

# КЗ-свойства языков. MFA, атрибутивные грамматики и типизация

---



Теория формальных языков  
2023 г.



## Кодирование LZ

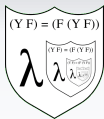
- Встретилось слово из одной буквы  $\Rightarrow$  добавляем его в словарь и создаём на него ссылку.
- Встретилось слово, максимально длинное и такое, что его префикс без последней буквы уже в словаре  $\Rightarrow$  добавляем его вместе с последней буквой в словарь и создаём на него ссылку.

В отличие от кодов Хаффмана, не разбираются с помощью конечных автоматов. Необходимо понятие обратных ссылок (backreferences) — актуальное в современных REGEX библиотеках. Языки, распознаваемые регулярными выражениями с backref-ами, обычно называют REGEX-языками (не «регулярными», т.к. они представляют собой более широкий класс).



## Общие соглашения про языки с обратными ссылками

- Если ссылка инициализируется внутри итерации, то активной считается только последняя её инициализация. Например, выражение  $((a^*)b)^+ \setminus 2$  описывает язык  $\{(a^*b)^n a^n b a^n b\}$ , а не  $\{a^n b (a^*b)^n a^n b\}$  и не  $\{a^n b a^n b \dots a^n b\}$ .
  - Принято в стандарте PCRE2 и всех формализациях.
- Неинициализированные ссылки либо считаются читающими пустое слово ( $\varepsilon$ -семантика), либо недопустимы ( $\emptyset$ -семантика). Например, в  $\varepsilon$ -семантике  $((a^*) \mid b) \setminus 2$  распознаёт язык  $\{a^{2n}, b\}$ ; в  $\emptyset$ -семантике  $\{a^{2n}\}$ .
  - В стандарте PCRE2 —  $\emptyset$ -семантика.
  - В различных формализациях семантика неинициализированных ссылок варьирует.



## Первая формализация языков с обратными ссылками

Формализация Кампеану–Саломаа–Ю (CSY): только безымянные группы захвата в память,  $\emptyset$ -семантика.

- Каждая скобочная группа ( $\tau$ ) считается группой захвата в память и автоматически нумеруется по очередности вхождений открывающих скобок.
- Ссылка  $\backslash k$  на скобочную группу с номером  $k$  может появиться, только если скобочная группа уже закрыта (это условие не выполняется в регулярках, принятых стандартом PCRE2).

Например, выражение  $((a^*)b^*)\backslash 2\backslash 1$  описывает язык  $\{a^n b^n a^{2n} b^n\}$ . Выражение  $((a^*)b^* | \backslash 1)^*\backslash 2$  некорректно в CSY: ссылка на первую скобочную группу осуществляется внутри неё самой. По стандарту PCRE2, оно корректно.



## Свойства CSY-REGEX

Если  $\mathcal{L}$  – CSY-язык, тогда

$$\exists N \in \mathbb{N} \forall \omega \in \mathcal{L} (|\omega| > N \Rightarrow \omega = x_0 y x_1 y \dots y x_m \ \& \ |x_0 y| < N \ \& \ |y| > 0 \ \& \ \forall i \in \mathbb{N} (i > 0 \Rightarrow x_0 y^i x_1 y^i \dots y^i x_m \in \mathcal{L})).$$

Например, с помощью леммы о накачке для CSY-языков легко доказать, что  $a^n b^n$  не описывается CSY-регуляркой.

- Описывают более узкий класс языков, чем PCRE2 — выражения с обратными ссылками.
- Но задача сопоставления слова с CSY-REGEX уже NP-сложна (теорема Англуин).



## Ref-words (Shmid, 2014)

Специальные символы —  $[_i, ]_i, x_i$ . Вхождения  $x_i$  и скобки с индексом  $i$  не могут встречаться внутри  $[_i \dots ]_i$ , однако разные скобочные блоки могут быть перепутаны:

$$[_1 a [_2 b ]_1 x_1 ]_2 x_2$$

Значение в скобках  $[_k \dots ]_k$  сохраняется в ячейку памяти с номером  $k$  и затем может быть прочитано из входной строки при чтении в backref-REGEX переменной  $x_k$ . К примеру, слово, описанное выше, определяет значение  $x_1 = ab$ , после чего можно вычислить, что  $x_2 = bab$ , и вся строка, которая сопоставляется с таким выражением, есть  $ababbab$ .



- с — «закрыть» ячейку памяти;
- о — «открыть» ячейку памяти;
- ◇ — не менять состояние ячейки.

$(u_i \in \Sigma^*, r_i \in \{o, c\})$  переходит по правилу

- если  $b \in \Sigma \cup \{\varepsilon\}$ , то  $v = b$ ;
- если  $b \in \{1, \dots, k\}$  и  $r'_k = c$ , то  $v = u_b$ ;
- $r'_i = r_i$ , если  $s_i = \diamond$ , и  $s_i$  в противном случае;
- $u'_i = u_i v$ , если  $r'_i = r_i = o$ ;  $u'_i = v$ , если  $r'_i = o$  и  $r_i = c$ ; и не меняется, если  $r'_i = c$ .



## Memory Finite Automata (MFA)

$k$  — MFA  $\mathcal{A}$  имеет функцию перехода из  $Q \times \Sigma \cup \{\varepsilon\} \cup \{1, \dots, k\}$  в подмножество  $Q \times \langle o, c, \diamond \rangle^k$ , где флаги управления памятью  $o, c, \diamond$  означают:

- $c$  — «закрыть» ячейку памяти;
- $o$  — «открыть» ячейку памяти;
- $\diamond$  — не менять состояние ячейки.

Начальная конфигурация памяти:  $\langle \langle \varepsilon, c \rangle, \dots, \langle \varepsilon, c \rangle \rangle$ . То есть все ячейки закрыты для записи и содержат пустое слово.

Заметим, что запись в ячейку слова, считанного с ленты по переходу, осуществляется не исходя из флагов памяти в предыдущем состоянии, а исходя из флагов памяти в состоянии, куда осуществляется переход. То есть мы сначала открываем (или закрываем) память, а уже потом читаем с ленты и пишем в открытые ячейки.

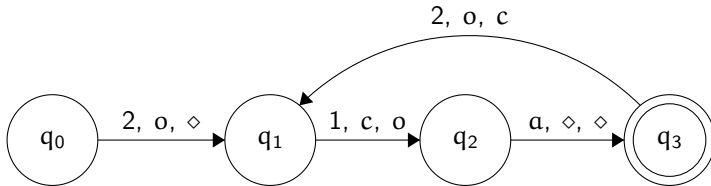




## MFA для $\{a^{n^2}\}$

Backref-REGEX для этого языка:  $([{}_1x_2]_1[{}_2x_1a]_2)^+$ . Здесь важно, что неинициализированная переменная (первое вхождение  $x_2$ ) получает значение  $\varepsilon$ .

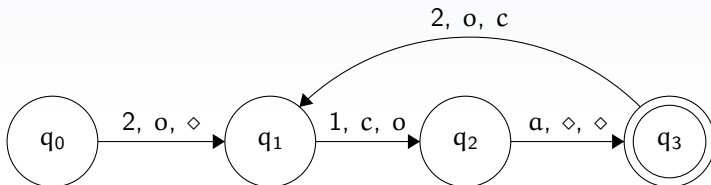
Посмотрим на 2-MFA, построенный по этому выражению:



Поскольку открытие и закрытие памяти происходит перед чтением с ленты (и, соответственно, записью в память), то на первом переходе в открытую память первой ячейки записывается пустое слово и оно же читается с ленты (т.к. начальная конфигурация второй ячейки памяти есть  $\varepsilon$ ).



## MFA для $\{a^{n^2}\}$

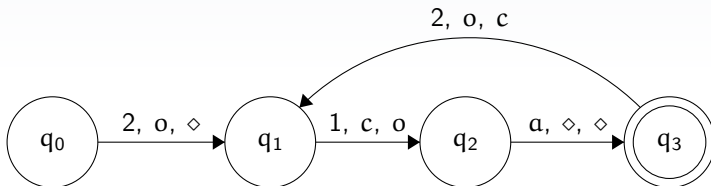


Поскольку открытие и закрытие памяти происходит перед чтением с ленты (и, соответственно, записью в память), то на первом переходе в открытую память первой ячейки записывается пустое слово и оно же читается с ленты (т.к. начальная конфигурация второй ячейки памяти есть  $\epsilon$ ). Затем память первой ячейки закрывается, и во вторую записывается содержимое первой ячейки (заодно оно же считывается с ленты). Затем считывается ещё одна буква  $a$  и также записывается во вторую ячейку.

Как итог, после достижения состояния  $q_3$  в первый раз память выглядит так:  $\langle\langle\epsilon, c\rangle, \langle a, o\rangle\rangle$



## МФА для $\{a^{n^2}\}$



Как итог, после достижения состояния  $q_3$  в первый раз память выглядит так:  $\langle \langle \varepsilon, c \rangle, \langle a, o \rangle \rangle$

После следующей итерации (на которой мы считаем с ленты слово  $a^3$  дополнительно к ранее прочитанному  $a$ ) получим конфигурацию памяти  $\langle \langle a, c \rangle, \langle aa, o \rangle \rangle$ . И вообще, каждый раз при  $k$ -ом посещении  $q_3$  получим состояние памяти вида  $\langle \langle a^{k-1}, c \rangle, \langle a^k, o \rangle \rangle$ .



## DMFA и Jumping Lemma

$k$  — MFA детерминированный, если

$\forall q \in Q, b \in \Sigma (|\bigcup_{i=1}^k \delta(q, i)| + |\delta(q, b)| \leq 1)$ . DMFL — такой язык, для которого существует DMFA.

- $[_x(a|b)^*]_x$  определяет DMFL;
  - $([_x y]_x [_y x a]_y)^*$  определяет DMFL;
  - $1^+ [_x 0^*]_x (1^+ x)^* 1^+$  — тоже DMFL (эквивалентен регулярке  $1(1^+ | 0[_x 0^*]_x 1^+ (0x1^+)^*)$ )).
- Замкнуты относительно пересечения с регулярным языком, а также относительно дополнения, но только если разрешается вводить неудачные переходы по обратным ссылкам;
  - Не замкнуты относительно объединения (и даже — объединения с регулярными языками), и относительно пересечения друг с другом.



## DMFA и Jumping Lemma

$k$  — MFA детерминированный, если

$\forall q \in Q, b \in \Sigma (|\bigcup_{i=1}^k \delta(q, i)| + |\delta(q, b)| \leq 1)$ . DMFL — такой язык, для которого существует DMFA.

Язык  $\mathcal{L} \in \text{REGEX}$  детерминированный, если либо он является регулярным, либо  $\forall m \exists n, p_n, v_n$  такие, что  $n \geq m$ ,  $p_n, v_n \in \Sigma^+$ , причём:

- $|v_n| = n$ ;
- $v_n$  — подслово  $p_n$ ;
- $p_n v_n$  — префикс какого-то слова из  $\mathcal{L}$ ;
- $\forall u \in \Sigma^+ (p_n u \in \mathcal{L} \Rightarrow v_n \text{ — префикс } u)$ .



## Пример применения JL

Язык  $\mathcal{L} = \{a^n w b^n h(w) \mid w \in a, b^* \text{ \& } h(a) = aa \text{ \& } h(b) = ab\}$  не является DMFL.

- Пересечём  $\mathcal{L}$  с  $a^* b^+$ . Получим язык  $\mathcal{L}' = \{a^n b^n\}$ .
- Предположим, что выполнены условия JL. Рассмотрим возможные значения  $v_n$ .
  - Первое  $v_n$ , для которого выполняется требуемое условие, обязано иметь вид  $a^n$ . Действительно, в противном случае при чтении префикса, состоящего из букв  $a$ , автомат мог бы принимать лишь конечное число состояний, однако классов эквивалентности по Майхиллу-Нероуду относительно языка  $a^n b^n$  в языке  $a^*$  бесконечно много.
  - $v_n = a^n$ . Тогда  $p_n = a^{n+k}$ . Слово  $a^{n+k} b^{n+k} \in \mathcal{L}'$ , но его суффикс  $b^{n+k}$  не начинается с  $v_n$ . Что доказывает непринадлежность  $\mathcal{L}'$  (а значит, и  $\mathcal{L}$ ) к DMFL.



## Отделение семантики и синтаксиса

- Все предыдущие примеры КЗ-языков выражали семантические свойства (повторения, синхронизации по аргументам, и т.д.) посредством синтаксических конструкций. В большинстве случаев это даёт выигрыш в скорости их проверки за счёт локальности алгоритмов (см. MFA или автоматы Треллиса). Но ограничивает в выразительных свойствах.
- Универсальный способ проверки семантических свойств — обход того же самого синтаксического дерева с дополнительными действиями.



## Атрибутные грамматики

Пусть  $A_0 \rightarrow A_1 \dots A_n$  — правило КС-грамматики. Припишем к нему конечное число атрибутивных свойств.

- Синтетические атрибуты вычисляются для  $A_0$  по атрибутам  $A_1, \dots, A_n$ ;
- Наследуемые атрибуты вычисляются для  $A_i$  по атрибутам  $A_0, \dots, A_{i-1}, A_{i+1}, \dots, A_n$ . Обычно — по атрибутам  $A_0$  и  $A_1, \dots, A_{i-1}$  (левосторонние атрибутные грамматики).

Повторные нетерминалы при присвоении атрибутов индексируются по вхождениям в правило слева направо. Т.е., например, если дано правило  $N \rightarrow N - N$ , тогда уравнение на атрибуты  $N_0.attr = N_1.attr - N_2.attr$  будет означать, что атрибут родителя есть атрибут левого потомка минус атрибут правого потомка, помеченных нетерминалами  $N$ .

Неповторные нетерминалы в уравнениях на атрибуты обычно не индексируются.





## Пример АГ для $\{a^n b^n c^n\}$

Атрибут нетерминала `iter` семантически означает число итераций. Чтобы не смешивать синтетические и наследуемые атрибуты, введём также атрибут `inh_iter`, означающий то же самое, но наследуемый сверху вниз по дереву разбора, а не снизу вверх. Здесь  $==$  — предикат;  $:=$  — операция присваивания.

$$\begin{aligned} S &\rightarrow AT && ; && T.iter == A.iter \\ A &\rightarrow aA && ; && A_0.iter := A_1.iter + 1 \end{aligned}$$

Синтетический вариант:

$$\begin{aligned} A &\rightarrow \varepsilon && ; && A.iter := 0 \\ T &\rightarrow bTc && ; && T_0.iter := T_1.iter + 1 \\ T &\rightarrow \varepsilon && ; && T.iter := 0 \end{aligned}$$

Вариант с наследованием:

$$\begin{aligned} S &\rightarrow AT && ; && B.inh\_iter := A.iter \\ A &\rightarrow aA && ; && A_0.iter := A_1.iter + 1 \\ A &\rightarrow \varepsilon && ; && A.iter := 0 \\ T &\rightarrow bTc && ; && T_1.inh\_iter := T_0.inh\_iter - 1 \\ T &\rightarrow \varepsilon && ; && T.inh\_iter == 0 \end{aligned}$$



## Определение типа

Понятие типа ограничивает возможные операции над его сущностями  $\Rightarrow$  исключает парадоксы (неожиданное/неприемлемое поведение программ).

Система типов — гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс



## Определение типа

Система типов — гибко управляемый синтаксический *метод доказательства* отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.

Записывается:  $\Gamma \vdash M : \sigma$ , где  $\Gamma$  — это перечисление  $x_i : \tau_i$  — ака контекст.

Читается: «в контексте  $\Gamma$  терм  $M$  имеет тип  $\sigma$ ».

Понимается: «если придать переменным  $x_i$  типы  $\tau_i$ , тогда можно установить, что тип выражения  $M$  есть  $\sigma$ ».



## Таблица связывания

КЗ-свойства имён вынуждают использовать таблицы связывания (имён и функций) с двумя базовыми операциями:

- $\text{bind} :: ([\text{таблица}], [\text{имя}], [\text{тип}]) \rightarrow [\text{таблица}];$
- $\text{lookup} :: ([\text{таблица}], [\text{имя}]) \rightarrow [\text{тип}].$



## Пример правил типизации

- Сорты (простые типы): Bool, Int.
- Операторы: =, +, условный, вызов функции.
- Синтаксис:

$$\begin{aligned} [\text{Prog}] &::= [\text{Fs}] & [\text{Fs}] &::= [\text{F}] \mid [\text{Fs}] \\ [\text{F}] &::= [\text{TypeId}] \text{ (} [\text{TIds}] \text{) } = [\text{Exp}] \\ [\text{Exps}] &::= [\text{Exp}] \mid [\text{Exp}], [\text{Exps}] \\ [\text{TypeId}] &::= (\text{Bool} \mid \text{Int}) \text{ id} & [\text{TIds}] &::= [\text{TypeId}], [\text{TIds}] \mid [\text{TypeId}] \\ [\text{Exp}] &::= \text{num} \mid \text{id} \mid [\text{Exp}] + [\text{Exp}] \mid [\text{Exp}] = [\text{Exp}] \mid \text{id} \text{ (} [\text{Exps}] \text{)} \\ &\quad \mid \text{if } [\text{Exp}] \text{ then } [\text{Exp}] \text{ else } [\text{Exp}] \mid \text{let id} = [\text{Exp}] \text{ in } [\text{Exp}] \end{aligned}$$



## Пример правил типизации

```
[Prog] ::= [Fs]           [Fs] ::= [F] | [Fs]
[F] ::= [TypeId] ([TIds]) = [Exp]
[Exps] ::= [Exp] | [Exp], [Exps]
[TypeId] ::= (Bool | Int) id [TIds] ::= [TypeId], [TIds] | [TypeId]
[Exp] ::= num | id | [Exp] + [Exp] | [Exp] = [Exp] | id ([Exps])
         | if [Exp] then [Exp] else [Exp] | let id = [Exp] in [Exp]
```



## Пример правил типизации

$$\begin{aligned} [\text{Prog}] &::= [\text{Fs}] & [\text{Fs}] &::= [\text{F}] \mid [\text{Fs}] \\ [\text{F}] &::= [\text{TypeId}] \text{ } ([\text{TIds}]) = [\text{Exp}] \\ [\text{Exps}] &::= [\text{Exp}] \mid [\text{Exp}], [\text{Exps}] \\ [\text{TypeId}] &::= (\text{Bool} \mid \text{Int}) \text{ } \mathbf{id} & [\text{TIds}] &::= [\text{TypeId}], [\text{TIds}] \mid [\text{TypeId}] \\ [\text{Exp}] &::= \mathbf{num} \mid \mathbf{id} \mid [\text{Exp}] + [\text{Exp}] \mid [\text{Exp}] = [\text{Exp}] \mid \mathbf{id} \text{ } ([\text{Exps}]) \\ &\quad \mid \text{if } [\text{Exp}] \text{ then } [\text{Exp}] \text{ else } [\text{Exp}] \mid \text{let } \mathbf{id} = [\text{Exp}] \text{ in } [\text{Exp}] \end{aligned}$$

tchExp(Exp, vtable, ftable) = case Exp of

<b>num</b>	int
<b>id</b>	$\begin{aligned} &\mid t == \text{undef} = \text{err}; \text{int} \\ &\mid \text{otherwise} = t \\ &\quad \text{where } t = \text{lookup}(\text{vtable}, \mathbf{id}) \end{aligned}$
$\text{Exp}_1 + \text{Exp}_2$	$\begin{aligned} &\mid t_1 \neq \text{int} \parallel t_2 \neq \text{int} = \text{err}; \text{int} \\ &\mid \text{otherwise} = \text{int} \\ &\quad \text{where } t_1 = \text{tchExp}(\text{Exp}_1, \text{vtable}, \text{ftable}), \\ &\quad \quad t_2 = \text{tchExp}(\text{Exp}_2, \text{vtable}, \text{ftable}) \end{aligned}$



## Пример правил типизации

$\text{tchExp}(\text{Exp}, \text{vtable}, \text{fable}) = \text{case Exp of}$

<b>num</b>	int
<b>id</b>	$  t == \text{undef} = \text{err}; \text{int}$ $  \text{otherwise} = t$ where $t = \text{lookup}(\text{vtable}, \text{id})$
$\text{Exp}_1 + \text{Exp}_2$	$  t_1 \neq \text{int} \parallel t_2 \neq \text{int} = \text{err}; \text{int}$ $  \text{otherwise} = \text{int}$ where $t_1 = \text{tchExp}(\text{Exp}_1, \text{vtable}, \text{fable})$ , $t_2 = \text{tchExp}(\text{Exp}_2, \text{vtable}, \text{fable})$
$\text{Exp}_1 = \text{Exp}_2$	$  t_1 == t_2 = \text{bool}$ $  \text{otherwise} = \text{err}; \text{bool}$ where $t_1 = \text{tchExp}(\text{Exp}_1, \text{vtable}, \text{fable})$ , $t_2 = \text{tchExp}(\text{Exp}_2, \text{vtable}, \text{fable})$





## Правила типизации в форме вывода

$$\frac{}{\Gamma \vdash \mathbf{num} : \text{int}} \quad \frac{\Gamma \vdash t_1 : \text{int}, \Gamma \vdash t_2 : \text{int}}{\Gamma \vdash t_1 + t_2 : \text{int}}$$

$$\frac{}{\Gamma, \mathbf{id} : \tau \vdash \mathbf{id} : \tau} \quad \frac{\Gamma \vdash t_1 : \sigma, \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 = t_2 : \text{bool}}$$

$$\frac{\Gamma \vdash t_1 : \text{bool}, \Gamma \vdash t_2 : \sigma, \Gamma \vdash t_3 : \sigma}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : \sigma}$$

$$\frac{\Gamma, \mathbf{f\_id} : (\tau_1, \dots, \tau_n) \rightarrow \tau_0 \vdash t_i : \tau_i}{\Gamma, \mathbf{f\_id} : (\tau_1, \dots, \tau_n) \rightarrow \tau_0 \vdash \mathbf{f\_id}(t_1, \dots, t_n) : \tau_0}$$

$$\frac{\Gamma, \mathbf{id} : \tau \vdash s : \sigma, \Gamma \vdash t : \tau}{\Gamma \vdash M, \text{let } \mathbf{id} = t \text{ in } s : \sigma}$$



## Пробное задание на РК-2

- 1 Язык, описывающийся следующей атрибутивной грамматикой:

$$S \rightarrow AT \quad ; \quad T.rng > A.iter$$

$$A \rightarrow aA \quad ; \quad A_0.iter := A_1.iter + 1$$

$$A \rightarrow \varepsilon \quad ; \quad A.iter := 0$$

$$T \rightarrow TcT \quad ; \quad T_0.rng := \max(T_1.rng, T_2.rng)$$

$$T \rightarrow K \quad ; \quad T.rng := K.rng$$

$$K \rightarrow aK \quad ; \quad K_0.rng := K_1.rng + 1$$

$$K \rightarrow bK \quad ; \quad K_0.rng := 0$$

$$K \rightarrow \varepsilon \quad ; \quad K.rng := 0$$

- 2 Язык  $\{wcvw_{\text{pref}}zw_{\text{suff}} \mid w, z \in \{a, b\}^* \text{ \& } v \in \{a, b, c\}^*\}$ .

Здесь  $w_{\text{pref}}$  — непустой префикс слова  $w$ ;  $w_{\text{suff}}$  — непустой суффикс слова  $w$ .



## Продолжение (третье задание)

- 3 Язык, описывающийся следующей атрибутивной грамматикой:

$$\begin{aligned} S &\rightarrow SbS && ; && S_0.iter := 2 \cdot S_1.iter, S_1.iter == S_2.iter \\ S &\rightarrow a && ; && S.iter := 1 \end{aligned}$$



## Классы задач

- Атрибутные грамматики: переводим в свёрточную форму или (мысленно) в неформальное описание. Далее действуем так же, как в других случаях.
- Языки SRS: определяем фрагменты разбиения на подслова, пытаемся выявить структуру подслов, при подозрении на выход из КС-языков или DCFL — пересекаем с регулярными языками.
- Языки с соотношением между частями слов: стараемся представить в более свёрточной форме через степенные соотношения над регулярками. Если не получается, то возможна не КС-структура (повторяемость подслов).



## Языки SRS

- ❶  $bb \rightarrow aa, ab \rightarrow aba$ , носитель  $b^*$ . Регулярен: правило  $ab \rightarrow aba$  влечёт правило  $ab \rightarrow aba^+$ . Далее разбиваем на подходящие под слова и итерируем их чередование.
- ❷  $ab \rightarrow baa$ , носитель  $ab^+$ . Не КС: пересечём с  $b^+a^+$ .
  - Кратчайшее слово:  $baa$ . Поскольку все  $b$  только перемещаются влево, можно предположить, что каждое очередное слово получается максимальным передвижением  $b$  с самой правой позиции на самую левую.
  - Рассмотрим цепочку превращений  $baab$ . Получаем  $baab \rightarrow babaa \rightarrow bbaaaa$ .
  - Аналогично из слова  $ba^n b$  получается слово  $bb a^{2 \cdot n}$ .
  - Полученный язык  $\{b^n a^{2^n}\}$  известен, как не КС.



## Языки SRS

- ❶  $ab \rightarrow ba$ , носитель  $ab^+$ . Не КС: пересечём с  $b^+a^+$ .
- Кратчайшее слово:  $ba$ . Поскольку все  $b$  только перемещаются влево, можно предположить, что каждое очередное слово получается максимальным передвижением  $b$  с самой правой позиции на самую левую.
  - Рассмотрим цепочку превращений  $baab$ . Получаем  $baab \rightarrow baba \rightarrow bbaaa$ .
  - Аналогично из слова  $ba^n b$  получается слово  $bb a^{2^n}$ .
  - Полученный язык  $\{b^n a^{2^n}\}$  известен, как не КС. Также не DFML: достаточно взять  $v_n = b^n$ ,  $p_n = b^{n+k}$ .  
 $b^{n+k} b^n a^{2^{n+k}}$  лежит в языке, но не всякое слово с префиксом  $p_n$ , лежащее в этом языке, продолжается на  $v_n$  (см. слово  $b^{n+k} a^{2^{n+k}}$ ).



## Языки соотношений

- Язык  $\{w_1 w_2 \mid w_i = z_{i,1} a z_{i,2} \ \& \ |z_{i,1}| = |z_{i,2}|\}$ .
- Частный случай:  $b^n a b^{n+m} a b^m$ . Похож на палиндромный, однако разница в том, что гораздо больше возможностей для выбора "середины палиндрома". Нужно их ограничить.
- Возьмём слова  $b^{p+1} a b^p b a$  и  $b^{p+1} a b^p b a b^{2p+3} a$ . Из-за невозможности взять во втором слове первую по счёту букву  $a$  за середину подслова, невозможно накачивать только префикс  $b^{p+1} a b^p$ .
- 2-исправляемый: вставляем букву  $c$  перед каждой центральной буквой  $a$  и получаем DCFL.



## Пробное-2

- 1 Язык SRS с правилами  $a \rightarrow bab$ ,  $ba \rightarrow ab$  над базисным словом  $aa$  (единственным). Подсказка: сначала решить задачу над базисным словом  $a$ .
- 2 Язык  $\{w_1 u u^R w_2 \mid |u| > 1 \text{ \& } w_i \neq z_1 u z_2\}$ . Т.е. подслово  $u$  не содержится нигде больше в слове, при любом выборе  $u$  таком, что  $u u^R$  входит в слово языка и притом  $|u| > 1$ . Подсказка: сначала рассмотреть  $|u| > 0$ .





## Продолжение (третье задание)

- 3 Язык, описывающийся следующей атрибутивной грамматикой (lookup — поиск по таблице значений Table, т.е. возвращает по [Name].id такое [Val].val, что  $([Name].id = [Val].val) \in Table$ ):

$[S] \rightarrow \{[Decl]\}[Exp]$	;	$[Exp].vars \subseteq [Decl].vars,$ $[Exp].val == 1, [Exp].inh\_table := [Decl].table$
$[Decl] \rightarrow$ $([Name] = [Val])[Decl]$	;	$[Name].id \notin [Decl]_1.vars,$ $[Decl]_0.table := [Decl]_1.table \cup \{[Name].id = [Val].val\},$ $[Decl]_0.vars := [Decl]_1.vars \cup \{[Name].id\}$
$[Decl] \rightarrow \varepsilon$	;	$[Decl].vars := \emptyset, [Decl].table := \emptyset$
$[Exp] \rightarrow [Name]$	;	$[Exp].val := lookup([Name].id, [Exp].inh\_table)$
$[Exp] \rightarrow [Exp] \& [Exp]$	;	$[Exp]_0.val := \min([Exp]_1.val, [Exp]_2.val),$ $[Exp]_1.inh\_table := [Exp]_0.inh\_table,$ $[Exp]_2.inh\_table := [Exp]_0.inh\_table$
$[Name] \rightarrow a[Name]$	;	$[Name]_0.id := a ++ [Name]_1.id$
$[Name] \rightarrow \varepsilon$	;	$[Name].id := \varepsilon$
$[Val] \rightarrow 0$	;	$[Val].val := 0$
$[Val] \rightarrow 1$	;	$[Val].val := 1$