

Visibly Pushdown Languages.

Конъюнктивные языки.

Древесные языки



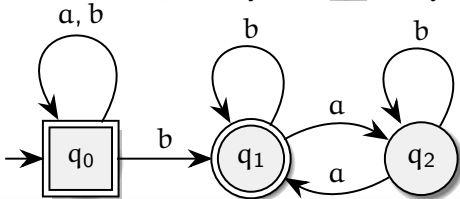
Теория формальных языков
2023 г.



AFA

Переключающиеся автоматы (Alternating Finite Automata) содержат узлы двух типов: конъюнктивные и дизъюнктивные (\forall -узлы и \exists -узлы, соответственно).

В примере AFA \mathcal{A} ниже  — \exists -узлы;  — \forall -узлы.



ДКА, содержащий только $\{q_1, q_2\}$, принимает слова, содержащие чётное число букв a . По каждой очередной букве b данный автомат должен быть принимающим по обеим исходящим из q_0 веткам. Значит, между каждыми двумя буквами b в словах из $\mathcal{L}(\mathcal{A})$ может стоять лишь чётное число букв a .



Выразительная сила AFA

- Все AFA языки регулярные, т.к. регулярные языки замкнуты относительно пересечений.
- Позволяют легко моделировать lookahead-выражения под итерацией, или lookahead-выражения внутри lookahead-выражений. Например, автомат на предыдущем слайде представляет выражение $(a \mid b(? = (b^* a b^* a)^* b^* \$))^*$.



Скобочные языки

Положим, что для каждого нетерминала правила имеют следующий вид:

$$N \rightarrow (N\Phi)N$$

Соответствующие языки будут замкнуты относительно пересечения и дополнения (но не итерации и конкатенации).
Очень ограниченный класс языков, в котором каждый символ является открывающей или закрывающей скобкой.



Visibly Pushdown Languages

Разделим входной алфавит PDA \mathcal{A} на три класса:

- Σ_c — вызывающий алфавит. При чтении его элементов в стек можно только класть.
- Σ_r — возвращающий алфавит. При чтении его элементов из стека можно только доставать.
- Σ (просто) — внутренний алфавит. Не меняет стек (a la конечный автомат).

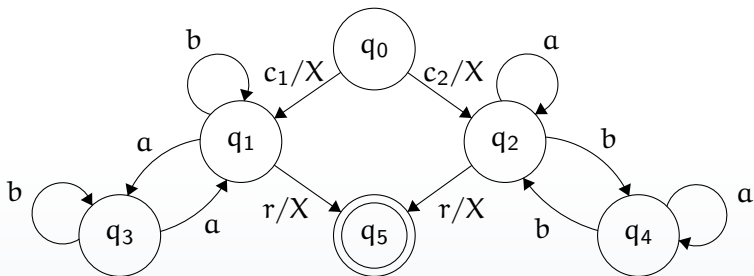
Дополнительные допущения:

- завершение по финальному состоянию;
- если достигли дна стека (символ \perp), то символы из Σ_r не меняют его (т.е. дно стека вытолкнуть из него нельзя).



Примеры: два эквивалентных VPDA

Алфавит $\Sigma_c = \{c_1, c_2\}$, $\Sigma_r = \{r\}$, $\Sigma = \{a, b\}$. Язык $c_1(b^*(ab^*ab^*)^*)r|c_2(a^*(ba^*ba^*)^*)r$. Он регулярный, и можно построить соответствующий автомат, который полностью игнорирует состояния памяти (пишем в память всегда X , и достаём тот же X).

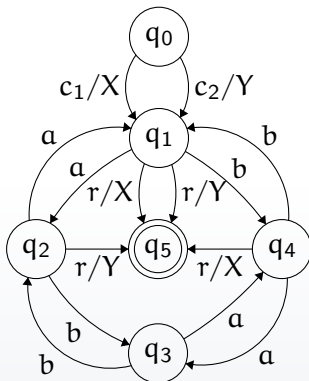




Примеры: два эквивалентных VPDA

Алфавит $\Sigma_c = \{c_1, c_2\}$, $\Sigma_r = \{r\}$, $\Sigma = \{a, b\}$. Язык $c_1(b^*(ab^*ab^*)^*)r|c_2(a^*(ba^*ba^*)^*)r$.

Можно пойти другим путём: считать сразу чётности всех букв, но корректность выхода в конечное состояние определять по символу, который сохранён в памяти. VPDA получится такого же размера.





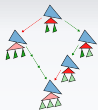
Теорема Майхилла–Нероуда

(упрощённая форма)

Язык \mathcal{L} является VPL \Rightarrow множество сбалансированных слов (т.е. таких, каждый символ из Σ_c в которых имеет соответствующий символ из Σ_r) разбивается на конечное число классов эквивалентности по Майхиллу–Нероуду относительно \mathcal{L} .

Напомним:

$$w_1 \equiv_{\mathcal{L}} w_2 \Leftrightarrow \forall u, v (u w_1 v \in \mathcal{L} \Leftrightarrow u w_2 v \in \mathcal{L})$$



Расширения КС-грамматик?

$O(n^3)$ для КС-грамматик — оценка с запасом. Что можно поместить в такой запас?

- Местность функций соответствует объявленной сигнатуре;
- ... даже при вложенных и перекрестных вызовах этих функций!

Примеры задач:

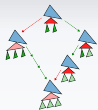
```
int f(int,...,int)
    { ... }
```

```
int g(int,...,int)
    { ... }
```

```
int main() {
    x1 = f(0,...,0);
    x2 = g(0,...,0);
}
```

```
int f(int,...,int)
    { ... }
```

```
int main() {
    x = f(
        f(0,...,0),
        ...,
        f(0,...,0))
}
```



Конъюнктивные грамматики

Конъюнктивная грамматика G — грамматика, правила которой имеют вид

$$A_i \rightarrow \Phi_1 \& \dots \& \Phi_n$$

где A_i — нетерминал; Φ_j — строки в смешанном алфавите терминалов и нетерминалов.

Грамматика для $\{(a^n b)^k \mid n, k \geq 1\}$:

$$S \rightarrow SA \& Cb \mid A$$

$$A \rightarrow aA \mid ab$$

$$C \rightarrow aCa \mid B$$

$$B \rightarrow BA \mid b$$



Язык равенства

Связанность переменных тоже можно проверить. См. грамматику для $\{wsw \mid w \in \{a, b\}^*\}$.

Язык неравенства:	Язык равенства:
$S_w \rightarrow C \mid ED$	$S_w \rightarrow C \ \& \ D$
$C \rightarrow XCX \mid XEc \mid cEX$	$C \rightarrow XCX \mid c$
$D \rightarrow aB \mid bA$	$D \rightarrow aA \ \& \ aD \mid bB \ \& \ bD \mid cE$
$A \rightarrow XAX \mid cEa$	$A \rightarrow XAX \mid cEa$
$B \rightarrow XBX \mid cEb$	$B \rightarrow XBX \mid cEb$
$E \rightarrow XE \mid \varepsilon$	$E \rightarrow XE \mid \varepsilon$
$X \rightarrow a \mid b$	$X \rightarrow a \mid b$



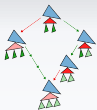
Вопросы парсинга

Это позволяет проверять условия вроде «все переменные объявлены» или «все вызываемые функции существуют». В любой позиции, где должен заканчиваться объявленный идентификатор, для этого вызываем конструкцию равенства по всем α из алфавита имён:

$$\begin{aligned}C &\rightarrow C_{\text{len}} \ \& \ C_{\text{iter}} \\C_{\text{len}} &\rightarrow LC_{\text{len}}L \mid LC_{\text{mid}}L \\C_{\text{mid}} &\rightarrow P \mid PAP \\C_{\text{iter}} &\rightarrow C_{\alpha}\alpha \ \& \ C_{\text{iter}}\alpha \\C_{\alpha} &\rightarrow LC_{\alpha}L \mid \alpha AP\end{aligned}$$

Здесь P — разделитель, L — буква из алфавита имён, A — произвольная строка.

Неудобство связано с линейным разрастанием количества правил при расширении алфавита имён.



Real-time клеточные автоматы

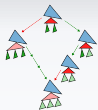
Если переписывание стартует с «листьев», но структурой является сеть, то получается т.н. «автомат Треллиса».

Выделенное красным значение — финальный нетерминал (то, что должно оказаться на вершине стека). Выделенные синим значения — нетерминалы (т.е. они не могут встречаться во входной строке).

Левая таблица определяет автомат Треллиса для слов из $\{a, b\}^*$, у которых центральная буква есть a . Правая таблица определяет автомат Треллиса для языка ПСП, где $a = ($, $b =)$.

a	a	b	c
a	c	b	
b	c	b	b
c		a	a

d	a	b	d	r
a	a	d	a	a
b	r	b		r
d		b		a
r	r	b	b	r



Real-time клеточные автоматы

Если переписывание стартует с «листьев», но структурой является сеть, то получается т.н. «автомат Треллиса».

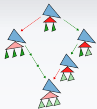
Языки, распознаваемые такими автоматами, — это в точности линейные конъюнктивные языки.

Преобразование автомата Треллиса в конъюнктивную грамматику (она будет линейной, т.к. в каждой базисной правой части может быть, самое большее, всего один нетерминал):

$$S \rightarrow A_{\text{final}}$$

$$A_{\text{init}(a)} \rightarrow a, a \in \Sigma$$

$$A_{\delta(q_1, q_2)} \rightarrow A_{q_1} c \ \& \ b A_{q_2}, \forall b, c \in \Sigma$$



Древесные автоматы

Древесный автомат задаётся входным алфавитом (сигнатурой конструкторов) $\Sigma \subset \{\langle f, n \rangle\}$ и конечным состоянием, а также правилами перехода:

$$\langle f, n \rangle \in \Sigma \Rightarrow (q_1, \dots, q_n, f) \rightarrow q_s.$$

Переписывание происходит снизу вверх (от листьев к корню).

- Описывают все деревья разбора КС-грамматик.
- Обладают свойствами регулярных языков (замкнутость относительно булевых операций, теорема Майхилла–Нероуда)



Грамматика древесных сопряжений (TAG)

Если деревья разрешается разрезать посередине и встраивать в них другие деревья из заданного базиса, то получаются так называемые «мягко контекстно-зависимые языки», которые также характеризуются КЗ-грамматиками, в которых каждый нетерминал оснащён стеком, и с правилами вида:

- $A[\circ \circ \eta] \rightarrow \Phi_1 A'[\circ \circ \eta'] \Phi_2$
- или $A[\eta] \rightarrow \Phi$

Пример такой грамматики для языка $\{a^n b^n c^n d^n\}$:

$$S[\circ \circ] \rightarrow aS[\circ \circ l]d \mid T[\circ \circ]$$

$$T[\circ \circ l] \rightarrow bT[\circ \circ]c$$

$$T[\varepsilon] \rightarrow \varepsilon$$



TAG-языки и конъюнктивные языки

	TAG	CG
Сложность разбора	$O(n^6)$	$O(n^3)$
Накачки	есть	нет
Язык $\{a^n b^n c^n d^n e^n\}$	не входит	входит
Язык $\{ww\}$	входит	неизвестно