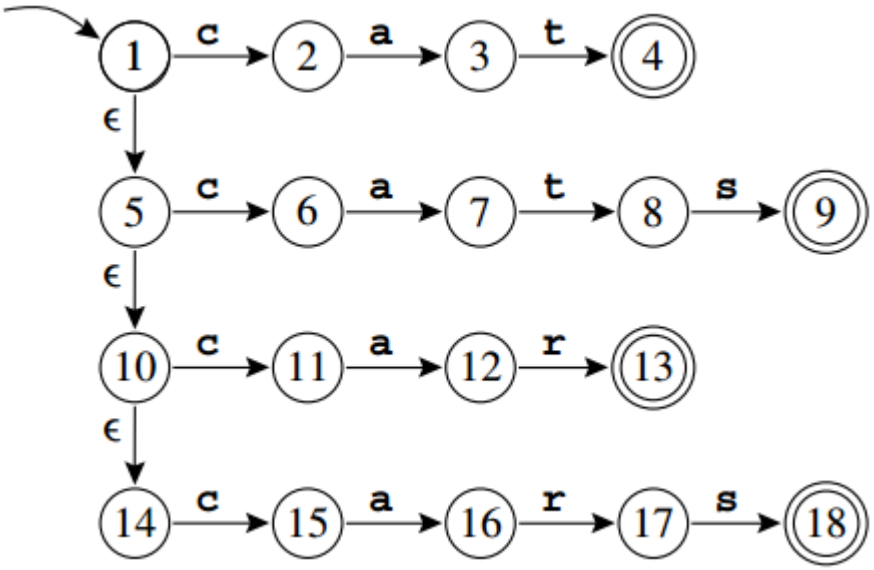
10. Chuyển đổi NFA sau thành DFA

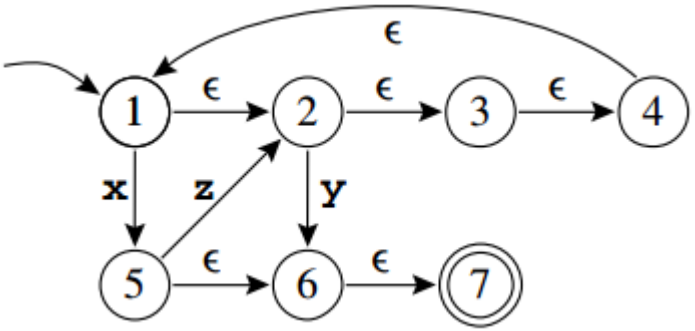


Bài tập

11. Chuyển đổi các NFA sau thành DFA







Bài tập

12. Xây dựng DFA tối ưu cho:
    1. (a | b)* a (a | b)
    2. (a | b)* a (a | b) (a | b)
    3. (a | b)* a (a | b) (a | b) (a | b)
13. Tối ưu hóa DFA dưới đây (nếu có thể)