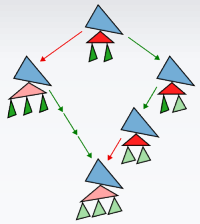
Другие регулярные модели. Синтаксический моноид

Теория формальных языков

2022 г.

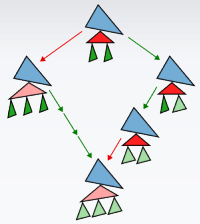


Леволинейные грамматики

Класс грамматик, симметричный праволинейным:

- виды правил — $A_i \to a_j$, $A_i \to A_k a_j$ и $S \to \varepsilon$, если S не встречается в правых частях правил;
- описывает тот же самый класс языков, что и праволинейные грамматики.

Преобразование в леволинейную форму легко сделать по обратным проходам из финального в начальное состояние в НКА, соответствующем грамматике.

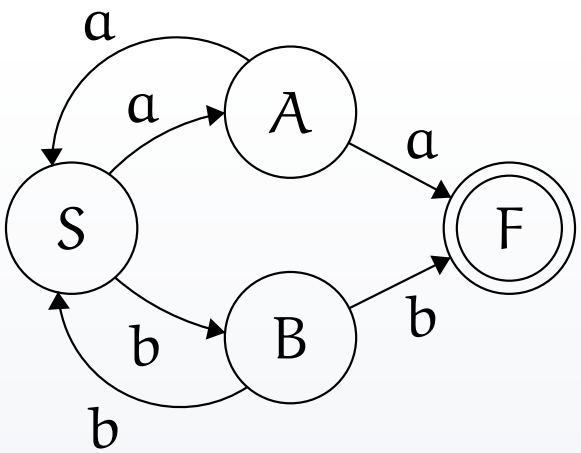


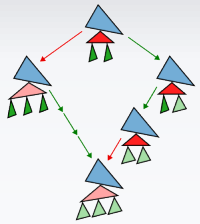
Леволинейные грамматики

Преобразование в леволинейную форму легко сделать по обратным проходам из финального в начальное состояние в НКА, соответствующем грамматике.

$$S \to aA \,|\, bB \quad A \to aS \,|\, a \quad B \to bS \,|\, b$$

Строим недетерминированный КА по грамматике. Здесь F — финальное состояние, куда добавляются переходы $\langle A_i, a_j \rangle \to F$, соответствующие правилам $A_i \to a_j$ грамматики.

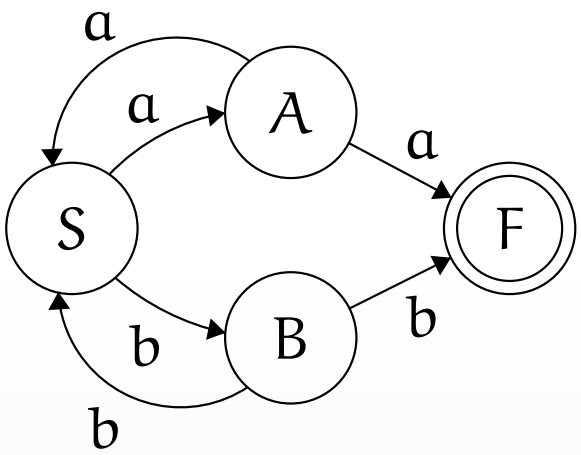


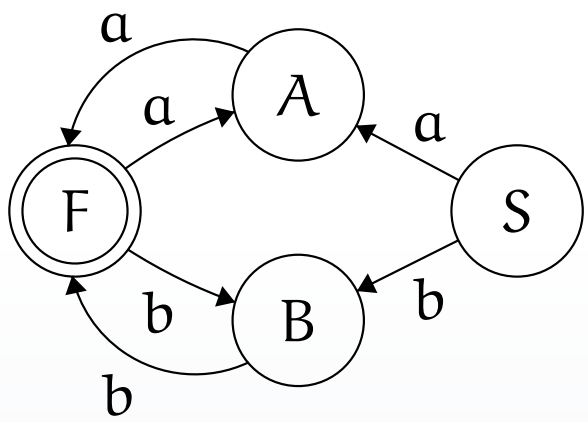


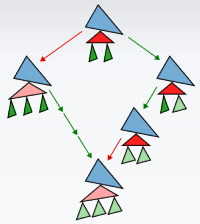
Леволинейные грамматики

$$S \to aA \,|\, bB \quad A \to aS \,|\, a \quad B \to bS \,|\, b$$

Теперь обращаем стрелки и меняем начальное и финальное состояния местами.



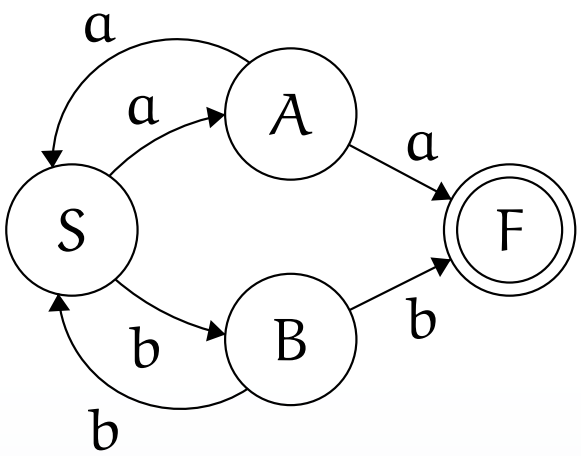


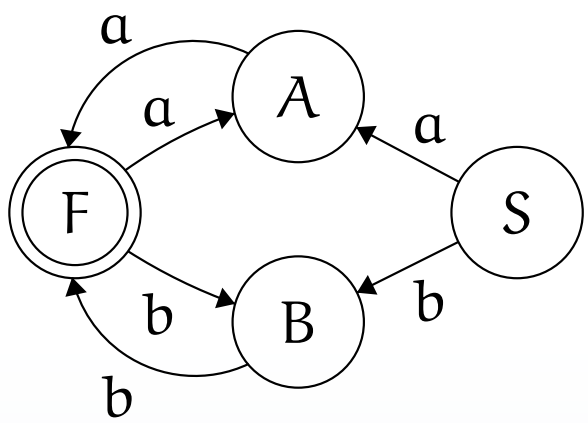


Леволинейные грамматики

$$S \to aA \mid bB \quad A \to aS \mid a \quad B \to bS \mid b$$

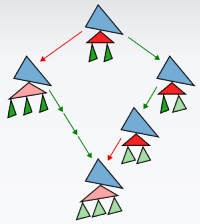
Теперь обращаем стрелки и меняем начальное и финальное состояния местами.





По полученному автомату строим леволинейную грамматику:

$$S \to Aa \mid Bb \quad A \to Fa \mid a \quad B \to Fb \mid b \quad F \to Aa \mid Bb$$

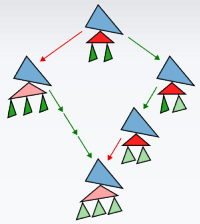


Леволинейные грамматики

Формальный алгоритм приведения к леволинейности

1. Добавляем в грамматику новый стартовый символ S' и для всех правил вида $A_i \to a_j$ — правила $S' \to A_i a_j$.
2. Правила вида $A_i \to a_j A_k$ преобразуем в $A_k \to A_i a_j$.
3. По правилам вида $S \to a_j A_k$ дополнительно порождаем правила $A_k \to a_j$.
4. По правилам вида $S \to a_j$ и $S \to \varepsilon$ порождаем правила $S' \to a_j$ и $S' \to \varepsilon$ соответственно.

Если в исходной грамматике S не встречался в правых частях правил, тогда этот алгоритм породит непродуктивные правила $A_k \to S a_j$. Поэтому шаг 2 для правил вида $S \to a_j A_k$ и шаг 1 для $S \to a_j$ в этом случае делать не надо.

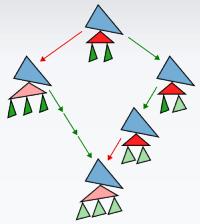


Обращение регулярного языка

Если $\mathscr{L}$ регулярен, то и $\mathscr{L}^R$ регулярен.

Очевидно для НКА (обращаем стрелки в НКА без перемены стартовых и финальных состояний), также очевидно для regex (почему?)

А что с ДКА?



Обращение регулярного языка

Если $\mathscr{L}$ регулярен, то и $\mathscr{L}^R$ регулярен.

Очевидно для НКА (обращаем стрелки в НКА без перемены стартовых и финальных состояний), также очевидно для regex (почему?)

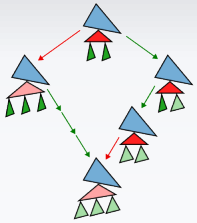
А что с ДКА?

Минимизация по Брзозовски

$\mathtt{det}(\mathtt{reverse}(\mathtt{det}(\mathtt{reverse}(\mathscr{A}))))$ является минимальным ДКА для любого НКА $\mathscr{A}$.

Реверсирование принципиально меняет структуру минимального автомата. Причина — асимметричность определения классов эквивалентности:

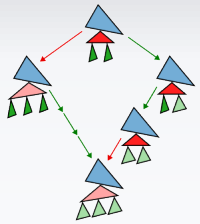
$$u \equiv_{\mathscr{L}} w \Leftrightarrow \forall x(u\,x \in \mathscr{L} \Leftrightarrow v\,x \in \mathscr{L})$$



Цена детерминизма

Утверждение

Имея регулярную грамматику с N нетерминалами, по ней можно построить ДКА (самое большее) с $O(2^N)$ состояниями. Эта оценка является точной.



Цена детерминизма

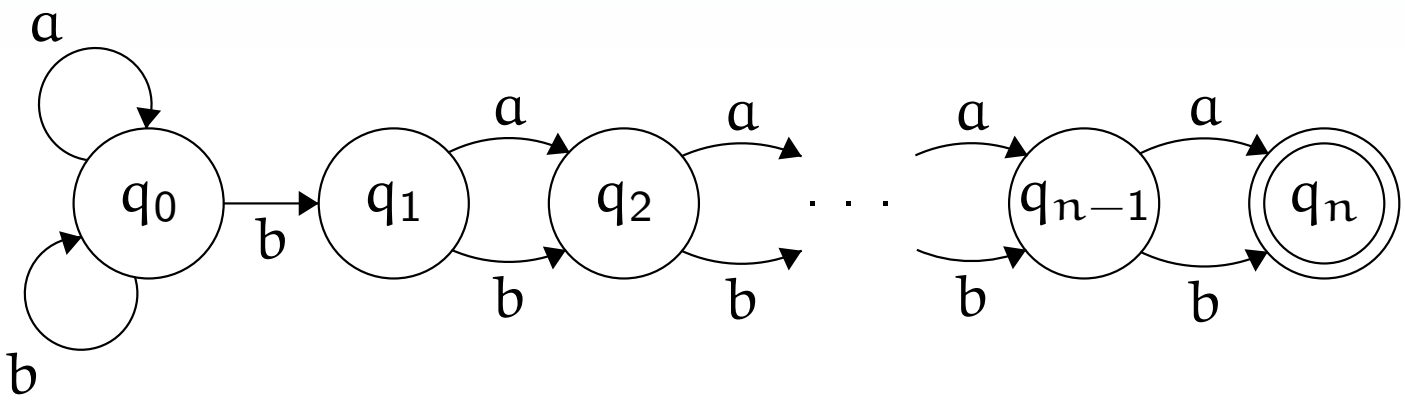
Утверждение

Имея регулярную грамматику с N нетерминалами, по ней можно построить ДКА (самое большее) с $O(2^N)$ состояниями. Эта оценка является точной.

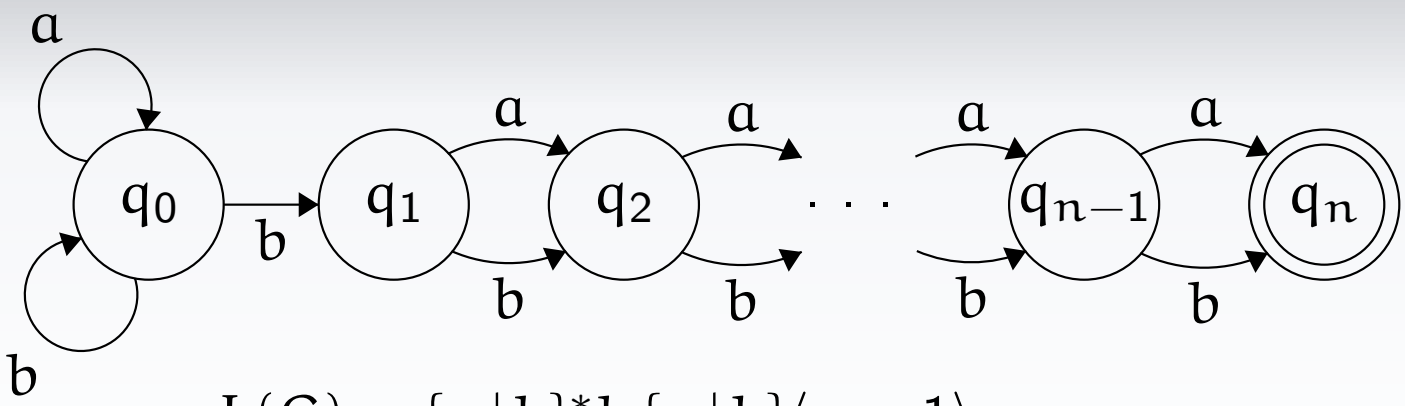
Рассмотрим грамматику G:

$$S \to aS \qquad S \to bS \qquad S \to bA_1$$
$$A_1 \to aA_2 \qquad A_1 \to bA_2 \qquad A_2 \to aA_3 \qquad A_2 \to bA_3$$
$$\dots$$
$$A_{n-1} \to aA_n \qquad A_{n-1} \to bA_n \qquad A_n \to a \qquad A_n \to b$$

Грамматике G соответствует следующий НКА. Её язык — это слова вида $\{a \mid b\}^* b \{a \mid b\} \langle n-1 \rangle$, то есть слова с n-ой буквой с конца, совпадающей с b.



Построим для этого языка таблицу классов эквивалентности и различающих слов по Майхиллу–Нероуду.



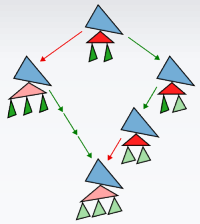
$$L(G) = \{a \mid b\}^* b \{a \mid b\} \langle n-1 \rangle.$$

	ε	a	$\ldots$	a^{n-3}	a^{n-2}	a^{n-1}
a^n	$-$	$-$	$\ldots$	$-$	$-$	$-$
$a^{n-1}b$	$-$	$-$	$\ldots$	$-$	$-$	$+$
$a^{n-2}ba$	$-$	$-$	$\ldots$	$-$	$+$	$-$
$a^{n-2}bb$	$-$	$-$	$\ldots$	$-$	$+$	$+$
$a^{n-3}baa$	$-$	$-$	$\ldots$	$+$	$-$	$-$
$a^{n-3}bab$	$-$	$-$	$\ldots$	$+$	$-$	$+$
$a^{n-3}bba$	$-$	$-$	$\ldots$	$+$	$+$	$-$
$a^{n-3}bbb$	$-$	$-$	$\ldots$	$+$	$+$	$+$
$\ldots$			$\ldots$		$\ldots$	
$a\,b^{n-1}$	$-$	$+$	$\ldots$	$+$	$+$	$+$
b^n	$+$	$+$	$\ldots$	$+$	$+$	$+$

$$L(G) = \{a \mid b\}^* b \{a \mid b\} \langle n-1 \rangle.$$

	ε	a	$\ldots$	a^{n-3}	a^{n-2}	a^{n-1}
a^n	$-$	$-$	$\ldots$	$-$	$-$	$-$
$a^{n-1}b$	$-$	$-$	$\ldots$	$-$	$-$	$+$
$a^{n-2}ba$	$-$	$-$	$\ldots$	$-$	$+$	$-$
$a^{n-2}bb$	$-$	$-$	$\ldots$	$-$	$+$	$+$
$a^{n-3}baa$	$-$	$-$	$\ldots$	$+$	$-$	$-$
$a^{n-3}bab$	$-$	$-$	$\ldots$	$+$	$-$	$+$
$a^{n-3}bba$	$-$	$-$	$\ldots$	$+$	$+$	$-$
$a^{n-3}bbb$	$-$	$-$	$\ldots$	$+$	$+$	$+$
$\ldots$			$\ldots$		$\ldots$	
$a\,b^{n-1}$	$-$	$+$	$\ldots$	$+$	$+$	$+$
b^n	$+$	$+$	$\ldots$	$+$	$+$	$+$

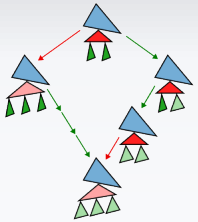
Если в слове w_i в k-ой позиции стоит b, а в w_j стоит a, тогда суффикс a^{k-1} различает w_i и w_j. Все w_i различны $\Rightarrow$ для каждой пары i, j есть такое $k \Rightarrow$ нашлось минимум 2^n классов эквивалентности, и ДКА для L имеет не меньше 2^n состояний.



Ещё раз о лемме о накачке

В отличие от теоремы Майхилла–Нероуда, лемма о накачке использует свойства НКА, а не ДКА. А именно, если константа накачки языка не может быть меньше k, то в НКА, распознающем этот язык, не меньше k состояний.

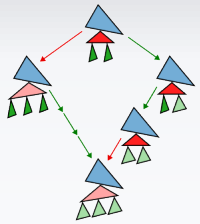
Рассмотрим язык $L = \{w_1 b w_2 \,|\, |w_2| = 3\}$. Мы знаем, что распознающий его ДКА имеет минимум 16 состояний. Накачку $w \in L$, такую что $w = xyz$, $xy^n z \in L$, найти очень просто для всякого слова длины $\geqslant 5$: достаточно взять первую букву этого слова в качестве y, а x принять пустым.
Теперь пусть длина накачки p меньше 5. Рассмотрим слово $baaa \in L$. Любая накачка только букв a (нулевая и нет) выводит слово из языка, и нулевая накачка подслова, содержащего букву b, также выводит из языка L. Поэтому НКА, распознающий L, не может иметь меньше 5 состояний.



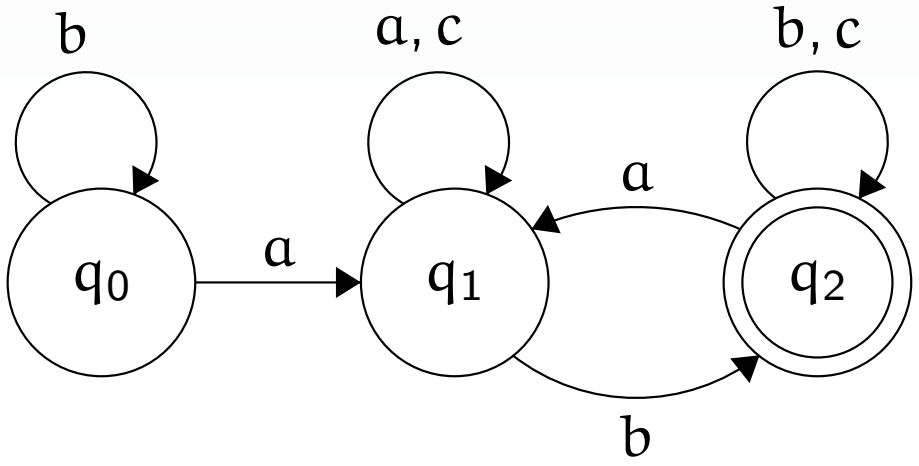
Трансформационный моноид

Посмотрим на правила перехода: $\langle q_1, a \rangle \to q_2$. Если частично специализировать их по элементам из Σ, то получится функция $F_a : Q \to Q$ (Q — множество состояний автомата). Такие же функции можно определить для слов по композиции.

Каждый автомат $\mathscr{A}$ определяет моноид $\mathcal{M} = \{w \mid w \in \Sigma^+\}$ такой, что $w_i = w_j \Leftrightarrow F_{w_i} = F_{w_j}$. Классы эквивалентности слов в $\mathcal{M}$ соответствуют функциям F_{w_i}.

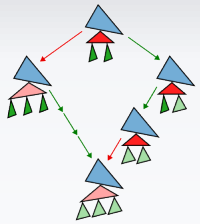


Пример построения Т.М.

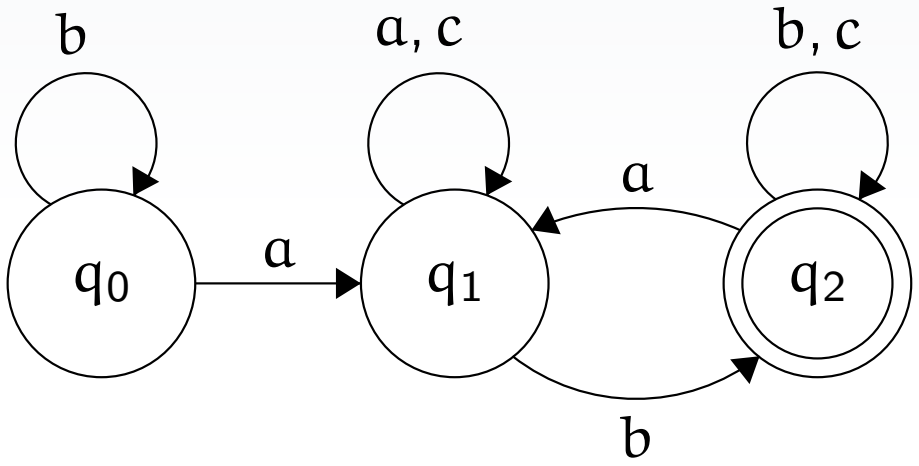


Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке.

$a := \{(0,1),(1,1),(2,1)\}$ $b := \{(0,0),(1,2),(2,2)\}$ $c := \{(1,1),(2,2)\}$



Пример построения Т.М.



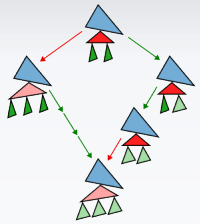
Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке.

$a := \{(0,1),(1,1),(2,1)\}$ $b := \{(0,0),(1,2),(2,2)\}$ $c := \{(1,1),(2,2)\}$

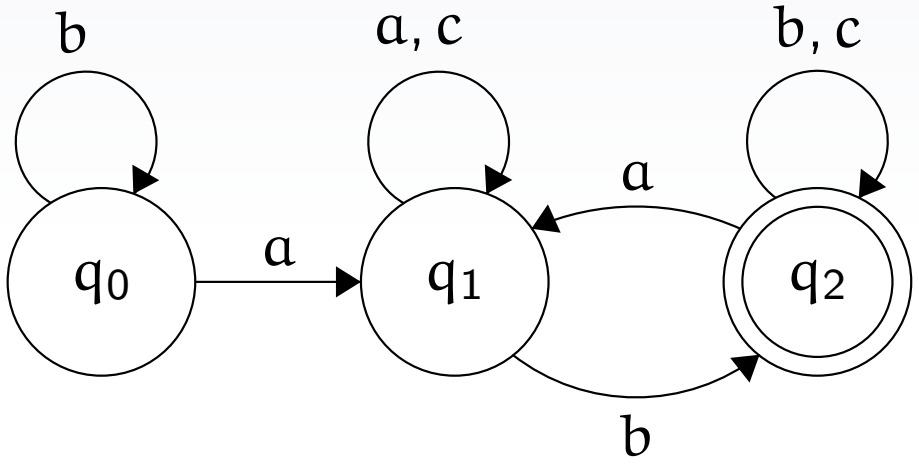
$ab := \{(0,2),(1,2),(2,2)\}$ $bc := \{(1,2),(2,2)\}$ $ca := \{(1,1),(2,1)\}$

$aa \to a$ $ac \to a$ $ba \to a$

$bb \to b$ $cc \to c$ $cb \to bc$



Пример построения Т.М.



Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке.

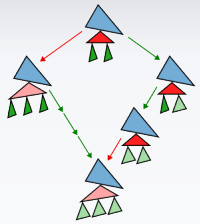
$a := \{(0,1),(1,1),(2,1)\}$ $b := \{(0,0),(1,2),(2,2)\}$ $c := \{(1,1),(2,2)\}$

$ab := \{(0,2),(1,2),(2,2)\}$ $bc := \{(1,2),(2,2)\}$ $ca := \{(1,1),(2,1)\}$

$aa \to a$ $ac \to a$ $ba \to a$

$bb \to b$ $cc \to c$ $cb \to bc$

$abc \to ab$ $bca \to ca$ $cab \to bc$

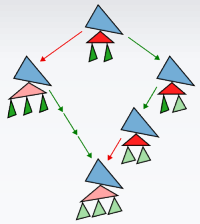


Синтаксический моноид

Положим $u \sim_{\mathscr{L}} v \Leftrightarrow \forall x, y (x u y \in \mathscr{L} \Leftrightarrow x v y \in \mathscr{L})$.
Синтаксический моноид $\mathcal{M}(\mathscr{L})$: $\{w \mid w_i = w_j \Leftrightarrow w_i \sim_{\mathscr{L}} w_j\}$.

Синтаксический моноид регулярного языка $\mathscr{L}$ совпадает с трансформационным моноидом минимального ДКА, его распознающего.

Одному классу эквивалентности синтаксического моноида может соответствовать несколько классов эквивалентности трансформационного, но не наоборот.

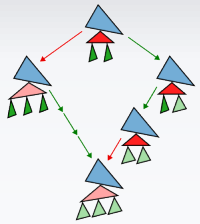


Синхронизирующиеся автоматы

ДКА $\mathscr{A}$ называется синхронизирующимся, если $\exists w, q_s \forall q_i (q_i \xrightarrow{w} q_s)$.

Критерий синхронизации

ДКА $\mathscr{A}$ синхронизирующийся $\Leftrightarrow \forall q, q' \exists w, q_x (q \xrightarrow{w} q_x \ \& \ q' \xrightarrow{w} q_x)$.



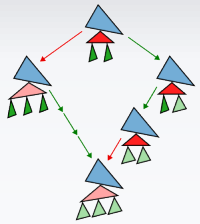
Синхронизирующиеся автоматы

ДКА $\mathscr{A}$ называется синхронизирующимся, если $\exists w, q_s \forall q_i (q_i \xrightarrow{w} q_s)$.

Критерий синхронизации

ДКА $\mathscr{A}$ синхронизирующийся $\Leftrightarrow \forall q, q' \exists w, q_x (q \xrightarrow{w} q_x \ \& \ q' \xrightarrow{w} q_x)$.

Рассмотрим слово w_1, синхронизирующее q_1 и q_2. Если w_1 синхронизирует все состояния, доказывать нечего. Иначе построим множество $Q_1 = \{q \mid q_i \xrightarrow{w_1} q\}$. По построению, $Q_1 \subset \{q_1, \ldots, q_n\}$. Выберем в нём два первых состояния, q_i, q_j, и слово w_2, синхронизирующее их. Построим множество $Q_2 = \{q \mid q_i \in Q_1 \ \& \ q_i \xrightarrow{w_2} q\}$. По построению, $Q_2 \subset Q_1$. Продолжив так не более чем $n - 1$ раз, построим синхронизирующее слово.



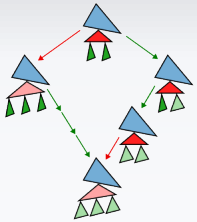
Синхронизирующиеся автоматы

ДКА $\mathscr{A}$ называется синхронизирующимся, если $\exists w, q_s \forall q_i (q_i \xrightarrow{w} q_s)$.

Критерий синхронизации

ДКА $\mathscr{A}$ синхронизирующийся $\Leftrightarrow \forall q, q' \exists w, q_x (q \xrightarrow{w} q_x \ \& \ q' \xrightarrow{w} q_x)$.

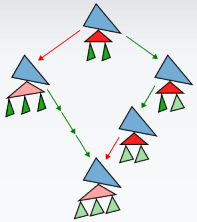
ДКА синхронизируется $\Leftrightarrow$ классы эквивалентности его трансформационного моноида содержат «константу», т.е. класс, переводящий все состояния в одно.



Задача Эшби о привидениях

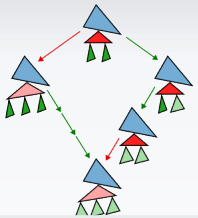
Дорогой друг! Недавно я купил старый дом, в котором обитают два призрака: Певун и Хохотун. Я установил, что их поведение подчиняется определенным законам, и что я могу воздействовать на них, играя на органе или сжигая ладан. В течение каждой минуты каждый из призраков либо шумит, либо молчит. Поведение же их в каждую минуту зависит только от минуты до этого, и эта зависимость такова.

Певун всегда ведет себя так же, как и в предыдущую минуту (звучит или шумит), если только в эту предыдущую минуту не было игры на органе при молчании Хохотуна. В последнем случае Певун меняет свое поведение на противоположное. Что касается Хохотуна, то, если в предыдущую минуту горел ладан, он будет вести себя так же, как Певун минутой раньше. Если, однако, ладан не горел, Хохотун будет вести себя противоположно Певуну в предыдущую минуту. Что мне делать, чтобы установить и поддерживать тишину в доме?



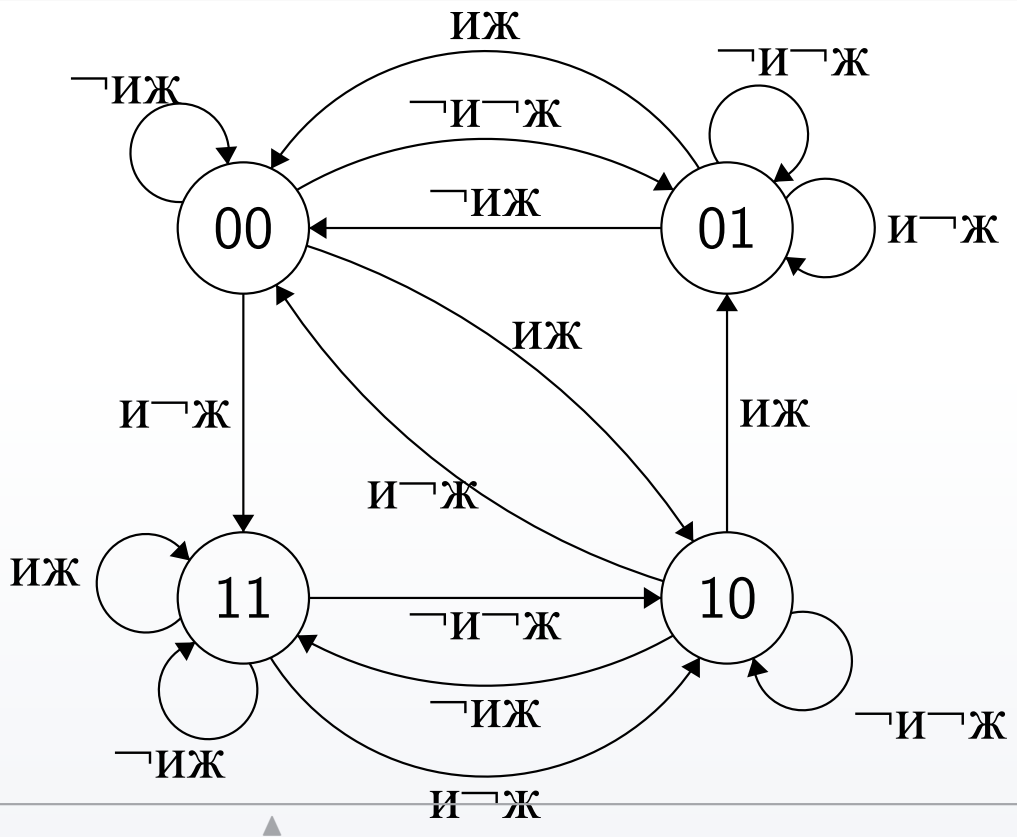
Задача Эшби о привидениях

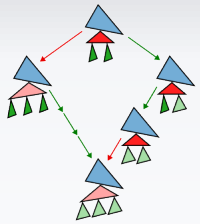
- Если не играли на органе или Хохотун шумел, Певун не меняет поведение, иначе меняет.
- Если горел ладан, Хохотун делает то же, что делал Певун, иначе — противоположное.



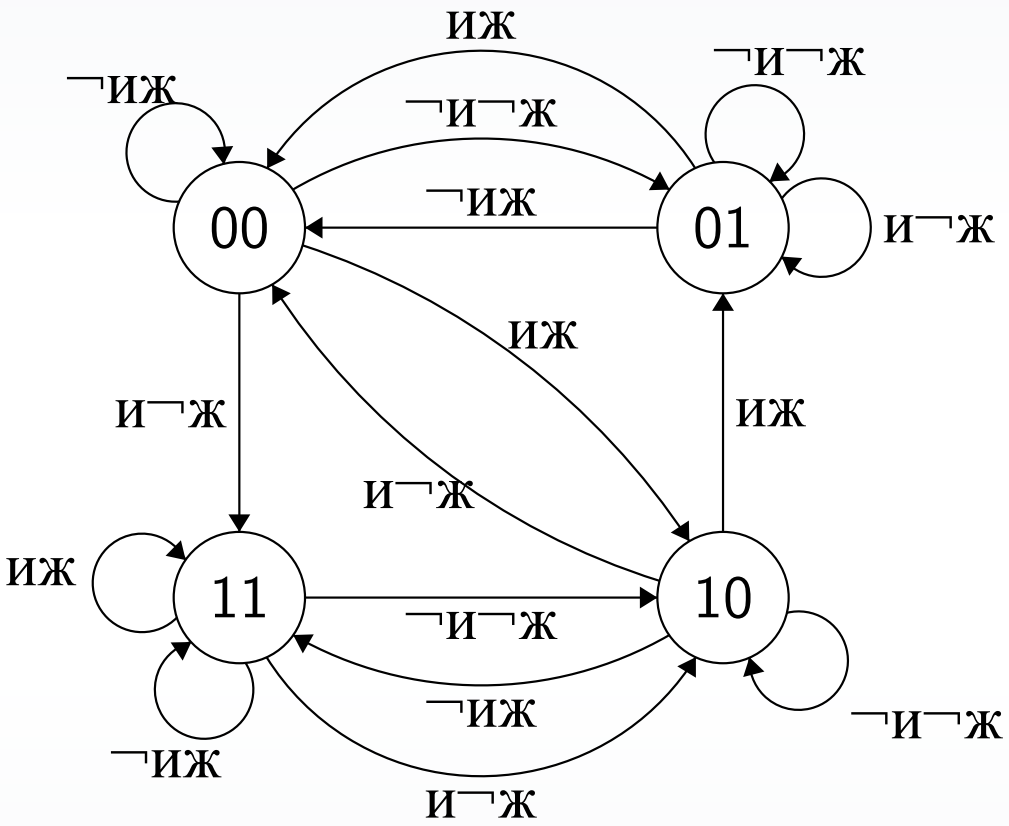
Задача Эшби о привидениях

- Если не играли на органе или Хохотун шумел, Певун не меняет поведение, иначе меняет.
- Если горел ладан, Хохотун делает то же, что делал Певун, иначе — противоположное.

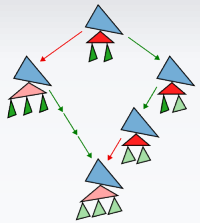




Задача Эшби о привидениях



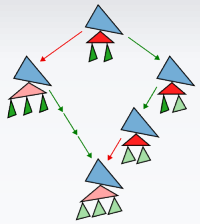
Синхронизирующее к состоянию 00 слово: ¬и¬ж, и¬ж, ¬иж.



Префиксное кодирование

Двоичное префиксное кодирование — это гомоморфизм $h : \Sigma^+ \to \{0,1\}^+$ такой, что
$\forall a, b \in \Sigma \, \forall w \in \{0,1\}^* (h(a) \neq h(b)w)$.

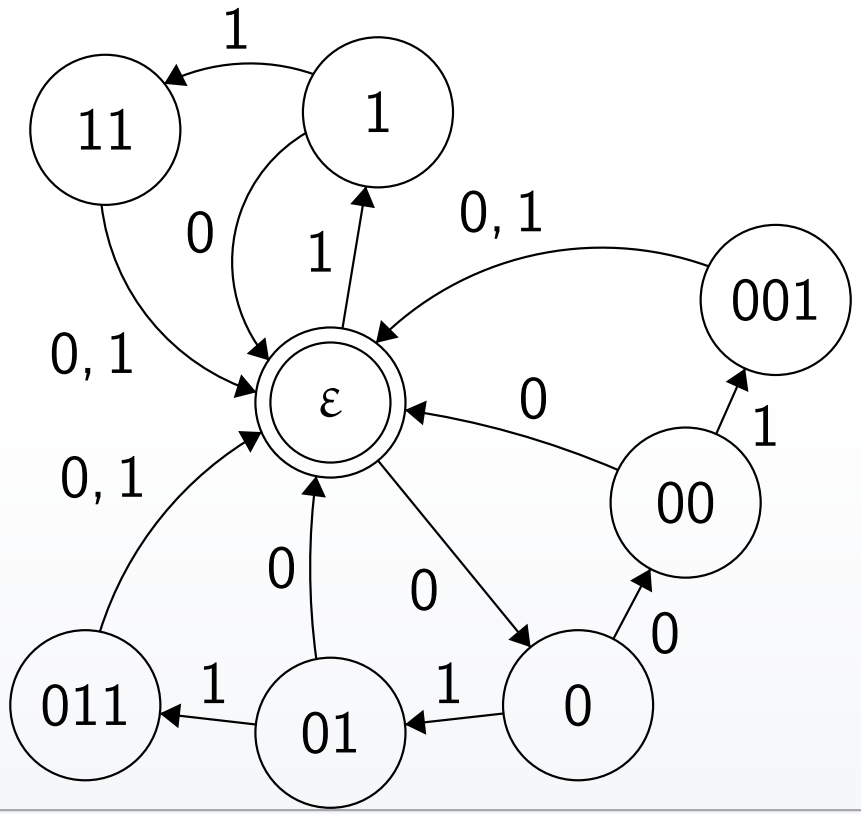
Рассмотрим префиксный код из 9-буквенного алфавита:
$\mathcal{C} = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$.

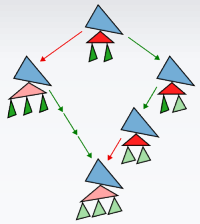


Префиксное кодирование

Рассмотрим префиксный код из 9-буквенного алфавита:
$\mathcal{C} = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$.

Автомат–декодер для $\mathcal{C}$ (возвращается в ε-состояние, дочитав очередной код):



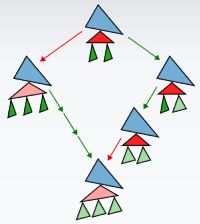


Коды, исправляющие ошибки

Префиксный код максимален, если к множеству кодирующих слов нельзя добавить ни одно слово без нарушения префикс-свойства (т.е. запрета слов из множества быть префиксами друг друга).

Максимальный префиксный двоичный код $\mathcal{C}$ называют синхронизированным, если $\exists z \in \{0,1\}^+$, такое что $\forall y \in \{0,1\}^+$ слово yz можно представить как конкатенацию слов из $\mathcal{C}$.

Если код $\mathcal{C}$ синхронизирован, тогда ошибки в передаче закодированного слова будут исправляться сами при передаче достаточно длинной закодированной последовательности.



Коды, исправляющие ошибки

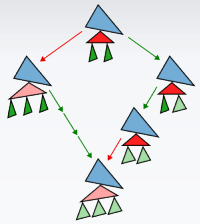
Префиксный код максимален, если к множеству кодирующих слов нельзя добавить ни одно слово без нарушения префикс-свойства (т.е. запрета слов из множества быть префиксами друг друга).

Максимальный префиксный двоичный код $\mathcal{C}$ называют синхронизированным, если $\exists z \in \{0,1\}^+$, такое что $\forall y \in \{0,1\}^+$ слово yz можно представить как конкатенацию слов из $\mathcal{C}$.

Если код $\mathcal{C}$ синхронизирован, тогда ошибки в передаче закодированного слова будут исправляться сами при передаче достаточно длинной закодированной последовательности.

Утверждение

Максимальный префиксный код синхронизирован $\Leftrightarrow$ его декодер — синхронизирующийся ДКА.



Алфавитные префиксные грамматики

Определение APG

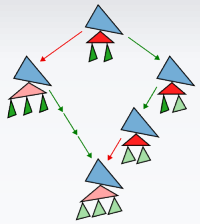
Дана SRS $\mathcal{S}$ с правилами переписывания двух видов:

$$a_i \to b_1 \dots b_n \qquad a_i \to \varepsilon$$

Разрешим применять правила только к первым буквам слова. Пусть дана пара $\langle \mathcal{S}, w_0 \rangle$, где w_0 — слово в алфавите Σ. Эта пара определяет алфавитную префиксную грамматику.

Утверждение

Язык $L\langle \mathcal{S}, w_0 \rangle$ регулярен.



Алфавитные префиксные грамматики

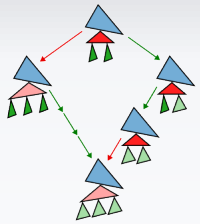
Утверждение

Язык $L\langle \mathcal{S}, w_0 \rangle$ регулярен.

Скажем, что $a \twoheadrightarrow \varepsilon$ (a коллапсирует), если либо $a \to \varepsilon \in \mathcal{S}$, либо $\exists b_1, \ldots, b_n (\forall b_i (b_i \twoheadrightarrow \varepsilon) \;\&\; a \to b_1 \ldots b_n \in \mathcal{S})$.

По APG $\langle \mathcal{S}, s_1 \ldots s_n \rangle$ породим праволинейную грамматику G. Каждому символу алфавита a_i сопоставим A_i — нетерминал G.

1. Пусть $a \to b_1 \ldots b_n$ и $\exists b_i (\neg(b_i \twoheadrightarrow \varepsilon) \;\&\; \forall j (j < i \Rightarrow b_j \twoheadrightarrow \varepsilon))$. Тогда добавим в G правила $A \to B_1 b_2 \ldots b_n$, $A \to B_2 b_3 \ldots b_n$,\ldots, $A \to B_i b_{i+1} \ldots b_n$, $A \to a$.
2. Если такого b_i нет, добавляем в G все правила вида $A \to B_1 b_2 \ldots b_n, \ldots, A \to B_{n-1} b_n, A \to B_n, A \to a$.
3. Вводим стартовый нетерминал S и для него добавляем развёртку в исходное слово $s_1 \ldots s_m$ по правилам выше.
4. Если все s_i коллапсируют, тогда добавляем в G правило $S \to \varepsilon$.



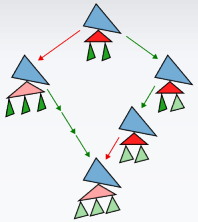
Алфавитные префиксные грамматики

Скажем, что $a \twoheadrightarrow \varepsilon$ (a коллапсирует), если либо $a \to \varepsilon \in \mathcal{S}$, либо $\exists b_1, \ldots, b_n (\forall b_i (b_i \twoheadrightarrow \varepsilon) \;\&\; a \to b_1 \ldots b_n \in \mathcal{S})$.

По APG $\langle \mathcal{S}, s_1 \ldots s_n \rangle$ породим праволинейную грамматику G. Каждому символу алфавита a_i сопоставим A_i — нетерминал G.

1. Пусть $a \to b_1 \ldots b_n$ и $\exists b_i (\neg (b_i \twoheadrightarrow \varepsilon) \;\&\; \forall j (j < i \Rightarrow b_j \twoheadrightarrow \varepsilon))$. Тогда добавим в G правила $A \to B_1 b_2 \ldots b_n$, $A \to B_2 b_3 \ldots b_n$,…, $A \to B_i b_{i+1} \ldots b_n$, $A \to a$.
2. Если такого b_i нет, добавляем в G все правила вида $A \to B_1 b_2 \ldots b_n, \ldots, A \to B_{n-1} b_n, A \to B_n, A \to a$.
3. Вводим стартовый нетерминал S и для него добавляем развёртку в исходное слово $s_1 \ldots s_m$ по правилам выше.
4. Если все s_i коллапсируют, тогда добавляем в G правило $S \to \varepsilon$.

Остается сделать развертку правил вида $A \to B_n$, либо перейти от G к НКА с ε-переходами.

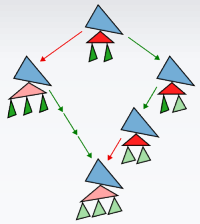


Неалфавитные грамматики

Если вместо правил $a_i \to b_1 \dots b_n$ к префиксам слов можно применять любые правила вида $a_1 \dots a_m \to b_1 \dots b_n$, такая грамматика называется (просто) префиксной. Для простоты предполагаем, что начальное слово также может быть не единственным.

Языки префиксных грамматик регулярны.

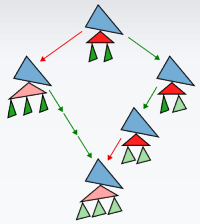
Доказательство использует ту же идею, что в случае АПГ: множество минимальных укорачивающихся комбинаций правил переписывания конечно.



От ДКА к префиксной грамматике

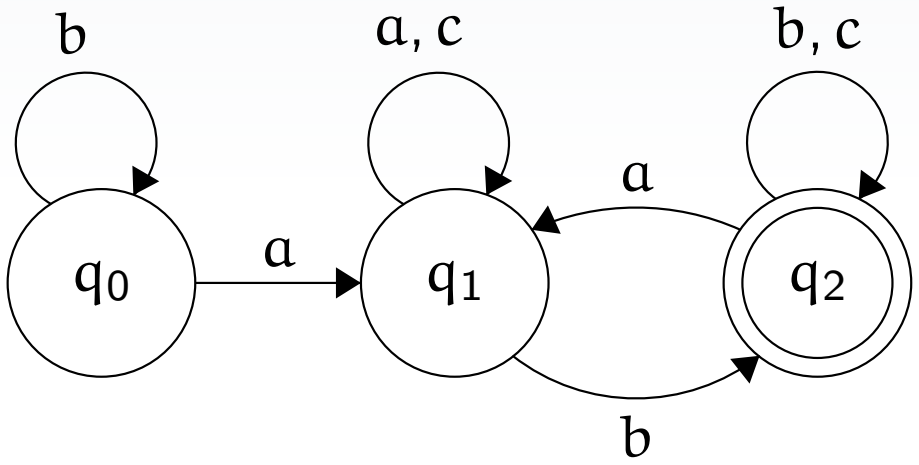
В данном алгоритме рассматривается минимальный ДКА для языка.

- Ведущими словами для нетерминалов (состояний) q_i объявим классы эквивалентности w_i такие, что $q_0 \xrightarrow{w_i} q_i$.
- Для всех стрелок, входящих в q_i из q_k и помеченных буквами a_j, построим правила переписывания: $w_i \to w_k a_j$.
- Начальными словами объявим слова из классов эквивалентности, лежащих в языке автомата.



От ДКА к префиксной грамматике

Построим префиксную грамматику для языка уже знакомого нам автомата:



Для q_0 ведущим словом будет b, для q_1 — a, для q_2 ведущее ab (оно же стартовое слово).

Правила префиксной грамматики:

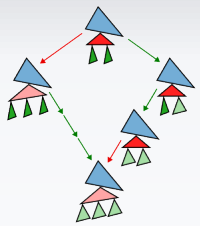
$b \to bb$ (в q_0 входит лишь одна стрелка)

$a \to aa$ $a \to ac$ (стрелки из q_1 в себя)

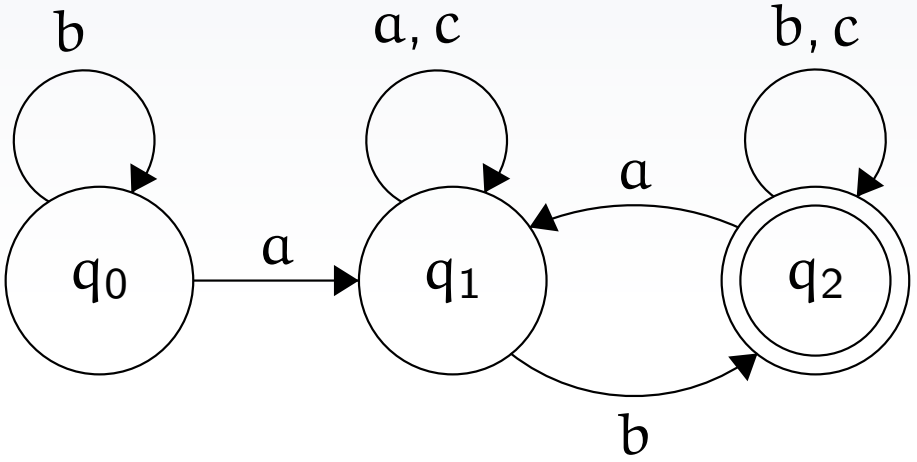
$a \to ba$ (стрелка из q_0 в q_1)

$ab \to ab$ (стрелка из q_1 в q_2)

$ab \to abc$ $ab \to abb$ (стрелки из q_2 в себя)



От ДКА к префиксной грамматике



Стартовое слово: ab. Правила переписывания:

$b \to bb$ (в q_0 входит лишь одна стрелка)

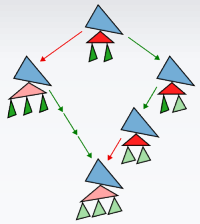
$a \to aa$ $a \to ac$ (стрелки из q_1 в себя)

$a \to ba$ (стрелка из q_0 в q_1)

$ab \to ab$ (стрелка из q_1 в q_2)

$ab \to abc$ $ab \to abb$ (стрелки из q_2 в себя)

Результат похож на обращенные правила трансформационного моноида, но учитывает префиксность: нет смысла переписывать $c \to cc$, если c может встретиться только после буквы a, либо после префикса ab.



Поведение стека в CBV-семантике

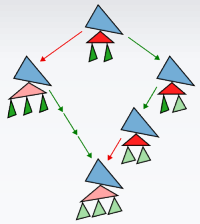
Рассмотрим стек с вершиной $\bullet_n$:

$\bullet_n \leftarrow \mathtt{f}_{n+1}(\dots)$, $\bullet_{n-1} \leftarrow \mathtt{f}_n(\bullet_n \dots) \dots$, $\bullet_0 \leftarrow \mathtt{f}_1(\bullet_1 \dots)$

Опишем его состояние перечислением имён функций в порядке их вхождения: $\mathtt{f}_{n+1}\mathtt{f}_n \dots \mathtt{f}_1$.

Шаги вычислений над такими состояниями стека описываются как применения правил в APG.

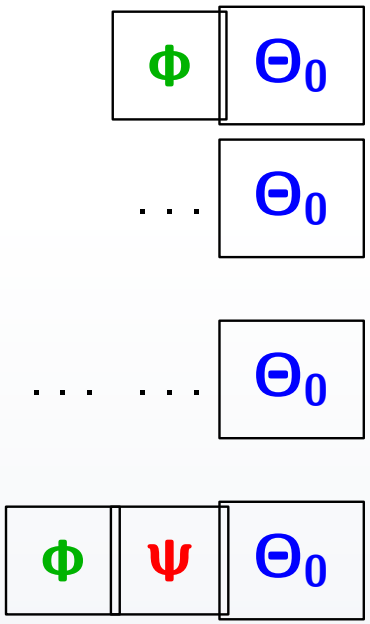
«Подозрительное» поведение — такое, при котором вершина стека повторяется, выбрасывая промежуточные вычисления.



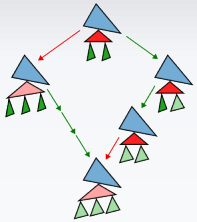
Поиск бесконечных циклов

Отношение Турчина

Пусть на пути развертки программы имеются два состояния стеков: $c_1 : \Phi\Theta_0$, $c_2 : \Phi\Psi\Theta_0$, такие что Θ_0 неизменна на всём отрезке пути от c_1 до c_2. Тогда скажем, что $c_1 \preceq c_2$ (связаны отношением Турчина).



Если вершина Φ действительно входит в бесконечный цикл, порождая всё новые состояния вида $\Phi\Psi^n\Theta_0$, тогда правдоподобно, что $c_1 \preceq c_2$. Однако может случиться, что $c_1 \preceq c_2$ и на развертке завершающегося вычисления (ложное срабатывание).

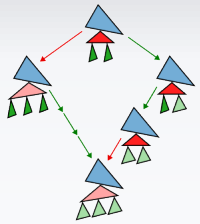


Теорема Турчина

Вариант для CBV

На любом бесконечном пути вычислений имеются два состояния стека, такие что $c_1 \preceq c_2$.

Теорема Турчина гарантирует, что существование $\preceq$-пар — необходимое условие бесконечного (зацикливающегося) вычисления. Поэтому $\preceq$ может использоваться для приблизительного анализа завершаемости программ (наряду с другими условиями).

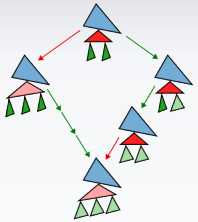


Пинг-понг протоколы

Определение

Пусть дано множество одноместных операций $\mathcal{V}_x = \mathcal{O}_x \cup \mathcal{P}_x$, задаваемое для участника x, причём для некоторых $p_1, p_2 \in \mathcal{V}_x$ выполняются тождества $p_1 \circ p_2 = \mathrm{id}$, и для всех $p_1, p_2, p_3((p_1 \circ p_2) \circ p_3 = p_1 \circ (p_2 \circ p_3))$.
Пинг-понг протокол для двух участников — это конечная последовательность инструкций $[p_1 \ldots p_n, [x, y]]$, $p_i \in \mathcal{V}_x \cup \mathcal{O}_y$.

$\mathcal{O}_x$ — публичные операции; $\mathcal{P}_x$ — приватные операции.

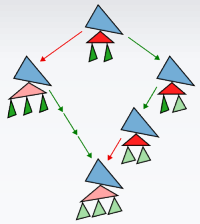


Модель угрозы Долева–Яо

Д. Долев & А. Яо — первая формальная модель угрозы и первое формальное понятие криптографического протокола (1983).

Злоумышленник по Долеву–Яо:

- Может перехватывать, пересылать и изменять любое сообщение в сети;
- Может играть роль любого пользователя (маскарад);
- Может убедить пользователей начать любой дозволенный протоколом сеанс передачи сообщений.
- Не может совершать битовые операции над сообщениями;
- Не может угадать свойства секретных операций.

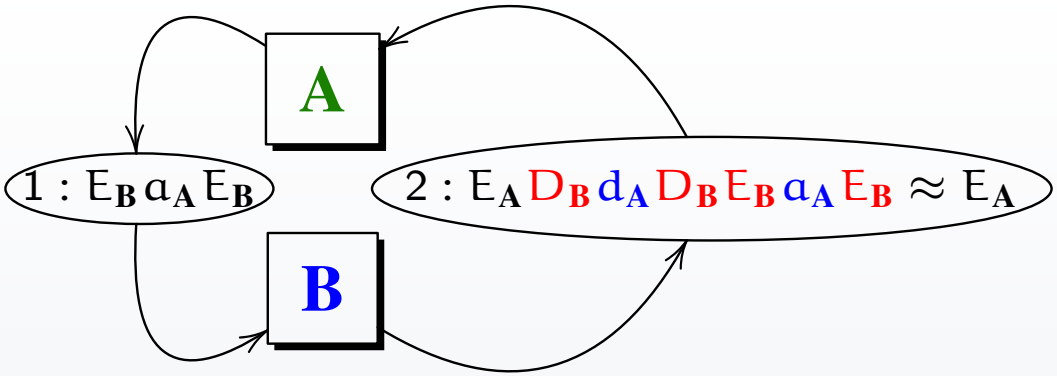


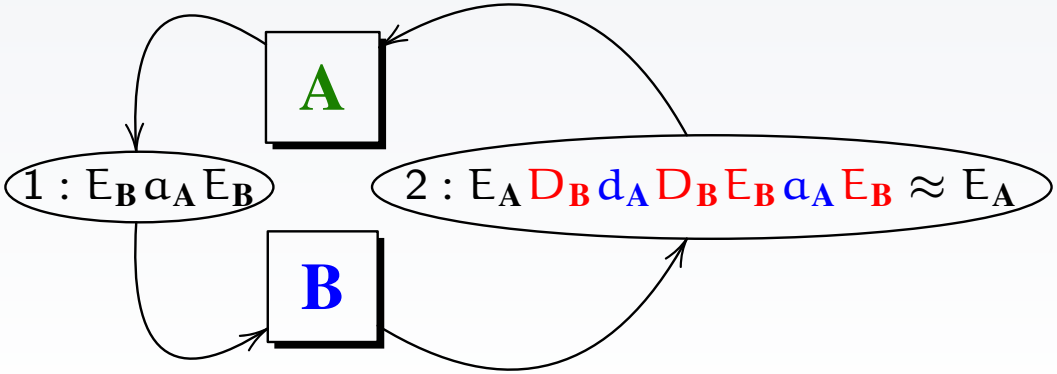
Протокол для двух участников

Легальные пользователи — **A**, **B**. **Злоумышленник** — **Z** (одного всегда достаточно).
Изначальное сообщение — M (обычно засекреченное).
Σ_x — словарь операторов x. E_x — зашифровка открытым ключом x, D_x — расшифровка E_x, a_x — приписывание к сообщению имени x, d_x — удаление префикса сообщения, совпадающего с именем x.

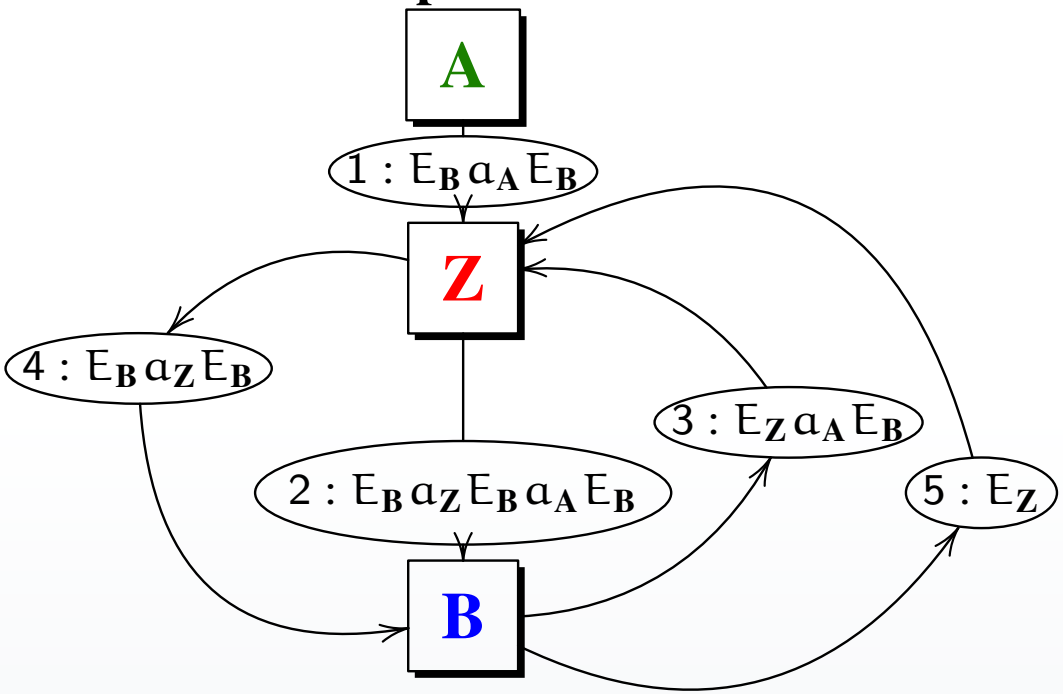
Протокол — набор α_i (**слов протокола**) и указаний, кто посылает α_i.
Атака — последовательность подстановок в α_i, порождающая пустое слово (т.е. демаскирующая сообщение M).

Пример протокола

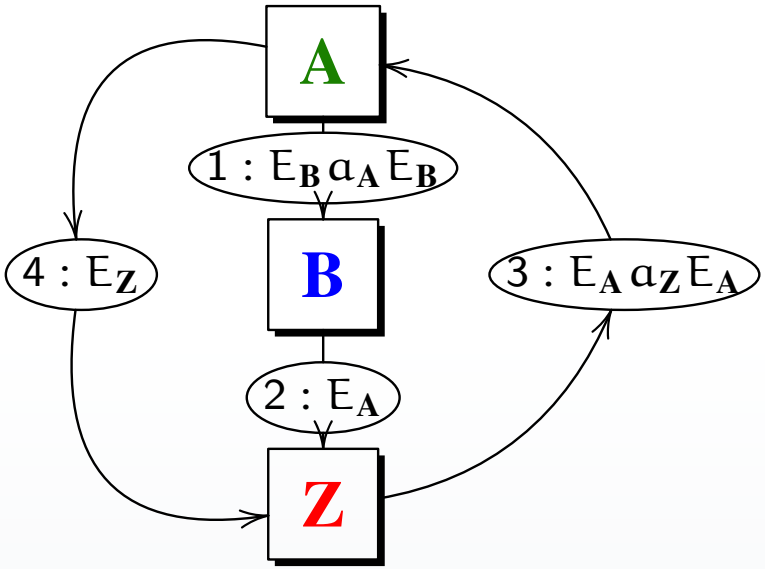


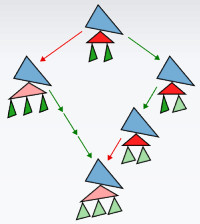


Первая атака



Вторая атака





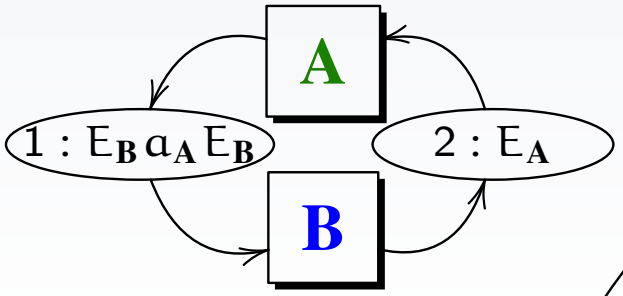
Автоматная модель $\mathscr{A}(\mathsf{P})$

- Строим все возможные подстановки в протокол P пар участников (включая злоумышленника);
- Строим начальное состояние 0 и конечное состояние 1, между ними — путь, соответствующий обращению первого слова протокола с двумя легальными участниками **A**, **B** (чтобы было что атаковать);
- Строим пути из 0 в 0, соответствующие реверсам (обращенным) словам-подстановкам в протокол P;
- Строим пути из 0 в 0, соответствующие всем возможным индивидуальным действиям злоумышленника **Z** — т.е. элементам $\mathcal{O}_{\mathbf{A}}$, $\mathcal{O}_{\mathbf{B}}$, $\mathcal{O}_{\mathbf{Z}}$ и $\mathcal{P}_{\mathbf{Z}}$.

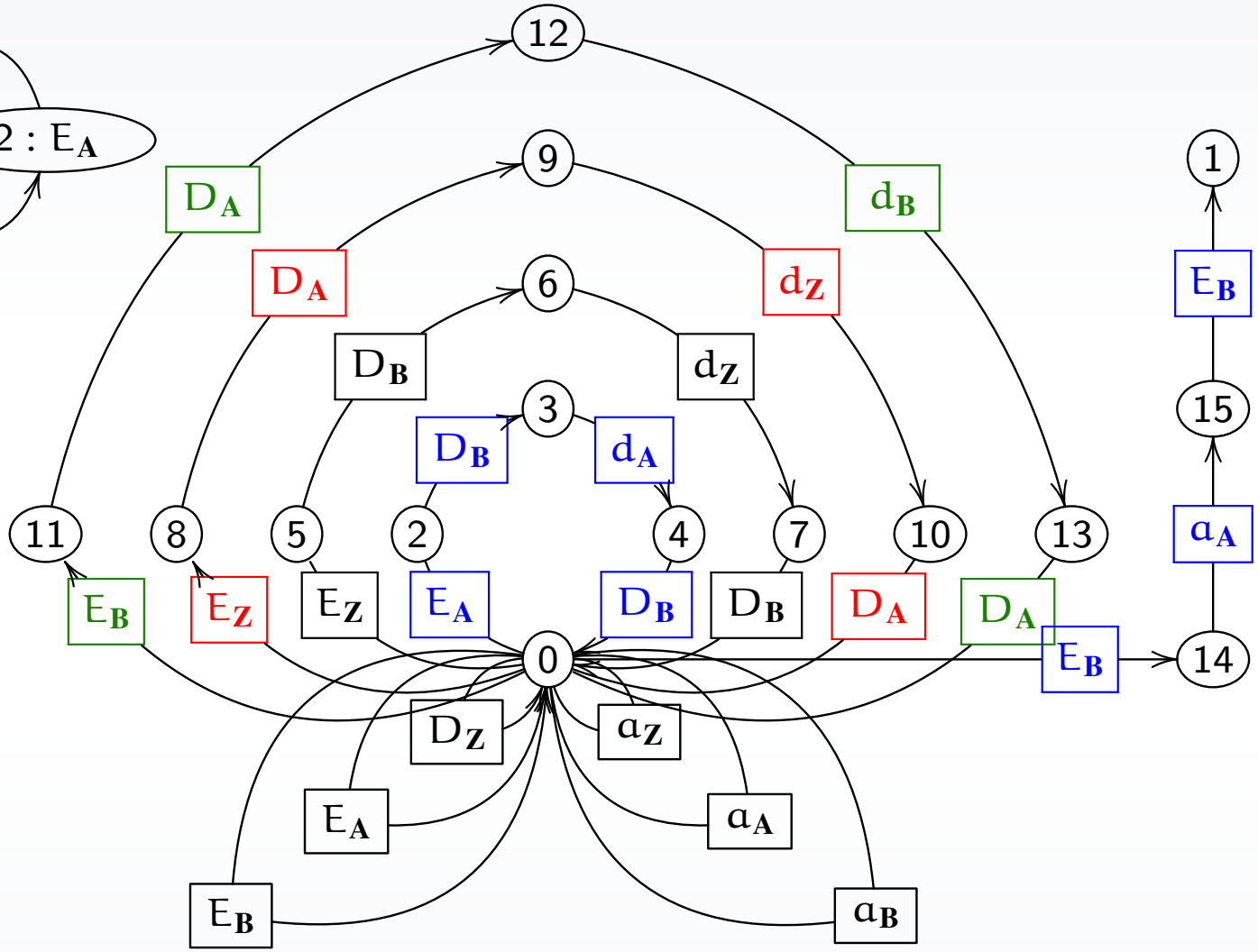
Утверждение

Протокол P ненадёжен в модели угрозы Долева–Яо тогда и только тогда, когда $\varepsilon \in \mathsf{L}(\mathscr{A}(\mathsf{P}))$.

Протокол



Автоматная модель



Случаи

$P_{Double}[A, B]$

$P_{Double}[B, A]$

$P_{Double}[A, Z]$

$P_{Double}[Z, B]$