Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

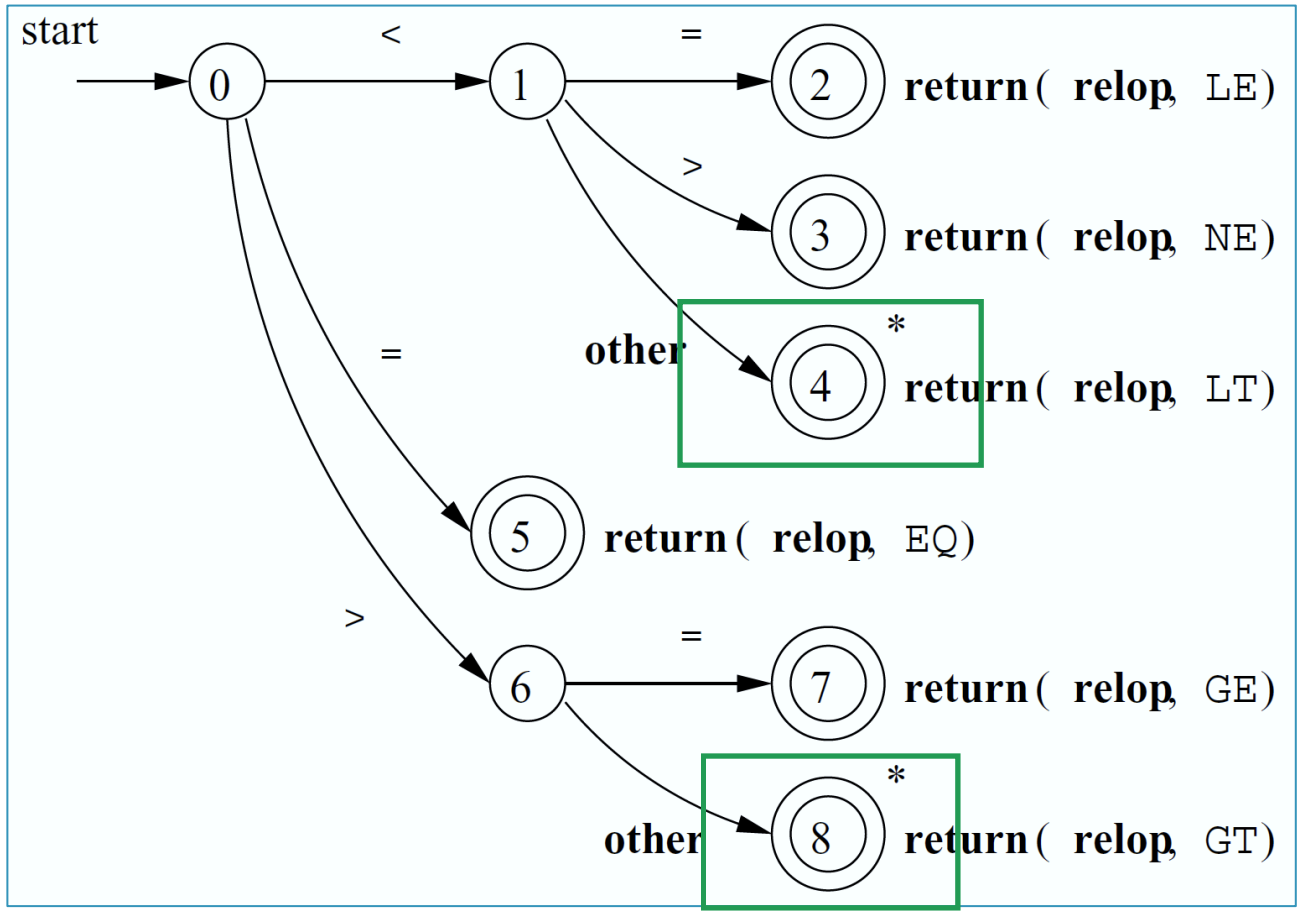
Transition Diagrams

- Transition diagrams have a collection of **nodes** or circles, called **states**
- **Edges** are directed from one state of the transition diagram to another
  - **Accepting or final states** indicate that a lexeme has been found
  - If it is necessary to retract the forward pointer one position, a ***** is placed near that accepting state
  - One state is designated the start state, or **initial state**

-
دیاگرام های انتقال همون عبارات منظم میشن به صورت شکلی
از روی این دیاگرام انتقال همون NFA , DFA تعریف میشه و در نهایت این DFA پیاده سازی میشه برای اینکه یک لکسر بتونه کار خودش رو انجام بده
دیاگرام انتقال یک دیاگرامی است که داره انتقال بین یک سری وضعیت ها رو نشون میده ینی یکسری استیت داره که بهشون نود گفته میشه و بین این استیت ها می تونیم حرکت کنیم با یال جهت دار
پس دیاگرام انتقال از یکسری استیت تشکیل شده که بین اون استیت ها هم وضعیت حرکت مشخص است
این دیاگرام انتقال تنها یک استیت شروع داره و یک یا چند استیت پایان داره

Transition Diagrams

- **Example:** A transition diagram that recognizes the lexemes matching the token relop

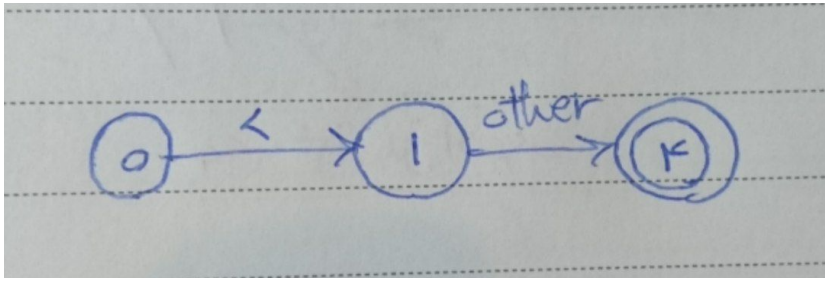


برای حرف کوچکتر مثلا > باید حرف بعدی هم بخونه تا متوجه بشه این کوچکتر خالی است یا چی --> اگر حرف بعدی رو خوند و حرف دیگه ای غیر از = or > بود ینی other در این صورت باید یکی به عقب برگرده واسه همین روی نود پایانی استار گذاشته توی این حالت و اینو در اخر به عنوان کوچکتر می شناسه

-
این دیاگرام انتقال برای شناسایی اپراتورهای مقایسه ای است

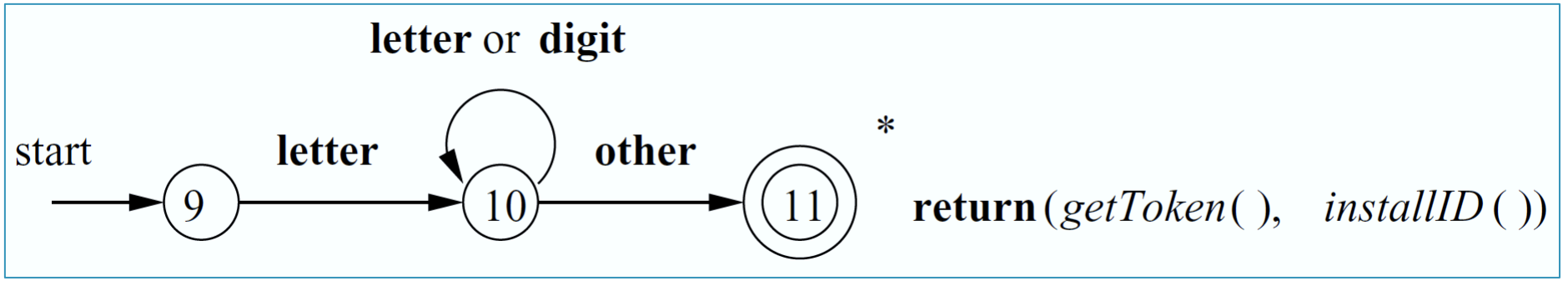
این دیاگرامی که اینجا کشیده شده صرفا برای اپراتورهای مقایسه ای است مثلا توی مثال زیر:
if(x<y توی این مثال فرض میکنیم رسیدیم به < و این کاراکتری که خونده میشه چون جز کاراکترهای مقایسه ای است اون تحلیل گر لغوی میاد به استیت شروع مربوط اپراتورهای مقایسه ای
و اون x با یک دیاگرام دیگه خونده شده و تشخیص داده شده به عنوان id و اومدیم کاراکتر بعدی و چون این کاراکتر جز اپراتورهای مقایسه ای است این اومده به استیت صفر مربوط به این دیاگرام اپراتور مقایسه ای --> شکل زیر

نکته: اون حرف هایی که داریم از ورودی می خونیم روی یال ها دیده میشه توی تحلیل گر لغوی



Transition Diagrams

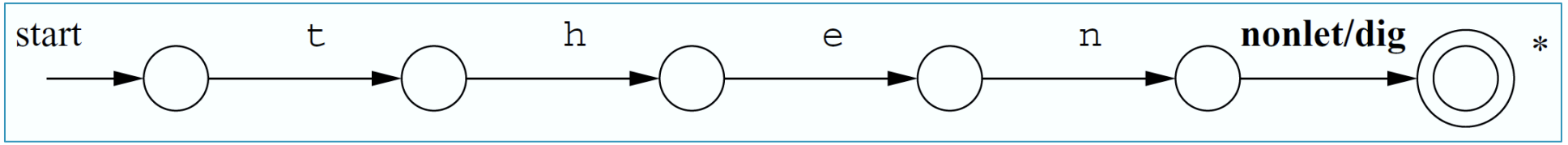
- **Example:** Recognition of Reserved Words and Identifiers



-
این دیاگرام انتقال برای شناسایی id است
استیت شروع می تونه از صفر هم باشه اصلا فرقی نمی کنه ولی چون ادامه مثال قبلی بوده از 9 اومده شروع کرده ولی حتی می تونست از 0 هم باز شروع بکنه اینجا
این دیاگرام مربوط به زبان پاسکال است چون توی زبان c ما آندرلاین هم می تونیم داشته باشیم به عنوان شروع
letter : حروف بزرگ یا کوچیک انگلیسی است
other اینجا می تونه اسپیس یا عملگر یا هر چیز دیگه ای می تونه باشه

Transition Diagrams

- There are two ways that we can handle reserved words that look like identifiers:
  1. ***Install the reserved words in the symbol table initially***
     - When an identifier is found, a call to **installID** places it in the symbol table if it is not already there
  2. ***Create separate transition diagrams for each keyword***
     - In this method we must prioritize the tokens so that the reserved-word tokens are recognized in preference to id



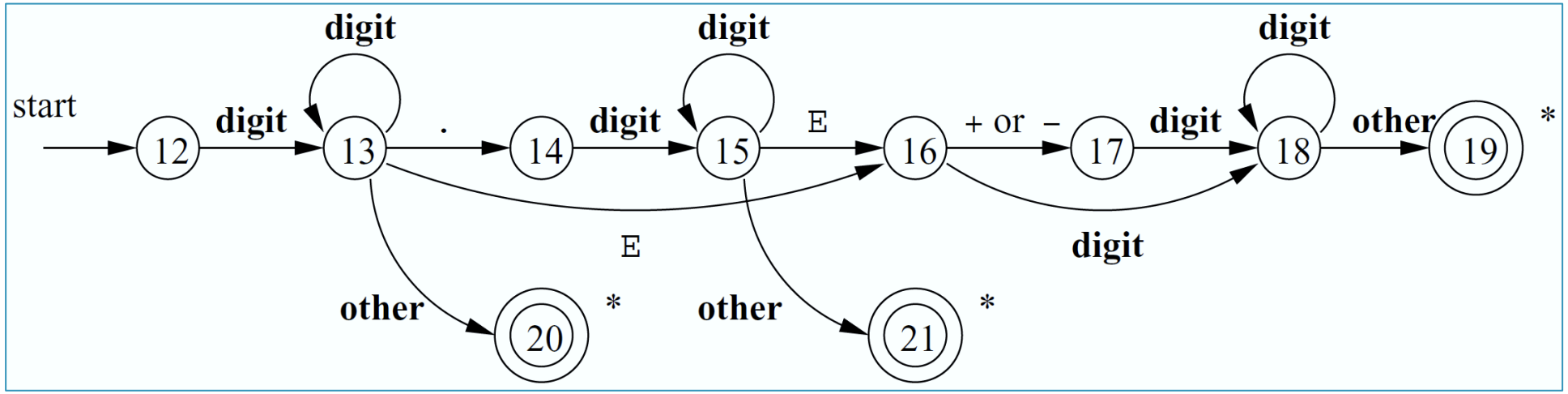
این دیاگرام یه دونه دیاگرام انتقال است برای کلمه کلیدی then

-
برای کلمات کلیدی کامپایلرها یه مقدار متفاوت هستند بعضی هاشون برای هر کلمه کلیدی یک دیاگرام انتقال می گیرن ولی بعضی هاشون یه دونه می گیرن برای همه
دو راه وجود داره برای اینکه کامپایلرها بتونن کلمات کلیدی رو مشخص کنن:
1- کلمات کلیدی از اول توی جدول نماد ذخیره بشن که تعداد اینا ثابت است ینی یه تعداد کلمات کلیدی برای هر زبان وجود داره که تعدادشون ثابت است --> اینجا دیاگرام های جدا برای هر کلمه کلیدی نداریم --> حالا هر لغتی که به عنوان id شناسایی بشه اول میاد چک میشه که توی این لغات ذخیره شده که کلمات کلیدی هستن وجود داره یا نداره و اگر وجود داشت دیگه id در نظر گرفته نمیشه و کلمه کلیدی در نظر گرفته میشه و می ره بعدی رو می خونه --> پس یه دونه دیاگرام برای id داشته باشیم و هر id که تشخیص داده میشه یک چکی می کنیم که این id کلمه کلیدی است یا نیست و یک راهش اینه که این کلمه های کلیدی همشون یه جا ذخیره شده باشن --> installID که توی مثال قبلی اومده بود منظورش همین بود که توکن رو که شناسایی کردی به عنوان id حالا بیا توی اون قسمتی که کلمات کلیدی هستن یه بار بررسی کن و ببین توی اینا هست یا نیست و اگر نبود میشه id و اگر بود میشه کلمه کلیدی
2- وقتی که برای هر کلمه کلیدی یک دیاگرام انتقال در نظر گرفته بشه صرفا اینه که باید اولویت بندی بشه ینی این دیاگرام نهایی که ساخته میشه باید توش اولویت باشه که اگر یک کلمه ای داره خونده میشه که با حرف مثلا t شروع شده اول باید بیاد همه دیاگرام های مربوط به کلمه های کلیدی چک بشه از اول تا اخر حالا یا با یکی از اون ها منطبق است یا نیست و اگر با هیچ کدوم منطبق نبود برسه سر دیاگرام id

Transition Diagrams

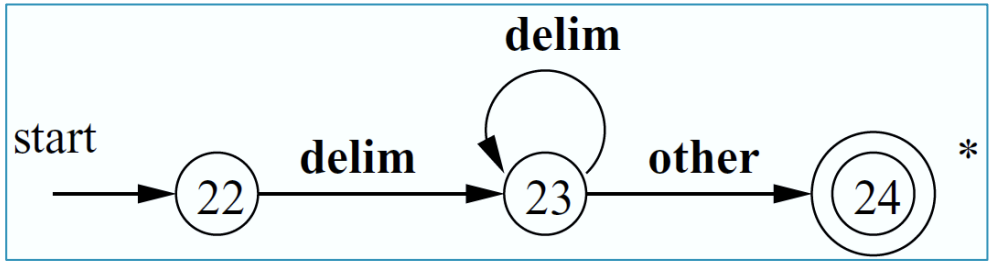
دیاگرام نامبر:
اعداد صحیح توی استیت 20 پذیرفته میشن
اعداد اعشاری توی استیت 21 پذیرفته میشن
نماد علمی توی استیت 19 پذیرفته میشن

- **Example:** The transition diagram for token number



- **Example:** The transition diagram for whitespace

دیاگرام فضای خالی مثل اسپیس یا تب یا اینتر یا...

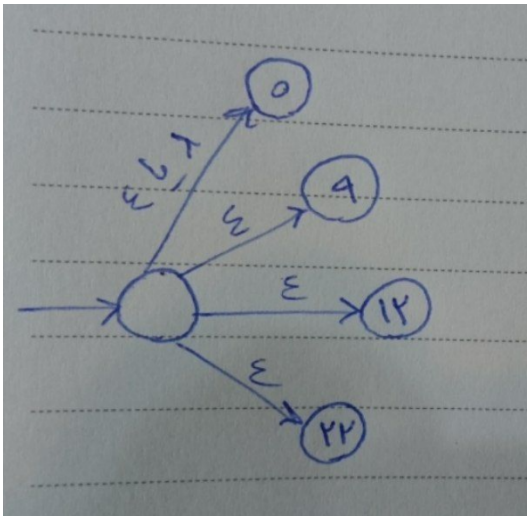


Transition Diagrams

- **Sketch of implementation of relop transition diagram**
- **Architecture of a Transition-Diagram-Based Lexical Analyzer**
  - Combining all the transition diagrams into one (Combining states 0, 9, 12, and 22 into one start state)

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```

-
اگر یک کامپایلر خیلی ساده داشتیم که همین 4 تا دیاگرامی که بالا گفتیم رو داشت الان باید اینا رو بذاریم کنار هم و پیاده سازیش رو انجام بدیم:
یک استیت شروع 0 برای اپراتورهای مقایسه ای
استیت شرو id هم = 9
استیت شروع عدد = 12
استیت شروع فضای خالی = 22
حالا لازمه برای اینکه لکسر کامل بشه یک استیت شروع کلی بذاریم و به هر 4 تای اینا حرکت داشته باشیم از این استیت شروع بدون هیچ کاراکتری --> شکل زیر:
مثلا اولین حرفی که توی برنامه خونده میشه p و بعد میاد می بینه با کدوم یکی از اینا می تونه شروع بکنه که این می ره توی استیت id و اونجا ادامه پیدا میکنه



Finite Automata

- Finite automata are recognizers
  - They simply say "yes" or "no" about each possible input string
- Finite automata come in two flavors
  - **Nondeterministic Finite Automata (NFA)**
    - It has no restrictions on the labels of their edges
    - A symbol can label several edges out of the same state, and ϵ is a possible label
  - **Deterministic Finite Automata (DFA)**
    - It has, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state
- Both deterministic and nondeterministic finite automata are capable of recognizing the same languages, **called the regular languages**

-
ساده ترین حالت اینه که ما به ازای هر یه دونه توکنی که می تونیم داشته باشیم مثلا id یا هر چی عبارت منظمش تعریف بشه و تبدیل بشه به یه دونه NFA و NFA خودش تبدیل بشه DFA و بعد این پیاده سازی بشه
NFA , DFA هر دوتاشون ماشین حالت متناهی هستن ولی NFA ممکنه وضعیت های نامشخصی داخلش باشه ولی توی DFA وضعیت نامشخص نداریم

هم NFA و هم DFA مجموعه زبان هایی که شناسایی می کنن می شه زبان های منظم ینی یک زبان نیست که NFA بتونه پیاده سازی بکنه و DFA نتونه چون کاملا با هم معادل هستن

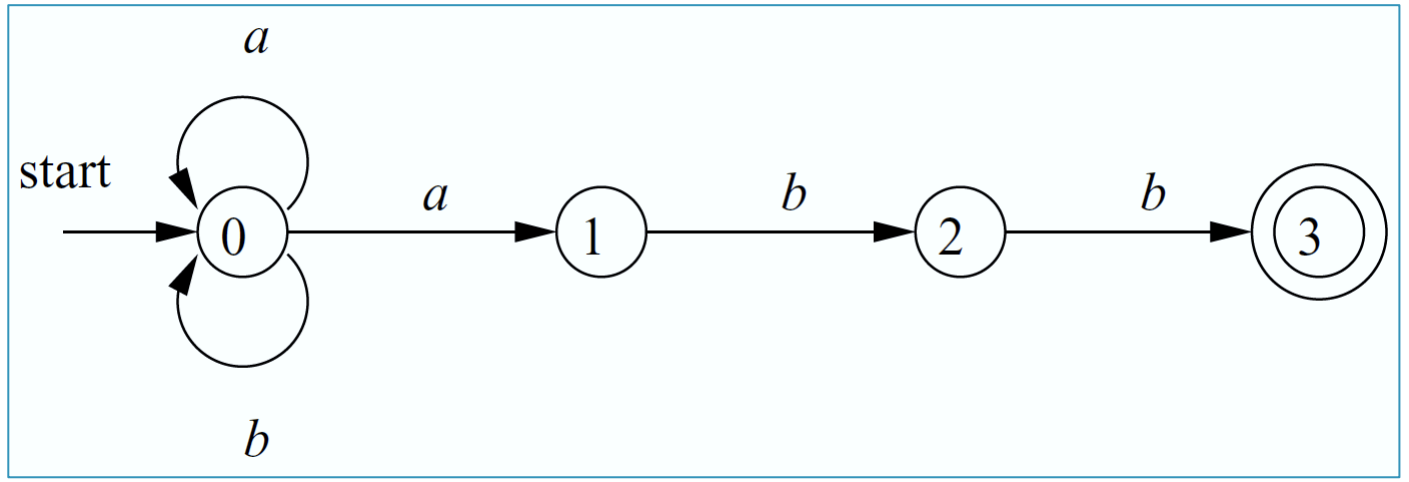
Nondeterministic Finite Automata

- **A nondeterministic finite automaton (NFA) consists of:**
  1. A finite set of states S
  2. A set of input symbols $\sum$, the input alphabet
  3. A transition function that gives, for each state, and for each symbol in $\sum \cup \{\epsilon\}$ a set of next states
  4. A state s_0 from S that is distinguished as the start state (or initial state)
  5. A set of states F, a subset of S, that is distinguished as the accepting states (or final states)

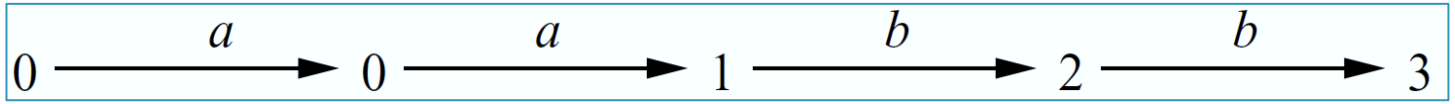
-
تعریف NFA:
1- یک مجموعه استیت داریم : S
2- الفبای ورودی رو داریم که با سیگما نشون میدیم
3- از هر استیتی به ازای هر حرف الفبا به چه استیت دیگه حرکت کنیم این میشه تابع انتقال --> تابع انتقال باید تعریف کنه این انتقال بین استیت ها رو --> توی NFA ما می تونیم از هر استیتی یه دونه از حروف الفبا رو ببینیم یا اپسیلون رو ببینیم و به استیت دیگه حرکت کنیم
4- استیت شروع داریم به اسم S0
5- استیت های پایانی می تونه بیشتر از یکی باشه --> یک زیر مجموعه ای از S که استیت های پایانی هستن --> F

Nondeterministic Finite Automata

- **Example**
  - The transition graph for an NFA recognizing the language of regular expression $(a|b)^*abb$



  - The string aabb is accepted by the above NFA



Nondeterministic Finite Automata

- **Transition Tables**
  - An NFA can also represented by a transition table, whose rows correspond to states and columns correspond to the input symbols and ϵ

STATE	a	b	ϵ
0	$\{0, 1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

تابع انتقال در قالب جدول نشون داده میشه معمولا

Deterministic Finite Automata

- A deterministic finite automaton (DFA) is a special case of an NFA where:
  1. There are no moves on input ϵ
  2. For each state s and input symbol a, there is exactly one edge out of s labeled a
- Every regular expression and every NFA can be converted to a DFA accepting the same language

-
DFA: یک NFA است که دوتا محدودیت داره:
1- اصلا اپسیلون نمی تونه داشته باشه
2- دقیقا به ازای هر حرف الفبا باید یه دونه یال داشته باشه نه کمتر و نه بیشتر

نکته: هر عبارت منظم با DFA , NFA قابل پیاده سازی است و NFA تبدیل میشه به DFA و هیچ فرقی با هم ندارن از نظر مجموعه زبان هایی که پذیرش می کنن

Simulating a DFA

```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s is in F ) return "yes";
else return "no";
```

DFA چون مشخصه از نظر پیاده سازی راحت تر است

Conversion of an NFA to a DFA

- The general idea
  - Each state of the constructed DFA corresponds to a set of NFA states
- Operations on NFA states

OPERATION	DESCRIPTION
ϵ-$closure(s)$	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-$closure(T)$	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}\ \epsilon$-$closure(s)$.
$move(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T.

-

تبدیل NFA به DFA:

ایده کلی این است که هر یک وضعیتی که برای DFA نهایی به دست میاد یک زیر مجموعه ای از وضعیت های همون NFA است که داریم

جدول: وقتی که می خوایم NFA به DFA تبدیل کنیم:

1- ؟؟