Курс «Компиляторные технологии»

Лекция 9

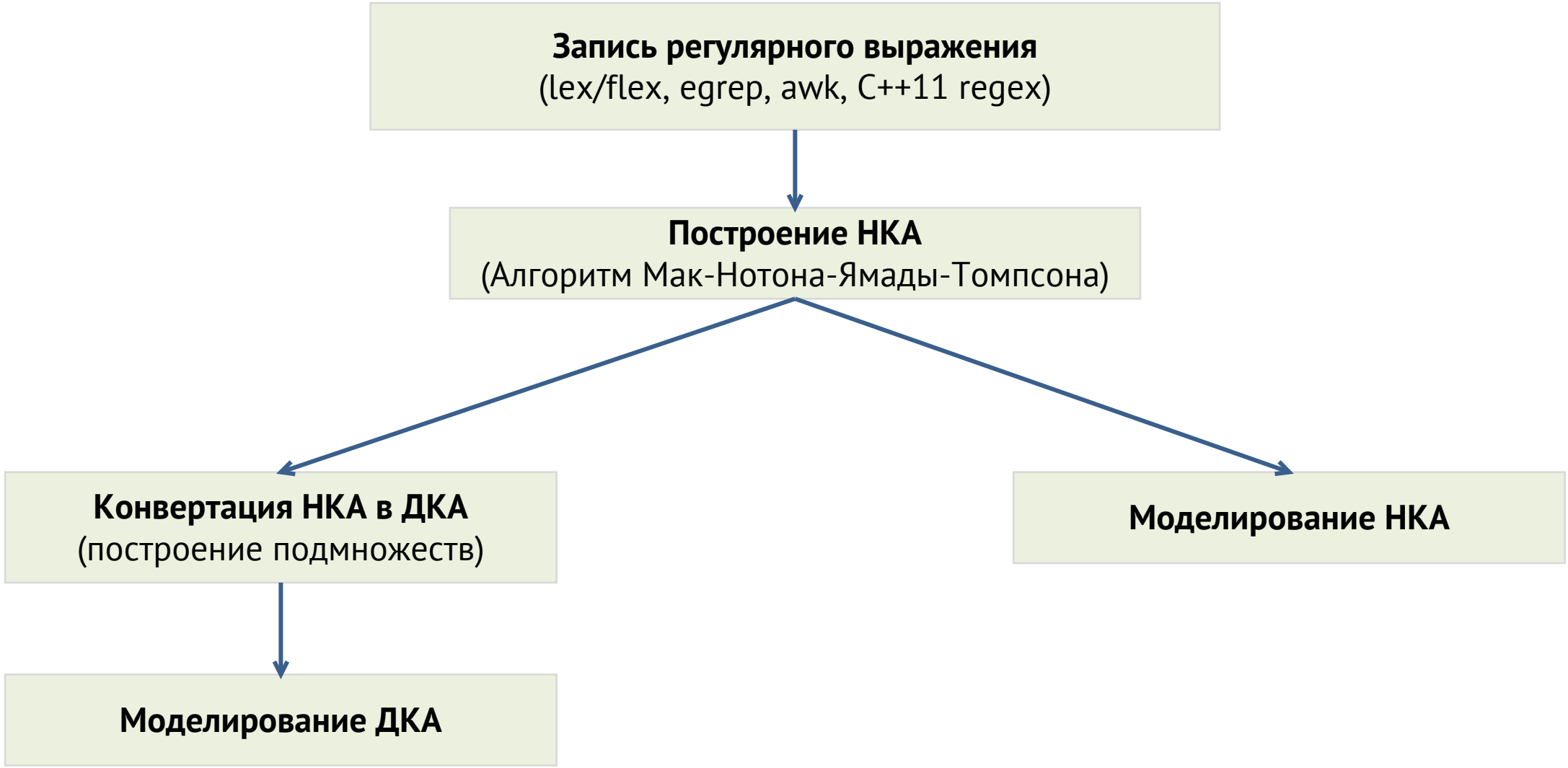
Лексический анализ (2)

Курносов Михаил Георгиевич

www.mkurnosov.net

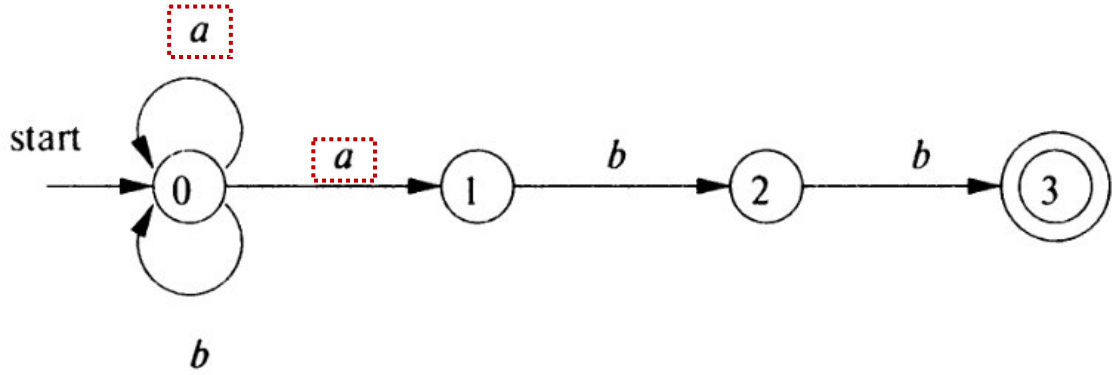
Сибирский государственный университет телекоммуникаций и информатики
Весенний семестр

Переход от регулярных выражений к конечным автоматам



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

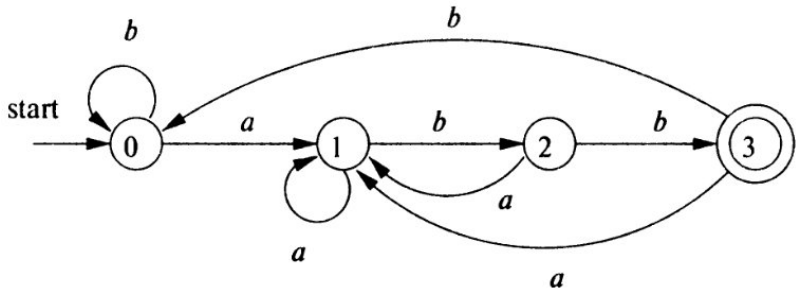
Детерминированные конечные автоматы (ДКА, DFA)

- **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

↓

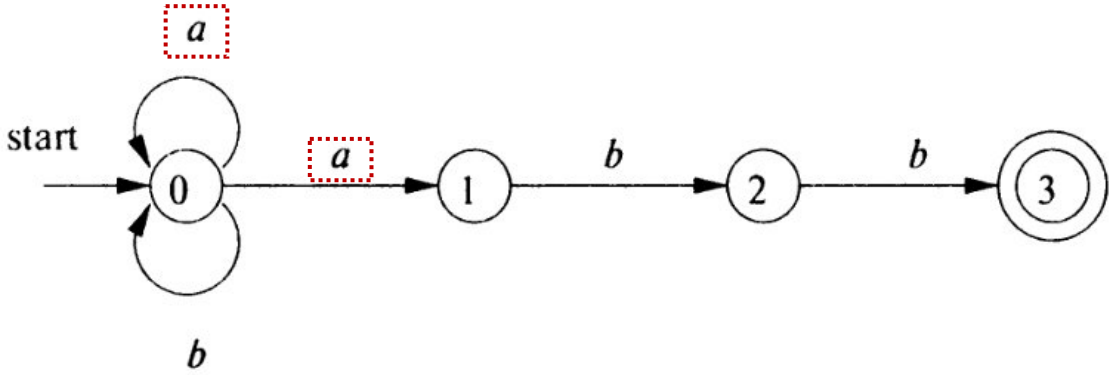
0, 1, 2, 1, 2, 3

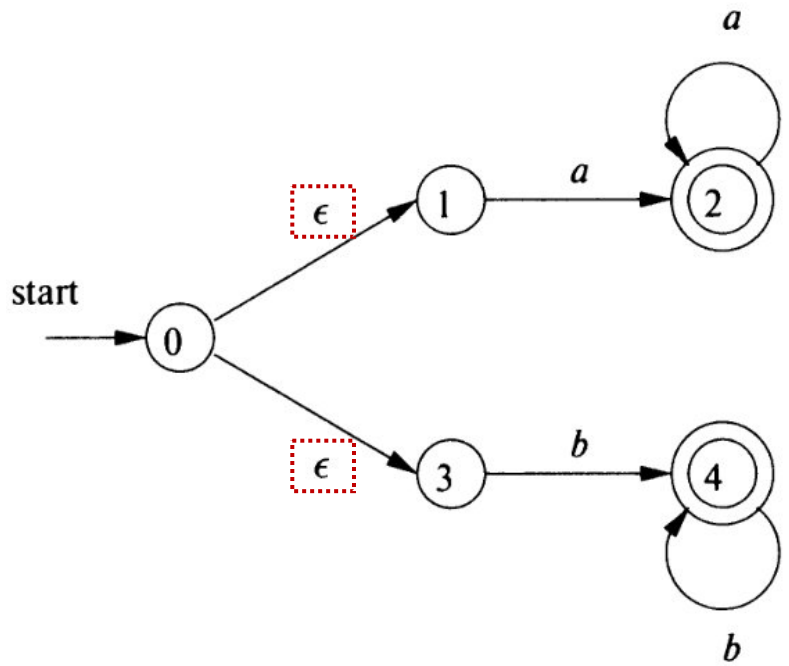
↓

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





От регулярных выражений к автоматам

- Общая идея — каждое состояние строящегося ДКА соответствует множеству состояний НКА
- После чтения входной строки $a_1a_2....a_n$ ДКА находится в состоянии, соответствующем множеству состояний, которых может достичь из своего стартового состояния НКА по пути, помеченному $a_1a_2....a_n$
- **Возможна ситуация, когда количество состояний ДКА экспоненциально зависит от количества состояний НКА, что может привести к сложностям при реализации такого ДКА**
- **Для реальных языков НКА и ДКА имеют примерно одинаковое количество состояний, без экспоненциального поведения**
- Два сценария
  - **Метод «построение подмножеств»** (subset construction) — конвертирование НКА в ДКА
  - Метод прямого моделирования НКА для случаев отличных от лексического анализа, когда преобразование НКА в ДКА требует больше времени, чем непосредственное моделирование

Конвертирование НКА в ДКА

Алгоритм. Построение подмножества (subset construction) ДКА из НКА
Вход: НКА *N*
Выход: ДКА D, принимающий тот же язык, что и *N*

Изначально в *Dstates* содержится только одно состояние, ϵ-*closure* (s_0), и оно не помечено
while (в *Dstates* имеется непомеченное состояние T) {
    Пометить T;
    for (каждый входной символ a) {
        $U = \epsilon$-*closure* $(move\,(T, a))$;
        if ($U \notin$ *Dstates*)
            Добавить U в *Dstates* как непомеченное состояние;
        Dtran $[T, a] = U$;
    }
}

- Строится таблица переходов *Dtran* для *D*
- Каждое состояние *D* — это множество состояний *НКА*
- *Dtran* строится так, чтобы «параллельно» моделировать все возможные переходы, которые *N* может выполнить для данной входной строки

Операция	Описание
ϵ-*closure* (s)	Множество состояний НКА, достижимых из состояния s при одном ϵ-переходе
ϵ-*closure* (T)	Множество состояний НКА, достижимых из состояния s из множества T при одном ϵ-переходе; $= \cup_{s \in T} \epsilon$-*closure* (s)
move (T, a)	Множество состояний НКА, в которые имеется переход из некоторого состояния $s \in T$ при входном символе a

Конвертирование НКА в ДКА

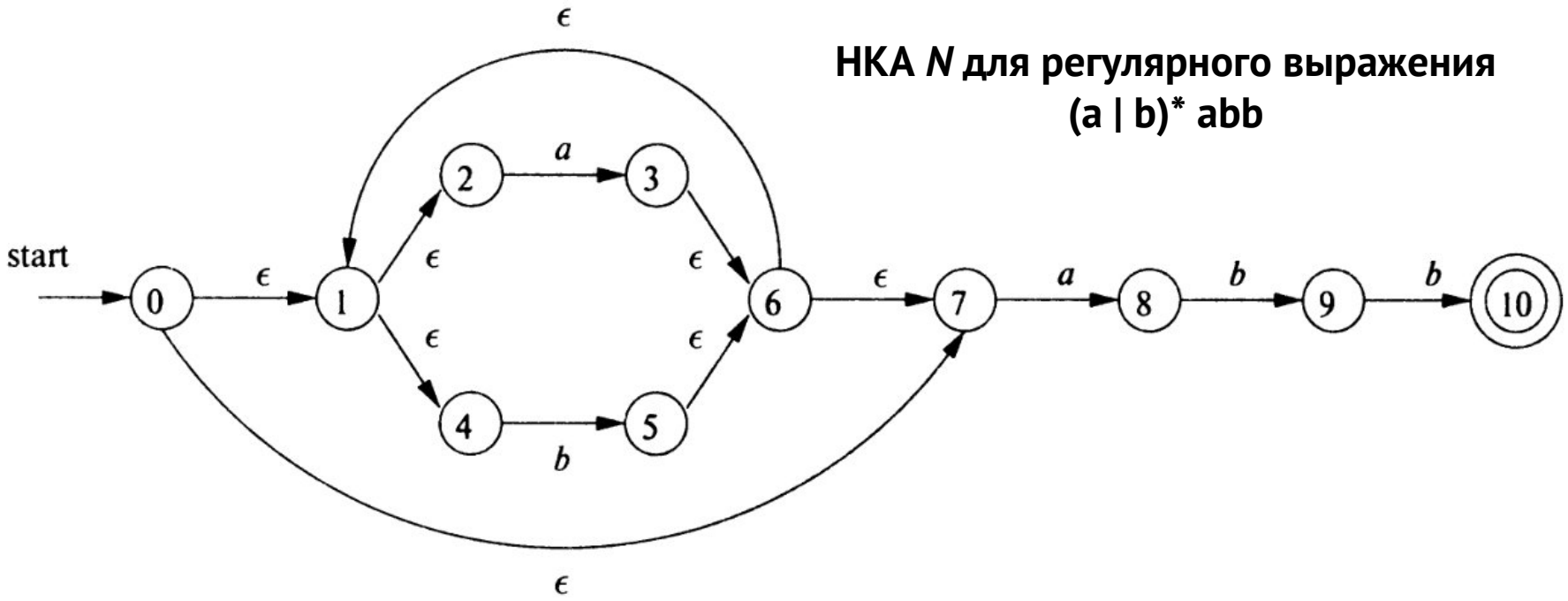
- Стартовым состоянием *D* является ε-*closure*(s_0), а принимающими состояниями *D* являются те множества состояний *N*, которые включают как минимум одно принимающее состояние *N*
- Для завершения описания построения подмножества требуется показать, как вычислить ε-*closure*(*T*) для произвольного множества состояний *T* недетерминированного конечного автомата

```
Поместить все состояния T в стек stack;
Инициализировать ϵ-closure (T) множеством T;
while ( stack не пуст ) {
        Снять со стека stack верхний элемент t;
        for ( каждое состояние u с дугой из t в u, помеченной ϵ )
                if ( u ∉ ϵ-closure (T) ) {
                        Добавить u в ϵ-closure (T);
                        Поместить u в stack;
                }
}
```

Вычисление ε-*closure*(*T*)

Конвертирование НКА в ДКА

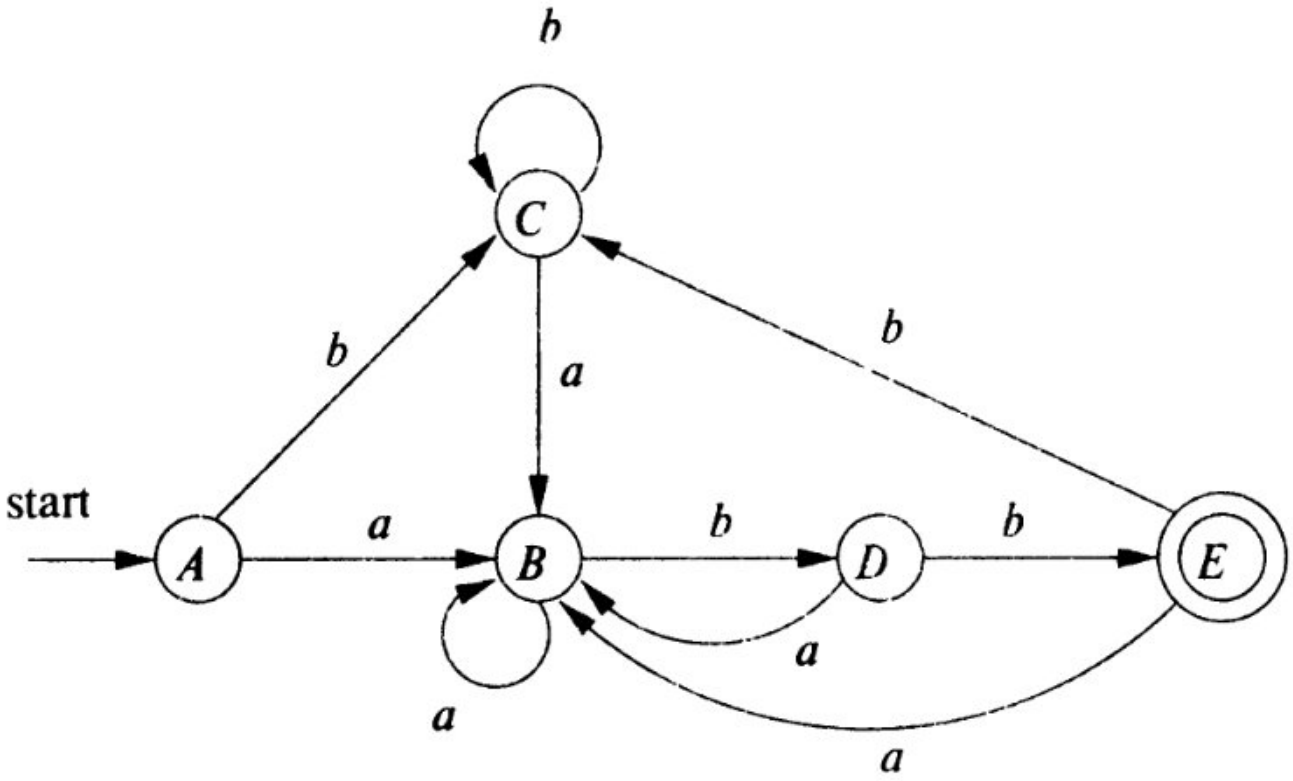
НКА *N* для регулярного выражения
(a | b)* abb



- Стартовое состояние *A* эквивалентного ДКА ε-*closure*(0) = {0, 1, 2, 4, 7}
- Пометим *A* и вычислим *Dtran*[*A*, *a*] = ε-*closure*(*move*(*A*, *a*)) и *Dtran*[*A*, *b*] = ε-*closure*(*move*(*A*, *b*))
- *move*(*A*, *a*) = {3, 8}, ε-*closure*(*move*(*A*, *a*)) = {1, 2, 3, 4, 6, 7, 8}
- *Dtran*[*A*, *a*] = {1, 2, 3, 4, 6, 7, 8} = *B*
- Найдем *Dtran*[*A*, *b*] = ε-*closure*(*move*(*A*, *b*)) = ε-*closure*({5}) = {1, 2, 4, 5, 6, 7} = *C*

Состояния НКА	Состояния ДКА	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 4, 5, 6, 7, 10\}$	E	B	C

Конвертирование НКА в ДКА



Результат применения построения подмножеств к НКА

Моделирование НКА

- Стратегия, использующаяся в некоторых текстовых редакторах — построение НКА из регулярного выражения и его моделировании с использованием методики, сходной с построением подмножеств "на лету"

- **Алгоритм. Моделирование НКА**
- Вход: входная строка *x* с завершающим символом eof; НКА *N* с начальным состоянием s_0, принимающими состояниями *F* и функцией переходов *move*.
- Выход: ответ "да", если *N* принимает *x*; ответ "нет" в противном случа

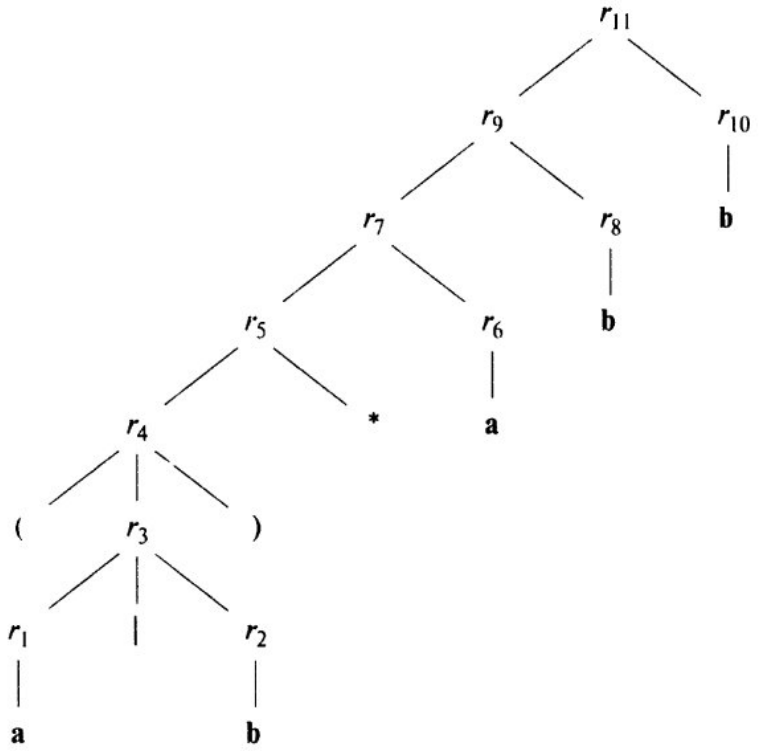
```
1)  S = ε-closure(s0);
2)  c = nextChar();
3)  while ( c != eof ) {
4)          S = ε-closure(move(S, c));
5)          c = nextChar();
6)  }
7)  if ( S ∩ F != ∅ ) return "да";
8)  else return "нет";
```

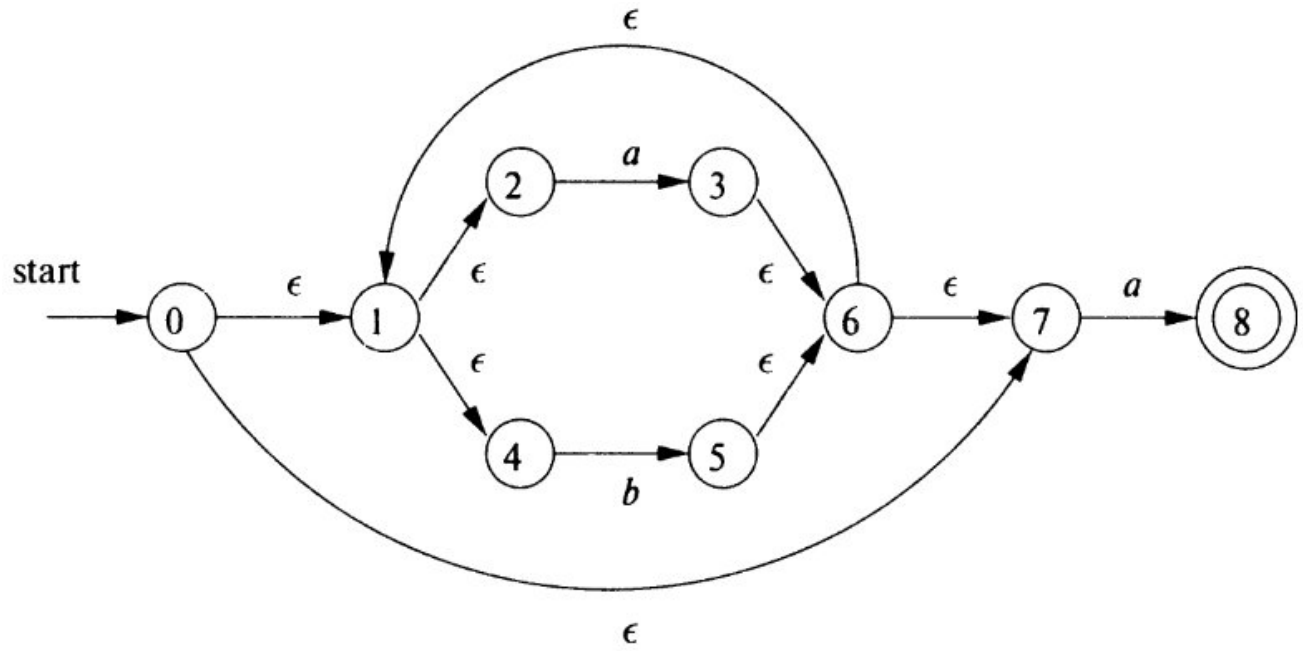
- Алгоритм поддерживает множество текущих состояний S, которые достигаются из s_0 по пути, помеченному считанными символами входной строки
- Если *c* — очередной входной символ, считанный *nextChar*(), то сначала вычисляется *move*(5, *c*), а затем — замыкание с применением ε-*closure*()

Вычислительная сложность: *0*(*k*(*n* + *m*)),
k — длина входной строки *x*, *n* — число состояний НКА, *m* — число переходов НКА

Построение НКА из регулярного выражения

- Алгоритм синтаксически управляемый — рекурсивно работает с деревом разбора регулярного выражения
- Для каждого подвыражения алгоритм строит НКА с единственным принимающим состоянием
- **Алгоритм** Мак-Нотона-Ямады-Томпсона (McNaughton-Yamada-Thompson) преобразования регулярного выражения в НКА
- Вход: регулярное выражение *r* над алфавитом *S*
- Выход: НКА *N*, принимающий регулярный язык *L*(*r*)





Эффективность обработки строк

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

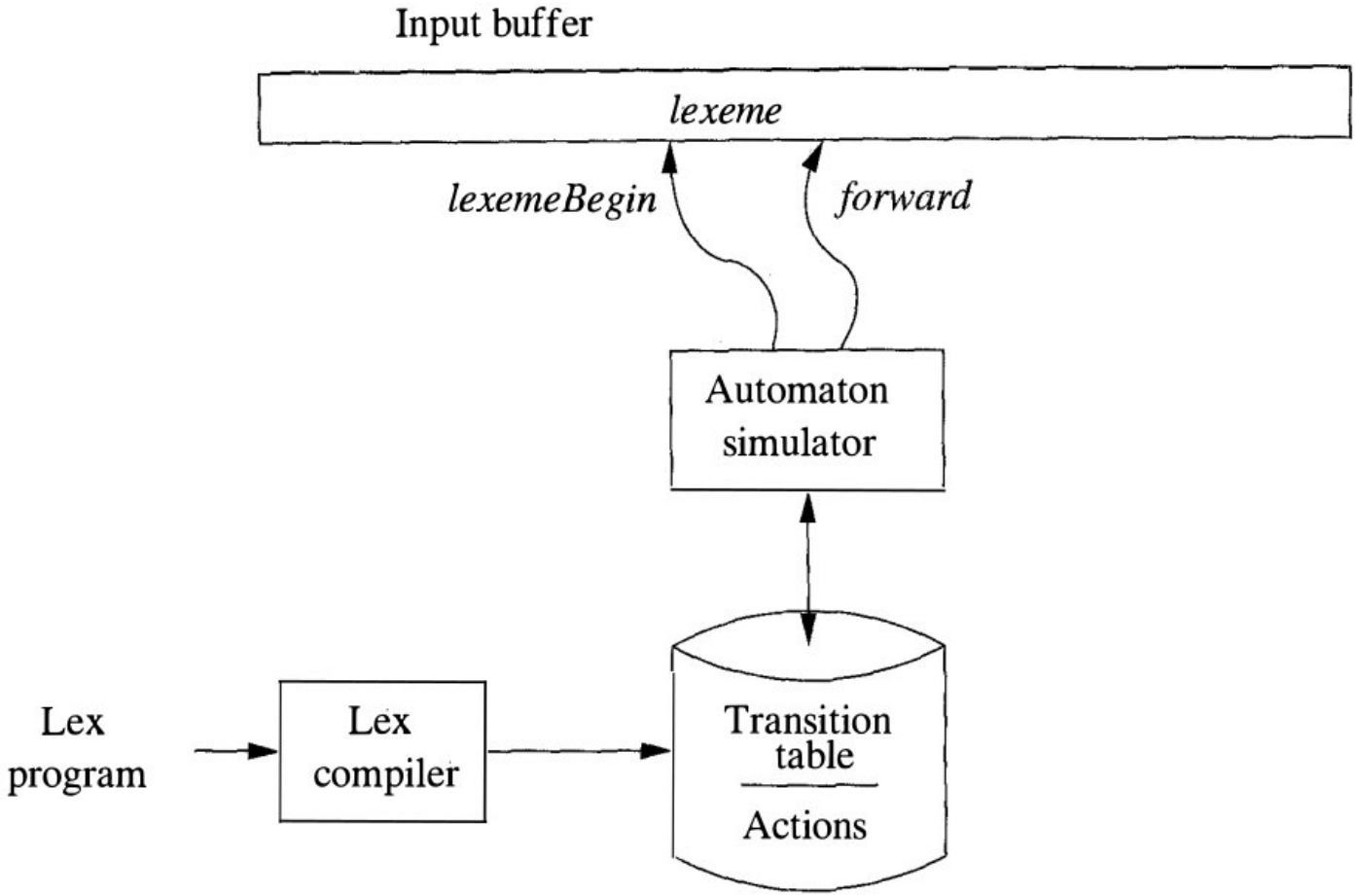
Автомат	Начальное построение	Работа над строкой
НКА	$O(\|r\|)$	$O(\|r\| \times \|x\|)$
ДКА: типичный случай	$O(\|r\|^3)$	$O(\|x\|)$
ДКА: наихудший случай	$O(\|r\|^2 2^{\|r\|})$	$O(\|x\|)$

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

Разработка генератора лексических анализаторов

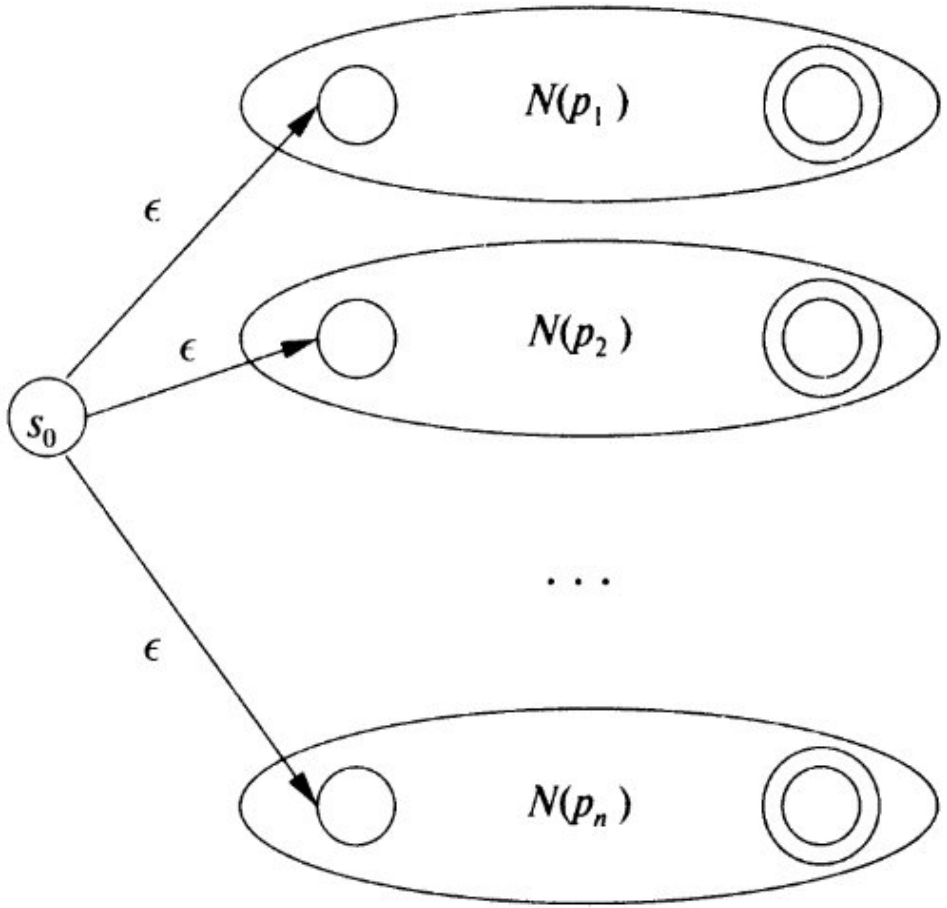
- Генераторы лексических анализаторов и системы обработки строк часто начинают работу с регулярного выражения
- Возможные варианты реализации — преобразовывать регулярные выражения в ДКА или в НКА



Разработка генератора лексических анализаторов: НКА

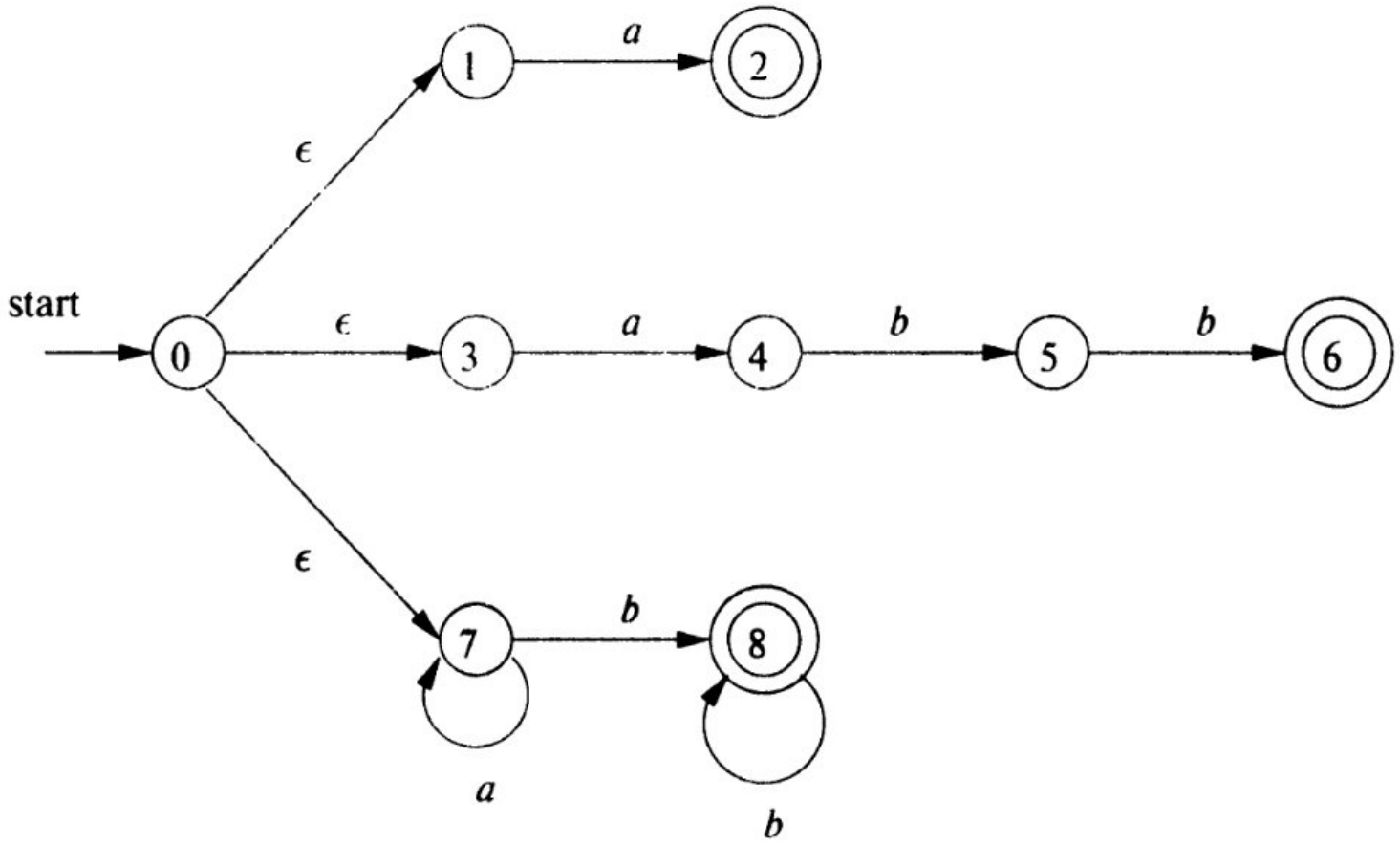
- Преобразование каждого регулярного выражения из Lex в недетерминированный конечный автомат (алгоритм Мак-Нотона-Ямады-Томпсона преобразования регулярного выражения в НКА)
- Объединение всех НКА в один с использованием ε-переходов в стартовые состояния всех НКА

a { действие A_1 для шаблона p_1 }
abb { действие A_2 для шаблона p_2 }
$\mathbf{a^*b^+}$ { действие A_3 для шаблона p_3 }



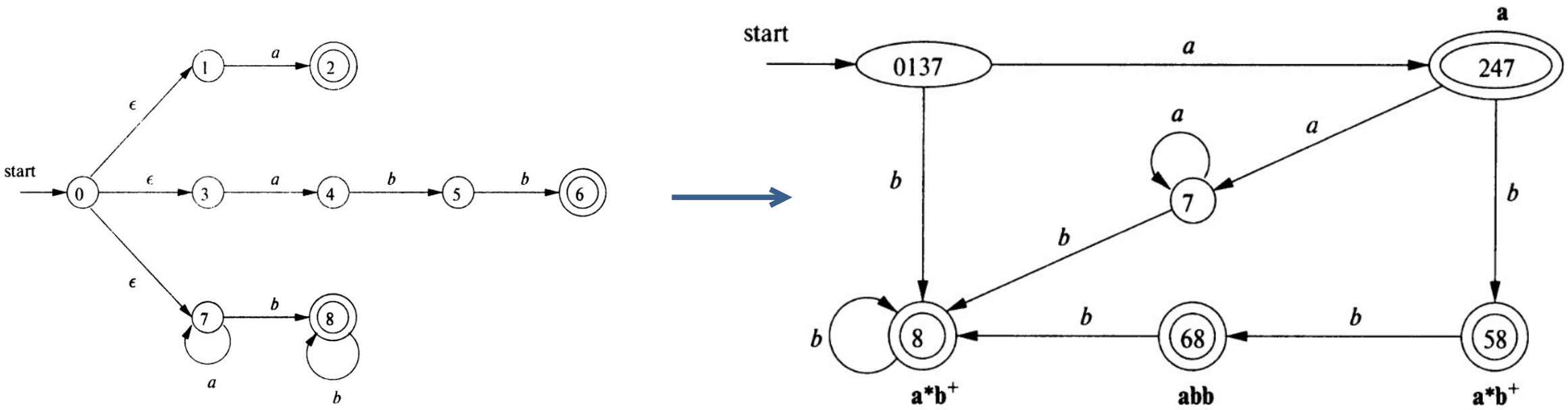
Разработка генератора лексических анализаторов: НКА

- Преобразование каждого регулярного выражения из Lex в недетерминированный конечный автомат (алгоритм Мак-Нотона-Ямады-Томпсона преобразования регулярного выражения в НКА)
- Объединение всех НКА в один с использованием ε-переходов в стартовые состояния всех НКА



Разработка генератора лексических анализаторов: ДКА

- Преобразование НКА для всех шаблонов в эквивалентный ДКА с использованием построения подмножеств
- ДКА при наличии в нем одного или нескольких принимающих состояний НКА определяется первый шаблон программы Lex, представленный принимающим состоянием, и этот шаблон является выходом данного состояния ДКА



Оптимизация распознавателей на основе ДКА

- Прямое построение ДКА из регулярного выражения
- Минимизация количества состояний ДКА — Для каждого ДКА существует ДКА с минимальным количеством состояний, принимающий тот же язык
- ДКА с минимальным количеством состояний для данного языка является единственным с точностью до имен состояний автомата