Курс «Компиляторные технологии»

Лекция 7

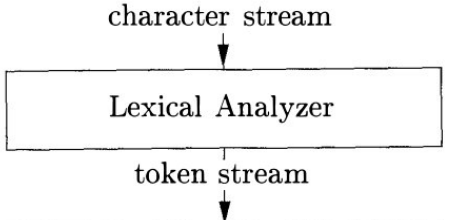
Лексический анализ (1)

Курносов Михаил Георгиевич

www.mkurnosov.net

Сибирский государственный университет телекоммуникаций и информатики
Весенний семестр

Лексический анализатор




```
<ID,0>, <ASSIGN>,
<ID,1>, <PLUS>,
<ID,2>, <MUL>,
<STR,3>,
<SEMICOLON>
```

- **Лексический анализатор** (lexical analyzer, lexer, scanner) — разбивает входную программу на последовательность *лексем* (lexeme), минимально значимых единиц входного языка
- Тип допустимых лексем определяется описанием языка
- Игнорирует пробельные символы, комментарии, отслеживает номер текущей строки для корректного информирования о положении возможных ошибок

```
// Увеличить сумму
globalSum = localSum + r * 16;
```

Лексемы: «globalSum», «=», «localSum», «+», «r», «*», «16», «;»

Symbol table	
ID	Symbol
0	globalSum
1	localSum
2	r
3	16

- Для каждой найденной лексемы анализатор формирует *токен* (token) — пара <*имя-токена*, *значение-атрибута*>, *имя-токена* — тип/класс лексемы, *значение-атрибута* — непосредственно лексема или ссылка на запись в таблице символом

Tokens: <ID,0>, <ASSIGN>, <ID,1>, <PLUS>, <ID,2>, <MUL>, <STR,3>, <SEMICOLON>

Token-names: ID, ASSIGN, PLUS, MUL, STR, SEMICOLON

Сложности распознавания

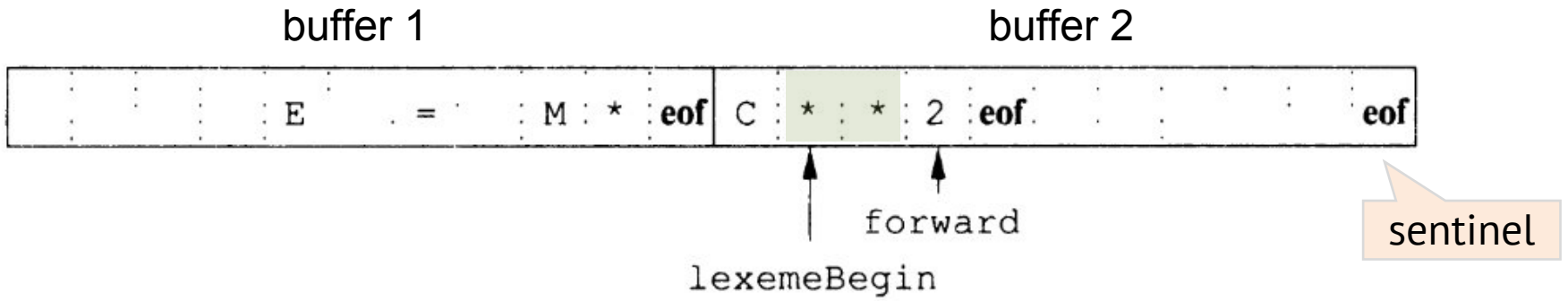
- В некоторых языках невозможно сразу определить по лексеме какому токену она соответствует
- **Fortran 90** (фиксированный формат — символы пробела игнорируются)
  - `DO 5 I = 1.25`
    DO5I — идентификатор, но анализатор понимает это только при встрече десятичной точки после 1
  - `DO 5 I = 1,125`
    DO — ключевое слово

Лексические ошибки

- **Что делать если лексический анализатор не способен продолжать работу — ни один из шаблонов не соответствует префиксу входного потока?**
- Простейшей стратегией в этой ситуации будет восстановление в "режиме паники" (panic mode) — пропускаем входные символы до тех пор, пока лексический анализатор не встретит распознаваемый токен в начале оставшейся входной строки
- Замена символа другим — попытка исправления программы

Буферизация ввода

- Для уменьшения накладных расходов на обработку одного входного символа применяется буферизация
- Метод **пары буферов** (buffer pairs) — буферизации включает два по очереди загружаемых буфера длины *N* (например, *N* = 4096 байт)
  - *lexemeBegin* — указатель на начало текущей лексемы, протяженность которой пытаемся определить
  - *forward* — указатель, который сканирует символы до тех пор, пока выполняется соответствие шаблону



- **Определение достижение конца буфера** — по текущей позиции или проверка чтения сигнального символа (ограничитель, sentinel)
- Ограничитель (sentinel) — специальный символ, который не может быть частью исходной программы, добавляется в конец каждого буфера

Спецификация токенов

- Регулярные выражения (regular expression) — способ записи спецификации шаблонов лексем
- **Алфавит** — любое конечное множество символов
  - Примеры: {0, 1}, {a, b, c, ..., z}
- **Строка** (слово, предложение, string, word, sentence) над некоторым алфавитом — это конечная последовательность символов из этого алфавита
  - Примеры: 0110010, hello, ε — пустая строка
- **Язык** — любое счетное множество строк над некоторым фиксированным алфавитом
  - Примеры: {∅}, {ε}, L = {A, B, ..., Z, a, b, ..., z}, D = {0, 1, ..., 9}
- **Операции над языками**

Операция	Определение и обозначение
Объединение (union) L и M	$L \cup M = \{s \mid s \in L \text{ или } s \in M\}$
Конкатенация (concatenation) L и M	$LM = \{st \mid s \in L \text{ и } t \in M\}$
Замыкание Клини (Kleene closure) языка L	$L^* = \bigcup_{i=0}^{\infty} L^i$
Позитивное замыкание (positive closure) языка L	$L^+ = \bigcup_{i=1}^{\infty} L^i$

- $L \cup D$ — язык с 62 строками единичной длины (буква или цифра)
- $L \cap D$ — язык с 520 строками длины 2 (буква, за которой следует цифра)
- L^* — множество строк из букв, включая пустую строку ε
- $L(L \cup D)^*$ — множество всех строк из букв и цифр, начинающихся с буквы
- D^+ — множество строк из одной или нескольких цифр

Регулярные выражения

- **Регулярные выражения** рекурсивно строятся из меньших регулярных выражений
- Каждое регулярное выражение *r* описывает язык *L*(*r*), который также рекурсивно определяется на основании языков, описываемых подвыражениями *r*
- **Правила определения регулярных выражений** над некоторым алфавитом A, и языки, описываемые этими регулярными выражениями
- **Базис**
  1. Пустая строка ε — регулярное выражение, *L*(ε) = {ε} — язык с пустой строкой
  2. Если a ∈ A (символ алфавита), то *a* — регулярное выражение, *L*(*a*) = {a} — язык со строкой длины 1
- **Индукция** (*r*, *s* — регулярные выражения)
  1. (*r*) | (*s*) — регулярное выражение, описывающее язык *L*(*r*) ∪ *L*(*s*)
  2. (*r*)(*s*) — регулярное выражение, описывающее язык *L*(*r*)*L*(*s*)
  3. (*r*)* — регулярное выражение, описывающее язык (*L*(*r*))*
  4. (*r*) — регулярное выражение, описывающее язык *L*(*r*) — можно заключить выражение в скобки без изменения описываемого им языка

- Унарный оператор * левоассоциативен и имеет наивысший приоритет
- Конкатенация имеет второй по величине приоритет и левоассоциативна
- Оператор | левоассоциативен и имеет наименьший приоритет

Алгебраические законы для регулярных выражений

Закон	Описание
$r \mid s = s \mid r$	Оператор $\mid$ коммутативен
$r \mid (s \mid t) = (r \mid s) \mid t$	Оператор $\mid$ ассоциативен
$r\,(st) = (rs)\,t$	Конкатенация ассоциативна
$r\,(s \mid t) = rs \mid rt;$ $(s \mid t)\,r = sr \mid tr$	Конкатенация дистрибутивна над $\mid$
$\epsilon r = r\epsilon = r$	ϵ является единичным элементом по отношению к конкатенации
$r^* = (r \mid \epsilon)^*$	ϵ гарантированно входит в замыкание
$r^{**} = r^*$	Оператор * идемпотентен

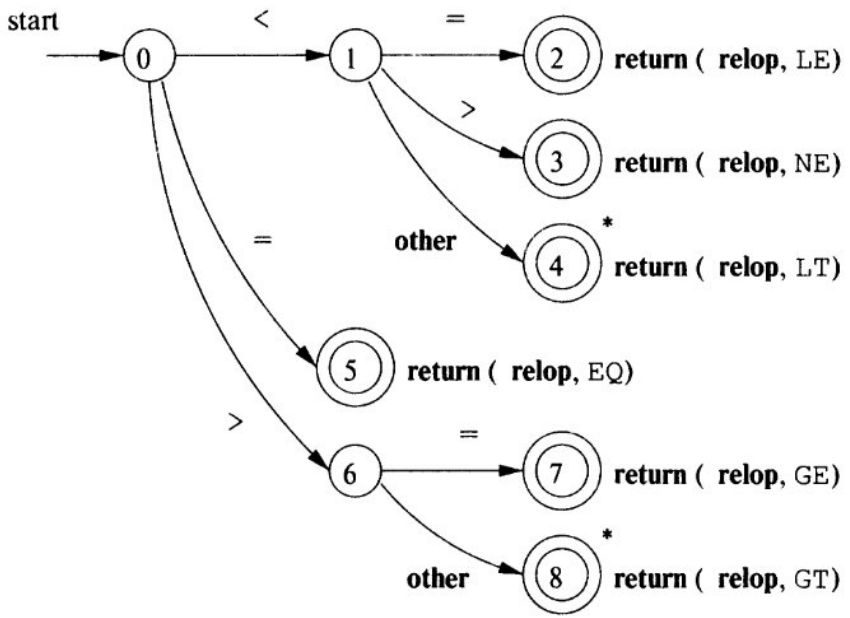
Расширения регулярных выражений

Выражение	Соответствие	Пример
c	Один неоператорный символ c	`a`
$\backslash c$	Символ c буквально	`\*`
"s"	Строка s буквально	`"**"`
`.`	Любой символ, кроме символа новой строки	`a.*b`
`^`	Начало строки	`^abc`
`$`	Конец строки	`abc$`
$[s]$	Любой символ из s	`[abc]`
$[\hat{}\,s]$	Любой символ, не входящий в s	`[^abc]`
$r*$	Нуль или более строк, соответствующих r	`a*`
$r+$	Одна или более строк, соответствующих r	`a+`
$r?$	Нуль или одно r	`a?`
$r\,\{m,n\}$	От m до n повторений r	`a{1,5}`
$r_1 r_2$	r_1, за которым следует r_2	`ab`
$r_1 \mid r_2$	r_1 или r_2	`a|b`
(r)	То же, что и r	`(a|b)`
r_1/r_2	r_1, если за ним следует r_2	`abc/123`

Диаграммы переходов

- **Диаграмма переходов** (transition diagram, state transition diagram) — ориентированный граф, задающий возможные состояния лексического анализатора и переходы между в ходе распознавания токенов
- **Состояния** (state) — узлы графа, представляет ситуацию, которая может возникнуть в процессе сканирования входного потока в поисках лексемы, соответствующей одному из нескольких шаблонов
- **Допускающее состояние** (конечное, принимающее, accepting, final) — искомая лексема найдена
- **Ребро** (дуга, edge) — показывает **переход** (transition) при чтении символа, которым помечена дуга
- **Детерминированная диаграмма переходов** (deterministic) — имеется не более одной дуги, выходящей из данного состояния с данным символом среди ее меток

Начальное состояние (start, initial)



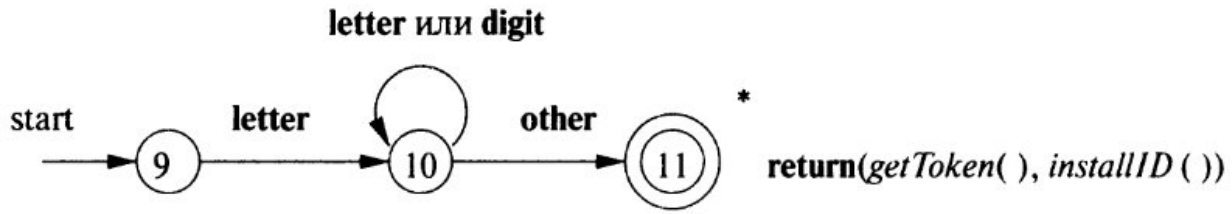
Конечное состояние

Диаграмма переходов для распознавания операторов отношений (relop)

Конечное состояние и возврат указателя *forward* на предыдущий символ

Распознавание зарезервированных слов и идентификаторов

- **Вариант 1** — внести зарезервированные слова в таблицу символов



- **Вариант 2** — создать отдельные диаграммы переходов для каждого ключевого слова

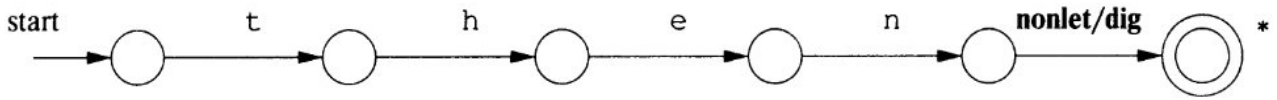
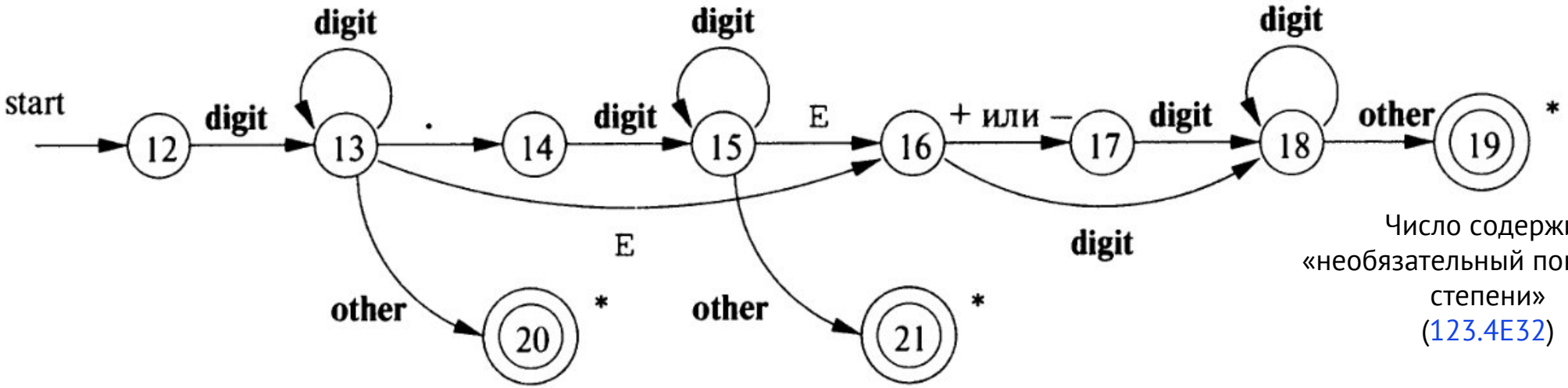


Диаграмма переходов для беззнаковых чисел

- Если первый встреченный символ — цифра, переходим в состояние 13
- В состоянии 13 можем считать любое количество дополнительных символов — если попадется символ, отличный от цифры, точки или E, значит, имеем дело с целым числом (состояние 20)



Целочисленный литерал (123)

Конец дробной части числа, у которого нет показателя степени (123.7438)

Число содержит «необязательный показатель степени» (123.4E32)

Архитектура лексического анализатора на основе диаграммы переходов

- Каждому состоянию (state) соответствует фрагмент кода, обрабатывающий его

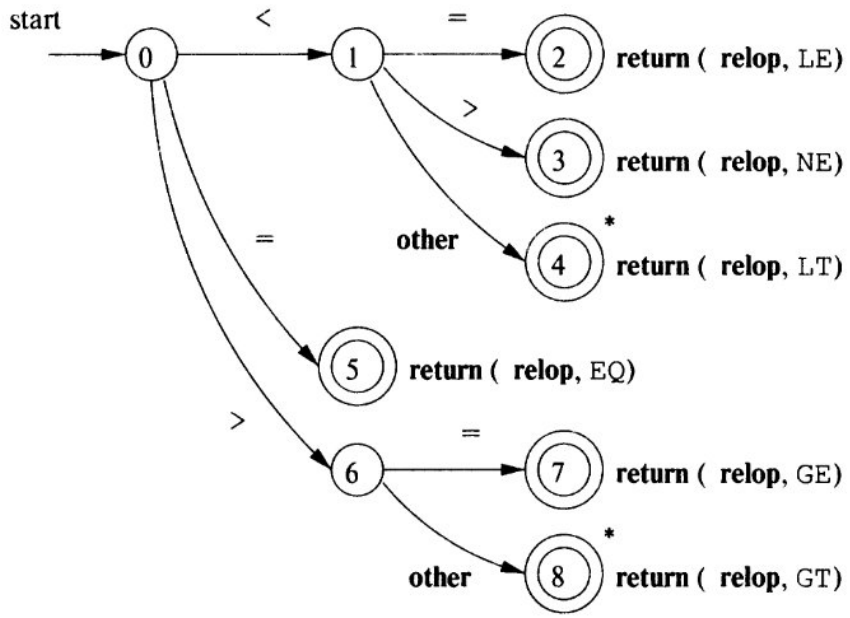


Диаграмма переходов для распознавания операторов отношений (relop)

```
TOKEN getRelop()
{
   TOKEN retToken = new(RELOP);
   while(1) { /* Обработка символов до тех пор, пока
                 не будет выполнен возврат из функции
                 или не будет обнаружена ошибка */
      switch(state) {
         case 0: c = nextChar();
                 if ( c == '<' ) state = 1;
                 else if ( c == '=' ) state = 5;
                 else if ( c == '>' ) state = 6;
                 else fail(); /* Это не лексема relop */
                 break;
         case 1: ...
         ...
         case 8: retract(); // вернуть с назад во входной поток
                 retToken.attribute = GT;
                 return(retToken);
      }
   }
}
```

Архитектура лексического анализатора на основе диаграммы переходов

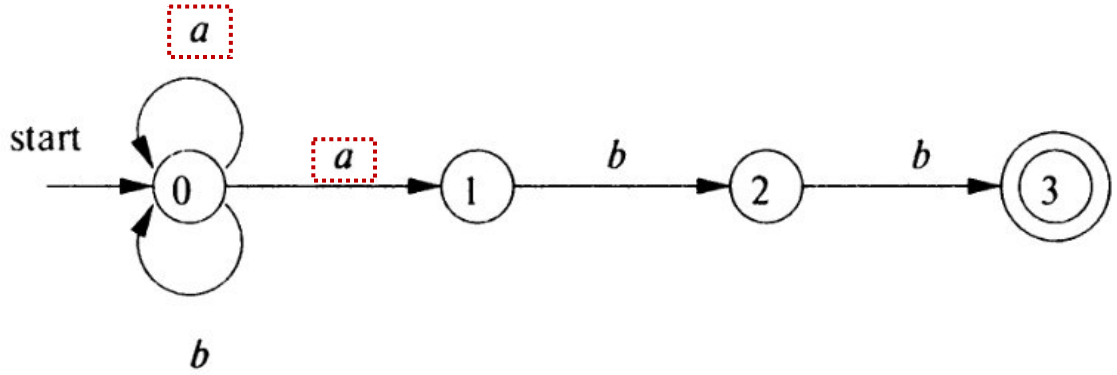
1. Можно последовательно испытывать **диаграммы переходов для каждого токена**
   - функция fail() сбрасывает значение указателя forward для обработки новой диаграммы переходов
2. Можно работать с **разными диаграммами переходов «параллельно»**, передавая очередной считанный символ им всем и выполняя соответствующий переход в каждой из диаграмм переходов
3. **Объединение всех диаграмм переходов в одну** — диаграмма переходов считывает символы до тех пор, пока возможные следующие состояния не оказываются исчерпаны, после этого выбирается наибольшая лексема, соответствующая некоторому шаблону

Конечные автоматы (finite automata)

- **Конечный автомат** — граф, подобный диаграмме переходов, включает:
  - входной алфавит *A* (конечное множество входных символов)
  - множество внутренних состояний *S*
  - начальное состояние s_0
  - множество конечных состояний *F*
  - функцию переходов *t*(*state*, *a*) —> *state*
- Конечные автоматы являются **распознавателями** (recognizer), отвечают "да" или "нет" для каждой возможной входной строки
- **Недетерминированные конечные автоматы** (НКА, nondeterministic finiteautomata — NFA) не имеют ограничений на свои дуги
  - символ может быть меткой нескольких дуг, исходящих из одного и того же состояния
  - одна из возможных меток — пустая строка ε
- **Детерминированные конечные автоматы** (ДКА, deterministic finiteautomata — DFA) — для каждого состояния и каждого символа входного алфавита имеют ровно одну дугу с указанным символом, покидающим это состояние
- Как детерминированные, так и недетерминированные конечные автоматы способны распознавать одни и те же языки — регулярные языки (regular language), которые могут быть описаны регулярными выражениями

Недетерминированные конечные автоматы (НКА, NFA)

- **Недетерминированный конечный автомат** (НКА):
  1. Множество состояний S
  2. Множество входных символов Σ (входной алфавит), не включает пустую строку ε
  3. Функция переходов — для каждого состояния и каждого символа из Σ U {ε} дает множество следующих состояний (next state)
  4. Стартовое состояние s_0 из S
  5. Множество допускающих (конечных) состояний *F*, являющееся подмножеством *S*



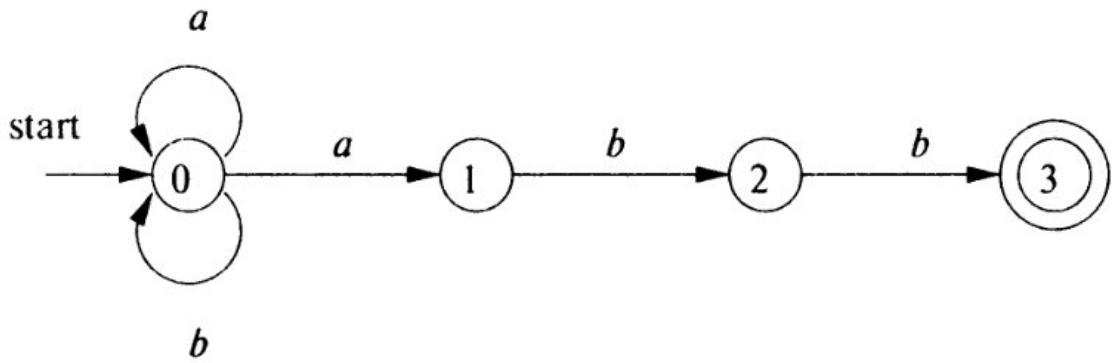
Состояние	*a*	*b*	ϵ
0	{0, 1}	{0}	∅
1	∅	{2}	∅
2	∅	{3}	∅
3	∅	∅	∅

Таблица переходов

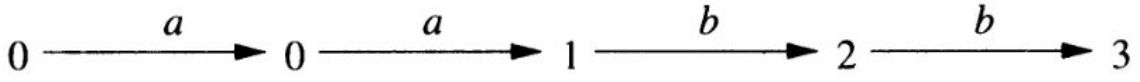
Недетерминированный конечный автомат
распознающий язык регулярного выражения (a | b)* abb — строки из a и b, заканчивающиеся подстрокой abb

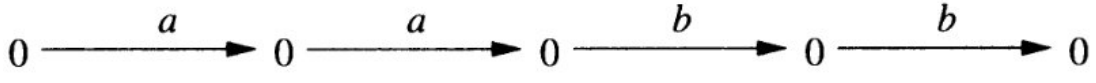
Недетерминированные конечные автоматы (НКА, NFA)

- НКА допускает, или **принимает** (accept), входную строку *x* тогда и только тогда, когда в графе переходов существует путь от начального состояния к одному из допускающих, такой, что метки дуг вдоль этого пути соответствуют строке *x*

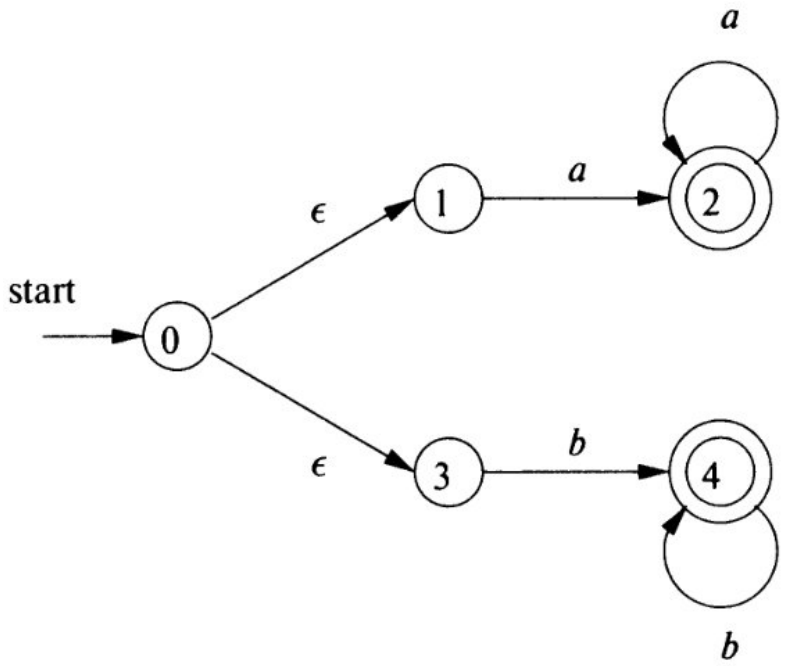


Строка aabb принимается НКА

Путь 1 

Путь 2 

Недетерминированные конечные автоматы (НКА, NFA)



НКА, принимающий aa* | bb*

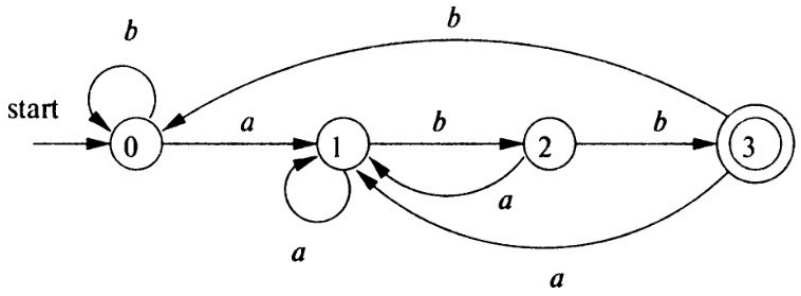
Детерминированные конечные автоматы (ДКА, DFA)

- **Детерминированный конечный автомат** (ДКА) — частный случай НКА:
  1. Отсутствуют переходы для входа ε
  2. Для каждого состояния *s* и входного символа *a* имеется ровно одна дуга, выходящая из *s* и помеченная *a*
- ДКА является конкретным алгоритмом распознавания строк
- Любое регулярное выражение и каждый НКА могут быть преобразованы в ДКА, принимающий тот же язык
- При построении лексического анализатора реализуется (симулируется, моделируется) детерминированный конечный автомат

Алгоритм моделирования ДКА (simulating a DFA)

- Вход: входная строка *x*, завершенная символом конца файла eof; детерминированный конечный автомат *D* с начальным состоянием s_0, принимающими состояниями *F* и функцией переходов move
- Выход: ответ «да», если *D* принимает (распознает) *x*, и «нет» в противном случае

```
s = s0
c = nextChar()
while (c != eof) {
  s = move(s, c);
  c = nextChar();
}
if (s in F) return "да"
else return "нет"
```



ДКА, принимающий язык (a | b)* abb

ababb

↓

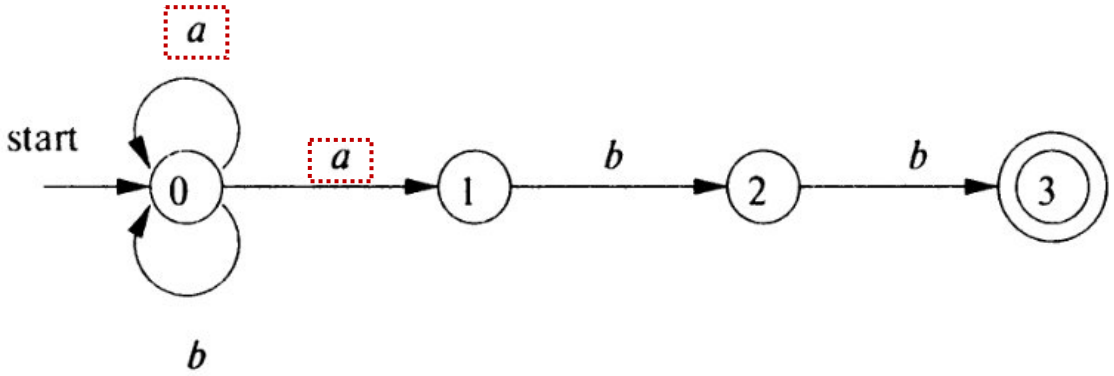
0, 1, 2, 1, 2, 3

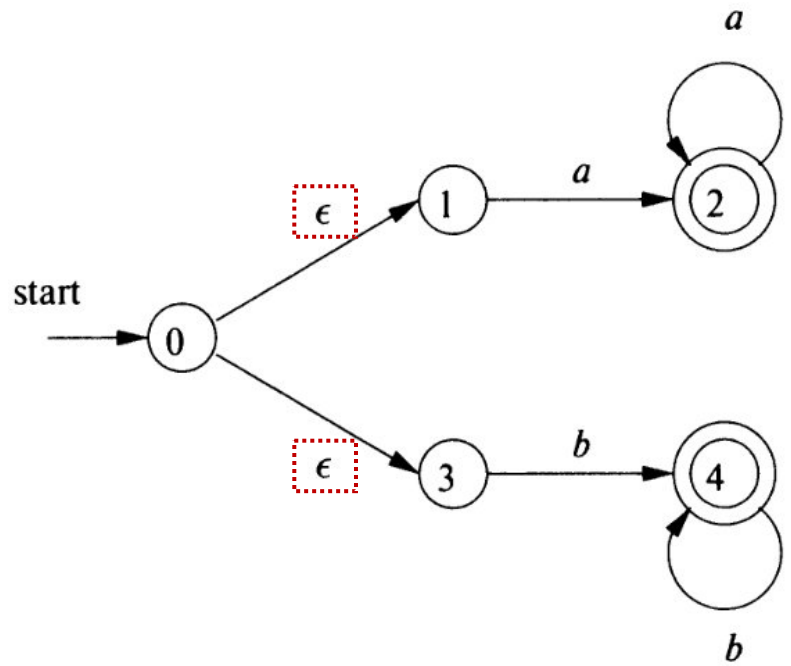
↓

да

Переход от регулярных выражений к конечным автоматам

- Регулярное выражение представляет собой способ описания лексических анализаторов
- **Реализация разбора регулярного выражения требует моделирования ДКА** или, возможно, моделирования НКА
- При работе с НКА может требоваться делать выбор перехода для входного символа или для ε, моделирование НКА существенно сложнее, чем моделирование ДКА
- Важной является **задача конвертации НКА в ДКА**, который принимает тот же язык





Переход от регулярных выражений к конечным автоматам

- Генераторы лексических анализаторов и системы обработки строк часто начинают работу с регулярного выражения
- Возможные варианты реализации — преобразовывать регулярные выражения в ДКА или в НКА

Автомат	Начальное построение	Работа над строкой
НКА	$O(\|r\|)$	$O(\|r\| \times \|x\|)$
ДКА: типичный случай	$O(\|r\|^3)$	$O(\|x\|)$
ДКА: наихудший случай	$O(\|r\|^2 2^{\|r\|})$	$O(\|x\|)$

Вычислительная сложность начального построения и обработки одной строки x различными методами распознавания языка регулярных выражений ($|r|$ — число состояний, $|x|$ — длина входной строки)

- Если доминирует время обработки одной строки, как в случае построения лексического анализатора, очевидно, что следует предпочесть ДКА
- В программах наподобие grep, в которых автомат работает только с одной строкой, обычно предпочтительнее использовать НКА
- Пока $|x|$ не становится порядка $|r|^3$, нет смысла переходить к ДКА