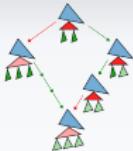


Базовые понятия теории формальных языков

Теория формальных языков
2022 г.



Формальные языки

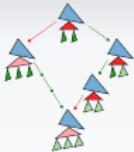
Традиционный подход

Формальный язык — это множество \mathcal{M} слов над алфавитом Σ (обозначается $\mathcal{M} \subseteq \Sigma^*$, здесь знак $*$ — итерация Клини). Обычно подразумевает наличие формальных правил, определяющих корректность формы (т.е. синтаксиса) слов из \mathcal{M} .

Теоретико-категорный подход

Формальный язык — это категория (т.е. способ задания объектов и их взаимосвязей в форме направленного графа). Отличие от теоретико-множественного подхода — способ описания не синтаксиса слов, а отношения между ними.

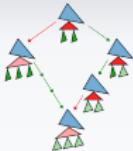
Классический пример: A man saw a dog with a telescope.



Перечислимость и разрешимость

- Язык \mathcal{M} разрешимый \Leftrightarrow для любого слова w существует алгоритм проверки принадлежности w к \mathcal{M} (всегда завершающийся и дающий точный, либо положительный, либо отрицательный ответ).
- Язык \mathcal{M} перечислимый \Leftrightarrow для любого слова w существует алгоритм, положительно отвечающий на вопрос принадлежности w к \mathcal{M} за конечное время (но, возможно, зацикливающийся, если $w \notin \mathcal{M}$).

Перечислимый, но не разрешимый: язык программ, завершающихся на входе 0 (на любом достаточно мощном ЯП). Далее разрешимые языки можно классифицировать по минимально необходимой сложности разрешающего алгоритма (P-разрешимые, ExpTime-разрешимые...)



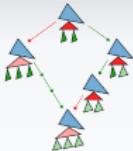
Примеры формальных языков

- $\{\underbrace{aa\dots a}_{n \text{ раз}} \underbrace{bb\dots b}_{n \cdot 3 \text{ раз}}\}$ (сокращаем до $\{a^n b^{3n}\}$);
- палиндромы чётной длины в русском языке;
- правильно записанные арифметические выражения с \cdot , $+$ над натуральными числами;
- правильные скобочные последовательности;
- язык тождественно истинных формул логики предикатов;
- язык правильно типизированных программ на ЯП со статическими типами;
- язык, описывающий все разрешимые за линейное время формальные языки.



Представления формальных языков

- Свёртки множеств
- Системы переписывания термов
- Распознающие / порождающие машины
- Алгебраические выражения
- Алгебраические структуры
- Формулы логики предикатов



Пример представления

Язык слов в алфавите $\{a, b\}$ с чётным количеством букв a .

- Свёртка: $\{w \in \{a, b\}^* \mid 2 \text{ делит } |w|_a\}$. $|w|_t$ — количество вхождений терма t в слово w .
- Система переписывания термов:

$$\begin{array}{lll} S \rightarrow "a" ++ T & S \rightarrow "b" ++ S & S \rightarrow \varepsilon \\ T \rightarrow "a" ++ S & T \rightarrow "b" ++ T & \end{array}$$

А можно и по-другому:

$$\begin{array}{ll} S \rightarrow "a" ++ S ++ "a" & S \rightarrow "b" ++ S \\ S \rightarrow S ++ "b" & S \rightarrow \varepsilon \end{array}$$

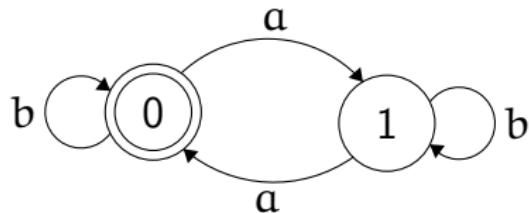
Здесь и далее ε — стандартное обозначение для пустого слова (строки нулевой длины). Поскольку структура данных — слова, то кавычки и знак конкатенации $++$ дальше опускаются.



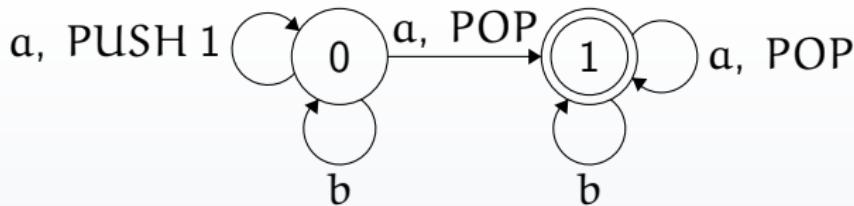
Пример представления

Язык слов в алфавите $\{a, b\}$ с чётным количеством букв a .

- Распознающие машины:



А можно иначе:





Пример представления

Язык слов в алфавите $\{a, b\}$ с чётным количеством букв a .

- Алгебраические выражения:

$$(b^*ab^*ab^*)^*$$

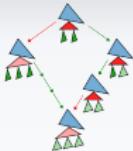
А можно и так:

$$(b^*ab^*a)^*b^*$$

- Алгебраические структуры:

Класс эквивалентности слова ε в полугруппе с соотношениями $aa \rightarrow \varepsilon$, $b \rightarrow \varepsilon$.

- Формулы логики предикатов: без введения считающих предикатов не выражима в логике предикатов первого порядка (но выражима в логике одноместных предикатов второго порядка).



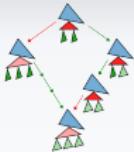
Анализ свойств языков

Проверить, действительно ли данная система переписывания термов порождает язык $\{w \mid |w|_a \text{ делится на } 2\}$, если начальным состоянием является S .

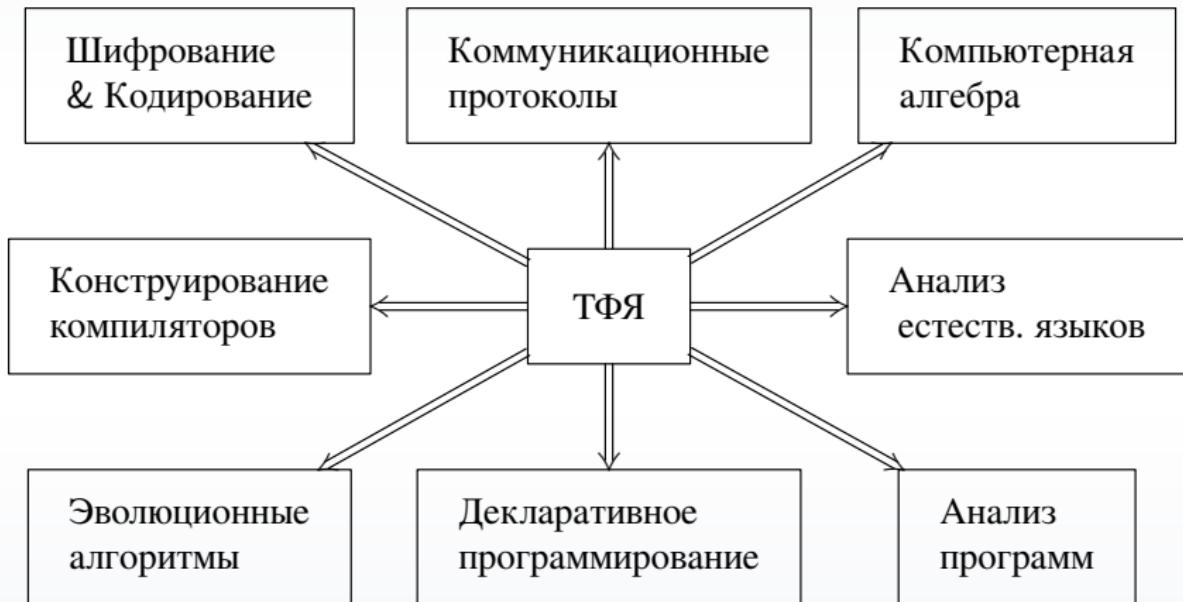
$$S \rightarrow a S a \quad S \rightarrow b S \quad S \rightarrow S b \quad S \rightarrow \epsilon$$

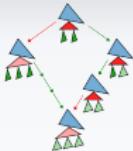
- Необходимо доказать, что все указанные слова порождаются системой (например, по индукции).
- А также что никакие другие не порождаются.

Предположим, что система порождает слова с нечётным числом букв a . Выберем из них такое, которое выводится из S за самое малое число шагов. Покажем, что каким бы ни был предпоследний шаг вывода, его можно поменять на $S \rightarrow \epsilon$ и получится слово с нечётным числом букв a , вывод которого ещё короче.



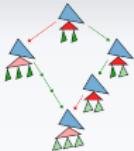
Области применения





Структура курса

- Два рубежных контроля × 15 баллов.
- Пять лабораторных работ × 8 баллов.
 - Java, Python, Go, JS — без бонуса
 - C/C++, Kotlin, TypeScript — бонус +1 балл
 - Rust, Dart, все лиспы, Scala — бонус +2 балла
 - Lua, Haskell, Erlang, Рефал — бонус +3 балла
 - Agda, Idris (с доказательствами) — 5 баллов за курс
- С момента выдачи лабораторной работы:
 - 0-14 дней — сдача за полный балл
 - 15-21 день — сдача со штрафом -1 балл
 - 22-28 дней — сдача со штрафом -2 балла
 - 29-∞ — сдача со штрафом -3 балла



Системы переписывания термов

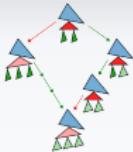
Определение

Сигнатура — множество пар $\langle f, n \rangle$ из имени конструктора f и его местности n .

Определение

Пусть V — множество переменных, F — множество конструкторов; множество термов $T(F)$ над F определяется рекурсивно:

- все элементы V — термы;
- если $\langle f, n \rangle$ — конструктор и t_1, \dots, t_n — термы, то $f(t_1, \dots, t_n)$ — терм;
- других термов нет.

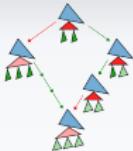


Term Rewriting Systems

Пусть V — множество переменных, F — множество конструкторов (сигнатура); $T(F)$ — множество термов над множеством конструкторов F . TRS — набор правил переписывания вида $\Phi_i \rightarrow \Psi_i$, где Φ_i, Ψ_i — термы в $T(F)$. Правило переписывания $\Phi_i \rightarrow \Psi_i$ применимо к терму t , если t содержит подтерм, который можно сопоставить (унифицировать) с Φ_i .

Если к терму t не применимо ни одно правило переписывания TRS, терм называется нормализованным.

Имея правила переписывания вида $f(g(x)) \rightarrow g(g(f(x)))$ и $g(g(x)) \rightarrow f(x)$, каждое из них можно применить к терму $f(g(g(f(g(g(g(g(Z))))))))$ тремя разными способами.



Конфлюэнтность

Определение

TRS называется конфлюэнтной, если для любых двух термов t, s , которые получаются переписыванием одного и того же терма u , существует терм v такой, что t, s оба переписываются в v .

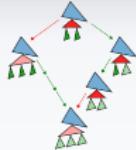
Формально:

$$\forall u, t, s (u \rightarrow^* t \& u \rightarrow^* s \Rightarrow \exists v (t \rightarrow^* v \& s \rightarrow^* v))$$

Конфлюэнтные системы поддаются распараллеливанию и легко оптимизируются.

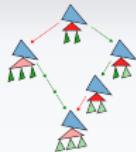
\rightarrow — переписывание за 1 шаг;

\rightarrow^* — переписывание за произвольное число шагов, начиная с 0.



Особенности TRS

- Недетерминированные.
- Нет ограничений на порядок применения правил.
- Не обязательно конфлюэнтны.
- Могут порождать бесконечные цепочки.



Особенности TRS

- Недетерминированные.
- Нет ограничений на порядок применения правил.
- Не обязательно конфлюэнтны.
- Могут порождать бесконечные цепочки.

Пример Хетта

$$f(x, x) \rightarrow a$$

$$f(x, g(x)) \rightarrow b$$

$$c \rightarrow g(c)$$

Терм, где нарушается конфлюэнтность?



Особенности TRS

- Недетерминированные.
- Нет ограничений на порядок применения правил.
- Не обязательно конфлюэнтны.
- Могут порождать бесконечные цепочки.

Пример Клопа

$$A \rightarrow CA$$

$$Cz \rightarrow Dz(Cz)$$

$$Dzz \rightarrow E$$

Способы преобразовать A?



Особенности TRS

- Недетерминированные.
- Нет ограничений на порядок применения правил.
- Не обязательно конфлюэнтны.
- Могут порождать бесконечные цепочки.

Пример Тойямы

- TRS 1:

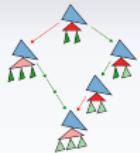
$$f(0, 1, x) \rightarrow f(x, x, x)$$

- TRS 2:

$$g(x, y) \rightarrow x$$

$$g(x, y) \rightarrow y$$

Как можно вычислить $f(g(0, 1), g(0, 1), g(0, 1))$?

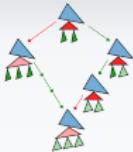


Фундированность

Определение

Частичный порядок \preceq является фундированным (wfo) на множестве M , если в M не существует бесконечных нисходящих цепочек относительно \preceq (говоря о множестве термов, иногда такой \preceq называют нётеровым).

Частичный порядок \preceq является монотонным в алгебре A , если $\forall f, t_1, \dots, t_n, s, s' (s \preceq s' \Rightarrow f(t_1, \dots, s, \dots, t_n) \preceq f(t_1, \dots, s', \dots, t_n))$ (строго монотонным, если при этом неверно обратное).



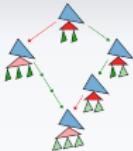
Завершаемость

Определение

Фундированная монотонная алгебра (ФуМА) над множеством функциональных символов F — это фундированное множество $\langle A, > \rangle$ такое, что для каждого функционального символа $f \in F$ существует функция $f_A : A^n \rightarrow A$, строго монотонная по каждому из аргументов.

Определим расширение произвольного отображения σ из множества переменных в A следующим образом:

- $[x, \sigma] = \sigma(x);$
- $[f(t_1, \dots, t_n), \sigma] = f_A([t_1, \sigma], \dots, [t_n, \sigma]).$



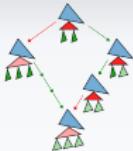
Завершаемость

Совместность

TRS $\{l_i \rightarrow r_i\}$ совместна с ФуМА $A \Leftrightarrow$ для всех i и для всех σ выполняется условие $[l_i, \sigma] > [r_i, \sigma]$.

Теорема

TRS не порождает бесконечных вычислений (завершается), если и только если существует совместная с ней ФуМА.

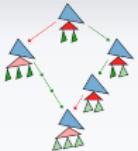


ФуМА, совместные с TRS

Стандартные способы определения f_A :

- лексикографический порядок на множестве имён F + отношение подтерма;
- построение монотонно возрастающей (по каждому аргументу) числовой функции, соответствующей f_A .

Оба случая подразумевают, что в построенной модели целое больше части, т.е. всегда выполняется $f(t) > t$.



Лексикографический порядок $>_{lo}$

Определение

$f(t_1, \dots, t_n) >_{lo} g(u_1, \dots, u_m)$ (этот порядок также называют порядком Кнута–Бендиекса) если и только если выполнено одно из условий:

- ❶ $\exists i (1 \leq i \leq n \ \& \ t_i = g(u_1, \dots, u_m));$
- ❷ $\exists i (1 \leq i \leq n \ \& \ t_i >_{lo} g(u_1, \dots, u_m));$
- ❸ $(f > g) \ \& \ \forall i (1 \leq i \leq m \Rightarrow f(t_1, \dots, t_n) >_{lo} u_i);$
- ❹ $(f = g) \ \& \ \forall i (1 \leq i \leq n \Rightarrow f(t_1, \dots, t_n) >_{lo} u_i)$ и n -ка (t_1, \dots, t_n) лексикографически больше, чем (u_1, \dots, u_n) (т.е. первый её не совпадающий с u_i элемент t_i удовлетворяет условию $t_i >_{lo} u_i$).

Примеры

Проверить завершаемость TRS методом $>_{\text{lo}}$:

$$f(g(x)) \rightarrow g(h(x, x))$$

$$g(f(x)) \rightarrow h(g(x), x)$$

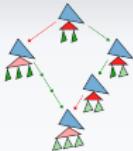
- Первое правило переписывания вынуждает либо $g(x) >_{\text{lo}} g(h(x, x))$ (по условию 1 или 2) — что невозможно, потому что x должно лексикографически оказаться больше $h(x, x)$ (по условию 4); либо $f > g$ и $f(g(x)) > h(x, x)$ (по условию 3). В этом случае можно взять также $f > h$. Неравенство $f(g(x)) > x$ выполняется тривиально.
- Второе правило переписывания удовлетворяет условию завершаемости по условию 2, например, если показать, что $f(x) >_{\text{lo}} h(g(x), x)$. Уже имеем $f > h$, поэтому достаточно показать $f(x) >_{\text{lo}} g(x)$ и $f(x) >_{\text{lo}} x$. Оба условия тривиально выполняются из допущений выше.

Примеры

Проверить завершаемость TRS методом построения монотонной функции:

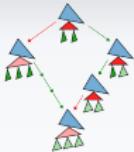
$$\begin{aligned}f(g(x, y)) &\rightarrow g(h(y), x) \\h(f(x)) &\rightarrow f(x).\end{aligned}$$

- Завершаемость по второму правилу переписывания автоматически выполняется по свойству подтерма. Поэтому то, что функция f стоит на двух его сторонах, не дает никаких указаний относительно того, стоит ли делать f_A быстро растущей или медленно. Все подсказки содержатся только в первом правиле переписывания.
- По первому правилу переписывания видно, что f_A надо делать большой (f стоит только слева), а h_A нет (h есть только справа). Положим $f_A(x) = 10 \cdot (x + 1)$, $h_A(x) = x + 1$. Тогда должно выполняться $10 \cdot (g_A(x, y) + 1) > g_A(y + 1, x)$. Этому неравенству удовлетворяет, например, $g_A(x, y) = x + y$.



Общие комментарии

- Не обязательно добиваться выполнения неравенства на образах f_A на всём множестве \mathbb{N} . Поскольку любой отрезок \mathbb{N} от k и до бесконечности фундирован, а все образы f_A монотонны, они замкнуты на этом отрезке. Поэтому, если неравенство не выполняется для нескольких первых чисел натурального ряда, этим можно пренебречь.
- Если не получается применить $>_{1_0}$ или подобрать числовую функцию, это ещё не значит, что TRS не завершается. См. пример Зантемы:
 $f(g(x)) \rightarrow g(f(f(x)))$.

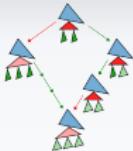


Терминалы и нетерминалы

Если TRS определена над алфавитом Σ , а нас интересует порождаемый ею язык в $\Sigma' \subset \Sigma$, то элементы Σ' обычно называются терминалами, а элементы $\Sigma \setminus \Sigma'$ — нетерминалами.

В этом случае значащие (порождающие) нетерминалы обязательно должны встречаться хотя бы в одной левой части правила переписывания (иначе такой нетерминал не сможет быть переписан в слово над Σ').

Терминалы также могут встречаться в левых частях правил (это не так только для некоторых классов систем переписывания термов).



Грамматики

Определение

Грамматика — это четвёрка $G = \langle N, \Sigma, P, S \rangle$, где:

- N — алфавит нетерминалов;
- Σ — алфавит терминалов;
- P — множество правил переписывания $\alpha \rightarrow \beta$ типа $\langle (N \cup \Sigma)^+ \times (N \cup \Sigma)^* \rangle$;
- $S \in N$ — начальный символ.

$\alpha \Rightarrow \beta$, если $\alpha = \gamma_1 \alpha' \gamma_2$, $\beta = \gamma_1 \beta' \gamma_2$, и $\alpha' \rightarrow \beta' \in P$.

\Rightarrow^* — рефлексивное транзитивное замыкание \Rightarrow .

Определение

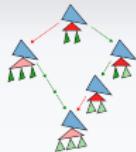
Язык $L(G)$, порождаемый G — множество
 $\{u \mid u \in \Sigma^* \text{ & } S \Rightarrow^* u\}$.



α -преобразование

По-разному воспринимают переименовку:

- Переменные vs конструкторы в TRS;
- Нетерминалы vs терминалы в грамматиках.



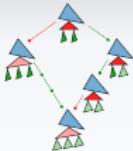
α -преобразование

По-разному воспринимают переименовку:

- Переменные vs конструкторы в TRS;
- Нетерминалы vs терминалы в грамматиках.

Для любой инъективной переименовки σ применение σ к правилам грамматики/trs для переменных и нетерминалов также называется α -преобразованием.

- α -преобразование не меняет терминальный язык;
- обычно термы различаются с точностью до α -преобразования.



α -преобразование

По-разному воспринимают переименовку:

- Переменные vs конструкторы в TRS;
- Нетерминалы vs терминалы в грамматиках.

Для любой инъективной переименовки σ применение σ к правилам грамматики/trs для переменных и нетерминалов также называется α -преобразованием.

- α -преобразование не меняет терминальный язык;
- обычно термы различаются с точностью до α -преобразования.

Неформально: контейнеры определяются не именем, а содержимым (см. экстенсиональность в логике).

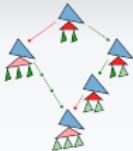


Иерархия Хомского без ε -правил

$A, B \in N, a \in \Sigma^*, \alpha, \beta \in (N \cup \Sigma)^*, \gamma \in (N \cup \Sigma)^+$

Иерархия грамматик

Тип 0	Рекурсивно-перечислимые	\forall
Тип 1	Контекстно-зависимые	$\alpha A \beta \rightarrow \alpha \gamma \beta, \gamma \neq \varepsilon$
Тип 2	Контекстно-свободные	$A \rightarrow \gamma$
Тип 3	Праволинейные (регулярные)	$A \rightarrow a, A \rightarrow aB$



Иерархия Хомского без ε -правил

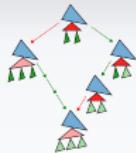
$A, B \in N, a \in \Sigma^*, \alpha, \beta \in (N \cup \Sigma)^*, \gamma \in (N \cup \Sigma)^+$

Иерархия грамматик

Тип 0	Рекурсивно-перечислимые	\forall
Тип 1	Контекстно-зависимые	$\alpha A \beta \rightarrow \alpha \gamma \beta, \gamma \neq \varepsilon$
Тип 2	Контекстно-свободные	$A \rightarrow \gamma$
Тип 3	Праволинейные (регулярные)	$A \rightarrow a, A \rightarrow aB$

Примеры языков

- Тип 0 $\{u \mid L(u) = L(r)\}, r — \text{фикс. regex}, u — \text{regex};$
- Тип 1 $\{ww \mid w \in \Sigma^+\}$
- Тип 2 непустые палиндромы в алфавите $\{a, b\}$
- Тип 3 $\{w \mid w = aw_1 \& (w_1 = a^{2k} \vee w_1 = a^{3k} \vee w_1 \neq a^{5k})\}$



Иерархия Хомского с ε -правилами

$A, B \in N, a \in \Sigma^*, \alpha, \beta \in (N \cup \Sigma)^*, \gamma \in (N \cup \Sigma)^+.$

Иерархия грамматик

Тип 0 Рекурсивно-перечислимые \forall

Тип 1 Контекстно-зависимые $\alpha A \beta \rightarrow \alpha \gamma \beta, \gamma \neq \varepsilon$

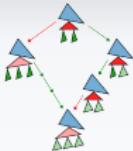
$\vee S \rightarrow \varepsilon \text{ & } \forall p : \alpha \rightarrow \beta \in P \forall \beta_1, \beta_2 (\beta \neq \beta_1 \wedge \beta_2)$

Тип 2 Контекстно-свободные $A \rightarrow \alpha$

Тип 3 Регулярные $A \rightarrow a, A \rightarrow aB, A \rightarrow \varepsilon$

Регулярные грамматики и выражения. Теорема Клини

Теория формальных языков
2022 г.



Грамматики

Определение

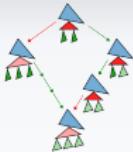
Грамматика — это четвёрка $G = \langle N, \Sigma, P, S \rangle$, где:

- N — алфавит нетерминалов;
- Σ — алфавит терминалов;
- P — множество правил переписывания $\alpha \rightarrow \beta$ типа $\langle (N \cup \Sigma)^+ \times (N \cup \Sigma)^* \rangle$;
- $S \in N$ — начальный символ.

$\alpha \Rightarrow \beta$, если $\alpha = \gamma_1 \alpha' \gamma_2$, $\beta = \gamma_1 \beta' \gamma_2$, и $\alpha' \rightarrow \beta' \in P$.

\Rightarrow^* — рефлексивное транзитивное замыкание \Rightarrow .

Язык $\mathcal{L}(G)$, порождаемый G — множество
 $\{u \mid u \in \Sigma^* \& S \Rightarrow^* u\}$. Сентенциальная форма — элемент
множества $\{u \mid u \in (N \cup \Sigma)^* \& S \Rightarrow^* u\}$.

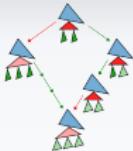


Регулярные грамматики и НКА

Регулярная грамматика имеет правила вида $S \rightarrow \varepsilon$ (причём S не встречается в правых частях никаких правил), $T_i \rightarrow a_i$, $T_i \rightarrow a_i T_j$.

НКА (неформально) определяется списком правил перехода и финальными состояниями.

- $T_i \rightarrow a_i T_j$ соответствует переходу $\langle T_i, a_i, T_j \rangle$;
- $T_i \rightarrow a_i$ соответствует переходу $\langle T_i, a_i, F \rangle$, где F — уникальное финальное состояние;
- $S \rightarrow \varepsilon$ соответствует объявлению S финальным.



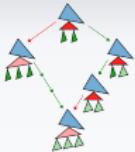
Лемма о накачке

Рассмотрим слово $w \in \mathcal{L}(G)$, $|w| \geq n + 1$. Оно получается применением не меньше, чем $n + 1$ правил \Rightarrow после применения хотя бы двух из них в сентенциальной форме справа будет стоять один и тот же нетерминал A .

$$S \rightarrow \cdots \rightarrow \underbrace{\Phi A \rightarrow \cdots \rightarrow \Phi \Psi A}_{\text{не больше } n \text{ шагов}} \rightarrow \cdots \rightarrow \Phi \Psi \Theta$$

\Downarrow

$$|\Phi| + |\Psi| \leq n$$



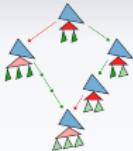
Лемма о накачке

Рассмотрим слово $w \in \mathcal{L}(G)$, $|w| \geq n + 1$. Оно получается применением не меньше, чем $n + 1$ правил \Rightarrow после применения хотя бы двух из них в сентенциальной форме справа будет стоять один и тот же нетерминал A .

Известно, что $|\Phi| + |\Psi| \leq n$.

$$S \xrightarrow{\underbrace{\quad\quad\quad}_{\rho_1: \text{вывод } \Phi A \text{ из } S}} \cdots \xrightarrow{\Phi A} \overbrace{\cdots \xrightarrow{\underbrace{\quad\quad\quad}_{\rho_2: \text{вывод } \Psi A \text{ из } A}} \Phi \Psi A}^{\rho_2: \text{вывод } \Psi A \text{ из } A} \cdots \xrightarrow{\Phi \Psi} \overbrace{\cdots \xrightarrow{\underbrace{\quad\quad\quad}_{\rho_3: \text{вывод } \Theta \text{ из } A}} \Phi \Psi \Theta}^{\rho_3: \text{вывод } \Theta \text{ из } A}$$

Все выводы вида $\rho_1 (\rho_2)^* \rho_3$ допустимы в $G \Rightarrow \forall k (\Phi \Psi^k \Theta \in \mathcal{L}(G))$.



Лемма о накачке

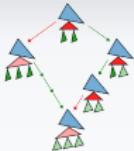
Утверждение

Если G — регулярная, то существует такое $n \in \mathbb{N}$, что
 $\forall w (w \in \mathcal{L}(G) \& |w| > n \Rightarrow \exists w_1, w_2, w_3 (|w_2| > 0 \& |w_1| + |w_2| \leq n \& w = w_1 w_2 w_3 \& \forall k (k \geq 0 \Rightarrow w_1 w_2^k w_3 \in \mathcal{L}(G))))$.

Известно, что $|\Phi| + |\Psi| \leq n$.

$$S \xrightarrow{\underbrace{\dots \xrightarrow{\Phi} A}_{\rho_1: \text{вывод } \Phi A \text{ из } S}} \overbrace{A \xrightarrow{\dots} \Phi}^{\rho_2: \text{вывод } \Psi A \text{ из } A} \overbrace{\Psi A \xrightarrow{\dots} \Phi}^{\rho_3: \text{вывод } \Theta \text{ из } A} \Psi \Theta$$

Все выводы вида $\rho_1 (\rho_2)^* \rho_3$ допустимы в $G \Rightarrow \forall k (\Phi \Psi^k \Theta \in \mathcal{L}(G))$.



Примеры применения леммы о накачке

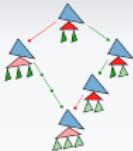
Обозначим обращение (reversal) слова w как w^R .

Рассмотрим язык $\mathcal{L} = \{ww^R \mid w \in \Sigma^+\}$.

Пусть длина накачки — n . Рассмотрим слово

$b^{n+1}a a b^{n+1} \in \mathcal{L}$. Поскольку $|\Phi| + |\Psi| \leq n$, то

$\Psi = b^k$, $k \geq 1$. Но $b^m a a b^n \notin \mathcal{L}$, если $m \neq n$. Поэтому \mathcal{L} — не регулярный.



Примеры применения леммы о накачке

Обозначим обращение (reversal) слова w как w^R .

Рассмотрим язык $\mathcal{L} = \{ww^R \mid w \in \Sigma^+\}$.

Пусть длина накачки — n . Рассмотрим слово

$b^{n+1}ab^n b^{n+1} \in \mathcal{L}$. Поскольку $|\Phi| + |\Psi| \leq n$, то

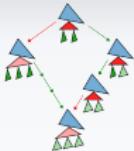
$\Psi = b^k$, $k \geq 1$. Но $b^m ab^n \notin \mathcal{L}$, если $m \neq n$. Поэтому \mathcal{L} — не регулярный.

Рассмотрим язык $\mathcal{L}' = \{a^n b^m \mid n \neq m\}$.

Пусть длина накачки — n . Рассмотрим множество слов

$a^n b^{n+n!} \in \mathcal{L}'$. Поскольку $|\Phi| + |\Psi| \leq n$, то $\Psi = a^k$, $k \geq 1$.

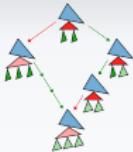
Но для всех $k \leq n \exists v (n + k * v = n + n!)$. Поэтому слово вида $a^{n+n!} b^{n+n!} \in \mathcal{L}'$, что абсурдно. Следовательно, \mathcal{L}' не является регулярным.



Нерегулярные языки

Пусть $\mathcal{L} = \{w \mid |w|_a = |w|_b\}$. Все слова вида $a^k b^k$ принадлежат \mathcal{L} . Пусть длина накачки равна n . Рассмотрим слово $a^n b^n$. Поскольку $|\Phi| + |\Psi| \leq n$, то $\Psi = a^k$, $k > 0$. Но слова $a^{n+k+i} b^n$ не принадлежат \mathcal{L} .

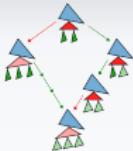
Совпадает ли \mathcal{L} с языком правильных скобочных последовательностей P (язык Дика)? Если да, доказать. Если нет, исследовать язык $L \setminus P$. Регулярен ли он?



Анализ на достаточность

Гипотеза

G — регулярная $\xleftrightarrow{??}$ существует такое $n \in \mathbb{N}$, что $\forall w (w \in \mathcal{L}(G) \& |w| > n \Rightarrow \exists w_1, w_2, w_3 (|w_2| > 0 \& |w_1| + |w_2| \leq n \& w = w_1 w_2 w_3 \& \forall k (k \geq 0 \Rightarrow w_1 w_2^k w_3 \in \mathcal{L}(G))))$.



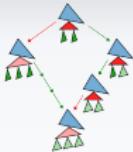
Анализ на достаточность

Гипотеза

G — регулярная $\xleftrightarrow{??}$ существует такое $n \in \mathbb{N}$, что $\forall w (w \in \mathcal{L}(G) \& |w| > n \Rightarrow \exists w_1, w_2, w_3 (|w_2| > 0 \& |w_1| + |w_2| \leq n \& w = w_1 w_2 w_3 \& \forall k (k \geq 0 \Rightarrow w_1 w_2^k w_3 \in \mathcal{L}(G))))$.

Рассмотрим язык $\mathcal{L} = \{ww^Rz \mid w \in \Sigma^+ \& z \in \Sigma^+\}$ и $n = 4$.

- Если $|w| = 1$, тогда можно разбить слово ww^Rz так: $\Phi = ww^R$, $\Psi = z[1]$, $\Theta = z[2..|z|]$. Тогда для всех k $\Phi \Psi^k \Theta \in \mathcal{L}$.
- Если $|w| \geq 2$, тогда разбиваем так: $\Phi = \varepsilon$, $\Psi = w[1]$, $\Theta = w[2..|w|] w^R z$. Слова $w[2..|w|] w^R z$ и $w[1]^k w[2..|w|] w^R z$ при $k \geq 2$ также принадлежат \mathcal{L} .



Смысл леммы о накачке

Структура доказательства указывает, что длина накачки n регулярного языка \mathcal{L} не больше (возможно, меньше) числа нетерминалов в минимальной грамматике для \mathcal{L} .

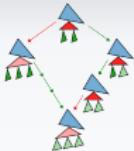
Рассмотрим $\mathcal{L} = a \mid b \mid (a \{a \mid b\}^* a) \mid (b \{a \mid b\}^* b)$. Если выбрать $n = 2$, то в качестве Ψ можно взять вторую букву слова из \mathcal{L} . Пусть G имеет два нетерминала S, T и распознаёт \mathcal{L} . Если G содержит правила $S \rightarrow aT$ и $S \rightarrow bT$ (или $S \rightarrow aS, S \rightarrow bS$), то для некоторого непустого z слова вида az и bz будут либо оба принадлежать \mathcal{L} , либо нет, чего не может быть. Значит, G содержит либо пару $S \rightarrow aT, S \rightarrow bS$, либо пару $S \rightarrow bT, S \rightarrow aS$. Рассмотрим первый случай. Тогда для некоторого непустого z имеем $az \in \mathcal{L} \Leftrightarrow b^+az \in \mathcal{L}$, что абсурдно.



Академические регулярные выражения \mathcal{RE}

Допустимые операции

- A^* — замыкание Клини — ноль или больше итераций A ;
- A^+ — одна или больше итерация A ;
- $A?$ — 0 или 1 вхождение A ;
- $A \mid B$ — альтернатива (вхождение либо A , либо B).



Академические регулярные выражения \mathcal{RE}

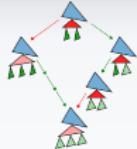
Допустимые операции

- A^* — замыкание Клини — ноль или больше итераций A ;
- A^+ — одна или больше итерация A ;
- $A?$ — 0 или 1 вхождение A ;
- $A \mid B$ — альтернатива (вхождение либо A , либо B).

Следствия

Если $r_1, r_2 \in \mathcal{RE}$, тогда

- $r_1 \mid r_2 \in \mathcal{RE}$;
- $r_1r_2 \in \mathcal{RE}$;
- $r_1^*, r_2^+ \in \mathcal{RE}$.

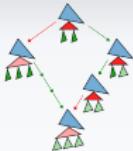


Операции в регулярных грамматиках

Объединение

Дано: G_1 и G_2 — праволинейные. Построить
 $G : \mathcal{L}(G) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.

- ① Переименовать нетерминалы из N_1 и N_2 , чтобы стало $N_1 \cap N_2 = \emptyset$ (сделать α -преобразование). Применить переименовку к правилам G_1 и G_2 .
- ② Объявить стартовым символом свежий нетерминал S и для всех правил G_1 вида $S_1 \rightarrow \alpha$ и правил G_2 вида $S_2 \rightarrow \beta$, добавить правила $S \rightarrow \alpha$, $S \rightarrow \beta$ в правила G .
- ③ Добавить в правила G остальные правила из G_1 и G_2 .



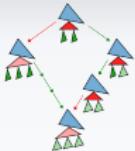
Операции в регулярных грамматиках

Конкатенация

Дано: G_1 и G_2 — праволинейные. Построить

$$G : \mathcal{L}(G) = \mathcal{L}(G_1) \mathcal{L}(G_2).$$

- ① Переименовать нетерминалы из N_1 и N_2 , чтобы стало $N_1 \cap N_2 = \emptyset$ (сделать α -преобразование).
- ② Построить из G_1 её вариант без ε -правил (см. ниже).
- ③ По всякому правилу из G_1 вида $A \rightarrow a$ строим правило G вида $A \rightarrow aS_2$, где S_2 — стартовый нетерминал G_2 .
- ④ Добавить в правила G остальные правила из G_1 и G_2 . Объявить S_1 стартовым.
- ⑤ Если $\varepsilon \in \mathcal{L}(G_1)$ (до шага 2), то по всем $S_2 \rightarrow \beta$ добавить правило $S_1 \rightarrow \beta$.

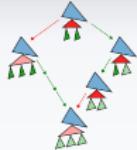


Операции в регулярных грамматиках

Положительная итерация Клини

Дано: G_1 — праволинейная. Построить
 $G : \mathcal{L}(G) = \mathcal{L}(G_1)^+$.

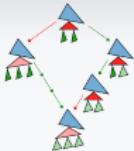
- ① Построить из G_1 её вариант без ε -правил.
- ② По всякому правилу из G_1 вида $A \rightarrow a$ строим правило G вида $A \rightarrow aS_1$, где S_1 — стартовый нетерминал G_1 .
- ③ Добавить в правила G все (включая вида $A \rightarrow a$) правила из G_1 . Объявить S_1 стартовым.
- ④ Если $\varepsilon \in \mathcal{L}(G_1)$ (до шага 2), добавить правило $S_1 \rightarrow \varepsilon$ и вывести S_1 из рекурсии.



Построение грамматики без ε -правил

Дано: G — праволинейная. Построить G' без правил вида $A \rightarrow \varepsilon$ такую, что $\mathcal{L}(G') = \mathcal{L}(G)$ или $\mathcal{L}(G') \cup \{\varepsilon\} = \mathcal{L}(G)$.

- ① Перенести в G' все правила G , не имеющие вид $A \rightarrow \varepsilon$.
- ② Если существует правило $A \rightarrow \varepsilon$, то по всем правилам вида $B \rightarrow aA$ дополнительно строим правила $B \rightarrow a$.



Пересечение регулярных грамматик

Дано: G_1, G_2 — праволинейные. Построить G' такую, что

$$\mathcal{L}(G') = \mathcal{L}(G_1) \cap \mathcal{L}(G_2).$$

- ❶ Построить стартовый символ G' — пару $\langle S_1, S_2 \rangle$, где S_i — стартовый символ грамматики G_i .
- ❷ Поместить $\langle S_1, S_2 \rangle$ в множество U неразобранных нетерминалов. Множество T разобранных нетерминалов объявить пустым.
- ❸ Для каждого очередного нетерминала $\langle A_1, A_2 \rangle \in U$:
 - ❶ если $A_1 \rightarrow a \in G_1, A_2 \rightarrow a \in G_2$, тогда добавить в G' правило $\langle A_1, A_2 \rangle \rightarrow a$;
 - ❷ если $A_1 \rightarrow aA_3 \in G_1, A_2 \rightarrow aA_4 \in G_2$, тогда добавить в G' правило $\langle A_1, A_2 \rangle \rightarrow a\langle A_3, A_4 \rangle$, а в U — нетерминал $\langle A_3, A_4 \rangle$, если его ещё нет в множестве T ;
 - ❸ если все пары правил, указанные выше, были обработаны, тогда переместить $\langle A_1, A_2 \rangle$ из U в T .
- ❹ Повторять шаг 3, пока множество U не пусто.
- ❺ Если $\varepsilon \in \mathcal{L}(G_1) \& \varepsilon \in \mathcal{L}(G_2)$, тогда добавить в G' правило $\langle S_1, S_2 \rangle \rightarrow \varepsilon$.



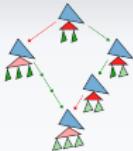
От \mathcal{RE} к НКА: конструкция Глушкова

Теорема

Если $E \in \mathcal{RE}$, то существует праволинейная регулярная грамматика G такая, что $\mathcal{L}(G) = \mathcal{L}(E)$.

Будем строить сразу же НКА, распознающий то же слово, что и E . Для этого определим следующие множества:

- $\text{First}(E)$ — множество символов, с которых может начинаться слово, распознаваемое E .
- $\text{Last}(E)$ — множество символов, которыми может заканчиваться слово, распознаваемое E .
- $\text{Next}(E)$ — множество пар символов, которые могут идти в словах, распознаваемых E , друг за другом.



От \mathcal{RE} к НКА: конструкция Глушкова

Теорема

Если $E \in \mathcal{RE}$, то существует праволинейная регулярная грамматика G такая, что $\mathcal{L}(G) = \mathcal{L}(E)$.

В E пронумеруем все символы из Σ разными номерами. Для полученного E' построим $\text{First}(E')$, $\text{Last}(E')$, $\text{Next}(E')$.

- Введём состояния, соответствующие буквам E' (нумерованным), а также входное состояние I .
- Если $\tau \in \text{First}(E')$, тогда порождаем переход из I в τ (по символу τ).
- Если $\tau_1\tau_2 \in \text{Next}(E')$, тогда порождаем переход из τ_1 в τ_2 по символу τ_2 .
- Если $\tau \in \text{Last}(E')$, тогда объявляем τ финальным.
- Стираем номера у символов на переходах. НКА, распознающий E , построен.

Построим НКА, распознающий $(a \mid (ab))^*b^+$.

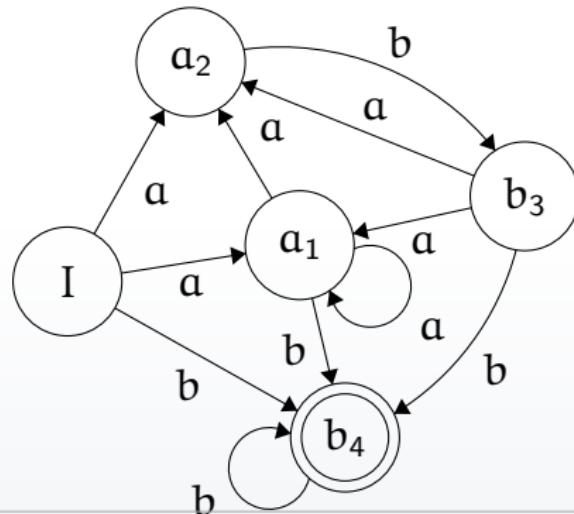
- Линеаризуем: $E' = (a_1 \mid (a_2 b_3))^* b_4^+$.
- Порождаем множества First, Last, Next:

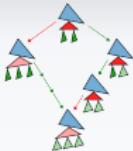
$$\text{First}(E') = \{a_1, a_2, b_4\}$$

$$\text{Last}(E') = \{b_4\}$$

$$\text{Next}(E') = \{a_1 a_1, a_1 a_2, a_2 b_3, b_3 a_1, b_3 a_2, a_1 b_4, b_3 b_4, b_4 b_4\}$$

- Строим конечный автомат:





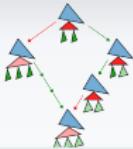
Производные \mathcal{RE}

Множество $a^{-1}U = \{w \mid aw \in U\}$ называется производным Бзозовски множества U относительно a . Если $\varepsilon \in a^{-1}U$, тогда a распознаётся выражением U .

Λ_E положим равным $\{\varepsilon\}$, если $\varepsilon \in E$, и пустым множеством иначе.

- $a^{-1}\varepsilon = \emptyset, a^{-1}\emptyset = \emptyset;$
- $a^{-1}a = \{\varepsilon\}, a^{-1}b = \emptyset;$
- $a^{-1}(\Phi \mid \Psi) = a^{-1}(\Phi) \cup a^{-1}(\Psi);$
- $a^{-1}(\Phi \Psi) = a^{-1}(\Phi)\Psi \cup \Lambda_\Phi a^{-1}(\Psi);$
- $a^{-1}(\Phi^*) = a^{-1}(\Phi)\Phi^*.$

С помощью последовательного взятия производных можно свести задачу $w \in \mathcal{L}(R)$ к задаче $\varepsilon \in w^{-1}R$. На этом построен ещё один способ преобразования \mathcal{RE} к автомату.



Пример преобразования

Рассмотрим всё то же выражение $(a | (ab))^*b^+$. Построим по нему автомат с помощью производных Брезовски.

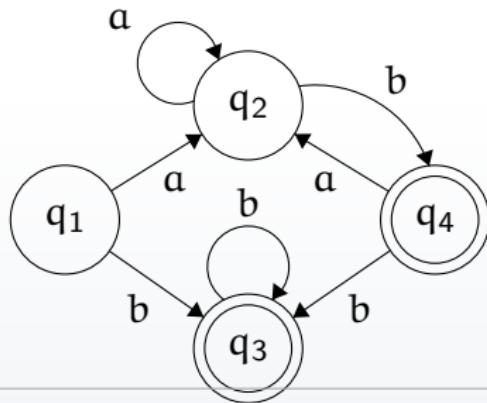
- $a^{-1}(a | (ab))^*b^+ = (a^{-1}(a | (ab))^*)b^+ \cup (a^{-1}b^+)$, но второе очевидно пусто, поэтому
 $a^{-1}(a | (ab))^*b^+ = (\varepsilon | b)(a | (ab))^*b^+$;
- $b^{-1}(a | (ab))^*b^+ = (b^{-1}(a | (ab))^*)b^+ \cup (b^{-1}b^+)$, и здесь как раз пусто первое, поэтому производная равна b^* .
- $a^{-1}b^* = \emptyset$; $b^{-1}b^* = b^*$.
- $a^{-1}((\varepsilon | b)(a | (ab))^*b^+)$ вынуждает первую альтернативу в $(\varepsilon | b)$ и порождает само себя.
- $b^{-1}((\varepsilon | b)(a | (ab))^*b^+)$ порождает $(a | (ab))^*b^+ \mid b^*$.
- $a^{-1}((a | (ab))^*b^+ \mid b^*)$ порождает $(\varepsilon | b)(a | (ab))^*b^+$,
 $b^{-1}((a | (ab))^*b^+ \mid b^*)$ порождает b^* .
- Переходы замкнулись. Осталось собрать производные в состояния автомата.

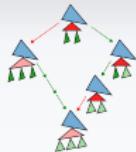


Пример преобразования

Рассмотрим всё то же выражение $(a \mid (ab))^*b^+$. Построим по нему автомат с помощью производных Брезовски.

Состояние	Производная
q_1	$(a \mid (ab))^*b^+$
q_2	$(\varepsilon \mid b) (a \mid (ab))^*b^+$
q_3	b^*
q_4	$(a \mid (ab))^*b^+ \mid b^*$





Неподвижная точка \mathcal{RE}

Неподвижная точка функции $f(x)$ — такое x , что $f(x) = x$.

Лемма Ардена

Пусть $X = (AX) \mid B$, где X — неизвестное \mathcal{RE} , а A, B — известные, причём $\varepsilon \notin \mathcal{L}(A)$. Тогда $X = (A)^*B$.

Рассмотрим систему уравнений:

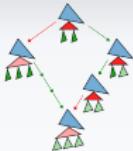
$$X_1 = (A_{11}X_1) \mid (A_{12}X_2) \mid \dots \mid B_1$$

$$X_2 = (A_{21}X_1) \mid (A_{22}X_2) \mid \dots \mid B_2$$

...

$$X_n = (A_{n1}X_1) \mid (A_{n2}X_2) \mid \dots \mid B_n$$

Положим $\varepsilon \notin A_{ij}$. Будем последовательно выражать X_1 через X_2, \dots, X_n , X_2 через $X_3 \dots X_n$ и т.д. Получим регулярное выражение для X_n .



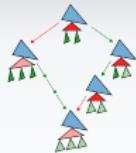
От грамматики и НКА к \mathcal{RE}

Теорема Клини

По каждому НКА можно построить \mathcal{RE} , распознающую тот же язык. Верно и обратное.

Здесь считаем, что в НКА нет ε -переходов.

- Объявляем каждый нетерминал (или состояние НКА) переменной и строим для него уравнение:
 - По правилу $A \rightarrow aB$ (или для стрелки из A в B) добавляем альтернативу aB ;
 - По правилу $A \rightarrow b$ (или для стрелки в финальное состояние) добавляем альтернативу без переменных.
 - Правило $S \rightarrow \varepsilon$ обрабатываем отдельно, не внося в уравнение: добавляем в язык альтернативу $(\mathcal{RE} \mid \varepsilon)$.
- Решаем систему относительно S .



От грамматики к \mathcal{RE}

Пример

Построим \mathcal{RE} по грамматике:

$$S \rightarrow aT \quad S \rightarrow abS$$

$$T \rightarrow aT \quad T \rightarrow bT \quad T \rightarrow b$$

Строим по правилам грамматики систему: $S = (abS) | (aT)$

$$T = ((a | b)T) | b$$

Решаем второе уравнение:

$$T = (a | b)^*b$$

Подставляем в первое:

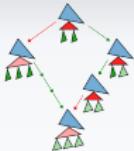
$$S = (abS) | (a(a | b)^*b)$$

Получаем ответ:

$$S = (ab)^*a(a | b)^*b$$

Представления регулярных языков. Критерий регулярности

Теория формальных языков
2022 г.



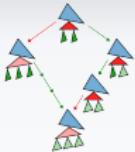
Недетерминированные КА

Определение

Недетерминированный конечный автомат (NFA) — это пятёрка $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, где:

- Q — множество состояний;
- Σ — алфавит терминалов;
- δ — множество правил перехода вида $\langle q_i, (a_i|\varepsilon), M_i \rangle$, где $q_i \in Q, a_i \in \Sigma, M_i \in 2^Q$;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество конечных состояний.

Сокращаем: $\langle q_1, a, q_2 \rangle \in \delta \Leftrightarrow \langle q_1, a, M \rangle \in \delta \ \& \ q_2 \in M$.

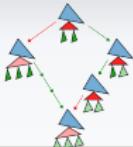


Недетерминированные КА

Определение

Недетерминированный конечный автомат (NFA) — это пятёрка $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$.

- $q \xrightarrow{\varepsilon} q' \Leftrightarrow (q = q') \vee \exists p_1, \dots, p_k (\langle q, \varepsilon, p_1 \rangle \in \delta \text{ & } \langle p_k, \varepsilon, q' \rangle \in \delta \text{ & } \forall i, 1 \leq i < k \langle p_i, \varepsilon, p_{i+1} \rangle \in \delta)$.
- $q \xrightarrow{a} q' \Leftrightarrow \exists p, p' (q \xrightarrow{\varepsilon} p \text{ & } \langle p, a, p' \rangle \in \delta \text{ & } p' \xrightarrow{\varepsilon} q')$.
- $q \xrightarrow{a_1 \dots a_k} q' \Leftrightarrow \exists p_1, \dots, p_{k-1} (q \xrightarrow{a_1} p_1 \text{ & } p_{k-1} \xrightarrow{a_k} q' \text{ & } \forall i, 1 \leq i < k-1 (p_i \xrightarrow{a_{i+1}} p_{i+1}))$.



Недетерминированные КА

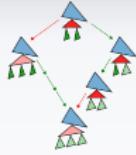
Определение

Недетерминированный конечный автомат (NFA) — это пятёрка $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$.

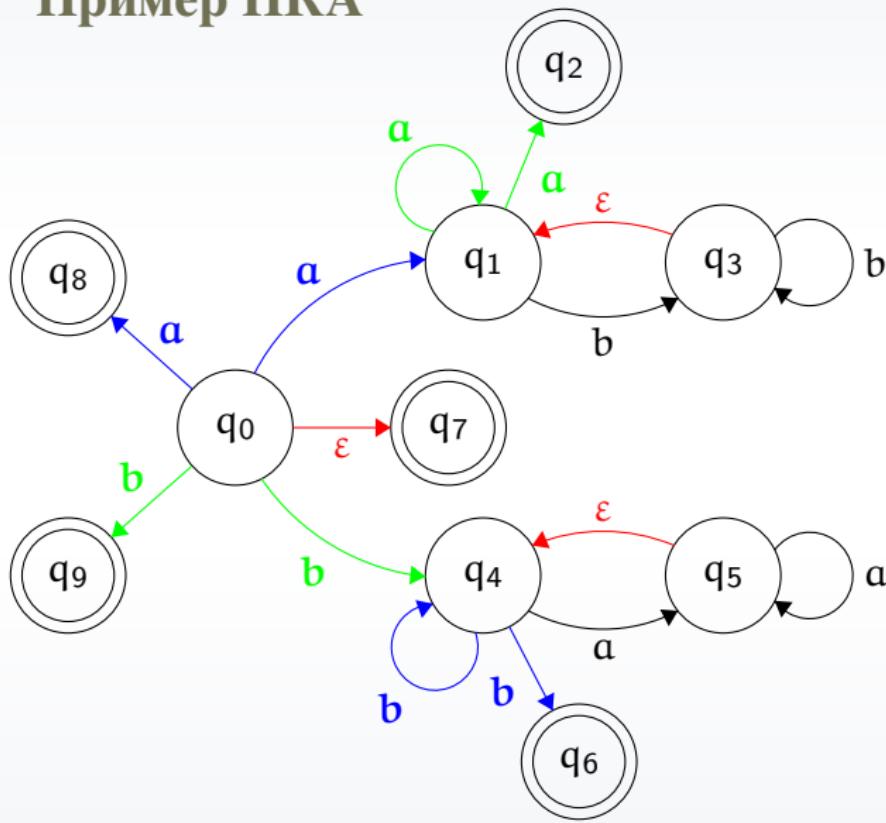
- $q \xrightarrow{\varepsilon} q' \Leftrightarrow (q = q') \vee \exists p_1, \dots, p_k (\langle q, \varepsilon, p_1 \rangle \in \delta \text{ & } \langle p_k, \varepsilon, q' \rangle \in \delta \text{ & } \forall i, 1 \leq i < k \langle p_i, \varepsilon, p_{i+1} \rangle \in \delta)$.
- $q \xrightarrow{a} q' \Leftrightarrow \exists p, p' (q \xrightarrow{\varepsilon} p \text{ & } \langle p, a, p' \rangle \in \delta \text{ & } p' \xrightarrow{\varepsilon} q')$.
- $q \xrightarrow{a_1 \dots a_k} q' \Leftrightarrow \exists p_1, \dots, p_{k-1} (q \xrightarrow{a_1} p_1 \text{ & } p_{k-1} \xrightarrow{a_k} q' \text{ & } \forall i, 1 \leq i < k-1 (p_i \xrightarrow{a_{i+1}} p_{i+1}))$.

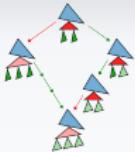
Определение

Язык \mathcal{L} , распознаваемый НКА \mathcal{A} — это множество слов $\{w \mid \exists q \in F (q_0 \xrightarrow{w} q)\}$.



Пример НКА





Детерминированный КА

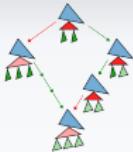
Определение

Детерминированный конечный автомат (DFA) — это пятёрка

$\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, где:

- Q — множество состояний;
- Σ — алфавит терминалов;
- δ — множество правил перехода вида $\langle q_i, a_i, q_j \rangle$, где $q_i, q_j \in Q, a_i \in \Sigma$, причём $\forall q_i, a_i \exists q_j (\langle q_i, a_i, q_j \rangle \in \delta \ \& \ \forall q_k (\langle q_i, a_i, q_k \rangle \in \delta \Rightarrow q_k = q_j))$;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество конечных состояний.

ϵ -переходов нет $\Rightarrow q \xrightarrow{a} q' \Leftrightarrow \langle q, a, q' \rangle \in \delta$.



Детерминированный КА

Определение

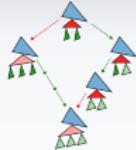
Детерминированный конечный автомат (DFA) — это пятёрка

$\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, где:

- Q — множество состояний;
- Σ — алфавит терминалов;
- δ — множество правил перехода вида $\langle q_i, a_i, q_j \rangle$, где $q_i, q_j \in Q, a_i \in \Sigma$, причём $\forall q_i, a_i \exists q_j (\langle q_i, a_i, q_j \rangle \in \delta \wedge \forall q_k (\langle q_i, a_i, q_k \rangle \in \delta \Rightarrow q_k = q_j))$;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество конечных состояний.

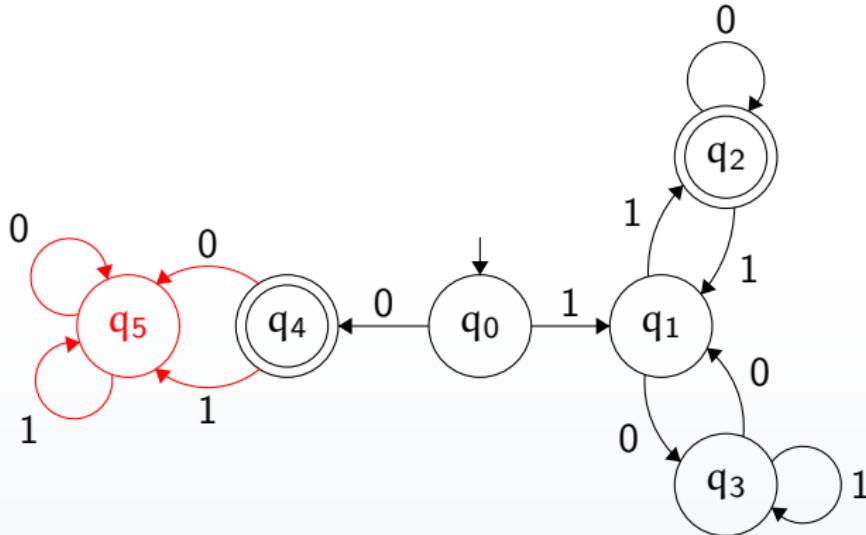
ϵ -переходов нет $\Rightarrow q \xrightarrow{a} q' \Leftrightarrow \langle q, a, q' \rangle \in \delta$.

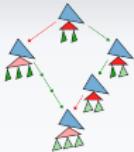
Язык \mathcal{L} , распознаваемый \mathcal{A} — это множество слов $\{w \mid \exists q \in F (q_0 \xrightarrow{w} q)\}$.



Sink/trap state (состояние–ловушка)

«Ловушка» — не конечное состояние с переходами лишь в себя. Нужны для корректного задания DFA, но иногда по умолчанию не описываются.





Детерминизация NFA

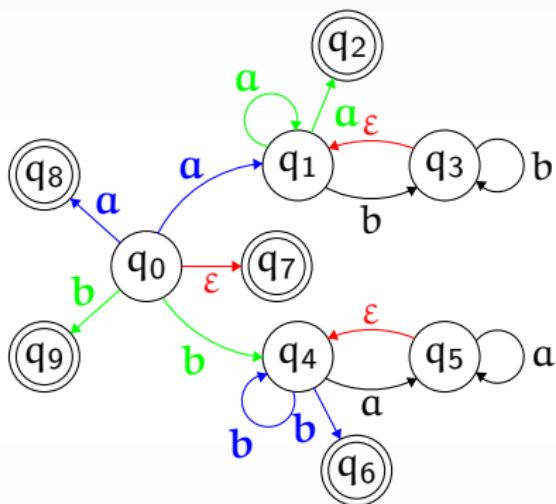
От \mathcal{A} к $D(\mathcal{A})$

Состояния DFA $D(\mathcal{A})$ — это состояния $m_i \in 2^Q$, где Q — состояния NFA \mathcal{A} .

- $m_0 = \{q_i \mid q_0 \xrightarrow{\epsilon} q_i\};$
- $m_i \in F_D \Leftrightarrow \exists q_i, q_j \{ q_i \in m_i \ \& \ q_j \in F(\mathcal{A}) \ \& \ q_i \xrightarrow{\epsilon} q_j \};$
- $\langle m, a, m' \rangle \in \delta_D \Leftrightarrow m' = \{q_i \mid \exists q_j \in m (q_j \xrightarrow{a} q_i)\}.$



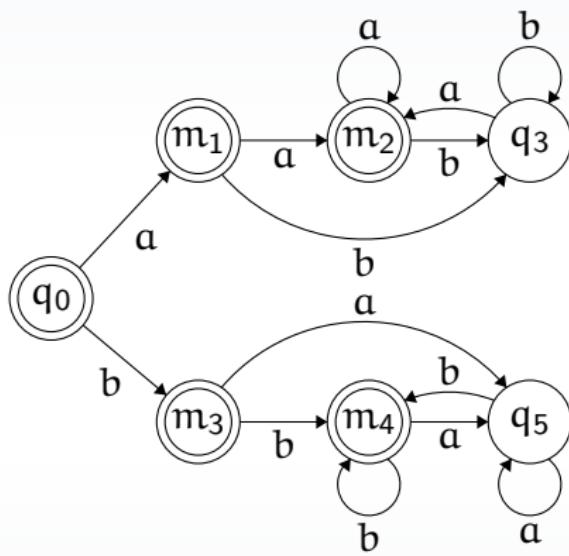
Пример детерминизации



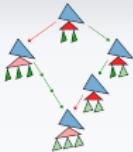
- $\{q_0\} \xrightarrow{a} \{q_1, q_8\},$
 $\{q_0\} \xrightarrow{b} \{q_4, q_9\};$
- $\{q_1, q_8\} \xrightarrow{a} \{q_1, q_2\},$
 $\{q_1, q_8\} \xrightarrow{b} \{q_3\}; \{q_1, q_8\} \sim m_1.$
- $\{q_1, q_2\} \xrightarrow{a} \{q_1, q_2\},$
 $\{q_1, q_2\} \xrightarrow{b} \{q_3\}; \{q_1, q_2\} \sim m_2.$
- $\{q_3\} \xrightarrow{a} \{q_1, q_2\}, \{q_3\} \xrightarrow{b} \{q_3\};$
- $\{q_4, q_9\} \xrightarrow{b} \{q_4, q_6\},$
 $\{q_4, q_9\} \xrightarrow{a} \{q_5\}; \{q_4, q_9\} \sim m_3;$
- $\{q_4, q_6\} \xrightarrow{b} \{q_4, q_6\},$
 $\{q_4, q_6\} \xrightarrow{a} \{q_5\}; \{q_4, q_6\} \sim m_4.$
- $\{q_5\} \xrightarrow{b} \{q_4, q_6\}, \{q_5\} \xrightarrow{a} \{q_5\}.$



Пример детерминизации



- $\{q_0\} \xrightarrow{a} \{q_1, q_8\},$
 $\{q_0\} \xrightarrow{b} \{q_4, q_9\};$
- $\{q_1, q_8\} \xrightarrow{a} \{q_1, q_2\},$
 $\{q_1, q_8\} \xrightarrow{b} \{q_3\}; \{q_1, q_8\} \sim m_1.$
- $\{q_1, q_2\} \xrightarrow{a} \{q_1, q_2\},$
 $\{q_1, q_2\} \xrightarrow{b} \{q_3\}; \{q_1, q_2\} \sim m_2.$
- $\{q_3\} \xrightarrow{a} \{q_1, q_2\}, \{q_3\} \xrightarrow{b} \{q_3\};$
- $\{q_4, q_9\} \xrightarrow{b} \{q_4, q_6\},$
 $\{q_4, q_9\} \xrightarrow{a} \{q_5\}; \{q_4, q_9\} \sim m_3;$
- $\{q_4, q_6\} \xrightarrow{b} \{q_4, q_6\},$
 $\{q_4, q_6\} \xrightarrow{a} \{q_5\}; \{q_4, q_6\} \sim m_4.$
- $\{q_5\} \xrightarrow{b} \{q_4, q_6\}, \{q_5\} \xrightarrow{a} \{q_5\}.$



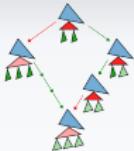
Замыкания регулярных языков

Гомоморфизм над свободной полугруппой (множеством слов) полностью определяется значениями на буквах, поскольку по определению $h(a_1 \circ a_2 \circ \dots \circ a_n) = h(a_1) \circ h(a_2) \circ \dots \circ h(a_n)$. Здесь \circ — конкатенация.

Утверждение

Пусть \mathcal{L} — регулярный язык над Σ . Тогда регулярны:

- язык $\Sigma^* \setminus \mathcal{L}$;
- для любого гомоморфизма h язык $\{h(w) \mid w \in \mathcal{L}\}$;
- для любого гомоморфизма h язык $\{w \mid h(w) \in \mathcal{L}\}$.



Замыкания регулярных языков

Утверждение

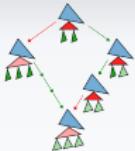
Пусть \mathcal{L} — регулярный язык над Σ . Тогда регулярны:

- язык $\Sigma^* \setminus \mathcal{L}$;
- для любого гомоморфизма h язык $\{h(w) \mid w \in \mathcal{L}\}$;
- для любого гомоморфизма h язык $\{w \mid h(w) \in \mathcal{L}\}$.

Рассмотрим DFA $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, распознающий \mathcal{L} .

Построим $\mathcal{A}' = \langle Q, \Sigma, q_0, Q \setminus F, \delta \rangle$. Тогда

$$w \notin \mathcal{L} \Leftrightarrow w \in \mathcal{L}(\mathcal{A}').$$



Замыкания регулярных языков

Утверждение

Пусть \mathcal{L} — регулярный язык над Σ . Тогда регулярны:

- язык $\Sigma^* \setminus \mathcal{L}$;
- для любого гомоморфизма h язык $\{h(w) \mid w \in \mathcal{L}\}$;
- для любого гомоморфизма h язык $\{w \mid h(w) \in \mathcal{L}\}$.

Рассмотрим регулярное выражение R такое, что $\mathcal{L}(R) = \mathcal{L}$.
Заменим в нём все $a_i \in \Sigma$ на $h(a_i)$. Полученное таким образом выражение R' также регулярно, причём $\mathcal{L}(R') = h(\mathcal{L})$.



Замыкания регулярных языков

Утверждение

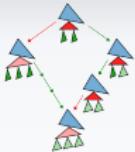
Пусть \mathcal{L} — регулярный язык над Σ . Тогда регулярны:

- язык $\Sigma^* \setminus \mathcal{L}$;
- для любого гомоморфизма h язык $\{h(w) \mid w \in \mathcal{L}\}$;
- для любого гомоморфизма h язык $\{w \mid h(w) \in \mathcal{L}\}$.

Рассмотрим DFA $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, распознающий \mathcal{L} .

Построим $\mathcal{A}' = \langle Q, \Sigma, q_0, F, \delta' \rangle$ такой, что

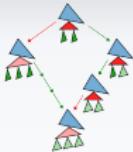
$\langle q_i, a, q_j \rangle \in \delta' \Leftrightarrow q_i \xrightarrow{h(a)} q_j$ в исходном автомате \mathcal{A} .



Примеры

Рассмотрим язык $\mathcal{L}' = \{a^n b^m \mid n \neq m\}$.

Предположим, \mathcal{L}' регулярен. Тогда $a^* b^* \setminus \mathcal{L}' = \{a^n b^n\}$ также регулярен, а мы знаем, что это не так. \perp



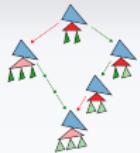
Примеры

Рассмотрим язык $\mathcal{L}' = \{a^n b^m \mid n \neq m\}$.

Предположим, \mathcal{L}' регулярен. Тогда $a^* b^* \setminus \mathcal{L}' = \{a^n b^n\}$ также регулярен, а мы знаем, что это не так. \perp

Рассмотрим язык $\mathcal{L}^f = \{(abaabb)^n b^n\}$.

Попытка доказать его нерегулярность леммой о накачке породит перебор по накачиваемым строкам $(abaabb)^+$, $(abaabb)^*a$, $(abaabb)^*ab$, $(abaabb)^*aba$, $(abaabb)^*abaa$, Рассмотрим гомоморфизм $h(a) = abaabb$, $h(b) = b$. $h^{-1}(\mathcal{L}^f) = \{a^n b^n\}$, который был бы регулярен, если бы L^f был регулярен. \perp

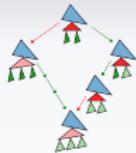


Эквивалентность слов в DFA

Пусть дан DFA \mathcal{A} . Положим

$$w_1 \equiv_{\mathcal{A}} w_2 \Leftrightarrow \exists q_i (q_0 \xrightarrow{w_1} q_i \text{ & } q_0 \xrightarrow{w_2} q_i).$$

Если $w_1 \equiv_{\mathcal{A}} w_2$, тогда $\forall z (w_1 z \in \mathcal{L}(\mathcal{A}) \Leftrightarrow w_2 z \in \mathcal{L}(\mathcal{A}))$.



Эквивалентность слов в DFA

Пусть дан DFA \mathcal{A} . Положим

$$w_1 \equiv_{\mathcal{A}} w_2 \Leftrightarrow \exists q_i (q_0 \xrightarrow{w_1} q_i \text{ & } q_0 \xrightarrow{w_2} q_i).$$

Если $w_1 \equiv_{\mathcal{A}} w_2$, тогда $\forall z (w_1 z \in \mathcal{L}(\mathcal{A}) \Leftrightarrow w_2 z \in \mathcal{L}(\mathcal{A}))$.

Рассмотрим более общее отношение. Положим

$w_1 \equiv_{\mathcal{L}} w_2 \Leftrightarrow \forall z (w_1 z \in \mathcal{L} \Leftrightarrow w_2 z \in \mathcal{L})$. Это отношение разбивает \mathcal{L} на классы эквивалентности.

Теорема Майхилла-Нероуда

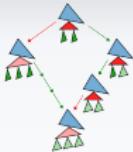
Язык \mathcal{L} регулярен тогда и только тогда, когда множество его классов эквивалентности по $\equiv_{\mathcal{L}}$ конечно.



Критерий регулярности языка

Теорема Майхилла-Нероуда

Язык \mathcal{L} регулярен тогда и только тогда, когда множество классов эквивалентности по $\equiv_{\mathcal{L}}$ конечно.

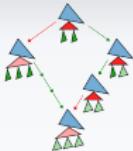


Критерий регулярности языка

Теорема Майхилла-Нероуда

Язык \mathcal{L} регулярен тогда и только тогда, когда множество классов эквивалентности по $\equiv_{\mathcal{L}}$ конечно.

\Rightarrow : Пусть \mathcal{L} регулярен. Тогда он порождается некоторым DFA \mathcal{A} с конечным числом состояний N . Значит, множество $\{q_i \mid q_0 \xrightarrow{w} q_i\}$ конечно, а для каждого двух w_1, w_2 таких, что $q_0 \xrightarrow{w_1} q_i$ и $q_0 \xrightarrow{w_2} q_i$, выполняется $w_1 \equiv_{\mathcal{L}} w_2$.



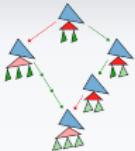
Критерий регулярности языка

Теорема Майхилла-Нероуда

Язык \mathcal{L} регулярен тогда и только тогда, когда множество классов эквивалентности по $\equiv_{\mathcal{L}}$ конечно.

\Leftarrow : Пусть все слова в Σ^* принадлежат N классам эквивалентности A_1, \dots, A_n по $\equiv_{\mathcal{L}}$. Построим по ним DFA \mathcal{A} , распознающий \mathcal{L} . Классы A_i объявим состояниями.

- Начальным состоянием объявим класс эквивалентности A_0 такой, что $\varepsilon \in A_0$.
- Конечными объявим такие A_j , что $\forall w \in A_j (w \in \mathcal{L})$.
- Если $w \in A_i, w a_k \in A_j$, тогда добавляем в δ правило $\langle A_i, a_k, A_j \rangle$. $\forall w_1, w_2 \in A_i, w_1 a_k$ и $w_2 a_k$ всегда принадлежат одному и тому же A_j .



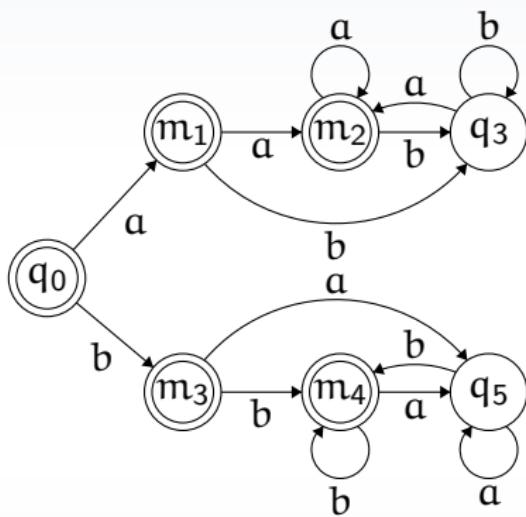
Минимизация DFA

- ❶ Построим таблицу всех двухэлементных множеств $\{q_i, q_j\}$, $q_i, q_j \in Q$.
- ❷ Пометим все множества $\{q_i, q_j\}$ такие, что одно из q_i, q_j из F , а второе нет.
- ❸ Пометим все множества $\{q_i, q_j\}$ такие, что $\exists a (q_i \xrightarrow{a} q'_1 \& q_j \xrightarrow{a} q'_2 \& \{q'_1, q'_2\} — \text{помеченная пара})$.
- ❹ Продолжаем шаг 3, пока не будет появляться новых помеченных пар.

Пары, оставшиеся непомеченными, можно объединить.



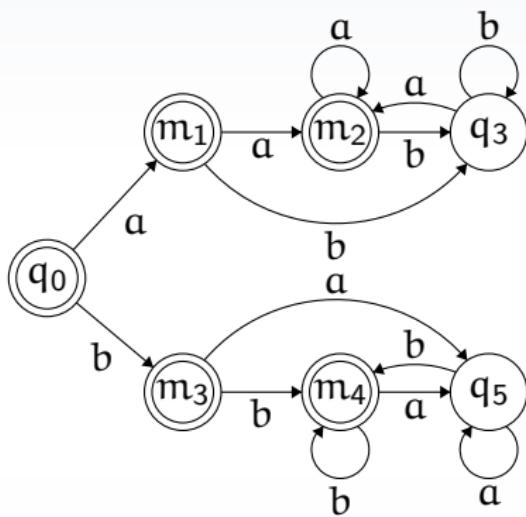
Пример минимизации



m_1	
m_2	
q_3	
m_3	
m_4	
q_5	
	q_0
	m_1
	m_2
	q_3
	m_3
	m_4
	q_5



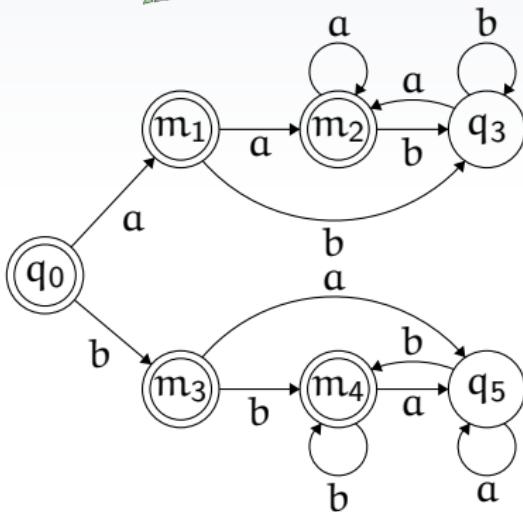
Пример минимизации



m_1						
m_2						
q_3	✓	✓	✓			
m_3				✓		
m_4				✓		
q_5	✓	✓	✓		✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4



Пример минимизации



$$q_0 \xrightarrow{a} m_1, m_1 \xrightarrow{a} m_2$$
$$q_0 \xrightarrow{a} m_1, m_2 \xrightarrow{a} m_2$$

$$q_0 \xrightarrow{b} m_3, m_1 \xrightarrow{b} q_3$$
$$q_0 \xrightarrow{b} m_3, m_2 \xrightarrow{b} q_3$$

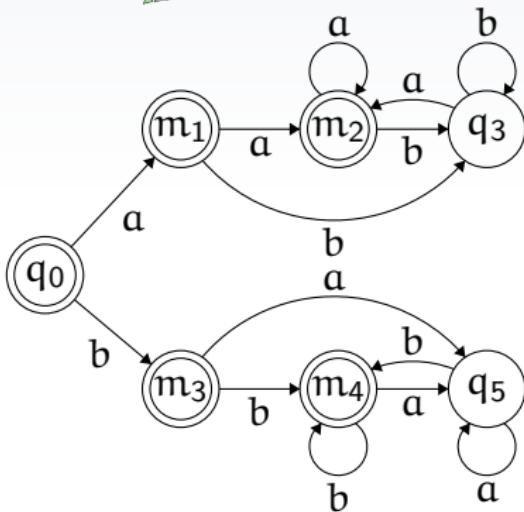
$$\{m_1, m_2\} \xrightarrow{a} m_2$$

$$\{m_1, m_2\} \xrightarrow{b} q_3$$

m_1						
m_2						
q_3	✓	✓	✓			
m_3					✓	
m_4					✓	
q_5	✓	✓	✓		✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4



Пример минимизации



$$q_0 \xrightarrow{a} m_1, m_3 \xrightarrow{a} q_5$$

$$m_2 \xrightarrow{a} m_2, m_3 \xrightarrow{a} q_5$$

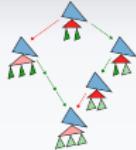
$$q_0 \xrightarrow{a} m_1, m_4 \xrightarrow{a} q_5$$

$$m_1 \xrightarrow{a} m_2, m_4 \xrightarrow{a} q_5$$

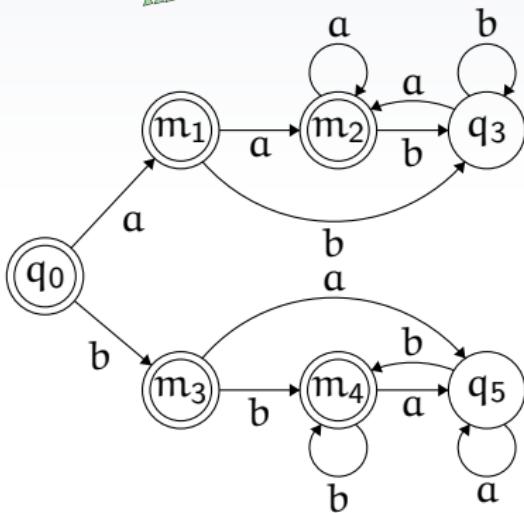
$$m_1 \xrightarrow{a} m_2, m_3 \xrightarrow{a} q_5$$

$$m_2 \xrightarrow{a} m_2, m_4 \xrightarrow{a} q_5$$

m_1	✓					
m_2	✓					
q_3	✓	✓	✓			
m_3					✓	
m_4					✓	
q_5	✓	✓	✓		✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4



Пример минимизации



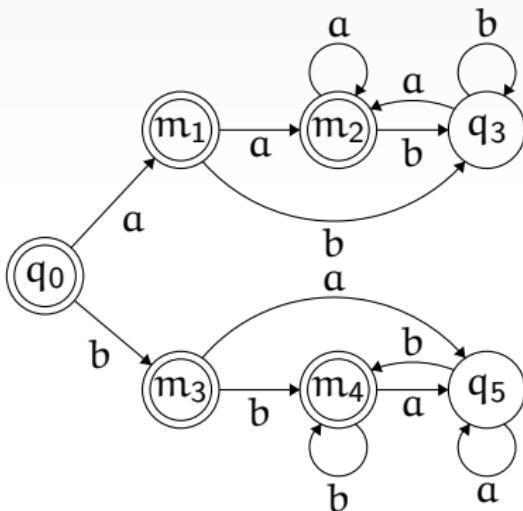
$$\{m_3, m_4\} \xrightarrow{a} q_5 \quad \{m_3, m_4\} \xrightarrow{b} m_4$$

$$q_3 \xrightarrow{a} m_2, q_5 \xrightarrow{a} m_4$$

m_1	✓					
m_2	✓					
q_3	✓	✓	✓			
m_3	✓	✓	✓	✓		
m_4	✓	✓	✓	✓		
q_5	✓	✓	✓		✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4

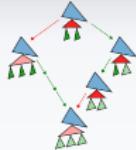


Пример минимизации

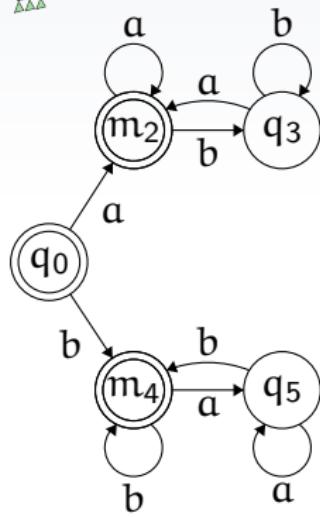


m_1	✓					
m_2	✓					
q_3	✓	✓	✓			
m_3	✓	✓	✓	✓		
m_4	✓	✓	✓	✓		
q_5	✓	✓	✓	✓	✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4

Можно объединить состояния m_1 и m_2 и состояния m_3 и m_4 .



Пример минимизации



m_1	✓					
m_2	✓					
q_3	✓	✓	✓			
m_3	✓	✓	✓	✓		
m_4	✓	✓	✓	✓		
q_5	✓	✓	✓	✓	✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4

Меньше чем пятью состояниями не обойтись. Рассмотрим слова ϵ , a , b , ab , ba . Каждые два из них различаются по $\equiv_{\mathcal{L}}$ при выборе одного из трёх z : ϵ , a или b .

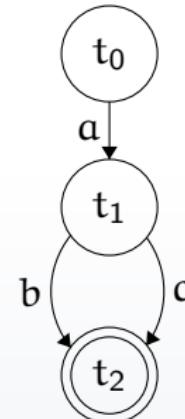
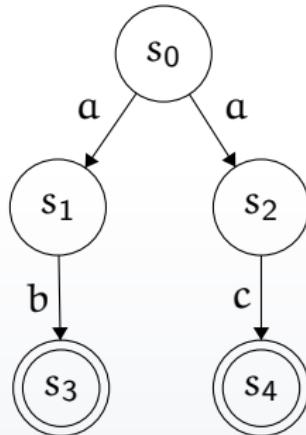


Бисимуляция

Скажем, что состояния s_1, s_2 системы переходов \mathcal{A} находятся в отношении бисимуляции ($s_1 \sim s_2$), если выполняются условия:

- $\forall t_1, a(s_1 \xrightarrow{a} t_1 \Rightarrow \exists t_2(s_2 \xrightarrow{a} t_2 \& t_1 \sim t_2))$;
- $\forall t_2, a(s_2 \xrightarrow{a} t_2 \Rightarrow \exists t_1(s_1 \xrightarrow{a} t_1 \& t_1 \sim t_2))$.

Бисимуляция — более сильное свойство, чем эквивалентность!

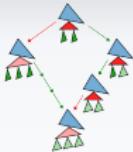




Связь M.–N. и производных

Пусть $w^{-1}U$ — это производная U по w , т.е. $\{v \mid wv \in U\}$.
Тогда выполнено $x \equiv_U y \Leftrightarrow x^{-1}U = y^{-1}U$.

- Количество производных (как языков) регулярного языка конечно.
- Конструкция Брзозовки порождает минимальный DFA.



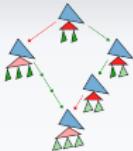
Связь M.–N. и производных

Пусть $w^{-1}U$ — это производная U по w , т.е. $\{v \mid wv \in U\}$.
Тогда выполнено $x \equiv_U y \Leftrightarrow x^{-1}U = y^{-1}U$.

- Количество производных (как языков) регулярного языка конечно.
- Конструкция Брзозовки порождает минимальный DFA.

Но проблема с правилами переписывания (ACI):

- $(w_1 \mid w_2) \mid w_3 = w_1 \mid (w_2 \mid w_3)$
- $w_1 \mid w_2 = w_2 \mid w_1$
- $w \mid w = w$



Применение теоремы М.–Н.

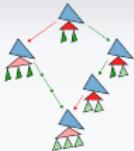
Задача

Дан язык \mathcal{L} . Показать, что он не регулярен, пользуясь теоремой Майхилла–Нероуда.

Стандартный подход

- ① Подобрать бесконечную последовательность префиксов w_1, \dots, w_n, \dots
- ② Подобрать бесконечную последовательность суффиксов z_1, \dots, z_n, \dots , такую, что $w_i ++ z_i \in \mathcal{L}$.
- ③ Доказать, что в таблице конкатенаций все строки различны (значит, $\forall i, j \exists k (w_i z_k \in \mathcal{L} \text{ & } w_j z_k \notin \mathcal{L})$).

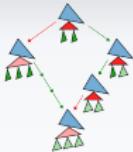
Диагональная конструкция (условие $w_i ++ z_i \in \mathcal{L}$) — одна из многих возможных, обычно она довольно удобна.



Диагональная конструкция

Рассмотрим язык $L = \{a^n b^n\}$. Положим $w_i = a^i$, $z_i = b^i$. Тогда таблица конкатенаций w_i, z_j будет выглядеть следующим образом. Здесь $+$ — это то же, что « $\in \mathcal{L}$ », $-$ читаем как « $\notin \mathcal{L}$ ».

	$z_1 = b$	$z_2 = b^2$	$z_3 = b^3$	\dots	$z_n = b^n$	\dots
$w_1 = a$	+	-	-		-	
$w_2 = a^2$	-	+	-		-	
$w_3 = a^3$	-	-	+		-	
\dots				\dots		
$w^n = a^n$	-	-	-		+	
\dots						



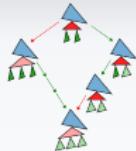
Доказательство минимальности

Так же можно обосновывать минимальность DFA. Рассмотрим минимальный автомат из примера выше. Его язык — слова в $\{a, b\}^*$, начинающиеся и заканчивающиеся одной и той же буквой. Построим таблицу классов эквивалентности по $w_i \in \{\epsilon, a, b, ab, ba\}$.

В этой таблице все строчки различны, значит, выбранные w_i действительно лежат в различных классах эквивалентности, и DFA, распознающий язык \mathcal{L} , не может иметь меньше пяти состояний.

При доказательстве минимальности DFA достаточно подобрать $\lceil \log_2 n \rceil + 1$ различных суффиксов z_i , где n — число состояний автомата.

	ϵ	a	b
ϵ	+	+	+
a	+	+	-
b	+	-	+
ab	-	+	-
ba	-	-	+



О порождении новых алгоритмов

Пусть \mathcal{A} — NFA. Тогда $\text{det}(\text{reverse}(\text{det}(\text{reverse}(\mathcal{A}))))$ — минимальный DFA, эквивалентный \mathcal{A} .

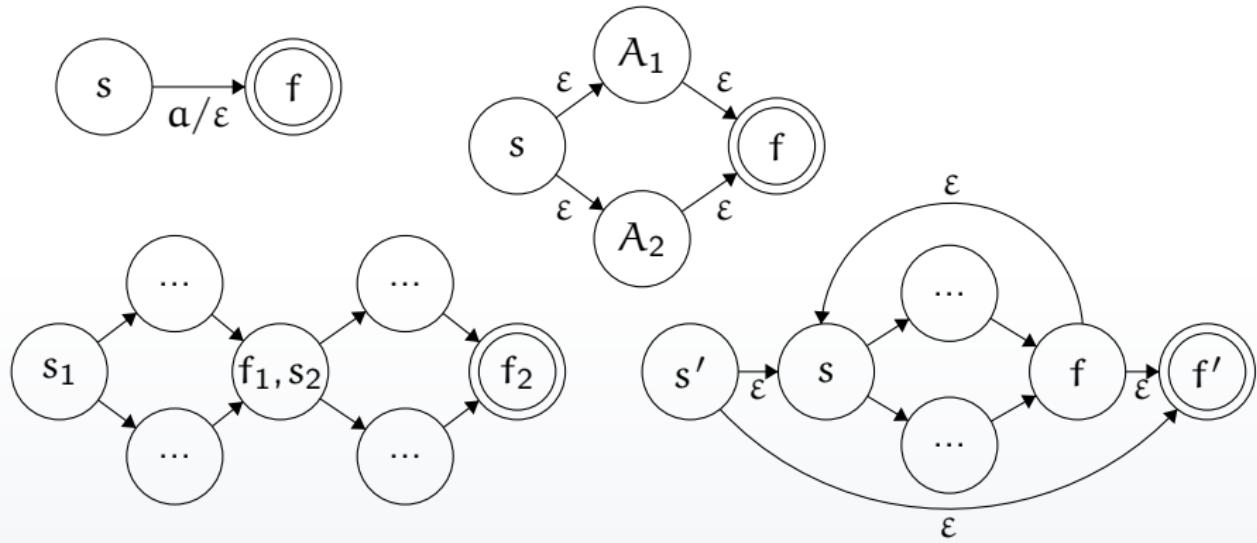
Многие алгоритмы для порождения малых (не минимальных) NFA являются комбинациями нескольких базовых операций.

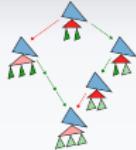
- Обращение автомата
- Детерминизация
- Удаление ε -правил
- Минимизация
- Разметка



Автомат Томпсона

- Единственное начальное состояние
- Единственное конечное состояние
- Не больше двух переходов из каждого состояния





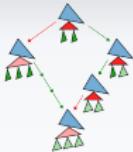
Несколько конструкций

- Автомат Глушкова: `rmeeps(Th(R));`
- Автомат Антимирова:
`rmeeps(deannotate(minimize(rmeeps(annotation_epsilon(Th(R))))));`
- Автомат Илия–Ю:
`deannotate(minimize(rmeeps(annotation(Th(R))))).`

Другие регулярные модели. Синтаксический моноид



Теория формальных языков
2022 г.

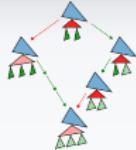


Леволинейные грамматики

Класс грамматик, симметричный праволинейным:

- виды правил — $A_i \rightarrow a_j$, $A_i \rightarrow A_k a_j$ и $S \rightarrow \varepsilon$, 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.

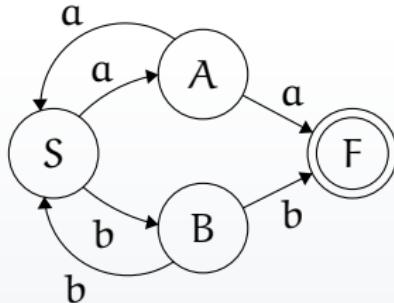


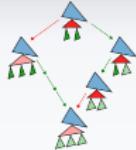
Леволинейные грамматики

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

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

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

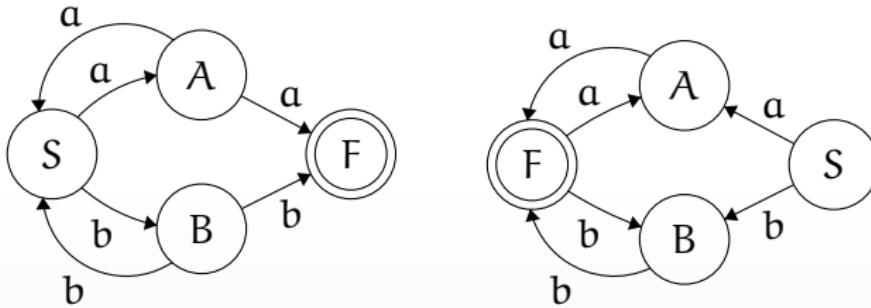


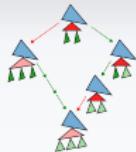


Леволинейные грамматики

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

Теперь обращаем стрелки и меняем начальное и финальное состояния местами.

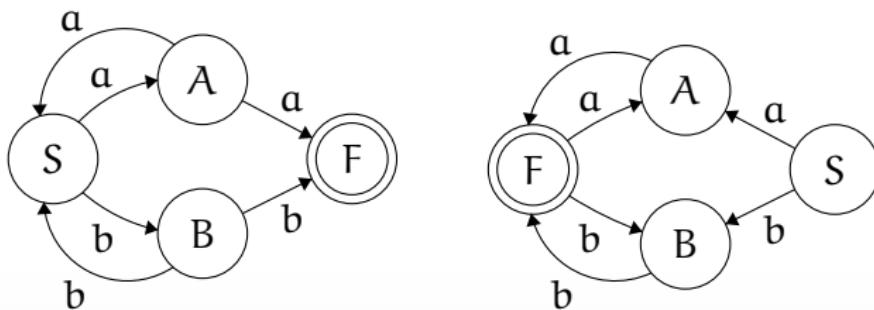




Леволинейные грамматики

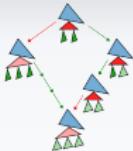
$$S \rightarrow aA \mid bB \quad A \rightarrow aS \mid a \quad B \rightarrow bS \mid b$$

Теперь обращаем стрелки и меняем начальное и финальное состояния местами.



По полученному автомату строим леволинейную грамматику:

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

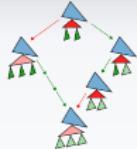


Леволинейные грамматики

Формальный алгоритм приведения к леволинейности

- ❶ Добавляем в грамматику новый стартовый символ S' и для всех правил вида $A_i \rightarrow a_j$ — правила $S' \rightarrow A_i a_j$.
- ❷ Правила вида $A_i \rightarrow a_j A_k$ преобразуем в $A_k \rightarrow A_i a_j$.
- ❸ По правилам вида $S \rightarrow a_j A_k$ дополнительно порождаем правила $A_k \rightarrow a_j$.
- ❹ По правилам вида $S \rightarrow a_j$ и $S \rightarrow \varepsilon$ порождаем правила $S' \rightarrow a_j$ и $S' \rightarrow \varepsilon$ соответственно.

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

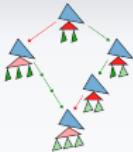


Обращение регулярного языка

Если \mathcal{L} регулярен, то и \mathcal{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 \mathcal{L} регулярен, то и \mathcal{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

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

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

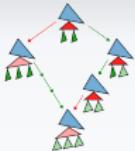
$$u \equiv_{\mathcal{L}} w \Leftrightarrow \forall x (ux \in \mathcal{L} \Leftrightarrow vx \in \mathcal{L})$$



Цена детерминизма

Утверждение

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



Цена детерминизма

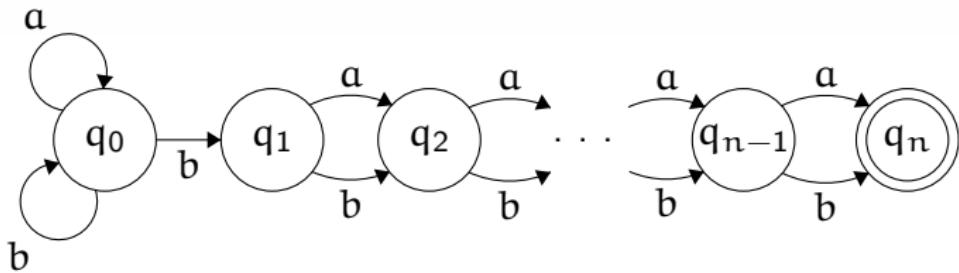
Утверждение

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

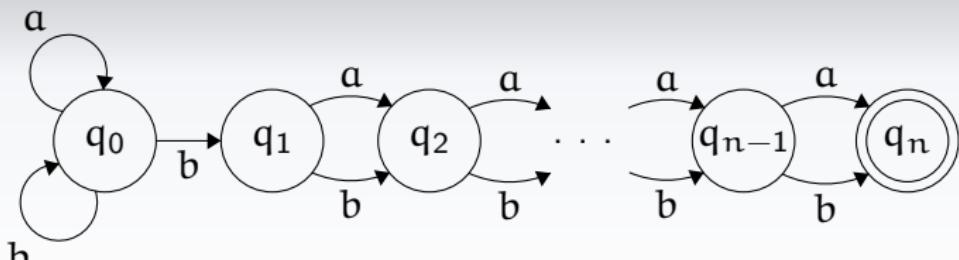
Рассмотрим грамматику G :

$$\begin{array}{lll} S \rightarrow aS & S \rightarrow bS & S \rightarrow bA_1 \\ A_1 \rightarrow aA_2 & A_1 \rightarrow bA_2 & A_2 \rightarrow aA_3 \\ \dots & & A_2 \rightarrow bA_3 \\ A_{n-1} \rightarrow aA_n & A_{n-1} \rightarrow bA_n & A_n \rightarrow a \\ & & A_n \rightarrow b \end{array}$$

Грамматике G соответствует следующий НКА. Её язык — это слова вида $\{a | b\}^* b \{a | b\} \langle n - 1 \rangle$, 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.



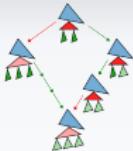
$$L(G) = \{a|b\}^* b \{a|b\} \langle n-1 \rangle.$$

	ε	a	...	a^{n-3}	a^{n-2}	a^{n-1}
a^n	—	—	...	—	—	—
$a^{n-1}b$	—	—	...	—	—	+
$a^{n-2}ba$	—	—	...	—	+	—
$a^{n-2}bb$	—	—	...	—	+	+
$a^{n-3}baa$	—	—	...	+	—	—
$a^{n-3}bab$	—	—	...	+	—	+
$a^{n-3}bba$	—	—	...	+	+	—
$a^{n-3}bbb$	—	—	...	+	+	+
...
$a b^{n-1}$	—	+	...	+	+	+
b^n	+	+	...	+	+	+

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

	ϵ	a	...	a^{n-3}	a^{n-2}	a^{n-1}
a^n	—	—	...	—	—	—
$a^{n-1}b$	—	—	...	—	—	+
$a^{n-2}ba$	—	—	...	—	+	—
$a^{n-2}bb$	—	—	...	—	+	+
$a^{n-3}baa$	—	—	...	+	—	—
$a^{n-3}bab$	—	—	...	+	—	+
$a^{n-3}bba$	—	—	...	+	+	—
$a^{n-3}bbb$	—	—	...	+	+	+
...
$a b^{n-1}$	—	+	...	+	+	+
b^n	+	+	...	+	+	+

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.

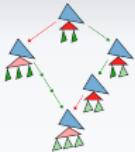


Ещё раз о лемме о накачке

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 $L = \{w_1bw_2 \mid |w_2| = 3\}$. 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 ≥ 5 : достаточно взять первую букву этого слова в качестве y , а x принять пустым.

Теперь пусть длина накачки r меньше 5. Рассмотрим слово $baaa \in L$. Любая накачка только букв a (нулевая и нет) выводит слово из языка, и нулевая накачка под слова, содержащего букву b , также выводит из языка L . Поэтому НКА, распознающий L , не может иметь меньше 5 состояний.



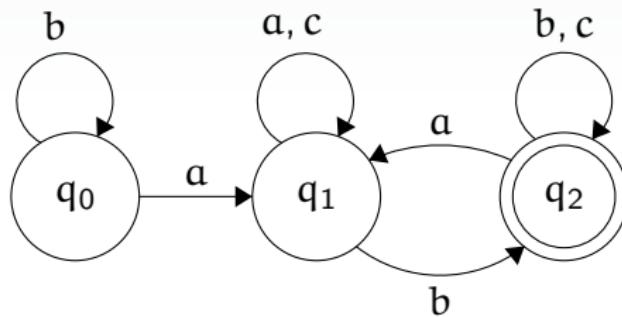
Трансформационный моноид

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

Каждый автомат \mathcal{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} .

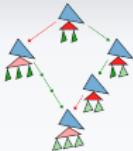


Пример построения Т.М.

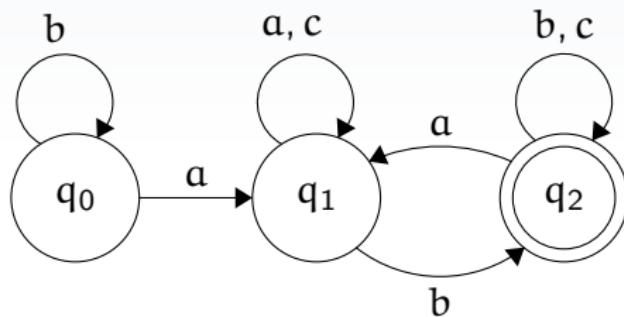


Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке.

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

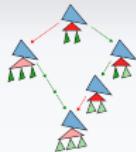


Пример построения Т.М.

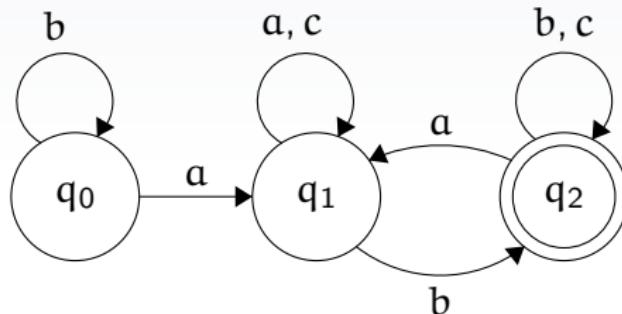


Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке.

$$\begin{array}{lll} a := \{(0, 1), (1, 1), (2, 1)\} & b := \{(0, 0), (1, 2), (2, 2)\} & c := \{(1, 1), (2, 2)\} \\ ab := \{(0, 2), (1, 2), (2, 2)\} & bc := \{(1, 2), (2, 2)\} & ca := \{(1, 1), (2, 1)\} \\ aa \rightarrow a & ac \rightarrow a & ba \rightarrow a \\ bb \rightarrow b & cc \rightarrow c & cb \rightarrow bc \end{array}$$



Пример построения Т.М.



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

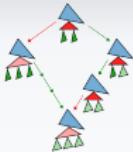
$$a := \{(0, 1), (1, 1), (2, 1)\} \quad b := \{(0, 0), (1, 2), (2, 2)\} \quad c := \{(1, 1), (2, 2)\}$$

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

$$aa \rightarrow a \quad ac \rightarrow a \quad ba \rightarrow a$$

$$bb \rightarrow b \quad cc \rightarrow c \quad cb \rightarrow bc$$

$$abc \rightarrow ab \quad bca \rightarrow ca \quad cab \rightarrow bc$$



Синтаксический моноид

Положим $u \sim_{\mathcal{L}} v \Leftrightarrow \forall x, y (xuy \in \mathcal{L} \Leftrightarrow xvy \in \mathcal{L})$.

Синтаксический моноид $\mathcal{M}(\mathcal{L})$: $\{w \mid w_i = w_j \Leftrightarrow w_i \sim_{\mathcal{L}} w_j\}$.

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

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

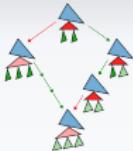


Синхронизирующиеся автоматы

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

Критерий синхронизации

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



Синхронизирующиеся автоматы

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

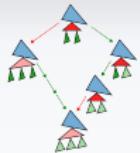
Критерий синхронизации

ДКА \mathcal{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, $Q_1 \subset \{q_1, \dots, q_n\}$.

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.



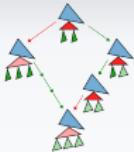
Синхронизирующиеся автоматы

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

Критерий синхронизации

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

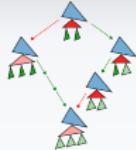
ДКА синхронизируется \Leftrightarrow классы эквивалентности его трансформационного моноида содержат «константу», т.е. класс, переводящий все состояния в одно.



Задача Эшби о привидениях

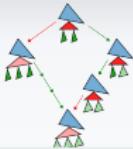
Дорогой друг! Недавно я купил старый дом, в котором обитают два призрака: Певун и Хохотун. Я установил, что их поведение подчиняется определенным законам, и что я могу воздействовать на них, играя на органе или сжигая ладан. В течение каждой минуты каждый из призраков либо шумит, либо молчит. Поведение 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 (звукит 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?



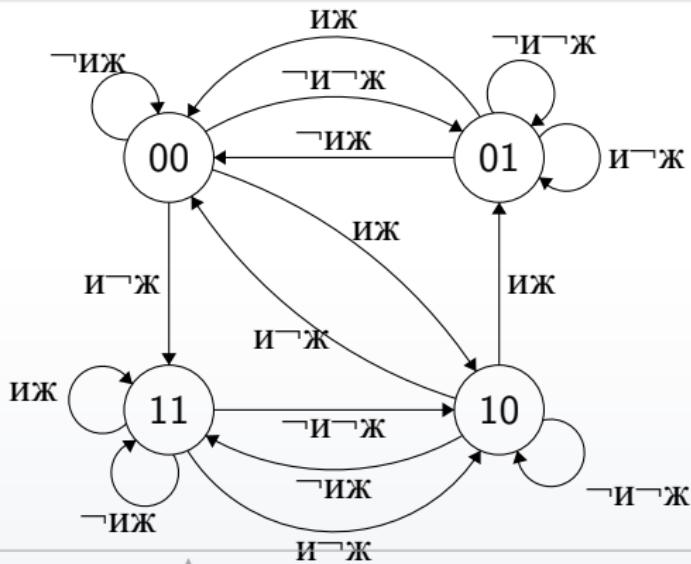
Задача Эшиби о привидениях

- 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.



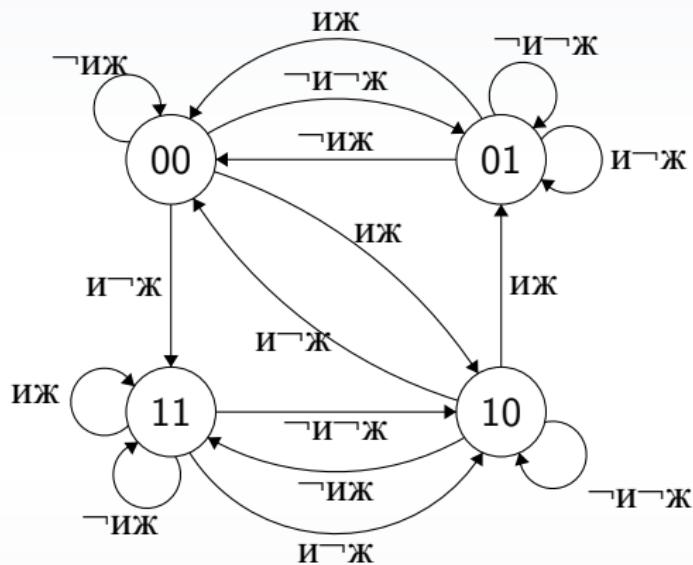
Задача Эшби о привидениях

- 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: $\neg\text{i}\neg\text{j}$, $\text{i}\neg\text{j}$, $\neg\text{иж}$.



Префиксное кодирование

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

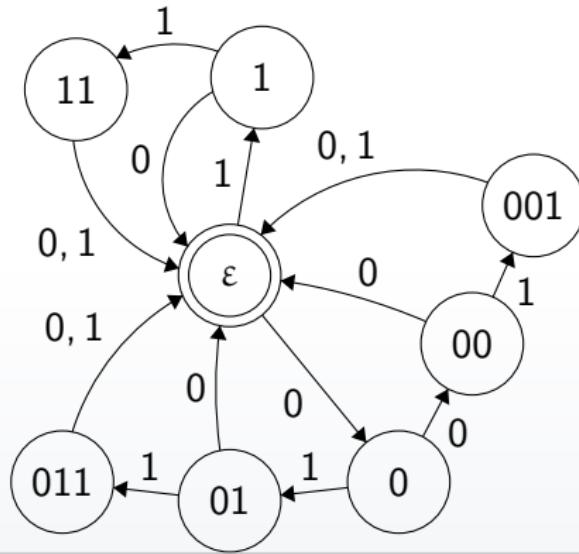
Рассмотрим префиксный код из 9-буквенного алфавита:
 $C = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$.



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):



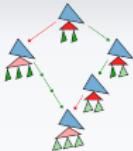


Коды, исправляющие ошибки

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

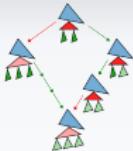
Префиксный код максимален, если к множеству кодирующих слов нельзя добавить ни одно слово без нарушения префикс-свойства (т.е. запрета слов из множества быть префиксами друг друга).

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

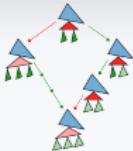
Дана SRS \mathcal{S} с правилами переписывания двух видов:

$$a_i \rightarrow b_1 \dots b_n \quad a_i \rightarrow \epsilon$$

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

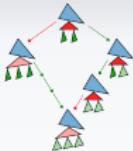
Утверждение

Язык $L(\mathcal{S}, w_0)$ регулярен.

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

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

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



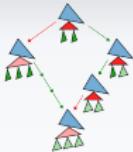
Алфавитные префиксные грамматики

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

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

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

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

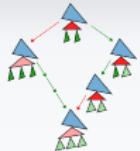


Неалфавитные грамматики

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

Языки префиксных грамматик регулярны.

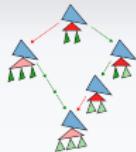
Доказательство использует ту же идею, что в случае АПГ: множество минимальных укорачивающихся комбинаций правил переписывания конечно.



От ДКА к префиксной грамматике

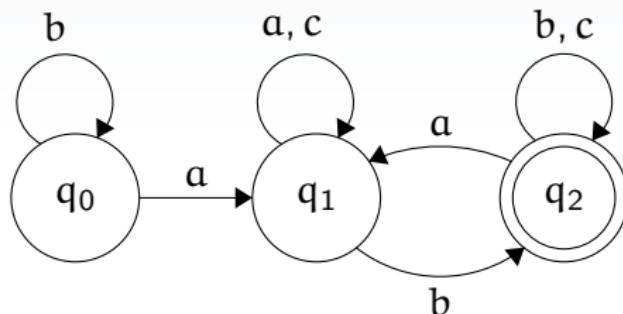
В данном алгоритме рассматривается минимальный ДКА для языка.

- 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: $w_i \rightarrow w_k a_j$.
- 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:

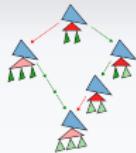
$b \rightarrow bb$ (в q_0 входит лишь одна стрелка)

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

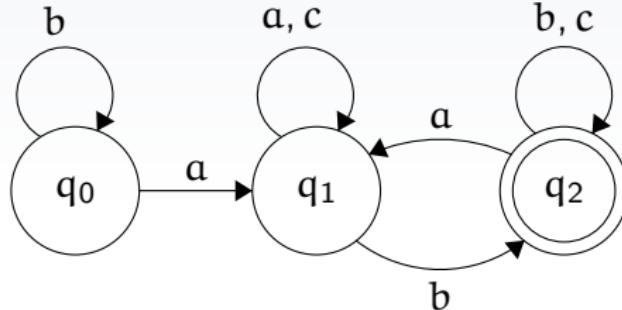
$a \rightarrow ba$ (стрелка из q_0 в q_1)

$ab \rightarrow ab$ (стрелка из q_1 в q_2)

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



От ДКА к префиксной грамматике



Стартовое слово: ab . Правила переписывания:

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

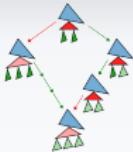
$a \rightarrow aa$ $a \rightarrow ac$ (стрелки из q_1 в себя)

$a \rightarrow ba$ (стрелка из q_0 в q_1)

$ab \rightarrow ab$ (стрелка из q_1 в q_2)

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

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



Поведение стека в СВ-семантике

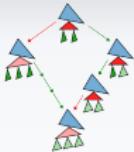
Рассмотрим стек с вершиной \bullet_n :

$$\bullet_n \leftarrow f_{n+1}(\dots), \bullet_{n-1} \leftarrow f_n(\bullet_n \dots) \dots, \bullet_0 \leftarrow f_1(\bullet_1 \dots)$$

Опишем его состояние перечислением имён функций в порядке их вхождения: $f_{n+1} f_n \dots f_1$.

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

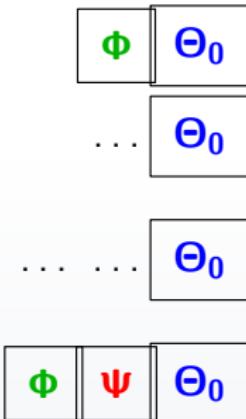
«Подозрительное» поведение — такое, при котором вершина стека повторяется, выбрасывая промежуточные вычисления.



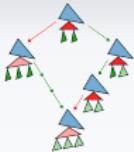
Поиск бесконечных циклов

Отношение Турчина

Пусть на путях 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).

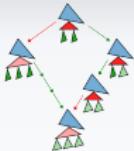


Теорема Турчина

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

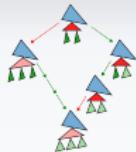
Определение

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 $p_1 \circ p_2 = \text{id}$, 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-pong протокол для двух участников — это конечная последовательность инструкций $[p_1 \dots p_n, [x, y]]$,
 $p_i \in \mathcal{V}_x \cup \mathcal{O}_y$.

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

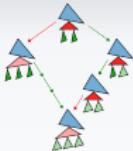


Модель угрозы Долева–Яо

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).

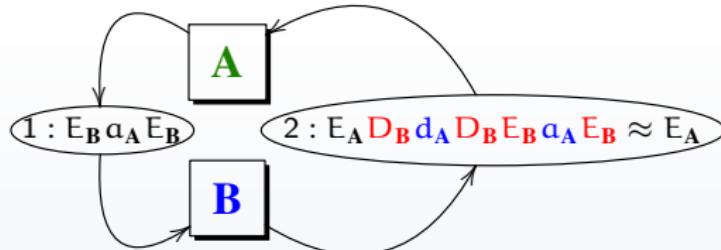
Изначальное сообщение — **M** (обычно засекрченное).

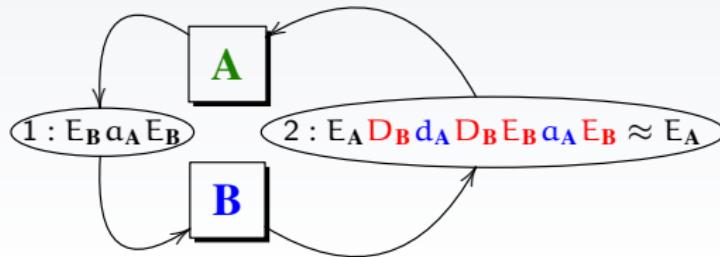
Σ_x — словарь операторов x . E_x — зашифровка открытым ключом x , D_x — расшифровка E_x , a_x — приписывание к сообщению имени x , d_x — удаление префикса сообщения, совпадающего с именем x .

Протокол — набор α_i (слов протокола) и указаний, кто посыпает α_i .

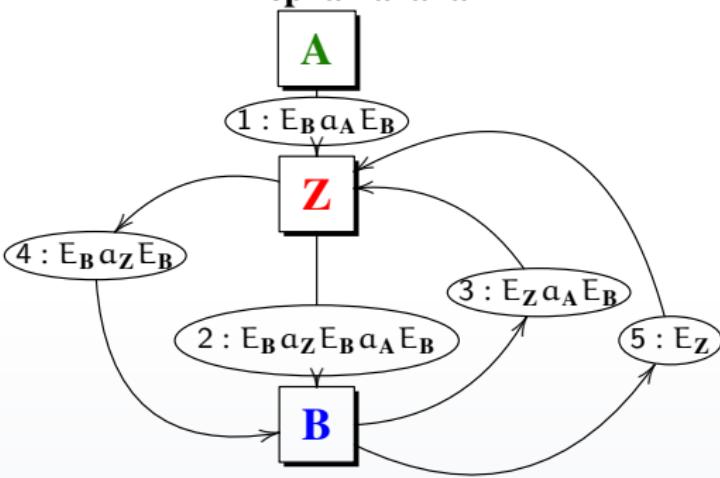
Атака — последовательность подстановок в α_i , порождающая пустое слово (т.е. демаскирующая сообщение **M**).

Пример протокола

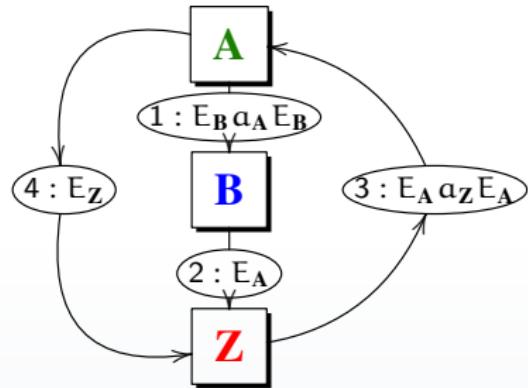


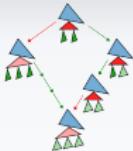


Первая атака



Вторая атака





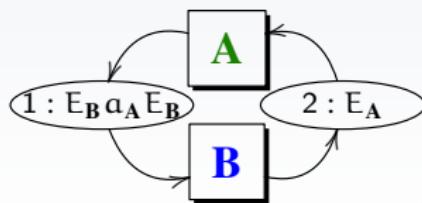
Автоматная модель $\mathcal{A}(P)$

- 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 $\mathcal{O}_A, \mathcal{O}_B, \mathcal{O}_Z$ и \mathcal{P}_Z .

Утверждение

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

Протокол



Автоматная модель

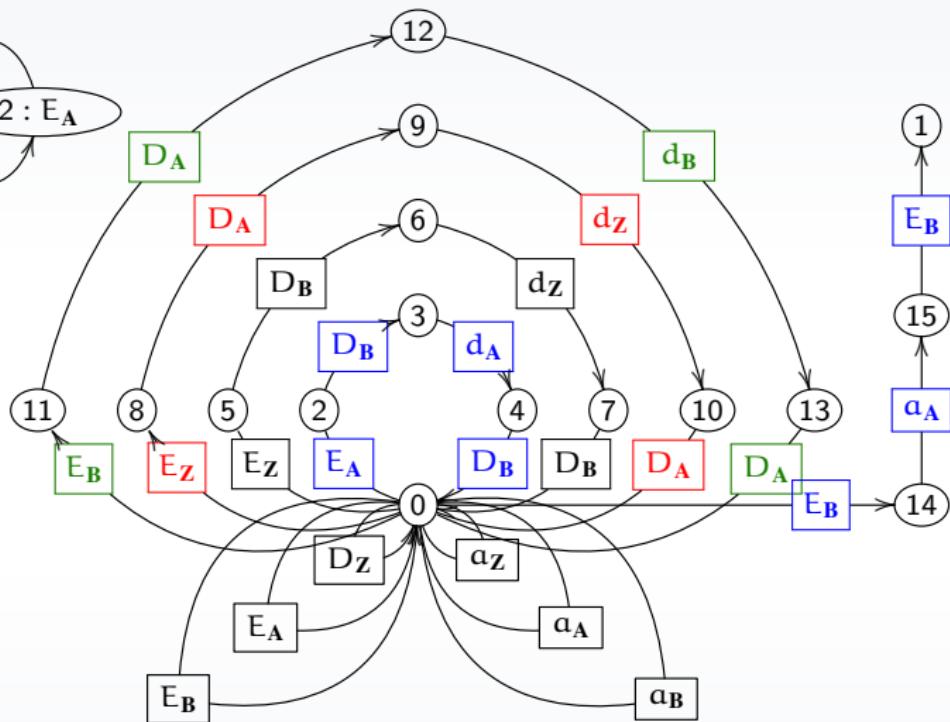
Случаи

P_{Double}[A, B]

$P_{\text{Double}}[B, A]$

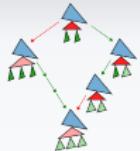
P_{Double}[A, Z]

$P_{\text{Double}}[Z, B]$



Контекстно-свободные грамматики.
Деревья разбора.
Нормальная форма Хомского.
Первая лемма о накачке

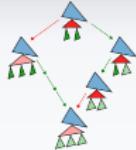
Теория формальных языков
2022 г.



Ограничения регулярных грамматик

- (синтаксический моноид) Слова лишь конечно различимы относительно правил переписывания
- (префиксные грамматики) Доступ лишь к началу (концу) слова

Что будет, если снять эти условия?



Ограничения регулярных грамматик

- (синтаксический моноид) Слова лишь конечно различимы относительно правил переписывания
- (префиксные грамматики) Доступ лишь к началу (концу) слова

Что будет, если снять эти условия?

Структура вывода — дерево, а не последовательность.

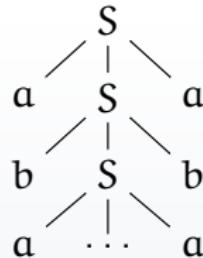
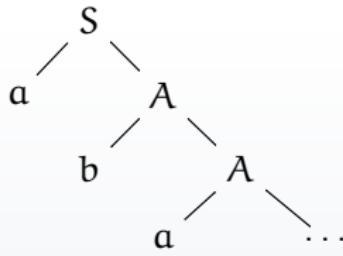


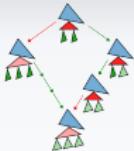
Ограничения регулярных грамматик

- (синтаксический моноид) Слова лишь конечно различимы относительно правил переписывания
- (префиксные грамматики) Доступ лишь к началу (концу) слова

Что будет, если снять эти условия?

Структура вывода — дерево, а не последовательность.



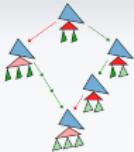


Контекстно-свободные грамматики

Определение

Контекстно-свободная грамматика (CFG) — это грамматика $\langle \Sigma, N, P, S \rangle$, где правила переписывания P имеют вид $A \rightarrow \alpha$, $A \in N$, $\alpha \in (\Sigma \cup N)^*$.

- Нетерминалы переписываются независимо друг от друга (можно понимать их как нульместные функции).
- Вывод в грамматике (разбор слова) не линеен.



Контекстно-свободные грамматики

Определение

Контекстно-свободная грамматика (CFG) — это грамматика $\langle \Sigma, N, P, S \rangle$, где правила переписывания P имеют вид $A \rightarrow \alpha$, $A \in N$, $\alpha \in (\Sigma \cup N)^*$.

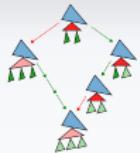
- Нетерминалы переписываются независимо друг от друга (можно понимать их как нульместные функции).
- Вывод в грамматике (разбор слова) не линеен.

Грамматика G_1

$$\begin{array}{l} S \rightarrow SS \\ S \rightarrow (S) \\ S \rightarrow \varepsilon \end{array}$$

Грамматика G_2

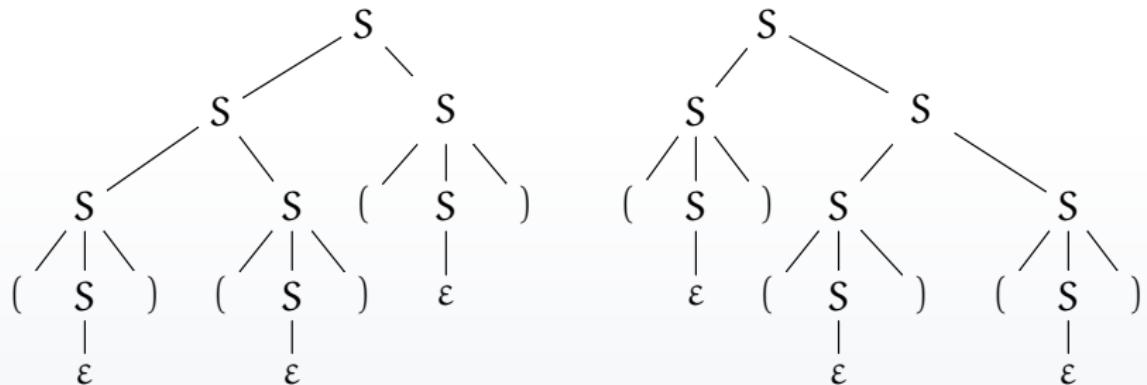
$$\begin{array}{ll} S \rightarrow B & R \rightarrow) \\ B \rightarrow (RB & R \rightarrow (RR \\ B \rightarrow \varepsilon & \end{array}$$

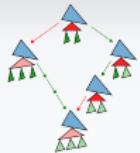


Неоднозначность разбора

Грамматика G_1 для языка Дика

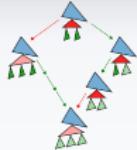
$$\begin{array}{lcl} S & \rightarrow & S\ S \\ S & \rightarrow & (S) \\ S & \rightarrow & \varepsilon \end{array}$$





Левосторонний разбор

Шаг левостороннего разбора с.ф. $\alpha_1 A \alpha_2$, где $\alpha_1 \in \Sigma^*$, $A \in N$, — замена выделенного вхождения A на правую часть $A \rightarrow \beta$. Левосторонний разбор S — разбор, каждый шаг которого левосторонний.

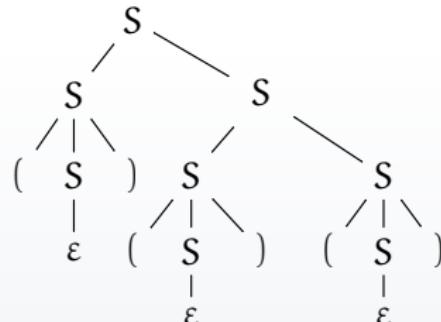
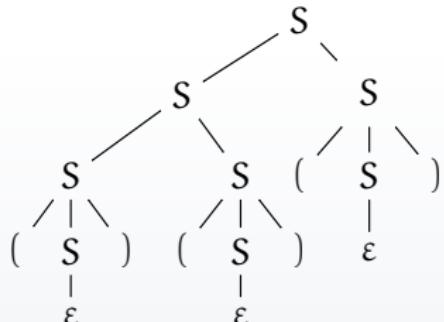


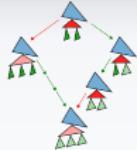
Левосторонний разбор

Шаг левостороннего разбора с.ф. $\alpha_1 A \alpha_2$, где $\alpha_1 \in \Sigma^*$, $A \in N$, — замена выделенного вхождения A на правую часть $A \rightarrow \beta$. Левосторонний разбор S — разбор, каждый шаг которого левосторонний.

Левосторонний разбор не обязательно единственный, см. ниже.

$$S \rightarrow SS \quad S \rightarrow (S) \quad S \rightarrow \epsilon$$



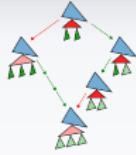


Левосторонний разбор

Шаг левостороннего разбора с.ф. $\alpha_1 A \alpha_2$, где $\alpha_1 \in \Sigma^*$, $A \in N$, — замена выделенного вхождения A на правую часть $A \rightarrow \beta$. Левосторонний разбор S — разбор, каждый шаг которого левосторонний.

Утверждение

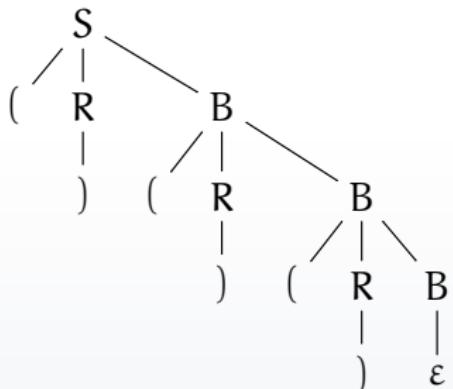
Между деревьями разбора слов $w \in L(G)$ и левосторонними разборами w есть взаимно-однозначное соответствие.



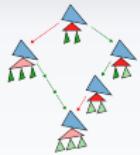
(Не)однозначность грамматик

Грамматика G_2 для языка Дика

$$\begin{array}{ll} S \rightarrow B & R \rightarrow) \\ B \rightarrow (RB & R \rightarrow (RR \\ B \rightarrow \varepsilon & \end{array}$$

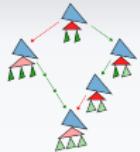


Грамматика G_2 однозначна — для всех $w \in L(G_2)$ существует единственный левосторонний разбор w . Достаточно заглянуть на 1 символ после разобранной позиции.



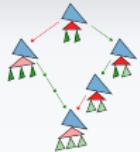
Другие проблемы контекстно-свободного разбора слов

- ε -правила (правила вида $A \rightarrow \varepsilon$);
- « ε -переходы», или цепные правила (правила вида $A \rightarrow B$).



Устранение ε -правил

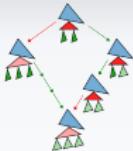
$A \in N$ коллапсирует, если $A \rightarrow \varepsilon \in P$ или $A \rightarrow \alpha \in P$ и все элементы α коллапсируют.



Устранение ε -правил

$A \in N$ коллапсирует, если $A \rightarrow \varepsilon \in P$ или $A \rightarrow \alpha \in P$ и все элементы α коллапсируют.

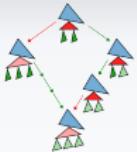
- Объявляем `Nullable` = \emptyset ;
- $\forall A \in N$, если $A \rightarrow \varepsilon$, тогда
`Nullable` = `Nullable` $\cup \{A\}$;
- Пока `Nullable` меняется:
 - для всех $A \in N$, если $A \rightarrow B_1 \dots B_n$, $B_i \in \text{Nullable}$
 $\Rightarrow \text{Nullable} = \text{Nullable} \cup \{A\}$.
- Итоговое множество `Nullable` — множество всех коллапсирующих нетерминалов.



Устранение ε -правил

$A \in N$ коллапсирует, если $A \rightarrow \varepsilon \in P$ или $A \rightarrow \alpha \in P$ и все элементы α коллапсируют.

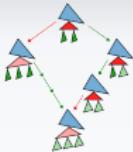
- Если $\varepsilon \in L(G)$, тогда добавляем новый стартовый символ S_0 и правила $S_0 \rightarrow \varepsilon, S_0 \rightarrow S$.
- Стираем все правила $B_i \rightarrow \varepsilon$, кроме $S_0 \rightarrow \varepsilon$.
- Для всех правил $A \rightarrow \alpha_1 B_i \alpha_2$, где $B_i \in \text{Nullable}$, добавляем правила $A \rightarrow \alpha_1 \alpha_2$.



Устранение ε -правил

$A \in N$ коллапсирует, если $A \rightarrow \varepsilon \in P$ или $A \rightarrow \alpha \in P$ и все элементы α коллапсируют.

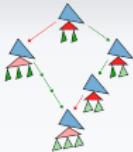
- Если $\varepsilon \in L(G)$, тогда добавляем новый стартовый символ S_0 и правила $S_0 \rightarrow \varepsilon, S_0 \rightarrow S$.
- Стираем все правила $B_i \rightarrow \varepsilon$, кроме $S_0 \rightarrow \varepsilon$.
- Для всех правил $A \rightarrow \alpha_1 B_i \alpha_2$, где $B_i \in \text{Nullable}$, добавляем правила $A \rightarrow \alpha_1 \alpha_2$. **И получаем новые ε -правила!** Порядок преобразований существенен.



Устранение ε -правил

$A \in N$ коллапсирует, если $A \rightarrow \varepsilon \in P$ или $A \rightarrow \alpha \in P$ и все элементы α коллапсируют.

- Если $\varepsilon \in L(G)$, тогда добавляем новый стартовый символ S_0 и правила $S_0 \rightarrow \varepsilon, S_0 \rightarrow S$.
- Для всех правил $A \rightarrow \alpha_1 B_i \alpha_2$ ($|\alpha_1 \alpha_2| \geq 1$), где $B_i \in \text{Nullable}$, добавляем правила $A \rightarrow \alpha_1 \alpha_2$.
- Стираем все правила $B_i \rightarrow \varepsilon$, кроме $S_0 \rightarrow \varepsilon$.



Уничтожение цепных правил

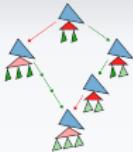
- Строим транзитивное замыкание $A \rightarrow_c^* B$ отношения $A \rightarrow_c B : A \rightarrow B \in P$.
- $\forall A, B : A \rightarrow_c B$, строим множество правил $A \rightarrow \phi_i$, для которых $\exists C, \phi_i(C \rightarrow \phi_i \in P \ \& \ (B \rightarrow_c^* C \vee C = B) \ \& \ (|\phi_i| > 1 \vee \phi_i = \varepsilon \vee (\phi_i = a \ \& \ a \in \Sigma)))$.
- Удаляем все правила $A \rightarrow B$.



Нормальная форма Хомского

Определение

Грамматика G находится в нормальной форме Хомского (CNF) \Leftrightarrow все её правила имеют вид либо $A \rightarrow a$, либо $A \rightarrow BC$, либо $S \rightarrow \epsilon$, причём S не входит в правую часть никакого правила из G .

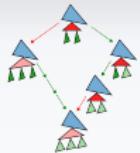


Нормальная форма Хомского

Определение

Грамматика G находится в нормальной форме Хомского (CNF) \Leftrightarrow все её правила имеют вид либо $A \rightarrow a$, либо $A \rightarrow BC$, либо $S \rightarrow \epsilon$, причём S не входит в правую часть никакого правила из G .

- Устранием ϵ -правила.
- Устранием цепные правила.
- $\forall a \in \Sigma$ таких, что a входит в правую часть правила, отличную от a , заводим нетерминал–охранник G_a , строим правило $G_a \rightarrow a$, и во всех правых частях, кроме совпадающих с a , заменяем a на G_a .
- $\forall A \rightarrow B_1 \dots B_n, n > 2$, вводим новый нетерминал B_{1f} и заменяем $A \rightarrow B_1 \dots B_n$ на два правила $A \rightarrow B_1 B_{1f}$, $B_{1f} \rightarrow B_2 \dots B_n$ (рекурсивно).



Смысл нормальной формы Хомского

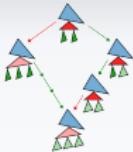
- ❶ Неукорачивающие применения правил
- ❷ Нет пустых переходов — правила либо финальные, либо удлиняющие
- ❸ Контролируемый рост длины сентенциальной формы от количества шагов разбора

Перевод грамматики в CNF позволяет легче анализировать свойства её языка и проводить разбор слов.



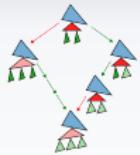
Недостижимость и зацикливание

- Стартовый нетерминал $S \in N$ достижим.
- Нетерминал $A \in N$ достижим, если существует правило $B \rightarrow \alpha$ такое, что $|\alpha|_A \geq 1$ и B достижим.



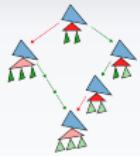
Недостижимость и зацикливание

- Стартовый нетерминал $S \in N$ достижим.
 - Нетерминал $A \in N$ достижим, если существует правило $B \rightarrow \alpha$ такое, что $|\alpha|_A \geq 1$ и B достижим.
-
- Если существует правило $A \rightarrow w$, $w \in \Sigma^*$, A порождающий.
 - Если $A \rightarrow \alpha$ и $\forall B_i (|\alpha|_{B_i} \geq 1 \Rightarrow B_i \text{ порождающий})$, то A порождающий.
-
- ❶ Удаляем из G все правила, в левых или правых частях которых стоят непорождающие нетерминалы.
 - ❷ Удаляем из G все правила, в левых или правых частях которых стоят недостижимые нетерминалы.



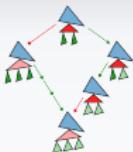
Проверка корректности рекурсивных алгоритмов

- ➊ Завершаемость — фундированность — искомое множество M нетерминалов не может уменьшаться, и количество нетерминалов грамматики конечно.



Проверка корректности рекурсивных алгоритмов

- ❶ Завершаемость — фундированность — искомое множество M нетерминалов не может уменьшаться, и количество нетерминалов грамматики конечно.
- ❷ Корректность — способ доказательства «*minimal bad sequence*» — пусть существуют элементы $k_i \in M$, которые не находятся рекурсивным алгоритмом. Выберем тот из них, до которого минимальный путь из S (варианты — из которого минимальный путь до Σ^* ; до ϵ). Покажем, что есть ещё какой-то с путём вывода ещё короче.



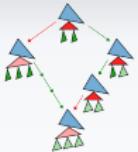
Н.Ф. Хомского и префиксные грамматики

При линеаризации правил, поведение КС-грамматики G в Н.Ф. Хомского в точности описывается алфавитной префиксной грамматикой на нетерминалах.

- Переведём АПГ в регулярную грамматику по методу, описанному в предыдущей лекции.
- У этой грамматики есть длина накачки, т.е. всякое достаточно длинное слово имеет вид $w_1(w_2)^n w_3$, причём $w_1 w_3$ также входит в язык сентенциальных форм G .

Поскольку G — КС-грамматика, то $w_1 \rightarrow \alpha_1$, $w_3 \rightarrow \alpha_3$, каждое из $w_2 \rightarrow \alpha_2$.

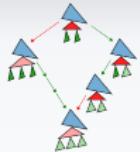
Но есть шаг порождения w_2 , также выбрасывающий последовательность терминалов.



Лемма о накачке КС-языков

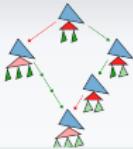
Лемма о накачке (разрастании)

Пусть G — КС-грамматика в форме Хомского. Тогда существует $p \in \mathbb{N}$ такое, что любое слово $w \in L(G)$ длины не меньше p имеет представление вида $x_1y_1z y_2x_2$, где $|y_1y_2| \geq 1$, $|y_1z y_2| \leq p$, и все слова вида $x_1y_1^k z y_2^k x_2$ также принадлежат $L(G)$.



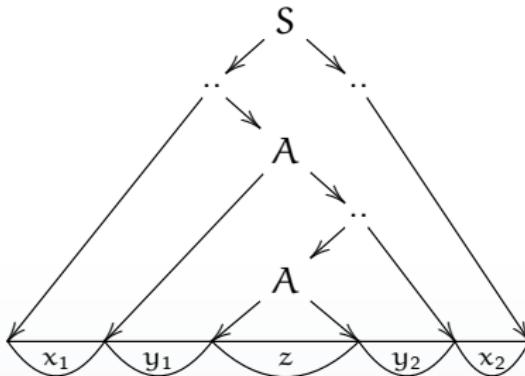
Лемма о накачке КС-языков

Пусть в н.ф. Хомского G n нетерминалов. Возьмём $r = 2^n$. Его вывод будет иметь минимум высоту $n + 1 \Rightarrow$ в нём будет существовать путь, содержащий два одинаковых нетерминала A .



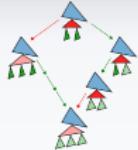
Лемма о накачке КС-языков

Пусть в н.ф. Хомского G n нетерминалов. Возьмём $p = 2^n$. Его вывод будет иметь минимум высоту $n + 1 \Rightarrow$ в нём будет существовать путь, содержащий два одинаковых нетерминала A .



Выберем самые нижние два одинаковых нетерминала \Rightarrow высота поддерева от первого из них не больше $n + 1 \Rightarrow$ длина выводимого слова $y_1 z y_2 \leq 2^n$ (т.е. $\leq p$).



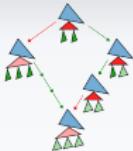


Пример применения

Парсинг в Python

Проанализировать язык

$$\{a^n z_1 a^n z_2 a^n | n \geq 1, |z_i|_a = 0, |z_i| \geq 1\}.$$



Пример применения

Парсинг в Python

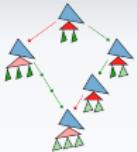
Проанализировать язык

$$\{a^n z_1 a^n z_2 a^n \mid n \geq 1, |z_i|_a = 0, |z_i| \geq 1\}.$$

Пусть длина накачки есть p . Рассмотрим слово $a^p b a^p b a^p$.

Заметим, что если $y_1 z y_2 = a^i b a^j$ (где i и j могут быть равны 0), тогда $|y_1 y_2|_b = 0$. Действительно, иначе нулевая накачка породит слово $a^m b a^p$, которое не принадлежит языку.

Значит, $y_1 = a^j$, $y_2 = a^i$. Однако слова $a^{p+i+j} b a^p b a^p$, $a^p b a^{p+i+j} b a^p$, $a^p b a^p b a^{p+i+j}$, $a^{p+j} b a^{p+i} b a^p$, $a^p b a^{p+j} b a^{p+i}$ ни одно не принадлежат требуемому языку \Rightarrow он не контекстно-свободен.



Теоретико-игровая интерпретация

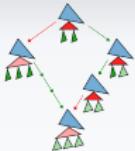
Достаточное условие непринадлежности языка L к КС по лемме о накачке:

$\forall p \exists w \in L (|w| > p \ \& \ \forall x_i, y_i, z (w = x_1 y_1 z y_2 x_2 \ \& \ |y_1 z y_2| < p \Rightarrow \exists i (x_1 y_1^i z y_2^i x_2 \notin L))).$

В пренексной форме этого условия кванторы образуют последовательность:

$\forall \exists \forall \exists$. Эта последовательность задаёт правила игры, где каждый квантор \exists — ход протагониста, квантор \forall — ход антагониста. Ходы антагониста назначают неопределённые параметры. Ходы протагониста дают выбор известной вам структуры, зависящей от ходов антагониста. В случае леммы о накачке это выглядит так.

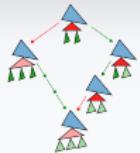
- Антагонист выбирает длину накачки p .
- Зная p , протагонист выбирает w .
- Антагонист выбирает разбиение w на пять подстрок.
- Возможно, в зависимости от этого разбиения, протагонист предъявляет i , для которого накачка не выполняется.



Теоретико-игровая интерпретация

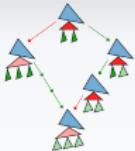
- Антагонист выбирает длину накачки p .
- Зная p , протагонист выбирает w .
- Антагонист выбирает разбиение w на пять подстрок.
- Возможно, в зависимости от этого разбиения, протагонист предъявляет i , для которого накачка не выполняется.

Иногда такая система анализа свойств, записанных в виде формул с чередующимися кванторами, также называется игрой Элоизы и Абеляра (по буквам, образующим кванторы \exists и \forall).



Замыкания

Пока без доказательства: множество КС-языков замкнуто относительно пересечения с регулярными языками.

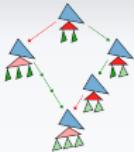


Техника применения

Сужение перебора

Если в язык L входят под слова произвольной формы из Σ^+ , где $|\Sigma| > 1$, тогда, скорее всего, потребуется пересечь L с регулярным языком, чтобы облегчить поиск свидетельства о ненакачиваемости. Пример: язык $\{w_1 w_1 w_2 \mid |w_1|_a = |w_2|_a\}$. Пересечение этого языка с $b a^* b b a^* b b a^*$ гораздо легче поддаётся анализу, поскольку такие слова разбиваются на подходящие w_1 и w_2 однозначно.

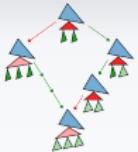
- Начальная буква b вынуждает w_1 содержать ровно две буквы b . Действительно, если $|w_1|_b = 1$, тогда второе вхождение w_1 должно будет начинаться с b^2 , что противоречит выбору w_1 .
- Последняя буква b навязывает позицию начала w_2 .



Техника применения

Работа с отрицанием

Если характеристическая функция L содержит предикат отрицания, связывающий две структуры неопределённого размера, в некоторых случаях это приводит к невозможности применения леммы о накачке. В других можно попробовать воспользоваться приёмом «всё включено». Поскольку мы знаем, что длина накачиваемого фрагмента y_1zy_2 меньше r , то выберем w так, чтобы в нём нашлись всевозможные фрагменты такой длины, удовлетворяющие желательному свойству.

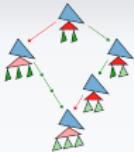


Техника применения

Покажем, что язык $L = \{w \mid w \neq a^{n^2} \& w \in \{a, b\}^*\}$ не является КС.

Для начала заметим, что слова L содержат произвольные подслова в $\{a, b\}^*$, и пересечём L с a^* . Получим $L' = \{a^k \mid k \neq n^2\}$ — если он не КС, то исходный язык также не КС.

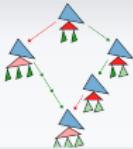
- Антагонист выбирает p .
- Наша задача — подобрать такое k , что $\forall p' \exists i, m(p' < p \Rightarrow k + p' * i = m^2)$. То есть включить возможность взятия любого такого p' в наше значение k как конструктивного элемента для построения квадрата числа.
- Возьмём $k = (p!)^2 + 1$. Тогда при любом значении p' , меньшем p , можно взять $i = \frac{p!}{p'} * 2$, и получим $k + p' * i = (p! + 1)^2$.



Ещё пример применения

Покажем, что язык $L = \{ww^R a^n \mid |w|_a = n\}$ не является КС. Опять сначала избавимся от произвольных подслов в L и пересечём его с языком $ba^+b^2a^+ba^+$. Пересечение с таким языком вынуждает w иметь вид $ba^i b$, а весь язык — вид $L' = \{ba^n bba^n ba^n\}$.

- Абеляр выбирает p . Элоиза строит слово $ba^p b^2a^p ba^p$. Абеляру предоставляется возможность построить его разбиение на $x_1y_1zy_2x_2$.
- Если Абеляр выберет $|y_1|_a > 0 \ \& \ |y_1|_b > 0$ (т.е. y_1 содержащим сразу буквы a и b), тогда ненулевая накачка сразу же выведет нас из языка. Аналогично с y_2 .
- Если Абеляр решит накачивать только b (т.е. выберет y_1 либо y_2 равными b или b^2), тогда любая накачка также будет выводить из языка.



Ещё пример применения

Покажем, что язык $L = \{ww^R a^n \mid |w|_a = n\}$ не является КС. Опять сначала избавимся от произвольных подслов в L и пересечём его с языком $ba^+b^2a^+ba^+$. Пересечение с таким языком вынуждает w иметь вид $ba^i b$, а весь язык — вид $L' = \{ba^n bba^n ba^n\}$.

- Абеляр выбирает p . Элоиза строит слово $ba^p b^2 a^p ba^p$. Абеляру предоставляется возможность построить его разбиение на $x_1 y_1 z y_2 x_2$.
- Остаётся только возможность $y_1 = a^i$, $y_2 = a^j$, что позволяет следующие накачки y_1 , y_2 на расстоянии не больше p :
 - $ba^{p+i+k} b^2 a^{p+j+k} ba^p$ — можно сохранить свойство палиндрома, но нельзя сохранить корректный подсчёт букв a , последний индекс не меняется.
 - $ba^p b^2 a^{p+i+k} ba^{p+j+k}$ — теряется свойство палиндрома.
 - $ba^{p+(i+j)*k} b^2 a^p ba^p$ — теряется свойство палиндрома, при накачке только второго подслова a^p аналогично.
 - $ba^p b^2 a^p ba^{p+(i+j)*k}$ — некорректный подсчёт букв a в w .



Языки, накачиваемые обманно

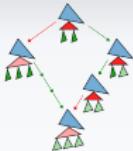
Некоторые не КС-языки тоже накачиваются, например,
 $\{a^m b^n c^n d^n \mid m > 0\} \cup \{b^i c^j d^k\}$.



Языки, накачиваемые обманно

Некоторые не КС-языки тоже накачиваются, например,
 $\{a^m b^n c^n d^n \mid m > 0\} \cup \{b^i c^j d^k\}$.

Действительно, если слово языка содержит буквы a , тогда мы можем взять $y_1 y_2 = a^i$. Иначе накачку можно выбрать произвольно.

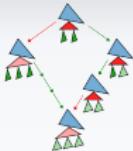


Языки, накачиваемые обманно

Некоторые не КС-языки тоже накачиваются, например,
 $\{a^m b^n c^n d^n \mid m > 0\} \cup \{b^i c^j d^k\}$.

Действительно, если слово языка содержит буквы a , тогда мы можем взять $y_1 y_2 = a^i$. Иначе накачку можно выбрать произвольно.

То, что этот язык — не КС, можно понять по тому факту, что его пересечение с регулярным языком $ab^*c^*d^*$ не контекстно-свободно.



Языки, накачиваемые обманно

Некоторые не КС-языки тоже накачиваются, например,
 $\{a^m b^n c^n d^n \mid m > 0\} \cup \{b^i c^j d^k\}$.

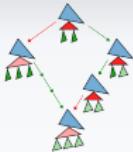
Действительно, если слово языка содержит буквы a , тогда мы можем взять $y_1 y_2 = a^i$. Иначе накачку можно выбрать произвольно.

То, что этот язык — не КС, можно понять по тому факту, что его пересечение с регулярным языком $ab^*c^*d^*$ не контекстно-свободно.

Иногда пересечение с регулярным языком делает язык «излишне накачиваемым»: например, пересекая $L = \{ww^R a^n \mid |w|_a = n\}$ с $ba^+b^*a^+ba^+$, мы даём возможность Абеляру выбрать в качестве y_1 пару букв из центрального блока b^* (положив $y_2 = \varepsilon$). Заметим, что слова без этого блока будут иметь вид $ba^{2n}ba^n$ — а такие слова тоже можно накачивать, выбрав y_1 из a^{2n} , y_2 — из a^n .

Семейство лемм о накачке.
Нормальная форма Грейбах.
Теорема Хомского–Шутценберже

Теория формальных языков
2022 г.



Связь КС-грамматик и АПГ

Рассмотрим язык сентенциальных форм, но только без учёта терминальных символов. Получим алфавитную префиксную грамматику (АПГ, см. лекцию 4). При этом каждое применение правила переписывания такой грамматики выбрасывает не более чем один терминальный символ слева, а стирающее правило — ровно один.

Рассмотрим КС-грамматику, соответствующую ей АПГ и путь вывода слова, получаемый с помощью АПГ.

$S \rightarrow aS bS aN_a bN_b$	$S \rightarrow S N_a N_b$
$N_a \rightarrow N_a E bN_a E cVB$	$N_a \rightarrow N_a E VB$
$N_b \rightarrow aN_b E bN_b E VA$	$N_b \rightarrow N_b E VA$
$V \rightarrow aV bV a b$	$V \rightarrow V \varepsilon$
$A \rightarrow a \quad B \rightarrow b \quad E \rightarrow a b$	$A \rightarrow \varepsilon \quad B \rightarrow \varepsilon \quad E \rightarrow \varepsilon$



Связь КС-грамматик и АПГ

$$S \rightarrow aS | bS | aN_a | bN_b$$

$$N_a \rightarrow N_a E | bN_a E | cVB$$

$$N_b \rightarrow aN_b E | bN_b E | VA$$

$$V \rightarrow aV | bV | a | b$$

$$A \rightarrow a \quad B \rightarrow b \quad E \rightarrow a | b$$

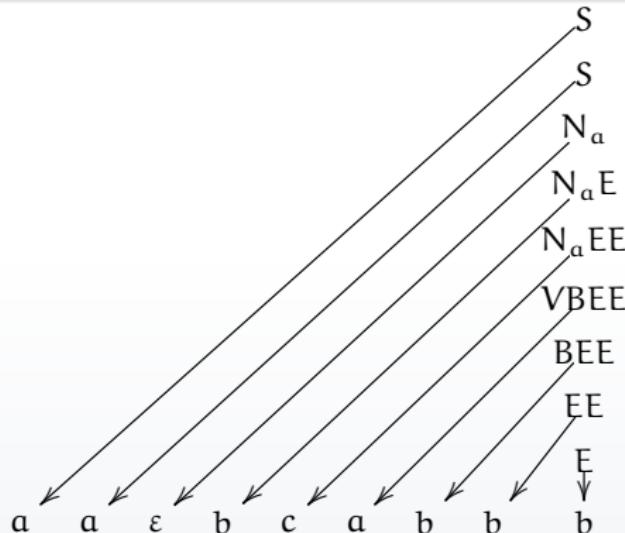
$$S \rightarrow S | N_a | N_b$$

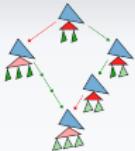
$$N_a \rightarrow N_a E | VB$$

$$N_b \rightarrow N_b E | VA$$

$$V \rightarrow V | \varepsilon$$

$$A \rightarrow \varepsilon \quad B \rightarrow \varepsilon \quad E \rightarrow \varepsilon$$





Основная теорема

Следствие теоремы Турчина

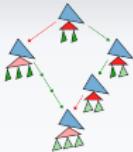
Пусть G — алфавитная префиксная грамматика, в которой N правил с непустой правой частью, и максимальная длина правой части правила равна M . Тогда любая последовательность порождаемых ею слов

$a_1 \dots a_n$

...

ε

длиной не менее $N^M \cdot (n + 1)$ содержит пару вида $\tau_1 = \Phi\Theta_0, \tau_2 = \Phi\Psi\Theta_0$, такую, что $|\Phi| \leq M$ и на отрезке $[\tau_1, \tau_2]$ нет слов длины меньше $|\Theta_0| + 1$.

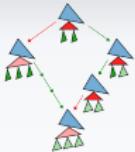


Первая лемма о накачке

Классическая лемма о накачке

Пусть G — КС-язык. Тогда существует $p \in \mathbb{N}$ такое, что любое слово $w \in L(G)$ длины не меньше p имеет представление вида $x_1y_1z y_2x_2$, где $|y_1y_2| \geq 1$, $|y_1z y_2| \leq p$, и все слова вида $x_1y_1^k z y_2^k x_2$ также принадлежат $L(G)$.

Доказательство: при левостороннем разборе выбираем самую последнюю пару сентенциальных форм.



Вторая лемма о накачке

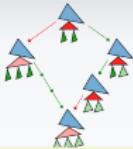
Пусть G — КС-язык. Тогда существует $p \in \mathbb{N}$ такое, что любое слово $w \in L(G)$ длины не меньше p имеет представление вида $x_1y_1z y_2x_2$, где $|y_2| \geq 1$, $|x_1y_1| \leq p$, $|y_2| \leq p$ либо y_2 накачивается отдельно, $|x_2| \leq p$ либо x_2 накачивается отдельно, и все слова вида $x_1y_1^k z y_2^k x_2$ также принадлежат $L(G)$.



Варианты лемм о накачке

- Хотелось бы сдвигать начало отрезка накачки вперёд на любое константное количество букв, аналогично — конец отрезка накачки (используя свойство реверса).
- Понятие накачки может быть применено рекурсивно к некоторым достаточно длинным подсловам выбранного слова. Т.е. можно выкидывать из слова под слова, накачиваемые отдельно, без риска выйти из языка.

Для первого нужна гарантия того, что если порождается k букв слова, то длина сентенциальных форм увеличивается не более чем в $f(k)$ раз (где f — хотя бы полином).



Бонус: лемма Огдена

Пусть L — КС-язык. Тогда существует такое число n , что в любом слове w , $|w| \geq n$, при отметке n или более букв, w представляется в виде $x_1y_1z\,y_2x_2$, причём либо во всех трех из x_1, y_1, z есть отмеченные буквы, либо они есть во всех трех из z, y_2, x_2 , в слове y_1zy_2 отмечено не более n букв, и $\forall k (x_1y_1^k z y_2^k x_2 \in L)$.

Исследуем «плохой» язык $\{a^m b^n c^n d^n \mid m > 0\} \cup \{b^i c^j d^k\}$ с помощью леммы Огдена. Абеляр (антагонист) выбирает n . Элоиза (т.е. мы) строит слово $ab^{2n}c^{2n}d^{2n}$ и отмечает n последних букв d . Абеляр может разбить слово $ab^{2n}c^{2n}d^{2n}$ на $x_1y_1z\,y_2x_2$ двумя способами:

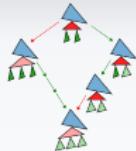
- отмечены x_1, y_1, z , накачиваться может только d^{2n} .
- отмечены x_2, y_2, z , накачивается либо d^{2n} , либо d^{2n} совместно с c^{2n}, b^{2n} или a .

Оба типа накачки выводят из языка, поскольку при любой положительной накачке число вхождений букв b или c расходится с числом вхождений d в слово.



Н.ф. Хомского и левосторонний вывод

- Могут быть непродуктивные левосторонние цепочки:
 $A \rightarrow AB \rightarrow \dots AB^n \rightarrow \dots$
- Есть гарантия роста слова при развертке, но нет определённости, по какому префиксу.

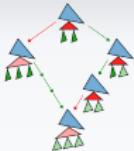


Нормальная форма Грейбах

Определение

Грамматика G ($\epsilon \notin L(G)$) находится в GNF (н.ф. Грейбах)
 \Leftrightarrow каждое её правило имеет вид $A_i \rightarrow a_j \alpha$, где $A_i \in N$,
 $\alpha \in N^*$, $a_j \in \Sigma$.

- Левосторонний разбор по грамматике в GNF на каждом шагу переписывания порождает терминальный символ.

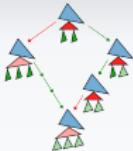


Нормальная форма Грейбах

Определение

Грамматика G ($\epsilon \notin L(G)$) находится в GNF (н.ф. Грейбах)
 \Leftrightarrow каждое её правило имеет вид $A_i \rightarrow a_j \alpha$, где $A_i \in N$,
 $\alpha \in N^*$, $a_j \in \Sigma$.

- Левосторонний разбор по грамматике в GNF на каждом шагу переписывания порождает терминальный символ.
- Для приведения к GNF нужно «вытащить из рекурсии» возможные first-терминалы, порождаемые нетерминалами грамматики.

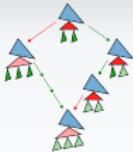


Нормальная форма Грейбах

Определение

Грамматика G ($\epsilon \notin L(G)$) находится в GNF (н.ф. Грейбах)
 \Leftrightarrow каждое её правило имеет вид $A_i \rightarrow a_j \alpha$, где $A_i \in N$,
 $\alpha \in N^*$, $a_j \in \Sigma$.

- Левосторонний разбор по грамматике в GNF на каждом шагу переписывания порождает терминальный символ.
- Для приведения к GNF нужно «вытащить из рекурсии» возможные first-терминалы, порождаемые нетерминалами грамматики.
 - явно найти все завершающиеся цепочки вывода;

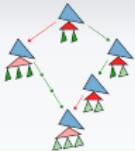


Нормальная форма Грейбах

Определение

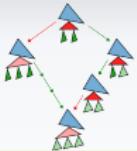
Грамматика G ($\epsilon \notin L(G)$) находится в GNF (н.ф. Грейбах)
 \Leftrightarrow каждое её правило имеет вид $A_i \rightarrow a_j \alpha$, где $A_i \in N$,
 $\alpha \in N^*$, $a_j \in \Sigma$.

- Левосторонний разбор по грамматике в GNF на каждом шагу переписывания порождает терминальный символ.
- Для приведения к GNF нужно «вытащить из рекурсии» возможные first-терминалы, порождаемые нетерминалами грамматики.
 - явно найти все завершающиеся цепочки вывода;
 - рассмотреть язык-реверс сентенциальных форм.
- По умолчанию считаем, что к GNF приводится CNF (н.ф. Хомского).



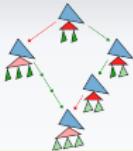
Первый способ приведения к GNF

- ① Нумеруем нетерминалы в правых частях правил в порядке их вхождения;
- ② (по исчерпанию по i , начиная с $i = 1$) Если имеется правило вида $A_i \rightarrow B_j \beta$, где $j < i$, тогда подставляем вместо B_j все правые части α_k правил вида $B_j \rightarrow \alpha_k$.
- ③ Если после этого все правила имеют вид либо $A_i \rightarrow a\alpha$, $a \in \Sigma$, либо $A_i \rightarrow B_j \beta$, причём $i < j$, тогда GNF получается последовательной развёрткой B_j .



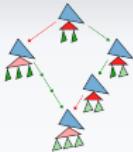
Первый способ приведения к GNF

- ① Нумеруем нетерминалы в правых частях правил в порядке их вхождения;
- ② (по исчерпанию по i , начиная с $i = 1$) Если имеется правило вида $A_i \rightarrow B_j \beta$, где $j < i$, тогда подставляем вместо B_j все правые части α_k правил вида $B_j \rightarrow \alpha_k$.
- ③ Если после этого все правила имеют вид либо $A_i \rightarrow a\alpha$, $a \in \Sigma$, либо $A_i \rightarrow B_j \beta$, причём $i < j$, тогда GNF получается последовательной развёрткой B_j . Существует лексикографический порядок на функциональных символах из N .



Первый способ приведения к GNF

- ① Нумеруем нетерминалы в правых частях правил в порядке их вхождения;
- ② (по исчерпанию по i , начиная с $i = 1$) Если имеется правило вида $A_i \rightarrow B_j \beta$, где $j < i$, тогда подставляем вместо B_j все правые части α_k правил вида $B_j \rightarrow \alpha_k$.
- ③ Если после этого все правила имеют вид либо $A_i \rightarrow a\alpha$, $a \in \Sigma$, либо $A_i \rightarrow B_j \beta$, причём $i < j$, тогда GNF получается последовательной развёрткой B_j .
Существует лексикографический порядок на функциональных символах из N .
- ④ Если есть правила вида $A_i \rightarrow A_i \alpha$, тогда устранием левую рекурсию.
- ⑤ Увеличиваем i , если ещё остались нетерминалы, не приведённые к ГНФ.



Устранение левой рекурсии

- ❶ Предположим, для A_i нашлось n леворекурсивных правил и m упорядоченных лексикографически:

$$A_i \rightarrow A_i \alpha_1$$

...

$$A_i \rightarrow A_i \alpha_n$$

$$A_i \rightarrow \beta_1$$

...

$$A_i \rightarrow \beta_m$$

- ❷ Вводим новый нетерминал A'_i такой, что его вес меньше всех прочих, и заменяем правила на:

$$A'_i \rightarrow \alpha_1 A'_i | \alpha_1$$

...

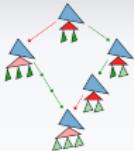
$$A'_i \rightarrow \alpha_n A'_i | \alpha_n$$

$$A_i \rightarrow \beta_1 | \beta_1 A'_i$$

...

$$A_i \rightarrow \beta_m | \beta_m A'_i$$

- ❸ После всех таких замен грамматика лексикографически упорядочена по левому разбору, и GNF получается последовательной левой развёрткой.

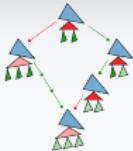


Второй способ приведения к GNF

Алгоритм Блюма–Коха (1999).

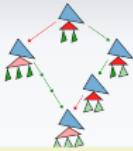
Неформальное описание

- Рассмотрим язык сентенциальных форм с переписыванием только по левому разбору. Он регулярен, и в конечное состояние его НКА ведут стрелки, помеченные терминалами.
- Для такого языка легко построить инверсный \Rightarrow множество терминалов-префиксов, которые может породить данный нетерминал.



Второй способ: порождение НКА

- ❶ По каждому нетерминалу B строим автомат $M_B = \langle N_B \cup \{S_B\}, \Sigma \cup N, B_B, \{S_B\}, \delta \rangle$ (S_B — новое состояние, N_B — множество нетерминалов CFG, индексированное нетерминалом B). Правила перехода δ :
 - $\langle C_B, E, M \rangle \Leftrightarrow M = \{D_B \mid C \rightarrow DE \in P\};$
 - $\langle C_B, a, \{S_B\} \rangle \Leftrightarrow C \rightarrow a \in P.$
- ❷ Строим реверс к M_B , получаем НКА M_B^R .
- ❸ Строим грамматику $G'_B = \langle N_B \cup \{S_B\}, \Sigma \cup N, R'_B, S_B \rangle$ для M_B^R с правилами переписывания:
 - $S_B \rightarrow aC_B \Leftrightarrow \langle S_B, a, C_B \rangle \in \delta^R$ и $C_B \neq B_B$ либо из B_B есть стрелки в M_B^R ;
 - $S_B \rightarrow a \Leftrightarrow \langle S_B, a, B_B \rangle \in \delta^R;$
 - $D_B \rightarrow EC_B \Leftrightarrow \langle D_B, E, C_B \rangle \in \delta^R$ и $C_B \neq B_B$ либо из B_B есть стрелки в M_B^R ;
 - $C_B \rightarrow E \Leftrightarrow \langle C_B, E, B_B \rangle \in \delta^R.$



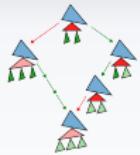
Окончание конструкции

Теперь по всем G'_i строим окончательный вариант грамматики $G_B = \langle N_B \cup \{S_B\}, \Sigma, R_B, S_B \rangle$ с правилами:

- $S_B \rightarrow aC_B, S_B \rightarrow aC_B \in R'_B;$
- $S_B \rightarrow a S_B \rightarrow a \in R'_B;$
- $D_B \rightarrow \alpha C_B \Leftrightarrow D_B \rightarrow E C_B \in R'_B \ \& \ S_E \rightarrow \alpha$ (по всем таким α и E);
- $D_B \rightarrow \alpha \Leftrightarrow D_B \rightarrow E \in R'_B \ \& \ S_E \rightarrow \alpha$ (по всем таким α и E).

Грамматика $\bigcup_{i \in N} G_i$ со стартовым символом S_S — это искомая GNF для исходной грамматики G .

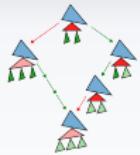
Последние два правила разворачивают неразмеченные нетерминалы в стартовые правила грамматик их сентенциальных форм, поэтому автоматы M_B имеет смысл строить только для тех нетерминалов, которые встречаются в исходной грамматике не только первыми в правых частях правил.



Пример преобразования грамматики по Блюму–Коху

Привести к GNF грамматику некорректных сумм двоичных чисел (почему некорректных?)

$$S \rightarrow S + S \mid D \quad D \rightarrow D0 \mid D1 \mid 1 \mid (S)$$



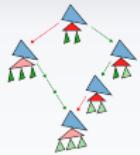
Пример преобразования грамматики по Блюму–Коху

Привести к GNF грамматику некорректных сумм двоичных чисел (почему некорректных?)

$$S \rightarrow S + S \mid D \quad D \rightarrow D0 \mid D1 \mid 1 \mid (S)$$

Сначала избавляемся от цепного правила $S \rightarrow D$. Потом строим порождающую структуру A_V сентенциальных форм по левостороннему разбору с финальным состоянием N_V и стартовым V_V . Каждому нетерминалу V соответствует своя структура.

$$\begin{array}{lllll} \text{Для } A_S : & S_S \xrightarrow{+S} S_S & S_S \xrightarrow{0} D_S & S_S \xrightarrow{1} D_S & S_S \xrightarrow{1} N_S \\ & S_S \xrightarrow{(S)} N_S & D_S \xrightarrow{0} D_S & D_S \xrightarrow{1} D_S & D_S \xrightarrow{(S)} N_S \end{array}$$



Пример преобразования грамматики по Блюму–Коху

Привести к GNF грамматику некорректных сумм двоичных чисел (почему некорректных?)

$$S \rightarrow S + S \mid D \quad D \rightarrow D0 \mid D1 \mid 1 \mid (S)$$

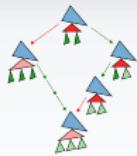
Сначала избавляемся от цепного правила $S \rightarrow D$. Потом строим порождающую структуру A_V сентенциальных форм по левостороннему разбору с финальным состоянием N_V и стартовым V_V . Каждому нетерминалу V соответствует своя структура.

Для A_S :

$$\begin{array}{lllll} S_S \xrightarrow{+S} S_S & S_S \xrightarrow{0} D_S & S_S \xrightarrow{1} D_S & S_S \xrightarrow{1} N_S \\ S_S \xrightarrow{(S)} N_S & D_S \xrightarrow{0} D_S & D_S \xrightarrow{1} D_S & D_S \xrightarrow{1} N_S & D_S \xrightarrow{(S)} N_S \end{array}$$

Для A_D :

$$D_D \xrightarrow{0} D_D \quad D_D \xrightarrow{1} D_D \quad D_D \xrightarrow{1} N_D \quad D_D \xrightarrow{(S)} N_D$$



Пример преобразования грамматики по Блюму–Коху

Превращаем структуры в праволинейные (меняя местами нетерминалы левых и правых частей правил и стартовые состояния с финальными):

Для A_S : $S_S \xrightarrow{+S} S_S \quad D_S \xrightarrow{0} S_S \quad D_S \xrightarrow{1} D_S \quad N_S \xrightarrow{1} S_S$
 $N_S \xrightarrow{(S)} S_S \quad D_S \xrightarrow{0} D_S \quad D_S \xrightarrow{1} D_S \quad N_S \xrightarrow{1} D_S \quad N_S \xrightarrow{(S)} D_S$

Для A_D : $D_D \xrightarrow{0} D_D \quad D_D \xrightarrow{1} D_D \quad N_D \xrightarrow{1} D_D \quad N_D \xrightarrow{(S)} D_D$

Извлекаем праволинейные (почти) грамматики. В правой части правила может не быть нетерминалов, если там стоял нетерминал V_V .

$$\begin{array}{lllll} q\text{-RLG } G_S : & S_S \rightarrow +SS_S & D_S \rightarrow 0S_S & D_S \rightarrow 1D_S & N_S \rightarrow 1S_S \\ & N_S \rightarrow (S)S_S & D_S \rightarrow 0D_S & D_S \rightarrow 1D_S & N_S \rightarrow 1D_S & N_S \rightarrow (S)D_S \\ S_S \rightarrow +S & D_S \rightarrow 0 & D_S \rightarrow 1 & N_S \rightarrow 1 & N_S \rightarrow (S) \end{array}$$

Извлекаем праволинейные (почти) грамматики. В правой части правила может не быть нетерминалов, если там стоял нетерминал V_V .

$$\begin{array}{lllll} \text{q-RLG } G_S : & S_S \rightarrow +SS_S & D_S \rightarrow 0S_S & D_S \rightarrow 1D_S & N_S \rightarrow 1S_S \\ & N_S \rightarrow (S)S_S & D_S \rightarrow 0D_S & D_S \rightarrow 1D_S & N_S \rightarrow 1D_S \\ & S_S \rightarrow +S & D_S \rightarrow 0 & D_S \rightarrow 1 & N_S \rightarrow 1 \\ & & & & N_S \rightarrow (S) \end{array}$$

$$\begin{array}{lllll} \text{q-RLG } G_D : & D_D \rightarrow 0D_D & D_D \rightarrow 1D_D & N_D \rightarrow 1D_D & N_D \rightarrow (S)D_D \\ & D_D \rightarrow 0 & D_D \rightarrow 1 & N_D \rightarrow 1 & N_D \rightarrow (S) \end{array}$$

Извлекаем праволинейные (почти) грамматики. В правой части правила может не быть нетерминалов, если там стоял нетерминал V_V .

$$\begin{array}{lllll} q\text{-RLG } G_S : & S_S \rightarrow +SS_S & D_S \rightarrow 0S_S & D_S \rightarrow 1D_S & N_S \rightarrow 1S_S \\ N_S \rightarrow (S)S_S & D_S \rightarrow 0D_S & D_S \rightarrow 1D_S & N_S \rightarrow 1D_S & N_S \rightarrow (S)D_S \\ S_S \rightarrow +S & D_S \rightarrow 0 & D_S \rightarrow 1 & N_S \rightarrow 1 & N_S \rightarrow (S) \end{array}$$

$$\begin{array}{lllll} q\text{-RLG } G_D : & D_D \rightarrow 0D_D & D_D \rightarrow 1D_D & N_D \rightarrow 1D_D & N_D \rightarrow (S)D_D \\ D_D \rightarrow 0 & D_D \rightarrow 1 & N_D \rightarrow 1 & N_D \rightarrow (S) \end{array}$$

Заменяем неразмеченные нетерминальные символы V исходной грамматики на N_V . В данном случае нет правил, в которых неразмеченные нетерминалы стояли бы первыми в правых частях, поэтому достаточно просто заменить их на N_V . Иначе пришлось бы заменять их на все возможные правые части α правил вида $N_V \rightarrow \alpha$. Стартовый символ — N_S . GNF почти построена!

Извлекаем праволинейные (почти) грамматики. В правой части правила может не быть нетерминалов, если там стоял нетерминал V_V .

Заменяем неразмеченные нетерминальные символы V исходной грамматики на N_V . В данном случае нет правил, в которых неразмеченные нетерминалы стояли бы первыми в правых частях, поэтому достаточно просто заменить их на N_V . Иначе пришлось бы заменять их на все возможные правые части α правил вида $N_V \rightarrow \alpha$. Стартовый символ — N_S . GNF почти построена!

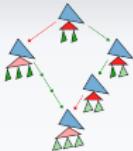
$$\begin{array}{lllll} q\text{-GNF для } G : & S_S \rightarrow +N_SS_S & D_S \rightarrow 0S_S & D_S \rightarrow 1D_S & N_S \rightarrow 1S_S \\ & N_S \rightarrow (N_S)S_S & D_S \rightarrow 0D_S & D_S \rightarrow 1D_S & N_S \rightarrow 1D_S \\ & S_S \rightarrow +N_S & D_S \rightarrow 0 & D_S \rightarrow 1 & N_S \rightarrow 1 & N_S \rightarrow (N_S)D_S \end{array}$$

Извлекаем праволинейные (почти) грамматики. В правой части правила может не быть нетерминалов, если там стоял нетерминал V_V .

Заменяем неразмеченные нетерминальные символы V исходной грамматики на N_V . В данном случае нет правил, в которых неразмеченные нетерминалы стояли бы первыми в правых частях, поэтому достаточно просто заменить их на N_V . Иначе пришлось бы заменять их на все возможные правые части α правил вида $N_V \rightarrow \alpha$. Стартовый символ — N_S . GNF почти построена!

$$\begin{array}{lllll} q\text{-GNF для } G : & S_S \rightarrow +N_SS_S & D_S \rightarrow 0S_S & D_S \rightarrow 1D_S & N_S \rightarrow 1S_S \\ & N_S \rightarrow (N_S)S_S & D_S \rightarrow 0D_S & D_S \rightarrow 1D_S & N_S \rightarrow 1D_S \\ & S_S \rightarrow +N_S & D_S \rightarrow 0 & D_S \rightarrow 1 & N_S \rightarrow 1 \\ & & & & N_S \rightarrow (N_S) \end{array}$$

Осталось обернуть в delay-нетерминалы терминальные символы правых частей правил, кроме первого. Здесь это символ $)$.



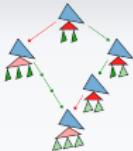
Теорема Хомского–Шутценберже

Пусть PAREN_n — язык из $4 * n$ элементов
 $\{[1,]_1, \dots, [n,]_n, (1,)_1, \dots, (n,)_n\}$.

Теорема

Любой CF-язык получается гомоморфизмом из языка
 $L' = \text{PAREN}_n \cap R$, где R — регулярный.

Пусть G — грамматика L в нормальной форме Хомского.
Пронумеруем правила G и поставим им в соответствие
следующие.



Теорема Хомского–Шутценберже

Пусть PAREN_n — язык из $4 * n$ элементов

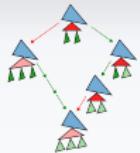
$\{[1,]_1, \dots, [n,]_n, (1,)_1, \dots, (n,)_n\}$.

Теорема

Любой CF-язык получается гомоморфизмом из языка
 $L' = \text{PAREN}_n \cap R$, где R — регулярный.

Пусть G — грамматика L в нормальной форме Хомского.
Пронумеруем правила G и поставим им в соответствие
следующие.

- ① Если правило n имеет вид $A \rightarrow BC$, тогда порождаем правило $A \rightarrow [n]BnC[n]$.
- ② Если правило n имеет вид $A \rightarrow a$, тогда порождаем правило $A \rightarrow [n]n(n)n$.



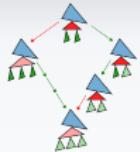
Свойства языка $L(G')$

- Все $]_n$ строго предшествуют $(_n$.



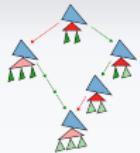
Свойства языка $L(G')$

- Все $]_n$ строго предшествуют $(_n$.
- Ни одна $)_n$ не предшествует непосредственно левой скобке.



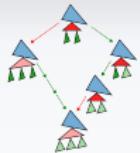
Свойства языка $L(G')$

- Все $]_n$ строго предшествуют $(_n$.
- Ни одна $)_n$ не предшествует непосредственно левой скобке.
- Если правило n — это $A \rightarrow BC$, тогда $[_n$ непосредственно предшествует некоторой $[_p$, так же как и $(_n$.



Язык R

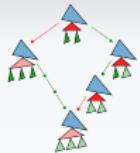
$R = \{x \in \{[j,]_j, (j,)_j\}^* \mid x \text{ начинается с } [_n \text{ для некоторого правила } n : A \rightarrow \dots \& \text{ все }]_n \text{ предшествуют } (n)\}.$



Язык R

$R = \{x \in \{[j,]_j, (j,)_j\}^* \mid x \text{ начинается с } [_n \text{ для некоторого правила } n : A \rightarrow \dots \& \text{ все }]_n \text{ предшествуют } (n)\}.$

Можно убедиться, что $L(G') = R \cap \text{PAREN}_n$.

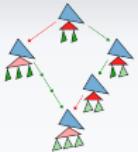


Язык R

$R = \{x \in \{[j,]_j,(j,)_j\}^* \mid x \text{ начинается с } [_n \text{ для некоторого правила } n : A \rightarrow \dots \& \text{ все }]_n \text{ предшествуют } (_n)\}.$

Можно убедиться, что $L(G') = R \cap \text{PAREN}_n$.

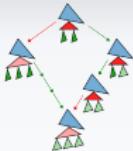
Осталось определить h . Если n — нефинальное правило, то $h([_n]) = h(]_n) = h((_n)) = h(_n) = \varepsilon$. Иначе $h([_n]) = a$, для остальных скобок так же.



Значение теоремы Х.-Ш.

Возможно разделить парсинг любого КС-языка на две стадии: лексический анализ (проверка условия R) и разбор правильных скобочных структур.

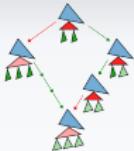
Замечание: поскольку гомоморфизм h не обязан быть инъективным, разбор ПСП не всегда можно определить однозначно. Пример: $\{a^n b^n\} \cup \{a^n b^{2n}\}$ (полностью неоднозначность устраниить нельзя, т.к. этот язык не является детерминированным). Однако Т.Х.Ш. даёт подсказку, как строить КС-грамматики: надо найти в языке все скрытые «скобочные структуры».



Построение грамматики по Х.-Ш.

Построить КС-грамматику для языка
 $\{a^n b^m c^k \mid n = 2 * m - k\}$.

Ищем возможную скобочную структуру. Для этого сначала избавимся от вычитания: $n + k = 2 * m$. Значит, буквы a должны балансироваться буквами b справа (т.е. буквы b являются «закрывающими скобками» для a), а буквы c — буквами b слева (т.е. буквы b являются «открывающими» для c). Возможны два случая: n и k оба чётны либо оба нечётны. Построим соответствующие им разбиения: $\{a^{2*n'} b^{n'} b^{k'} c^{2*k'}\}$ и $\{aa^{2*n'} b^{n'} bb^{k'} c^{2*k'} c\}$. Дальнейшее построение грамматики уже очевидно. Заметим, что гомоморфизм подразумевает минимум четыре вида скобок: пара $(_{2a},)_b$, пара $(_b,)_{2c}$, внешняя пара $[_a,]_c$ (для нечётного варианта) и $[_b,]_\varepsilon$ для него же, чтобы породить внутреннюю букву b .



Построение грамматики по Х.-Ш.

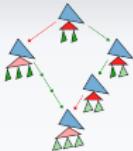
Построить КС-грамматику для языка
 $\{a^n b^m c^k \mid n = 2 * m - k\}$.

Как итог, получаем язык, гомоморфно порождаемый языком Дика над $\{(2a,)_b, (b,)_{2c}, [b,]_\varepsilon, [a,]_c\}$ со следующим лексером:

- ① До $(2a$ может идти лишь единственная $[a$.
- ② После $)_b$ распознаётся одна $[b$, если распознавалась $[a$.
- ③ После $)_b$ или $]_\varepsilon$ не может идти ничего другого, кроме $(_b$ или $]_c$ (последняя — только после $]_\varepsilon$).
- ④ После $)_{2c}$ не может быть ничего, кроме $)_{2c}$ или $]_c$.

Дополнительное условие на существование $[a$ уже не требуется — оно следует из сбалансированности ПСП.

Конструкция выше отличается от используемой в доказательстве теоремы — в целях экономии, в ней почти нет скобок, гомоморфно отображаемых в пустое слово.

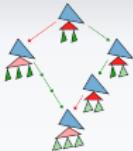


Дополнительный пример

Построить КС-грамматику для $L_{\neq} = \{w_1 c w_2 \mid w_i \in \{a, b\}^+ \text{ & } w_1 \neq w_2\}$.

Классический пример грамматики с не-КС дополнением. Чтобы расшифровать неравенство, раскроем его в дизъюнкцию: «слово w_1 короче, чем w_2 ; либо w_2 короче, чем w_1 ; либо существует такое i , что w_1 и w_2 различаются в i -й позиции». Здесь условия не взаимоисключающие: достаточно одного из них, чтобы слово принадлежало L_{\neq} , но могут выполняться и два сразу.

Перепишем первое условие: $w_1 c w'_1 w_2$, где $|w_2| > 0$ и $|w_1| = |w'_1|$. Очевидно, что «открывающими скобками» будут буквы из w_1 , «закрывающими» — из w'_1 , а «скобки» для w_2 замкнуты на самом w_2 . Чтобы обеспечить четыре вида соответствий букв по счёту, придётся ввести четыре пары скобок для w_1 и w'_1 : $\{(a,)_a, (b,)_b, [a,]_b, [b,]_a\}$. И две пары скобок для w_2 : $\{\{a, \}_\epsilon, \{b, \}'_\epsilon\}$ и пара скобок для порождения с: $(c \text{ и })_\epsilon$ (в нижних индексах — гомоморфные образы скобок).



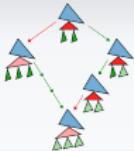
Дополнительный пример

Построить КС-грамматику для $L_{\neq} = \{w_1 c w_2 \mid w_i \in \{a, b\}^+ \text{ & } w_1 \neq w_2\}$.

Осталось построить регулярные условия. Для языка $w_1 c w'_1 w_2$, где $|w_2| > 0$ и $|w_1| = |w'_1|$, их можно описать следующим образом:

- ① После $(_a, (_b, [_a, [_b$ всегда идёт либо опять одна из таких скобок, либо $(_c$. Скобка $(_c$ единственна.
- ② После $)_\varepsilon$, а также скобок $)_a,)_b,]_b,]_a$, могут идти либо $)_a,)_b,]_b,]_a$, либо фигурные скобки.
- ③ В каждом слове есть хотя бы одна фигурная скобка. После открывающей фигурной скобки обязательно сразу идёт закрывающая, и после первой встреченной фигурной скобки все остальные скобки — тоже фигурные.

Представление Хомского–Шутценбергера для языка $w_1 w_2 c w'_1$, где $|w_2| > 0$ и $|w_1| = |w'_1|$, строится симметрично.



Дополнительный пример

Построить КС-грамматику для $L_{\neq} = \{w_1cw_2 \mid w_i \in \{a, b\}^+ \text{ & } w_1 \neq w_2\}$.

Осталось разобрать $\{w_1t_1w_2cw_3t_2w_4 \mid |w_1| = |w_3| \text{ & } t_1 \neq t_2\}$. Очевидно, что в нём «открывающими» будут элементы w_1 , закрывающими — элементы w_3 , t_1 и t_2 — уникальные скобки, отображающиеся в разные элементы алфавита, а w_2 и w_4 закрываются сами собой (как w_2 в предыдущем языке). Для $t_1 + t_2$ -скобок назначим пары $[^t_a,]^t_b$ и $[^t_b,]^t_a$, остальные обозначения сохраним те же. Лексер языка:

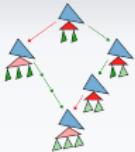
- ❶ После $(_a, (_b, [_a, [_b$ идёт либо опять одна из таких скобок, либо $[^t_-$ -скобка. $[^t_-$ единственна, за ней следует либо $\{_a$, либо $\{_b'$, либо $(_c$.
- ❷ За открывающей фигурной скобкой следует закрывающая, и после первой встреченной фигурной скобки все остальные скобки — тоже фигурные, до конца строки либо до чтения единственной $(_c$.
- ❸ После $)_\varepsilon$, а также скобок $)_a,)_b,]_b,]_a$, могут идти либо $)_a,)_b,]_b$, $]_a$, либо скобка $]^t_-$. За $]^t_-$ следует EOL или $\{_a$ или $\{_b'$.

Чтобы получить прообраз языка L_{\neq} , объединим все три лексера.

Теорема Париха. Стековые автоматы



Теория формальных языков
2022 г.

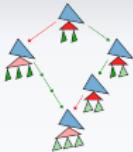


Теорема Париха

Скажем, что множество векторов полулинейно, если оно является конечным объединением множеств векторов вида $(i_1 + \sum p_{t,1} \cdot j_{t,1}, \dots, i_k + \sum p_{t,k} \cdot j_{p,k})$.

КС-язык \mathcal{L} в алфавите Σ можно описать количественно:
как множество $V_{\mathcal{L}}$ векторов
 $\{(k_{j,1}, \dots, k_{j,n}) \mid k_{j,i} = |w_j|_{a_i} \& w_j \in \mathcal{L}\}$.

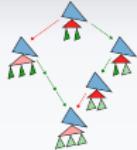
. Если \mathcal{L} — КС-язык, тогда $V_{\mathcal{L}}$ полулинейно.



Теорема Париха

Пусть $V_{\mathcal{L}} = \{(k_{j,1}, \dots, k_{j,n}) \mid k_{j,i} = |w_j|_{a_i} \& w_j \in \mathcal{L}\}$. Если \mathcal{L} — КС-язык, тогда $V_{\mathcal{L}}$ полулинейно.

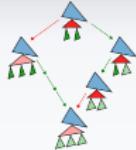
- Если $V_{\mathcal{L}}$ полулинеен, то $V_{h(\mathcal{L})}$ полулинеен (h — гомоморфизм).
- Если w принадлежит языку Шютценберже, то $\forall n (|w|_{(n)} = |w|_n = |w|_{[n]} = |w|_{]n})$.
- Рассматриваем префиксные трассы вывода над ГНФ языка Шютценберже и выкидываем из них наиболее короткие отрезки накачки. Их длина ограничена \Rightarrow ограничено их множество.



Следствия теоремы Париха

Множества регулярных и КС-языков над однобуквенным алфавитом совпадают.

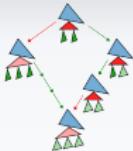
Коммутативным образом всякого КС-языка является регулярный язык.



Свойства замкнутости КС-языков

Пусть стартовый нетерминал грамматики G_i — это S_i .

- КС-языки тривиально замкнуты относительно объединения и конкатенации. Объединение: добавим правило $S' \rightarrow S_1 | S_2$, конкатенация: добавим правило $S' \rightarrow S_1 S_2$.
- КС-языки замкнуты относительно реверсирования (достаточно реверсировать левые части правил), а также префиксных и суффиксных замыканий (упражнение после освоения материала по PDA).

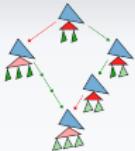


Свойства замкнутости КС-языков

- КС-языки не замкнуты относительно пересечения.

Универсальный контрпример: пусть $\$$ – символ-разделитель (отсутствует в словаре). Рассмотрим язык $\{w_1\$().^*\$w_3\}$, где слова w_1 и w_2 связаны порождающими правилами (т.е. структура $w_1\$w_3$ не регулярна), и язык $\{w_1\$w_2\$().^*\}$, где такими правилами связаны w_1 и w_2 . Их пересечение вынудит существование нерегулярной зависимости между w_1 и одновременно w_2 и w_3 , что порождает не две, как в КС-языках, а минимум три области накачки.

Конкретные примеры: пара $L_1 = \{a^n\$a^n\$a^*\}$, $L_2 = \{a^n\$a^*\$a^n\}$; или пара $L'_1 = \{w\$w^R\$a^*\}$, $L'_2 = \{w\$(a|b)^*\$a^n \mid |w|_a = n\}$.



Свойства замкнутости КС-языков

- КС-языки не замкнуты относительно дополнения. В противном случае пересечение также не нарушало бы свойство контекстной свободы. Контпримеры строятся на основе этой же идеи: берём язык-пересечение КС-языков L_1 и L_2 , не являющийся КС. Чаще всего его дополнение — КС-язык.

Конкретные примеры: КС-язык

$L' = \overline{L_1 \cap L_2} = \{a^m \$ a^n \$ a^k \mid m \neq n \vee m \neq k \vee n \neq k\}$ с не КС-дополнением; КС-язык

$L'' = \overline{L'_1 \cap L'_2} = \{w \$ v \$ a^n \mid v \neq w^R \vee n \neq |w|_a\}$ с не КС-дополнением.



Свойства замкнутости КС-языков

- КС-языки тривиально замкнуты относительно объединения и конкатенации.
- КС-языки замкнуты относительно реверсирования (достаточно реверсировать левые части правил), а также префиксных и суффиксных замыканий (упражнение после освоения материала по PDA).
- КС-языки не замкнуты относительно пересечения.
- КС-языки не замкнуты относительно дополнения.
- КС-языки замкнуты относительно пересечения с регулярным языком (см. ниже).



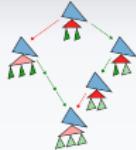
Пересечение КС-грамматики и рег. языка

Утверждение

Даны КС-грамматика G и конечный автомат \mathcal{A} . Можно построить КС-грамматику G' такую, что $L(G') = L(G) \cap L(\mathcal{A})$.

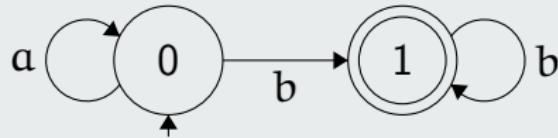
Предположим, что G — в k -нормальной форме Хомского (т.е. с максимум k нетерминалами в правых частях), q — множество состояний автомата \mathcal{A} , q_f — единственное финальное состояние, N — множество нетерминалов грамматики G . Множество нетерминалов G' — множество $\langle q_i, A, q_j \rangle$, $q_i, q_j \in q$, $A \in N$.

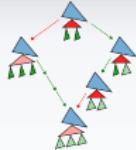
- По каждому правилу $A \rightarrow A_1 \dots A_n$ из G строим правила $\langle p, A, q \rangle \rightarrow \langle p, A_1, q_1 \rangle \langle q_{n-1}, A_n, q \rangle$ для всех возможных p , q , q_i .
- По правилу вида $A \rightarrow t$ из G и переходу $p \xrightarrow{t} q$ строим правило $\langle p, A, q \rangle \rightarrow t$.
- Нетерминал $\langle q_0, S, q_f \rangle$ объявляем стартовым.



Пример

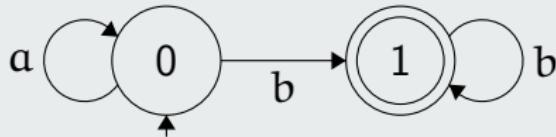
Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:





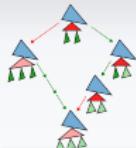
Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



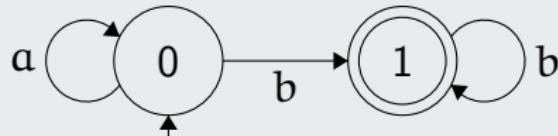
Сначала разберёмся с правилами вида $X \rightarrow t$. Если $t = a$, тогда подходящий нетерминал — только G_A , состояния — только 0+0. Если $t = b$, получается четыре комбинации состояний и нетерминалов.

$$\begin{array}{ll} \langle 0, G_B, 1 \rangle \rightarrow b & \langle 1, G_B, 1 \rangle \rightarrow b \\ \langle 0, T, 1 \rangle \rightarrow b & \langle 1, T, 1 \rangle \rightarrow b \end{array} \quad \begin{array}{l} \langle 0, G_A, 0 \rangle \rightarrow a \end{array}$$



Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



$$\langle 0, G_B, 1 \rangle \rightarrow b \quad \langle 1, G_B, 1 \rangle \rightarrow b$$

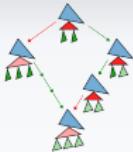
$$\langle 0, T, 1 \rangle \rightarrow b \quad \langle 1, T, 1 \rangle \rightarrow b \quad \langle 0, G_A, 0 \rangle \rightarrow a$$

Рассмотрим возможные подстановки состояний в правила,
порождаемые $S \rightarrow G_A T$, $T \rightarrow SG_B$. Соответствующие уравнения:

$$\langle X_1, S, X_2 \rangle \rightarrow \langle X_1, G_A, X_3 \rangle \langle X_3, T, X_2 \rangle$$

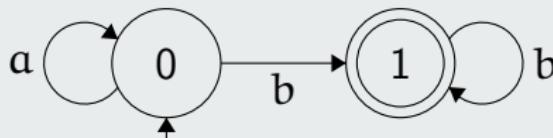
$$\langle Y_1, T, Y_2 \rangle \rightarrow \langle Y_1, S, Y_3 \rangle \langle Y_3, G_B, Y_2 \rangle$$

Чтобы правила были порождающими, необходимо положить
 $X_1 = X_3 = 0$, $Y_2 = 1$. Выпишем все такие правила. Заметим, что
получившийся в одном из них нетерминал $\langle 0, T, 0 \rangle$ — непорождающий,
и удалим это правило.



Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



$$\langle 0, G_B, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow b$$

$$\langle 0, S, 0 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 0 \rangle$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 0 \rangle \langle 0, G_B, 1 \rangle$$

$$\langle 1, T, 1 \rangle \rightarrow \langle 1, S, 0 \rangle \langle 0, G_B, 1 \rangle$$

$$\langle 1, G_B, 1 \rangle \rightarrow b$$

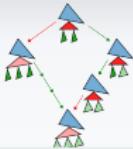
$$\langle 1, T, 1 \rangle \rightarrow b$$

$$\langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

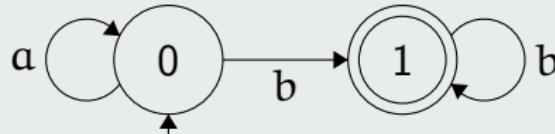
$$\langle 1, T, 1 \rangle \rightarrow \langle 1, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

Осталось разобраться с правилами, порождёнными $S \rightarrow SS$. Выпишем их общий вид: $\langle X_1, S, X_2 \rangle \rightarrow \langle X_1, S, X_3 \rangle \langle X_3, S, X_2 \rangle$.



Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



$$\langle 0, G_B, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 0 \rangle \langle 0, G_B, 1 \rangle \quad \langle 0, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

$$\langle 1, T, 1 \rangle \rightarrow \langle 1, S, 0 \rangle \langle 0, G_B, 1 \rangle \quad \langle 1, T, 1 \rangle \rightarrow \langle 1, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

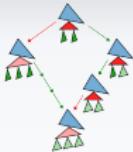
$$\langle 1, G_B, 1 \rangle \rightarrow b$$

$$\langle 1, T, 1 \rangle \rightarrow b \quad \langle 0, G_A, 0 \rangle \rightarrow a$$

$$\langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle$$

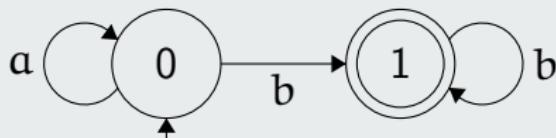
Осталось разобраться с правилами, порождёнными $S \rightarrow SS$. Выпишем их общий вид: $\langle X_1, S, X_2 \rangle \rightarrow \langle X_1, S, X_3 \rangle \langle X_3, S, X_2 \rangle$.

Если положить $X_1 = 1$, $X_2 = 0$, получим саморекурсивное правило $\langle 1, S, 0 \rangle \rightarrow \alpha_1 \langle 1, S, 0 \rangle \alpha_2$. Но в построенной части грамматики нет правил вида $\langle 1, S, \dots \rangle \rightarrow \beta$. Поэтому нетерминал $\langle 1, S, 0 \rangle$ — непорождающий. Удалим правила с его вхождением.



Пример

Построим пересечение языков CFG $S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



$$\langle 0, G_B, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 0 \rangle \langle 0, G_B, 1 \rangle$$

$$\langle 1, G_B, 1 \rangle \rightarrow b$$

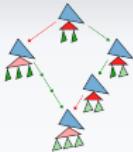
$$\langle 1, T, 1 \rangle \rightarrow b$$

$$\langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

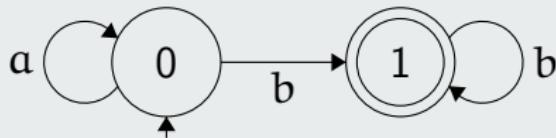
$$\langle 1, T, 1 \rangle \rightarrow \langle 1, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

Теперь если $X1 = X2 = 1$, то единственный вариант развёртки $S \rightarrow SS$ без участия нетерминала $\langle 1, S, 0 \rangle$ будет иметь вид
 $\langle 1, S, 1 \rangle \rightarrow \langle 1, S, 1 \rangle \langle 1, S, 1 \rangle$, так что нетерминал $\langle 1, S, 1 \rangle$ тоже непорождающий.



Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



$$\langle 0, G_B, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow \langle 0, S, 0 \rangle \langle 0, G_B, 1 \rangle \quad \langle 1, G_B, 1 \rangle \rightarrow b$$

$$\langle 1, G_B, 1 \rangle \rightarrow b$$

$$\langle 1, T, 1 \rangle \rightarrow b$$

$$\langle 1, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle \quad \langle 0, G_A, 0 \rangle \rightarrow a$$

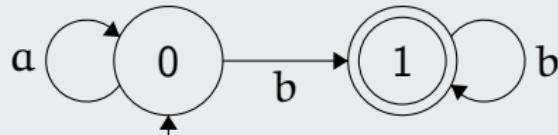
$$\langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle$$

Аналогичным образом устанавливаем бесполезность нетерминала $\langle 0, S, 0 \rangle$, который обязан ссылаться либо дважды на себя, либо на непорождающий $\langle 1, S, 0 \rangle$.



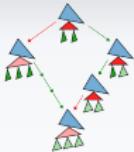
Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



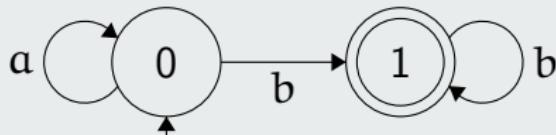
$$\begin{array}{ll} \langle 0, G_B, 1 \rangle \rightarrow b & \langle 1, G_B, 1 \rangle \rightarrow b \\ \langle 0, T, 1 \rangle \rightarrow b & \langle 1, T, 1 \rangle \rightarrow b \quad \langle 0, G_A, 0 \rangle \rightarrow a \\ & \langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle \\ & \langle 0, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle \end{array}$$

Теперь получается, что все варианты раскрытия нетерминала $\langle 0, S, 1 \rangle$ по правилу $S \rightarrow SS$ включают непорождающие нетерминалы, поэтому никаких других правил в грамматику добавлять не надо. Осталось только удалить правила с недостижимыми нетерминалами $\langle 0, G_B, 1 \rangle$, $\langle 1, T, 1 \rangle$.



Пример

Построим пересечение языков $CFG\ S \rightarrow G_A T | SS$,
 $T \rightarrow b | SG_B$, $G_A \rightarrow a$, $G_B \rightarrow b$, и следующего автомата:



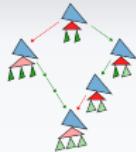
$$\langle 0, G_A, 0 \rangle \rightarrow a$$

$$\langle 1, G_B, 1 \rangle \rightarrow b$$

$$\langle 0, T, 1 \rangle \rightarrow b$$

$$\langle 0, S, 1 \rangle \rightarrow \langle 0, G_A, 0 \rangle \langle 0, T, 1 \rangle \quad \langle 0, T, 1 \rangle \rightarrow \langle 0, S, 1 \rangle \langle 1, G_B, 1 \rangle$$

Грамматика пересечения языков построена.



Алгоритм Кока–Янгера–Касами (CYK)

Задача

Дано слово $w_1 \dots w_n \in \Sigma^+$ и грамматика G в CNF.

Проверить, выполнено ли $w \in L(G)$.

Идея алгоритма: переход к более простым задачам порождения подстрок w .



Алгоритм Кока–Янгера–Касами (CYK)

Задача

Дано слово $w_1 \dots w_n \in \Sigma^+$ и грамматика G в CNF.

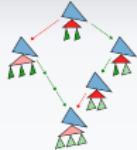
Проверить, выполнено ли $w \in L(G)$.

Идея алгоритма: переход к более простым задачам порождения подстрок w .

Определим функцию $f(A, i, j)$ (где $i \leq j$), возвращающую ответ, можно ли вывести слово $w_i \dots w_j$ из $A \in N$.

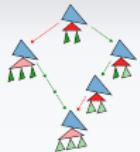
- Если $i = j$, тогда $f(A, i, j) = T \Leftrightarrow A \rightarrow w_i \in P$, и $f(A, i, j) = F$ иначе.
- Если $i < j$, тогда

$$f(A, i, j) = \bigvee_{(A \rightarrow BC \in P)} \bigvee_{k=i+1}^j (f(B, i, k-1) \& f(C, k, j)).$$



Стековая память

Пусть G — CFG. Неформально представим, что G — это стековый автомат, где состояния стека — нетерминальные сент. формы, порождаемые G . Скажем, что G распознаёт только слова, соответствующие пустому стеку.



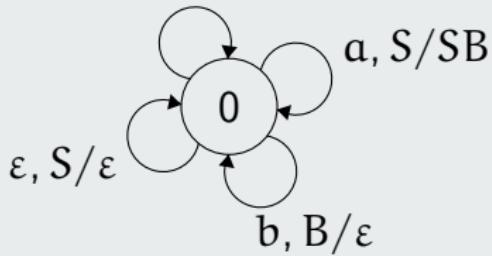
Стековая память

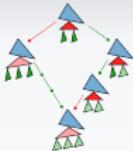
Пусть G — CFG. Неформально представим, что G — это стековый автомат, где состояния стека — нетерминальные сент. формы, порождаемые G . Скажем, что G распознаёт только слова, соответствующие пустому стеку.

Грамматика и её стек

$$S \rightarrow aSB \mid SS \mid \varepsilon \quad B \rightarrow b$$

$$\varepsilon, S/SS$$





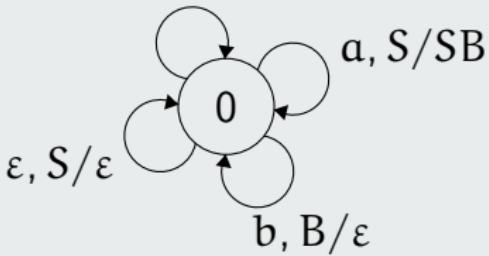
Стековая память

Пусть G — CFG. Неформально представим, что G — это стековый автомат, где состояния стека — нетерминальные синтаксические формы, порождаемые G . Скажем, что G распознаёт только слова, соответствующие пустому стеку.

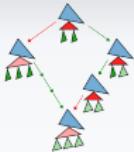
Грамматика и её стек

$$S \rightarrow aSB \mid SS \mid \epsilon \quad B \rightarrow b$$

$$\epsilon, S/SS$$



А если в такие автоматы добавить ещё состояния?

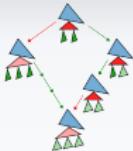


Pushdown Automata

Определение

Стековый автомат \mathcal{A} — кортеж $\langle \Pi, \Sigma, Q, \delta, q_0, Z_0 \rangle$, где:

- Π — алфавит стека;
- Σ — алфавит языка;
- Q — множество состояний;
- δ — правила перехода вида $\langle q_i, t, P_i \rangle \rightarrow \langle q_j, \alpha \rangle$, где $t \in \Sigma \cup \{\varepsilon\}$, $\alpha \in \Pi^*$;
- q_0 — стартовое состояние, Z_0 — дно стека.



Pushdown Automata

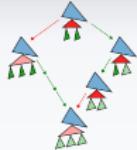
Определение

Стековый автомат \mathcal{A} — кортеж $\langle \Pi, \Sigma, Q, \delta, q_0, Z_0 \rangle$, где:

- Π — алфавит стека;
- Σ — алфавит языка;
- Q — множество состояний;
- δ — правила перехода вида $\langle q_i, t, P_i \rangle \rightarrow \langle q_j, \alpha \rangle$, где $t \in \Sigma \cup \{\epsilon\}$, $\alpha \in \Pi^*$;
- q_0 — стартовое состояние, Z_0 — дно стека.

Два варианта допуска слова:

- если слово полностью прочитано, и стек пуст;
- если слово полностью прочитано, и состояние финальное.



Виды допуска

Утверждение

PDA с допуском по конечному состоянию распознают те же языки, что и PDA с допуском по пустому стеку.



Виды допуска

Утверждение

PDA с допуском по конечному состоянию распознают те же языки, что и PDA с допуском по пустому стеку.

- Пусть PDA допускает пустой стек. Добавим новый символ дна Z_1 и добавим по нему ϵ -переходы из всех состояний в новое финальное состояние.

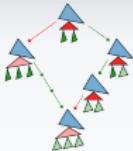


Виды допуска

Утверждение

PDA с допуском по конечному состоянию распознают те же языки, что и PDA с допуском по пустому стеку.

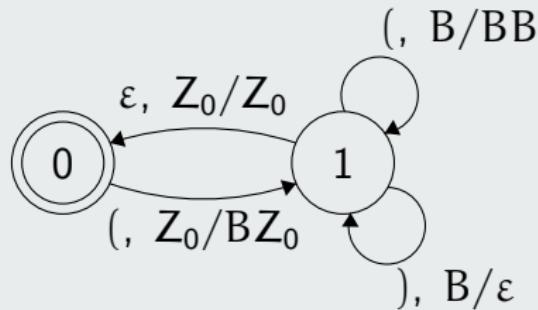
- Пусть PDA допускает пустой стек. Добавим новый символ дна Z_1 и добавим по нему ϵ -переходы из всех состояний в новое финальное состояние.
- Пусть PDA допускает финальные состояния. Добавим из них ϵ -переходы в состояние, опустошающее стек, а также новый символ стека Z_1 и новое стартовое состояние q'_0 с переходом $\langle q'_0, \epsilon, Z_0 \rangle \rightarrow \langle q_0, Z_0 Z_1 \rangle$.



Пример оформления PDA

Обычно PDA изображается в виде автомата, в котором стрелки помечены сигнатурой α , T/Φ , где α — это символ терминального алфавита (или пустое слово), T — символ на вершине стека, Φ — последовательность стековых символов, помещаемая на вершину стека вместо T .

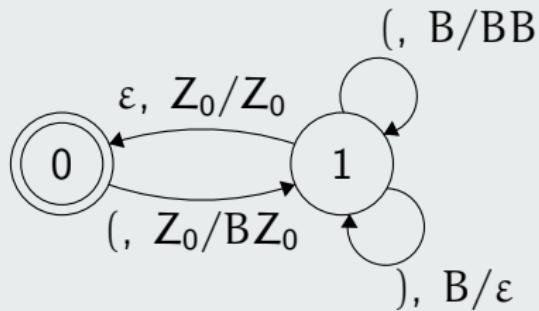
Следующий PDA распознаёт правильные скобочные последовательности (включая пустое слово).





Пример оформления PDA

Следующий PDA распознаёт правильные скобочные последовательности (включая пустое слово).



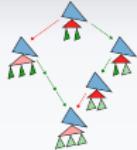
Заметим, что перехода из состояния 0 по символу) нет. Так же как и в случае конечных автоматов, можно добавить для такого перехода состояние-ловушку, потому что он порождает слова, в префиксе которых количество закрывающих скобок превышает количество открывающих, а такие слова не являются ПСП.



От CFG к PDA

Утверждение

По всякой CFG G можно построить PDA \mathcal{A} такой, что $L(G) = L(\mathcal{A})$.



От CFG к PDA

Утверждение

По всякой CFG G можно построить PDA \mathcal{A} такой, что $L(G) = L(\mathcal{A})$.

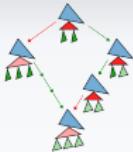
Переведём G в GNF и построим по ней PDA с единственным состоянием 0 и допуском по пустому стеку, такой что $Z_0 = S$, правилу $A \rightarrow a$ соответствует переход $(0, a, A) \rightarrow (0, \varepsilon)$; правилу $A \rightarrow aB_1 \dots B_n$ — переход $(0, a, A) \rightarrow (0, B_1 \dots B_n)$.



От PDA к CFG

Утверждение

По всякому PDA \mathcal{A} можно построить CFG G такую, что $L(G) = L(\mathcal{A})$.



От PDA к CFG

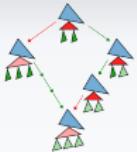
Утверждение

По всякому PDA \mathcal{A} можно построить CFG G такую, что $L(G) = L(\mathcal{A})$.

Пусть \mathcal{A} допускает слова по пустому стеку.

- Построим по стеку \mathcal{A} вспомогательную G' :

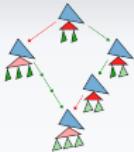
- введём новые стековые символы и заменим правила $(q_i, t, A) \rightarrow (q_j, A_1 \dots A_n)$ ($n \geq 1$) на пары $(q_i, \epsilon, A) \rightarrow (q_i, A_0 \dots A_n), (q_i, t, A_0) \rightarrow (q_j, \epsilon)$.
- переход $(q_i, \epsilon, A) \rightarrow (q_j, A_0 A_1 \dots A_n)$ поставим в соответствие правилу $A \rightarrow A_0 A_1 \dots A_n$; переход $(q_i, t, A) \rightarrow (q_j, \epsilon)$ поставим в соответствие правилу $A \rightarrow t_{i,j}$. Z_0 объявим стартовым символом.
Пустой символ введём явно и так же пометим.



От PDA к CFG

Пусть \mathcal{A} допускает слова по пустому стеку.

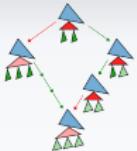
- Построим по стеку \mathcal{A} вспомогательную G' :
 - введём новые стековые символы и заменим правила $(q_i, t, A) \rightarrow (q_j, A_1 \dots A_n)$ ($n \geq 1$) на пары $(q_i, \epsilon, A) \rightarrow (q_i, A_0 \dots A_n)$, $(q_i, t, A_0) \rightarrow (q_j, \epsilon)$.
 - переход $(q_i, \epsilon, A) \rightarrow (q_j, A_0 A_1 \dots A_n)$ поставим в соответствие правилу $A \rightarrow A_0 A_1 \dots A_n$; переход $(q_i, t, A) \rightarrow (q_j, \epsilon)$ поставим в соответствие правилу $A \rightarrow t_{i,j}$. Z_0 объявим стартовым символом.
Пустой символ введём явно и так же пометим.
- Построим $\mathcal{A}' — FA$ с правилами вида $(q_i, t_{i,j}) \rightarrow q_j$, если для каких-нибудь A, α $(q_i, t, A) \rightarrow (q_j, \alpha)$ — переход \mathcal{A} . Все состояния объявим финальными.



От PDA к CFG

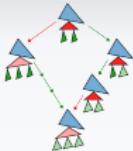
Пусть \mathcal{A} допускает слова по пустому стеку.

- Построим по стеку \mathcal{A} вспомогательную G' :
 - введём новые стековые символы и заменим правила $(q_i, t, A) \rightarrow (q_j, A_1 \dots A_n)$ ($n \geq 1$) на пары $(q_i, \epsilon, A) \rightarrow (q_i, A_0 \dots A_n)$, $(q_i, t, A_0) \rightarrow (q_j, \epsilon)$.
 - переход $(q_i, \epsilon, A) \rightarrow (q_j, A_0 A_1 \dots A_n)$ поставим в соответствие правилу $A \rightarrow A_0 A_1 \dots A_n$; переход $(q_i, t, A) \rightarrow (q_j, \epsilon)$ поставим в соответствие правилу $A \rightarrow t_{i,j}$. Z_0 объявим стартовым символом. Пустой символ введём явно и так же пометим.
- Построим \mathcal{A}' — FA с правилами вида $(q_i, t_{i,j}) \rightarrow q_j$, если для каких-нибудь A, α $(q_i, t, A) \rightarrow (q_j, \alpha)$ — переход \mathcal{A} . Все состояния объявим финальными.
- Теперь построим CFG — пересечение G' и \mathcal{A}' и сотрем все $\epsilon_{i,j}$ и разметку терминалов. Грамматика G готова!



PDA в CFG формально

- Нетерминалы — тройки $[p, A, q]$, где $p, q \in Q, A \in \Pi$.
- По каждому переходу вида $(q, t, A) \rightarrow (p, A_1 \dots A_n)$ добавим правила для всех возможных q_i вида $[q, A, q_n] \rightarrow t[p, A_1, q_1] \dots [q_{n-1}, A_n, q_n]$.
- По каждому переходу вида $(q, t, A) \rightarrow (p, \varepsilon)$ добавим правило $[q, A, p] \rightarrow t$.
- Разрешим стартовому состоянию переписываться в любое из $[q_0, Z_0, q]$.



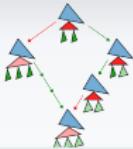
DPDA

Определение

PDA \mathcal{A} детерминированный, если:

- если есть переход $\langle q, \varepsilon, Z \rangle \rightarrow \dots$, то больше никаких переходов по Z из состояния q нет;
- каждой тройке $\langle q, a, Z \rangle$, $a \in \Sigma$, соответствует не больше одной правой части.

DPDA слабее, чем NPDA. Например, язык $\{a^n b^m \mid n = m \vee m = 2 * n\}$ не распознается DPDA. DPDA с допуском по пустому стеку ещё слабее — язык $\{a^n\}$ не может быть распознан DPDA с таким допуском.

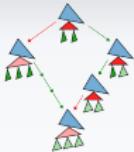


DPDA

DPDA слабее, чем NPDA. Например, язык $\{a^n b^m \mid n = m \vee m = 2 * n\}$ не распознается DPDA. DPDA с допуском по пустому стеку ещё слабее — язык $\{a^n\}$ не может быть распознан DPDA с таким допуском.

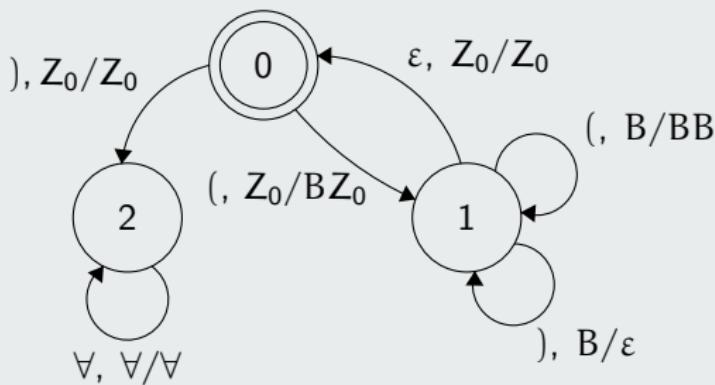
Предположим, что существует DPDA, распознающий язык $\{a^n b^m \mid n = m \vee m = 2 * n\}$. Тогда после чтения префикса $a^n b^n$ слова $a^n b^{2n}$ он должен находиться в финальном состоянии. Далее он должен распознать ровно n букв b . Заменим часть автомата, распознающую этот фрагмент слова, на изоморфную ей, но читающую только буквы c . Получим PDA, распознающий язык $\{a^n b^n\} \cup \{a^n b^n c^n\}$, не являющийся КС.

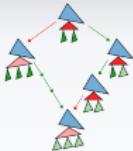
Предположим, что существует DPDA с допуском по пустому стеку, распознающий язык $\{a^n\}$. Тогда на слове a стек этого автомата должен быть уже точно пуст \Rightarrow в этом состоянии вообще невозможно сделать дальнейшие переходы.



Пример DPDA

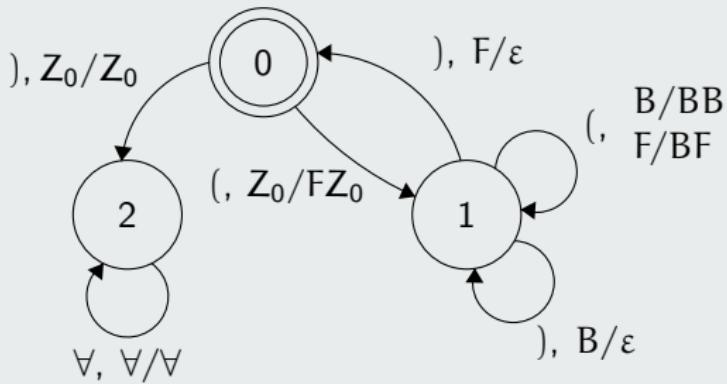
PDA для ПСП, приведённый выше, является DPDA, в чём нетрудно убедиться, проверив, что ϵ -переход совершается лишь в том случае, когда никакие другие совершились невозможны. Добавим в него состояние-ловушку.

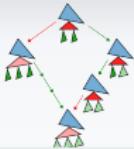




Пример DPDA

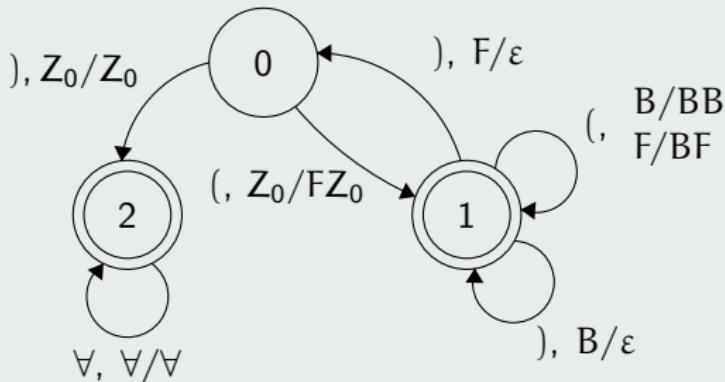
Чтобы не описывать многочисленные переходы из состояния-ловушки в себя по всем парам «символ ленты — символ стека», мы воспользовались сокращённым обозначением \forall , \forall/\forall , подразумевая следующее: «по любой паре $(\text{терминал}, \text{символ стека})$ в состоянии 2 переходим в себя, сохраняя символ стека на вершине». Также избавимся от ε -перехода, введя символ стека F , т.е. «самая первая скобка».

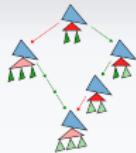




Пример DPDA

Поскольку автомат \mathcal{A} — детерминированный, в нём существуют переходы по всем комбинациям \langle терминал, символ стека \rangle , и нет ϵ -переходов, связывающих нефинальное и финальное состояния, то автомат, в котором все конечные состояния \mathcal{A} заменены на нефинальные и наоборот, распознаёт дополнение языка, распознаваемого PDA \mathcal{A} . Значит, мы показали, что дополнение языка ПСП контекстно-свободно, и предъявили PDA, который распознаёт его.





Двухсторонние PDA

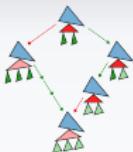
Утверждение

Двухсторонние PDA распознают больше языков, чем односторонние.

Доказательство: язык $\{a^n b^n c^n\}$ распознаем двухсторонним PDA.

Кодирующие КС-языки Нисходящий разбор

Теория формальных языков
2022 г.



Кодировка путей праволинейной грамматики

Рассмотрим путь вывода произвольного слова $a_1 \dots a_n$ в праволинейной грамматике. Он имеет вид

$S \rightarrow a_1 A_1; A_1 \rightarrow a_2 A_2; \dots A_n \rightarrow a_n$. Применим к нему обратный гомоморфизм $h(A_i; A_i \rightarrow) = \varepsilon$ и сотрём префикс $S \rightarrow$, получим искомое слово.

Алфавит: $\Sigma \cup N \cup \{; , \rightarrow\}$. Описание языка:
 $\{S \rightarrow a_i (A_i; A_i \rightarrow a_j)^*\}$.

Описание языка привязано к множеству нетерминалов в рассматриваемой грамматике, но левые и правые вхождения нетерминалов можно было бы закодировать скобками, по количеству считающими номер нетерминала.

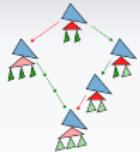


Кодировка путей КС-грамматики

Аналогично — с кодировкой путей вывода в КС-грамматике. Но здесь есть проблема с "перепутанными скобками": в пути $A_1 \rightarrow a_1 A_2 A_3; A_2 \rightarrow \dots; A_3 \rightarrow \dots$ имена A_2 и A_3 состоят в зависимости с соответствующими левыми частями, но перемежаются.

Решение

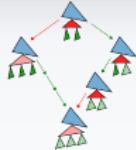
Поменять местами кодировки для A_2 и A_3 в правых частях.



Язык Грейбах

Здесь ε -free вариант. D — язык сбалансированных скобочных структур над $\{(,), [,]\}$.

$L_0 = \{x_1cy_1cz_1d \dots dx_ncy_ncz_nd \mid y_1 \dots y_n \in eD \text{ &} z_i, x_i \text{ не содержат } e \text{ &} y_1 \in e\{(,), [,]\}^* \text{ &} y_{i+1} \in \{(,), [,]\}^*\}$



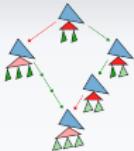
Язык Грейбах

Здесь ε -free вариант. D — язык сбалансированных скобочных структур над $\{(,), [,]\}$.

$L_0 = \{x_1cy_1cz_1d \dots dx_ncy_ncz_nd \mid y_1 \dots y_n \in eD \text{ &} \\ z_i, x_i \text{ не содержат } e \text{ &} y_1 \in e\{(,), [,]\}^* \text{ &} y_{i+1} \in \{(,), [,]\}^*\}$

Утверждение

Если L — КС-язык, тогда существует $h \in \text{Hom}$ такой, что $h^{-1}(L_0) = L$.

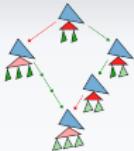


Гомоморфизм Грейбах

Пусть G — грамматика для L в форме Грейбах (Шейлы!). Пронумеруем нетерминалы G так, чтобы стартовый был первым. Построим вспомогательную функцию ξ :

- для правил $A_i \rightarrow a$ положим $\xi(i) =])^i)$
- для правил $A_i \rightarrow aA_{j_1} \dots A_{j_n}$ положим
 $\xi(i) =])^i)([^m(\dots ([^1($
- если $i = 1$, тогда дополнительно припишем префикс $e([[(.$

Пусть терминалом a начинаются левые части правил k_1, \dots, k_m . Тогда $h(a) = c\xi(k_1)c \dots c\xi(k_m)d$.



Коммутативный образ по Пиллингу

Лемма Ардена для коммутативных образов

Пусть правила грамматики имеют следующий вид:

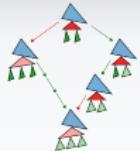
$$T \rightarrow W_1 T^{i_1} T | \dots | W_{k-1} T^{i_{k-1}} T | W_k$$

где W_i не содержит T , и степени i_j различны. При этом W_i могут содержать любые регулярные операции над константами и нетерминалами, отличными от T .

Коммутативный образ правил для T есть

$$T = (W_1(W_k)^{i_1} + \dots + W_{k-1}(W_k)^{i_{k-1}})^* W_k.$$

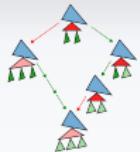
Ограничение: меняет местами подвыводы, соединяя те, которые накачиваются отдельно.



μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X)$, $r(r(X))$, \dots , $r^n(X)$.

Что будет, если добавить в регулярные выражения μ -оператор?



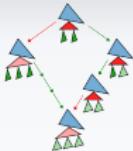
μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X)$, $r(r(X))$, \dots , $r^n(X)$.

Что будет, если добавить в регулярные выражения μ-оператор?

μ-оператор по переменным + регулярные операции
определяют множество всех КС-языков.

Пример: $\mu y.a(\mu x. axb + y + a)$ определяет грамматику $Y \rightarrow a X$, $X \rightarrow a X b | Y | a$.



μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X)$, $r(r(X))$, \dots , $r^n(X)$.

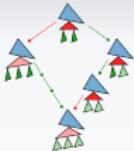
Что будет, если добавить в регулярные выражения μ-оператор?

μ-оператор по переменным + регулярные операции
определяют множество всех КС-языков.

Пример: $\mu y.a(\mu x. axb + y + a)$ определяет грамматику $Y \rightarrow a X$, $X \rightarrow a X b \mid Y \mid a$.

Неудобно, если язык некоторого нетерминала используется в нескольких правилах (например, $S \rightarrow A B \mid B A$ заставит дважды выписывать μ-выражения для A и B).

В действительности аналогом μ-регулярных выражений выступает РБНФ: в правых частях правил разрешены любые регулярные операции.

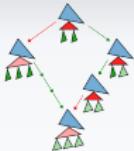


Проблемы PDA и сила Рефала

Не всегда по стеку и входным символам можно понять, что делать со стеком. Пример: грамматика палиндромов.

Неформально: заглядывание вперёд на произвольное, но заранее ограниченное число символов не даёт никакой информации о том, что делать со стеком.

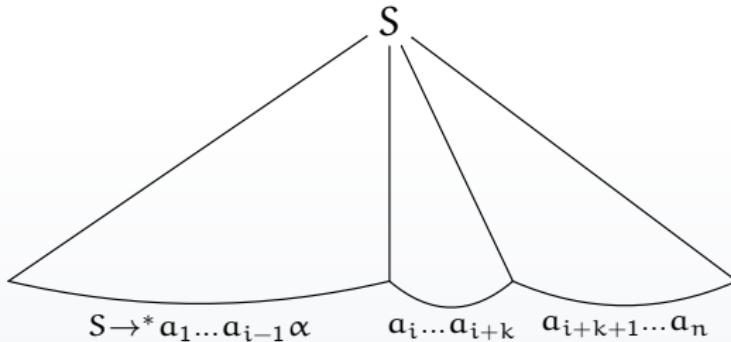
- Возьмём слова $a^{2p} b a^{2p}$ и a^{2p} .
- После чтения p символов в первом случае нужно продолжать накапливать стек, а во втором — начинать вынимать из него.
- Но впереди одни и те же a^p букв...

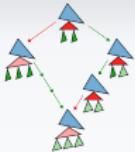


Нисходящий разбор

Пусть PDA-анализатор хранит пару $\langle \alpha, a_i \dots a_{i+k} \rangle$, где α — сент. форма, $a_i \dots a_{i+k}$ — k следующих символов в строке.

- Если $\alpha = A\alpha'$, тогда по правилу $A \rightarrow \beta$ стековый анализатор переходит в $\langle \beta\alpha', a_i \dots a_{i+k} \rangle$, либо сообщает об ошибке.
- Если $\alpha = a_i\alpha'$, тогда a_i одновременно снимается со стека и читается во входной строке.





LL(k)-грамматики

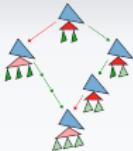
Определение

Грамматика G называется LL(k) (*left-to-right, leftmost derivation*) \Leftrightarrow в ситуации, когда существуют выводы

$$S \rightarrow^* w_1 A \alpha \rightarrow w_1 \xi \alpha \rightarrow^* w_1 c w_2,$$

$$S \rightarrow^* w_1 A \alpha \rightarrow w_1 \eta \alpha \rightarrow^* w_1 c w_3, \text{ причём } c \in \Sigma^k,$$

$w_1, w_2, w_3 \in \Sigma^*$, или $c \in \Sigma^{<k}$, $w_2 = w_3 = \varepsilon$, всегда $\eta = \xi$.



LL(k)-грамматики

Определение

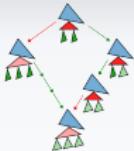
Грамматика G называется LL(k) (*left-to-right, leftmost derivation*) \Leftrightarrow в ситуации, когда существуют выводы

$$S \rightarrow^* w_1 A \alpha \rightarrow w_1 \xi \alpha \rightarrow^* w_1 c w_2,$$

$$S \rightarrow^* w_1 A \alpha \rightarrow w_1 \eta \alpha \rightarrow^* w_1 c w_3, \text{ причём } c \in \Sigma^k,$$

$$w_1, w_2, w_3 \in \Sigma^*, \text{ или } c \in \Sigma^{<k}, w_2 = w_3 = \varepsilon, \text{ всегда } \eta = \xi.$$

Неформально: неоднозначность в выборе правила грамматики при разборе сверху вниз устраняется заглядыванием вперёд на k букв.

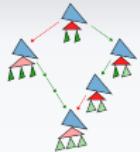


Критерий LL(k)-грамматики

Определим

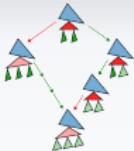
- $\text{FIRST}_k(\eta) = \{a_1 \dots a_j \mid (j < k \ \& \ \eta \rightarrow a_1 \dots a_j) \vee (j = k \ \& \ \eta \rightarrow a_1 \dots a_k \alpha)\} \cup \{\varepsilon \mid \eta \rightarrow^* \varepsilon\}$
- $\text{FOLLOW}_k(\eta) = \{a_1 \dots a_j \mid (j < k \ \& \ S \rightarrow^* \beta \eta a_1 \dots a_j) \vee (j = k \ \& \ S \rightarrow^* \beta \eta a_1 \dots a_k \alpha)\} \cup \{\$\mid S \rightarrow^* \beta \eta\}$

Тогда G — LL(k)-грамматика $\Leftrightarrow \forall A \rightarrow \alpha, A \rightarrow \beta$
 $(\text{FIRST}_k(\alpha) \text{ FOLLOW}_k(A)) \cap$
 $\text{FIRST}_k(\beta) \text{ FOLLOW}_k(A) = \emptyset$.



Вычисление FIRST_k

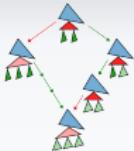
- $\text{FIRST}_k(A) = \{a_1 \dots a_k \mid A \rightarrow a_1 \dots a_k \alpha\}$
 $\cup \{a_1 \dots a_j \mid j < k \ \& \ A \rightarrow a_1 \dots a_j\};$
- До исчерпания: $\forall A \rightarrow B_1 \dots B_n, \text{FIRST}_k(A) = \text{FIRST}_k(A) \cup \text{first}_k(\text{FIRST}_k(B_1) \dots \text{FIRST}_k(B_n))$, где first_k — это k первых символов строки.



Вычисление FIRST_k

- $\text{FIRST}_k(A) = \{a_1 \dots a_k \mid A \rightarrow a_1 \dots a_k \alpha\}$
 $\cup \{a_1 \dots a_j \mid j < k \ \& \ A \rightarrow a_1 \dots a_j\};$
- До исчерпания: $\forall A \rightarrow B_1 \dots B_n, \text{FIRST}_k(A) = \text{FIRST}_k(A) \cup \text{first}_k(\text{FIRST}_k(B_1) \dots \text{FIRST}_k(B_n))$, где first_k — это k первых символов строки.

Алгоритм задаёт фундированный порядок на множестве конфигураций $\text{FIRST}_k(A_i)$, поэтому рано или поздно множества FIRST_k перестанут изменяться.



Вычисление FOLLOW_k

Добавляем правило из нового стартового символа $S_0 \rightarrow S\$$.

- $\text{FOLLOW}_k(A) = \emptyset$ для всех $A \neq S$.
- До исчерпания: $\forall B \rightarrow \beta$ и разбиений $\beta = \eta_1 A \eta_2$,
 $\text{FOLLOW}_k(A) =$
 $\text{FOLLOW}_k(A) \cup \text{first}_k(\text{FIRST}_k(\eta_2)\text{FOLLOW}_k(B))$, где
 first_k — это k первых символов строки.

Алгоритм также задаёт wfo на множестве конфигураций
 $\text{FOLLOW}_k(A_i)$.

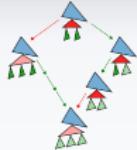
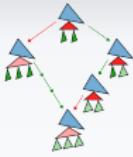


Таблица LL(k)-разбора

Таблица $T_k(A, x)$ по нетерминалу A и lookahead-символам x вычисляет правило для парсинга.

```
for all  $A \rightarrow \alpha$ 
  for all  $x \in \text{first}_k(\text{FIRST}_k(\alpha) \text{ FOLLOW}_k(A))$ 
    if  $T_k(A, x)$  не задано, тогда  $T_k(A, x) = A \rightarrow \alpha$ 
    else объявление о конфликте
```



PDA для LL(1)-грамматик

- Чтение терминалов с ленты происходит только в одном состоянии, при этом не меняется стек, но делается переход в другое состояние (учёт lookahead-ов).
- Во всех остальных состояниях только меняется стек. Переход ничего не читает с ленты и возвращает автомат в читающее состояние.

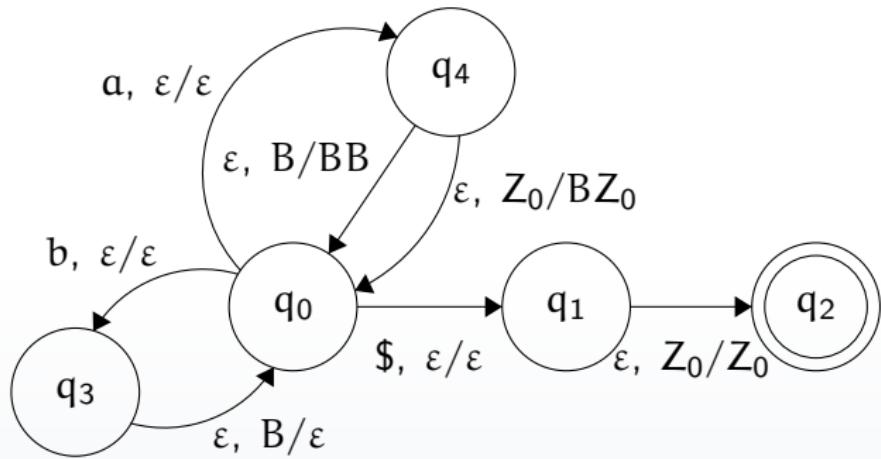
$$S \rightarrow aBS$$

$$S \rightarrow \epsilon$$

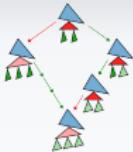
$$B \rightarrow aBB$$

$$B \rightarrow b$$

$S \rightarrow Z_0$ в
PDA («дно»).



Для LL(k) аналогично, но состояния будут соответствовать k последним прочитанным буквам.

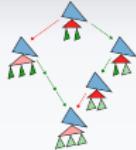


LL(k)-грамматики без ε -правил

Пример

Стандартный алгоритм удаления ε -правил приводит к разрушению LL-свойств:

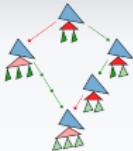
$$\begin{array}{lcl} S & \rightarrow & ABC \\ A & \rightarrow & aA \mid d \\ B & \rightarrow & b \mid \varepsilon \\ C & \rightarrow & c \mid \varepsilon \end{array}$$



LL(k)-грамматики без ε -правил

Утверждение (Куроки–Суонио)

Всякая LL(k)-грамматика может быть преобразована в LL($k+1$)-грамматику без ε -правил.



LL(k)-грамматики без ε -правил

Утверждение (Куроки–Суонио)

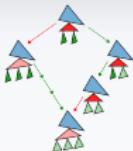
Всякая LL(k)-грамматика может быть преобразована в LL($k+1$)-грамматику без ε -правил.

Идея доказательства

Сначала избавимся от всех правил, начинающихся с nullable-нетерминала, путём введения notnull-двойников.

Затем присоединим все nullable-нетерминалы отрезки, стоящие в правых частях, к предшествующим им notnull-нетерминалам, и объявим полученные строки новыми нетерминалами.

В новых правилах nullable-кусок приписывается к самому последнему нетерминалу в правой части.

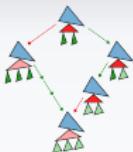


Пример устранения ε без разрушения LL-свойства

$$\begin{array}{ll} S \rightarrow ABC | Bk & A \rightarrow aA | d \\ B \rightarrow b | \varepsilon & C \rightarrow c | \varepsilon \end{array}$$

Сначала избавимся от обнуляемого нетерминала в начале правой части правила стандартным приёмом:

$$\begin{array}{ll} S \rightarrow ABC | B'k | k & A \rightarrow aA | d \\ B \rightarrow b | \varepsilon & C \rightarrow c | \varepsilon \quad B' \rightarrow b \end{array}$$



Пример устранения ε без разрушения LL-свойства

$$\begin{array}{ll} S \rightarrow ABC | B'k | k & A \rightarrow aA | d \\ B \rightarrow b | \varepsilon & C \rightarrow c | \varepsilon \quad B' \rightarrow b \end{array}$$

Теперь присоединим правый обнуляемый контекст к A и протащим его по правилу переписывания для A :

$$\begin{array}{ll} S \rightarrow [ABC] | B'k | k & [ABC] \rightarrow a[ABC] | [dBC] \\ B \rightarrow b | \varepsilon & C \rightarrow c | \varepsilon \quad B' \rightarrow b \end{array}$$

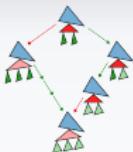


Пример устранения ε без разрушения LL-свойства

$$\begin{array}{ll} S \rightarrow [ABC] | B'k | k & [ABC] \rightarrow a[ABC] | [dBC] \\ B \rightarrow b | \varepsilon & C \rightarrow c | \varepsilon \quad B' \rightarrow b \end{array}$$

Определяем правила переписывания для нетерминала $[dBC]$, соответствующие коллапсу всего обнуляемого контекста, его префиксов, и непустой развёртке префикса.

$$\begin{array}{ll} S \rightarrow [ABC] | B'k | k & [ABC] \rightarrow a[ABC] | [dBC] \\ [dBC] \rightarrow d[bC] | dc | d & B \rightarrow b | \varepsilon \\ C \rightarrow c | \varepsilon & B' \rightarrow b \end{array}$$

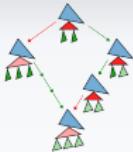


Пример устранения ε без разрушения LL-свойства

$$\begin{array}{ll} S \rightarrow [ABC] | B'k | k & [ABC] \rightarrow a[ABC] | [dBC] \\ [dBC] \rightarrow d[bC] | dc | d & B \rightarrow b | \varepsilon \\ C \rightarrow c | \varepsilon & B' \rightarrow b \end{array}$$

Аналогично обрабатываем нетерминал $[bC]$ и удаляем все правила для обнуляемых нетерминалов из грамматики.

$$\begin{array}{ll} S \rightarrow [ABC] | B'k | k & [ABC] \rightarrow a[ABC] | [dBC] \\ [dBC] \rightarrow d[bC] | dc | d & [bC] \rightarrow bc | b \\ & B' \rightarrow b \end{array}$$



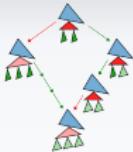
Лемма Розенкранца–Стирнса

Утверждение

Если G — $LL(k)$ -грамматика, тогда вышеописанный алгоритм устранения ϵ -правил всегда завершается, поскольку ни на одном шаге не появляется новых нетерминалов, дважды присоединяющих в правый контекст один и тот же обнуляемый нетерминал грамматики G .

Покажем, что в общем случае алгоритм может и не завершаться. Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, $LL(k)$ ни для какого значения k).

$$S \rightarrow SS \mid a \mid \epsilon$$



Лемма Розенкранца–Стирнса

Утверждение

Если G — $LL(k)$ -грамматика, тогда вышеописанный алгоритм устранения ϵ -правил всегда завершается.

Покажем, что в общем случае алгоритм может и не завершаться.

Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, $LL(k)$ ни для какого значения k).

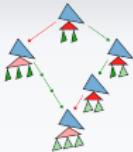
$$S \rightarrow SS \mid a \mid \epsilon$$

По алгоритму, устраним сначала обнуляемый правый нетерминал.

$$S \rightarrow S'S \mid S' \mid a \mid \epsilon \quad S' \rightarrow S'S \mid S' \mid a$$

Теперь присоединим обнуляемые контексты (это оставшиеся вхождения S) и посмотрим, какие правила получатся для нового нетерминала $[S'S]$.

$$S \rightarrow [S'S] \mid S' \mid a \mid \epsilon \quad S' \rightarrow [S'S] \mid S' \mid a \quad [S'S] \rightarrow S'SS \mid S'S \mid aS$$



Лемма Розенкранца–Стирнса

Покажем, что в общем случае алгоритм может и не завершаться.

Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, LL(k) ни для какого значения k).

$$S \rightarrow SS \mid a \mid \epsilon$$

По алгоритму, устраним сначала обнуляемый правый нетерминал.

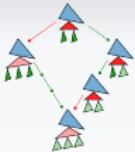
$$S \rightarrow S'S \mid S' \mid a \mid \epsilon \quad S' \rightarrow S'S \mid S' \mid a$$

Теперь присоединим обнуляемые контексты (это оставшиеся вхождения S) и посмотрим, какие правила получатся для нового нетерминала $[S'S]$.

$$S \rightarrow [S'S] \mid S' \mid a \mid \epsilon \quad S' \rightarrow [S'S] \mid S' \mid a \quad [S'S] \rightarrow S'SS \mid S'S \mid aS$$

Видно, что нужно порождать новый нетерминал, потому что обнуляемый контекст у правила $[S'S] \rightarrow S'SS$ — это уже два вхождения S . После его порождения опять получим правило вида $[S'SS] \rightarrow S'SSS$, и вообще, для каждого нового $[S'S^n]$ — правило $[S'S^n] \rightarrow S'S^{n+1}$.

Значит, ни на какой итерации процесс не сходится.



GNF для LL(k)-грамматики

Утверждения

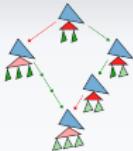
- Ни одна леворекурсивная грамматика — не LL(k) ни для какого k .
- Всякая LL(k)-грамматика может быть преобразована в LL($k+1$)-грамматику в GNF.



LL(k)-языки

Определение

CFL L — это LL(k)-язык, если для него существует LL(k)-грамматика.



LL(k)-языки

Определение

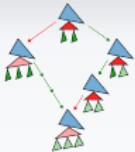
CFL L — это LL(k)-язык, если для него существует LL(k)-грамматика.

Существуют LL(k), но не LL(k-1)-языки.

Рассмотрим язык $\{a^n(b^k d \mid b \mid c)^n\}$. Это LL(k)-язык:

$$\begin{array}{lcl} S & \rightarrow & aCA \\ C & \rightarrow & aCA \mid \epsilon \\ A & \rightarrow & bB \mid c \\ B & \rightarrow & b^{k-1}d \mid \epsilon \end{array}$$

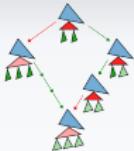
Неформально: не LL(k-1) потому, что иначе бы имелась LL(k)-грамматика в GNF для L. В стеке разбора для префикса a^n оказалось бы больше, чем $2k - 1$ символов. Подставляя варианты суффиксов к a^{n+k-1} , получаем противоречие (метод подмены).



Метод подмены

Общая схема метода

- Рассматриваем сразу LL(k)-грамматику в GNF.
- Показываем, что в её стеке в состоянии α должно находиться не меньше, чем $f(k)$ разных символов при предъявлении тех же самых lookahead k символов.
- После данных k символов предъявляем длинный суффикс, отрезок которого должен контекстно-свободно выводиться из некоторого нетерминала в стеке.
- Предъявляем другой суффикс к тому же префиксу и lookahead, в котором не может быть такого отрезка, и подменяем вывод нетерминала.



Техника метода подмены

- Ищем такие два слова xfy , xfz , что $|f| = k$ и при чтении префикса x стек мог неограниченно разрастись (т.е. есть синхронная накачка хотя бы одной из пар x и y и x и z). Предполагаем высоту стека равной хотя бы $k + t$ (где t выбирается в зависимости от задачи).
- k символов в стеке при выталкивании точно распознают фрагмент f (lookahead). Рассматриваем, что может распознаться остальными t символами. Комбинируем распознанные фрагменты и показываем, что их комбинация не входит в язык.
- Либо если в стеке осталось t символов, а длина, например, y меньше t , тогда с этим стеком точно нельзя распознать y .



Пример подмены

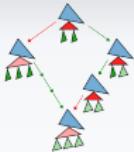
Может ли язык $\{a^n b^n\} \cup \{a^n c^n\}$ задаваться LL(k)-грамматикой для некоторого k ?

Пусть такое k нашлось (без ограничения общности — в GNF). Рассмотрим слово $a^{n+k}b^{n+k}$ и подберём такое n , что после чтения a^n LL-анализатор имеет в стеке не меньше $k+2$ символов. Пусть последний символ — это Y . Он распознает некоторое подслово суффикса b^{n+k} , а именно b^s . Теперь подменим строку на $a^{n+k}c^{n+k}$. В этом случае Y должен распознать некоторое c^t . Но значит, $a^{n+k}b^{n+k-s}c^t \in L$, что невозможно. □



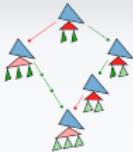
Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geq m\}$.



Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geq m\}$. Положим $w = a^{n+k} b^{n+k}$, где n таково, что в стеке на момент чтения a^n не меньше, чем $k+2$ нетерминала. Тогда при подмене w на a^{n+k+1} из стека достанется только k нетерминалов, и слово a^{n+k+1} распознаться не сможет.



Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geq m\}$.

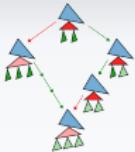
Положим $w = a^{n+k} b^{n+k}$, где n таково, что в стеке на момент чтения a^n не меньше, чем $k+2$ нетерминала. Тогда при подмене w на a^{n+k+1} из стека достанется только k нетерминалов, и слово a^{n+k+1} распознаться не сможет.

Следствие

Опциональный else не парсится с помощью LL-разбора.

«Плавающий else» делает грамматику неоднозначной.

Договорённость по приоритетам (связывание с ближайшим if) снимает неоднозначность, но по префиксу выражения всё ещё невозможно установить, какой if имеется в виду: короткий или длинный.



Свойства LL(k)-языков

Утверждение

- LL(k)-языки не замкнуты относительно объединения.
- LL(k)-языки не замкнуты относительно пересечения с регулярным языком (пример: $\{a^nba^n b\} \cup \{a^nc a^n c\}$ — не LL(k)-язык).
- Если L — LL(k)-нерегулярный язык, то его дополнение — не LL(k) ни для какого k .
- (Более сильное утверждение) Если $L_1 \cup L_2$ — регулярный язык, и L_1 — нерегулярный LL(k)-язык, то L_2 — не LL(k) ни для какого k .

Восходящий разбор. Неоднозначность и детерминированность

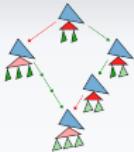


Теория формальных языков
2022 г.



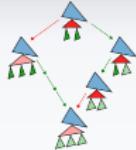
(Не)лирическое отступление





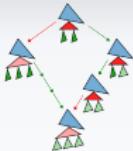
(Не)лирическое отступление

- Дополнительные задания не требуют больших объёмов кода, доступны всем (в том числе группам) и стоят минимум 2 балла. Следим за горячими сроками!
- Запись на доделывание закрывается после дедлайна.
- Планируем действия заранее:
 - 3 лабораторная по объёму большая — используем чужой код (с контеста, от прошлых цивилизаций), но не забываем про структуры (и Рефал-стайл).
 - 4 лабораторная будет сложной (и прошлые цивилизации помогут мало). Зато будет 2 допзадания!
 - 5 лабораторная будет легче, но выполняться в мини-группах. Допзадание будет одно.
 - На последней неделе(!!) будет биг-фарма: возможность добрать баллы по всем темам решением задач.



Неоднозначные КС-языки

Рассмотрим КС-язык $\{a^n b^m c^m\} \cup \{a^n b^n c^n\}$. Слова $a^n b^n c^n$ этого языка гарантированно имеют минимум два дерева разбора.

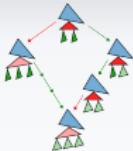


Неоднозначные КС-языки

Рассмотрим КС-язык $\{a^n b^m c^m\} \cup \{a^n b^n c^n\}$. Слова $a^n b^n c^n$ этого языка гарантированно имеют минимум два дерева разбора.

Определение

КС-грамматика G неоднозначная, если существует слово $w \in L(G)$ такое, что в G у него больше одного дерева разбора. КС-язык L существенно неоднозначен, если всякая его грамматика неоднозначна.



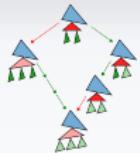
Неоднозначные КС-языки

Рассмотрим КС-язык $\{a^n b^m c^m\} \cup \{a^n b^n c^m\}$. Слова $a^n b^n c^n$ этого языка гарантированно имеют минимум два дерева разбора.

Определение

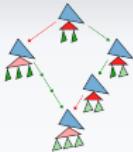
КС-грамматика G неоднозначная, если существует слово $w \in L(G)$ такое, что в G у него больше одного дерева разбора. КС-язык L существенно неоднозначен, если всякая его грамматика неоднозначна.

Существование однозначной грамматики не гарантирует существования DPDA: см. $\{a^n b^n\} \cup \{a^n b^{2n}\}$.



Алгоритм Касами–Тории

Алгоритм Кока–Янгера–Касами — таблица
 $T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$; изменим её на
 $T'_j[A] = \{i \mid a_{i+1} \dots a_j \in L_G(A)\}$.



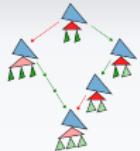
Алгоритм Касами–Тории

Алгоритм Кока–Янгера–Касами — таблица

$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$; изменим её на
 $T'_j[A] = \{i \mid a_{i+1} \dots a_j \in L_G(A)\}$.

Идея алгоритма СТ

Читаем очередной символ, вычисляем множество $T'_j[A]$ для всех $A \in N$, постепенно достраивая его как список, упорядоченный по возрастанию. Если $A \rightarrow BC$, тогда если $k \in T'_j[C]$, то для всех $x \in T'_k[B]$ выполнено $x \in T'_j[A]$.

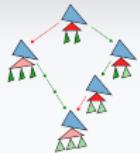


Алгоритм Касами–Тории

Алгоритм Кока–Янгера–Касами — таблица

$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$; изменим её на
 $T'_j[A] = \{i \mid a_{i+1} \dots a_j \in L_G(A)\}$.

```
forall j, A(T'_j[A] = ∅)
for j=1..n
    for all A ∈ N if A → a_j ∈ R then T'_j[A] = {j - 1}
    for k=j-1..1
        for all A → BC ∈ R
            if k ∈ T'_j[C] then for all i ∈ T'_k[B] T'_j[A] = T'_j[A] ∪ {i}
```



Алгоритм Касами–Тории

Алгоритм Кока–Янгера–Касами — таблица

$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$; изменим её на

$T'_j[A] = \{i \mid a_{i+1} \dots a_j \in L_G(A)\}$.

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n$

 for all $A \in N$ if $A \rightarrow a_j \in R$ then $T'_j[A] = \{j - 1\}$

 for $k=j-1..1$

 for all $A \rightarrow BC \in R$

 if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

Для однозначных грамматик ACT работает за $O(n^2)$.

$$S \rightarrow G_A A | G_B B | a | b \quad A \rightarrow a | SG_A \quad B \rightarrow b | SG_B$$

$$G_A \rightarrow a \quad G_B \rightarrow b \quad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n$

for $k=j-1..1$

for all $A \rightarrow BC \in R$

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

Инициализация таблицы:

	S	A	B	G_A	G_B
1 – a					
2 – b					
3 – b					
4 – a					

$$S \rightarrow G_A A \mid G_B B \mid a \mid b \quad A \rightarrow a \mid SG_A \quad B \rightarrow b \mid SG_B$$

$$G_A \rightarrow a \quad G_B \rightarrow b \quad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j=1$

for all $A \in N$ if $A \rightarrow a_j \in R$ then $T'_j[A] = \{j - 1\}$

for $k=j-1..1$

for all $A \rightarrow BC \in R$

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$j = 1$, проход по терминальным правилам, цикла по нетерминальным нет, т.к. $k = 0$:

	S	A	B	G_A	G_B
1 – a	0	0		0	
2 – b					
3 – b					
4 – a					

$$\begin{array}{lll}
 S \rightarrow G_A A \mid G_B B \mid a \mid b & A \rightarrow a \mid SG_A & B \rightarrow b \mid SG_B \\
 G_A \rightarrow a & G_B \rightarrow b & \text{слово } abba
 \end{array}$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 2$

for all $A \in N$ if $A \rightarrow a_j \in R$ then $T'_j[A] = \{j - 1\}$

for $k=j-1..1$

for all $A \rightarrow BC \in R$

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$j = 2$, проход по терминальным правилам:

	S	A	B	G_A	G_B
1 – a	0	0		0	
2 – b	1		1		1
3 – b					
4 – a					

$$S \rightarrow G_A A \mid G_B B \mid a \mid b \quad A \rightarrow a \mid SG_A \quad B \rightarrow b \mid SG_B$$

$$G_A \rightarrow a \quad G_B \rightarrow b \quad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 2$

for $k=j-1..1 \backslash\backslash k = 1$

for all $A \rightarrow BC \in R \backslash\backslash$ если второй нетерминал правой части есть в строке 2 с индексом 1

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$\backslash\backslash$ тогда добавляем в ячейку второй строки для левого нетерминала содержимое ячейки первого нетерминала в строке 1

Подходящие правила есть для B и G_B . Но для нетерминала G_B (правило $S \rightarrow G_B B$) ячейка в первой строке пуста, так что добавляем только содержимое ячейки для S в ячейку B .

	S	A	B	G_A	G_B
1 – a	0	0		0	
2 – b	1		1, 0		1
3 – b					
4 – a					

$$S \rightarrow G_A A \mid G_B B \mid a \mid b \quad A \rightarrow a \mid SG_A \quad B \rightarrow b \mid SG_B$$

$$G_A \rightarrow a \quad G_B \rightarrow b \quad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j=3$

for all $A \in N$ if $A \rightarrow a_j \in R$ then $T'_j[A] = \{j - 1\}$

for $k=j-1..1$

for all $A \rightarrow BC \in R$

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$j = 3$, проход по терминальным правилам:

	S	A	B	G_A	G_B
$1 - a$	0	0		0	
$2 - b$	1		1, 0		1
$3 - b$	2		2		2
$4 - a$					

$$\begin{array}{l}
 S \rightarrow G_A A \mid G_B B \mid a \mid b \\
 G_A \rightarrow a \\
 G_B \rightarrow b
 \end{array}
 \quad
 \begin{array}{l}
 A \rightarrow a \mid SG_A \\
 B \rightarrow b \mid SG_B
 \end{array}
 \quad
 \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 3$

for $k=j-1..1 \backslash\backslash k = 2$

for all $A \rightarrow BC \in R \backslash\backslash$ если второй нетерминал правой части есть в строке 3 с индексом 2

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$\backslash\backslash$ тогда добавляем в ячейку третьей строки для левого нетерминала содержимое ячейки первого нетерминала в строке 2

Подходящие правила: $S \rightarrow G_B B$, $B \rightarrow SG_B$, причём во второй строке ячейки G_B и S обе не пустые. Добавляем их содержимое в ячейки левых частей, S и B :

	S	A	B	G_A	G_B
$1 - a$	0	0		0	
$2 - b$	1		1, 0		1
$3 - b$	2, 1		2, 1		2
$4 - a$					

$$\begin{array}{l}
 S \rightarrow G_A A \mid G_B B \mid a \mid b \\
 G_A \rightarrow a \\
 G_B \rightarrow b
 \end{array}
 \quad
 \begin{array}{l}
 A \rightarrow a \mid SG_A \\
 B \rightarrow b \mid SG_B
 \end{array}
 \quad
 \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 3$

for $k=j-1..1 \backslash\backslash k = 1$

for all $A \rightarrow BC \in R \backslash\backslash$ если второй нетерминал правой части есть в строке 3 с индексом 1

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$\backslash\backslash$ тогда добавляем в ячейку третьей строки для левого нетерминала содержимое ячейки первого нетерминала в строке 1

1 в третьей строке есть у нетерминалов S и B, но первый никогда не бывает вторым в правой части. Остается правило $S \rightarrow G_B B$, оно также ничего не даёт, т.к. в первой строке ячейка G_B пуста.

	S	A	B	G_A	G_B
1 – a	0	0		0	
2 – b	1		1, 0		1
3 – b	2, 1		2, 1		2
4 – a					

$$S \rightarrow G_A A \mid G_B B \mid a \mid b \quad A \rightarrow a \mid SG_A \quad B \rightarrow b \mid SG_B$$

$$G_A \rightarrow a \qquad \qquad G_B \rightarrow b \qquad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 4$

for all $A \in N$ if $A \rightarrow a_j \in R$ then $T'_j[A] = \{j - 1\}$

for $k=j-1..1$

for all $A \rightarrow BC \in R$

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$j = 4$, проход по терминальным правилам:

	S	A	B	G_A	G_B
$1 - a$	0	0		0	
$2 - b$	1		1, 0		1
$3 - b$	2, 1		2, 1		2
$4 - a$	3	3		3	

$$\begin{array}{lll} \mathbf{S} \rightarrow \mathbf{G_A A} | \mathbf{G_B B} | \mathbf{a} | \mathbf{b} & \mathbf{A} \rightarrow \mathbf{a} | \mathbf{SG_A} & \mathbf{B} \rightarrow \mathbf{b} | \mathbf{SG_B} \\ \mathbf{G_A} \rightarrow \mathbf{a} & \mathbf{G_B} \rightarrow \mathbf{b} & \text{слово abba} \end{array}$$

$$\forall j, A(T'_j[A] = \emptyset)$$

for j=1..n \j=4

```
for k=j-1..1 \& k = 3
```

for all $A \rightarrow BC \in R$ \\ если второй нетерминал правой части есть в строке 4 с индексом 3

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

\\\ тогда добавляем в ячейку 4-ой строки для левого нетерминала содержимое ячейки первого нетерминала в строке 3

Подходящие правила: $S \rightarrow G_A A$, $A \rightarrow SG_A$, однако ячейка G_A в третьей строке пуста. Добавляем содержимое ячейки S в ячейку A в четвёртой строке.

	S	A	B	G_A	G_B
1 - a	0	0		0	
2 - b	1		1, 0		1
3 - b	2, 1		2, 1		2
4 - a	3	3, 2, 1		3	

$$\begin{array}{l}
 S \rightarrow G_A A \mid G_B B \mid a \mid b \\
 G_A \rightarrow a \\
 \quad\quad\quad A \rightarrow a \mid SG_A \\
 G_B \rightarrow b \\
 \quad\quad\quad B \rightarrow b \mid SG_B \\
 \quad\quad\quad \text{слово } abba
 \end{array}$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 4$

for $k=j-1..1 \backslash\backslash k = 2$

for all $A \rightarrow BC \in R \backslash\backslash$ если второй нетерминал правой части есть в строке 4 с индексом 2

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

$\backslash\backslash$ тогда добавляем в ячейку 4-ой строки для левого нетерминала содержимое ячейки первого нетерминала в строке 2

Подходящее правило: $S \rightarrow G_A A$, однако ячейка G_A во второй строке пуста. На этом шаге таблица не меняется.

	S	A	B	G_A	G_B
1 – a	0	0		0	
2 – b	1		1, 0		1
3 – b	2, 1		2, 1		2
4 – a	3	3, 2, 1		3	

$$S \rightarrow G_A A \mid G_B B \mid a \mid b \quad A \rightarrow a \mid SG_A \quad B \rightarrow b \mid SG_B$$

$$G_A \rightarrow a \quad G_B \rightarrow b \quad \text{слово } abba$$

$\forall j, A(T'_j[A] = \emptyset)$

for $j=1..n \backslash\backslash j = 4$

for $k=j-1..1 \backslash\backslash k = 1$

for all $A \rightarrow BC \in R \quad \backslash\backslash$ если второй нетерминал

правой части есть в строке 4 с индексом 1

if $k \in T'_j[C]$ then for all $i \in T'_k[B]$ $T'_j[A] = T'_j[A] \cup \{i\}$

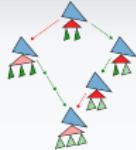
$\backslash\backslash$ тогда добавляем в ячейку 4-ой строки для левого нетерминала содержимое ячейки первого нетерминала в строке 1

Подходящее правило опять

$S \rightarrow G_A A$, и на сей раз нужная ячейка G_A не пуста.

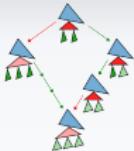
To, что теперь $0 \in T'_4[S]$, показывает, что $abba \in L(G)$, поскольку $T'_4[S] = \{i \mid a_{i+1} \dots a_4 \in L_G(S)\}$, где $a_1 a_2 a_3 a_4 = abba$.

	S	A	B	G_A	G_B
$1 - a$	0	0		0	
$2 - b$	1		1, 0		1
$3 - b$	2, 1		2, 1		2
$4 - a$	3, 0	3, 2, 1		3	



Детерминированные КС-языки

Язык L обладает префикс-свойством (prefix-free), если
 $\forall w (w \in L \Rightarrow \forall v (v \neq \epsilon \Rightarrow wv \notin L))$.

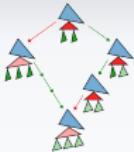


Детерминированные КС-языки

Язык L обладает префикс-свойством (prefix-free), если $\forall w (w \in L \Rightarrow \forall v (v \neq \epsilon \Rightarrow wv \notin L))$.

Детерминированные языки с префикс-свойством — языки, распознаваемые DPDA с допуском по пустому стеку.

Рассмотрим язык a^+ . Предположим, он распознаётся DPDA с допуском по пустому стеку. Тогда на элементе a стек уже обязательно пуст. А значит, работа DPDA не может быть продолжена, и элемент aa не может быть им распознан.

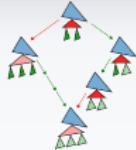


Детерминированные КС-языки

Язык L обладает префикс-свойством (prefix-free), если $\forall w (w \in L \Rightarrow \forall v (v \neq \epsilon \Rightarrow wv \notin L))$.

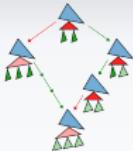
Детерминированные языки с префикс-свойством — языки, распознаваемые DPDA с допуском по пустому стеку.

Рассмотрим язык L , $w_1, w_1w_2 \in L$, $w_2 \neq \epsilon$. Предположим, он распознаётся DPDA с допуском по пустому стеку. Тогда на элементе w_1 стек уже обязательно пуст. А значит, работа DPDA не может быть продолжена, и элемент w_1w_2 не может быть им распознан.



Эндмаркеры

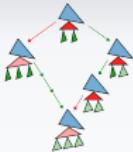
Рассмотрим язык $a^+ \$$ (алфавит терминалов $\Sigma = \{a, \$\}$). В этом языке ни одно слово не является префиксом другого.



Эндмаркеры

Рассмотрим язык $\{w\$ \mid w \in L\}$ (алфавит терминалов $\Sigma = \Sigma_L \cup \{\$\}$, $\$ \notin \Sigma_L$). Независимо от L , в этом языке ни одно слово не является префиксом другого.

- Хорошие новости: любой детерминированный КС-язык легко преобразовать в язык, распознаваемый DPDA с допуском по пустому стеку.



Эндмаркеры

Рассмотрим язык $\{w\$ \mid w \in L\}$ (алфавит терминалов $\Sigma = \Sigma_L \cup \{\$\}$, $\$ \notin \Sigma_L$). Независимо от L , в этом языке ни одно слово не является префиксом другого.

- Хорошие новости: любой детерминированный КС-язык легко преобразовать в язык, распознаваемый DPDA с допуском по пустому стеку.
- Плохие новости: существенно неоднозначные контекстно-свободные языки с префикс-свойством. Стандартный пример: $\{a^n b^n c^m d\} \cup \{a^m b^n c^n d\}$.



Языки нередуцируемых префиксов

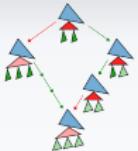
Определим понятие свёртки — перехода справа налево в правиле переписывания $A \rightarrow \alpha$. Что можно сказать о всех возможных префиксах сентенциальных форм, порождаемых грамматикой G , к которым нельзя применить ни одну свёртку?



Языки нередуцируемых префиксов

Определим понятие свёртки — перехода справа налево в правиле переписывания $A \rightarrow \alpha$. Что можно сказать о всех возможных префиксах сентенциальных форм, порождаемых грамматикой G , к которым нельзя применить ни одну свёртку?

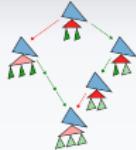
Такие с.ф. образуют регулярный язык. Идея обоснования: в распознающем их PDA из стека ничего не читается, т.е. PDA учитывает только символы сент. формы и свои состояния.



Автомат нередуцируемых префиксов

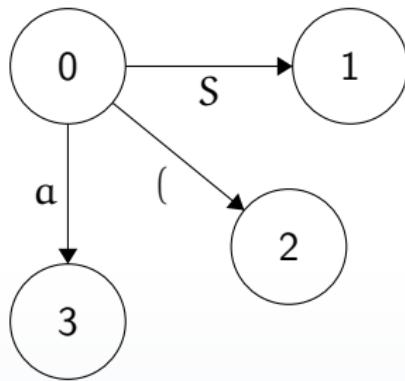
Описание конструкции

- Отмеченная позиция в правиле: \bullet . В правиле с правой частью $\xi_1 \dots \xi_n$ есть $n + 1$ таких позиций.
- Правило $A \rightarrow \alpha \bullet B\beta$ и правило $B \rightarrow \bullet\gamma$ — одно и то же множество переходов по символу, не приводящих к редукции \Rightarrow в одном состоянии.
- При чтении элемента правой части сдвигаем \bullet вправо на позицию.



Автомат нередуцируемых префиксов

$$S' \rightarrow S \quad S \rightarrow a \quad S \rightarrow (S)$$

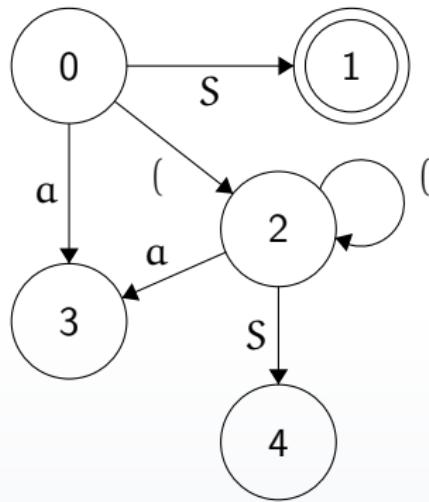


0	$S' \rightarrow \bullet S$
	$S \rightarrow \bullet(S)$
	$S \rightarrow \bullet a$
<hr/>	<hr/>
1	$S' \rightarrow S\bullet$
<hr/>	<hr/>
2	$S \rightarrow (\bullet S)$
<hr/>	<hr/>
3	$S \rightarrow a\bullet$

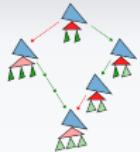


Автомат нередуцируемых префиксов

$$S' \rightarrow S \quad S \rightarrow a \quad S \rightarrow (S)$$

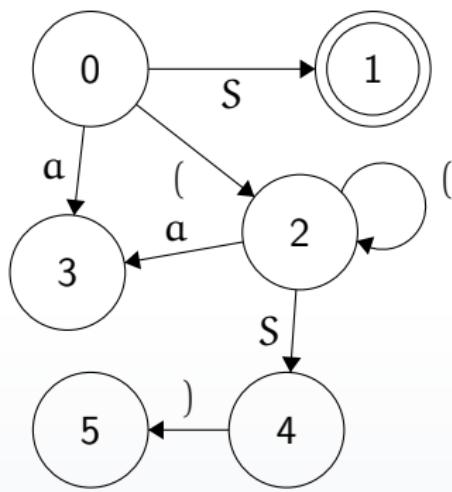


0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$
1	$S' \rightarrow S\bullet$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$
3	$S \rightarrow a\bullet$
4	$S \rightarrow (S\bullet)$

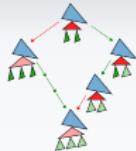


Автомат нередуцируемых префиксов

$$S' \rightarrow S \quad S \rightarrow a \quad S \rightarrow (S)$$



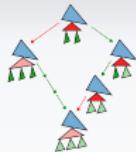
0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$
1	$S' \rightarrow S\bullet$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$
3	$S \rightarrow a\bullet$
4	$S \rightarrow (S\bullet)$
5	$S \rightarrow (S)\bullet$



Типы состояний автомата

- ① Финальное (свёртка в S').
- ② Не финальное, но свёртка.
- ③ Сдвиг по символу сентенциальной формы.

Что хранить в стеке PDA, построенного по такому
автомату?

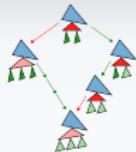


Типы состояний автомата

- ❶ Финальное (свёртка в S').
- ❷ Не финальное, но свёртка.
- ❸ Сдвиг по символу сентенциальной формы.

Что хранить в стеке PDA, построенного по такому
автомату?

- Хранить сами сентенциальные формы плохо —
проблема с извлечением нескольких подряд символов.

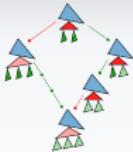


Типы состояний автомата

- ❶ Финальное (свёртка в S').
- ❷ Не финальное, но свёртка.
- ❸ Сдвиг по символу сентенциальной формы.

Что хранить в стеке PDA, построенного по такому
автомату?

- Хранить сами сентенциальные формы плохо —
проблема с извлечением нескольких подряд символов.
- Логично хранить последовательности последних
символов с.ф., которые могут привести к разным
свёрткам, закодированными одним символом стека.

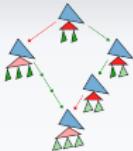


Типы состояний автомата

- ❶ Финальное (свёртка в S').
- ❷ Не финальное, но свёртка.
- ❸ Сдвиг по символу сентенциальной формы.

Что хранить в стеке PDA, построенного по такому
автомату?

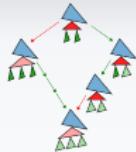
- Хранить сами сентенциальные формы плохо —
проблема с извлечением нескольких подряд символов.
- Логично хранить последовательности последних
символов с.ф., которые могут привести к разным
свёрткам, закодированными одним символом стека.
- А это — в точности состояния автомата.



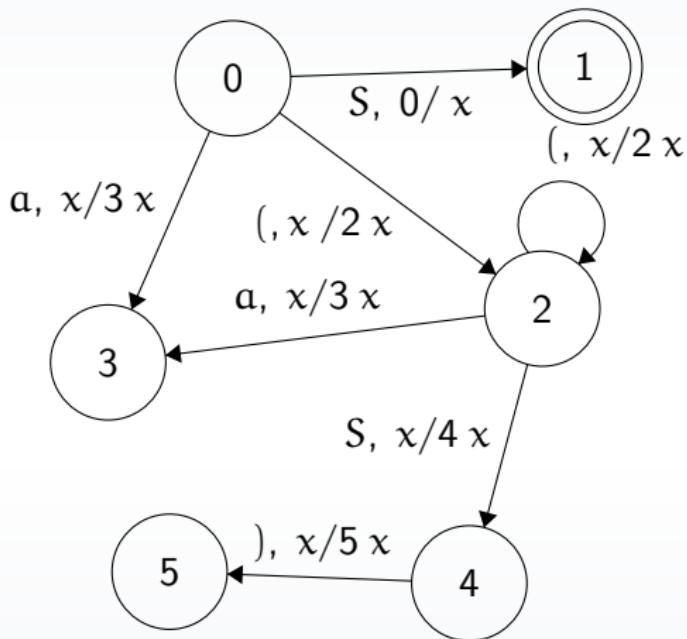
PDA по LR(0)-автомату

Общая конструкция

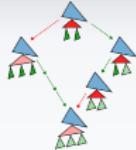
- При каждом сдвиге кладём в стек номер состояния, в которое приходим в конечном автомате.
- При каждой свёртке извлекаем из стека n символов, где n — длина правой части β правила $A \rightarrow \beta$, после чего переходим в состояние с номером $n + 1$ -ого символа в стеке, подразумевая на ленте символ A .
- Совершаем переход по символу A из полученного состояния (этот шаг мы на графике объединили с предыдущим).



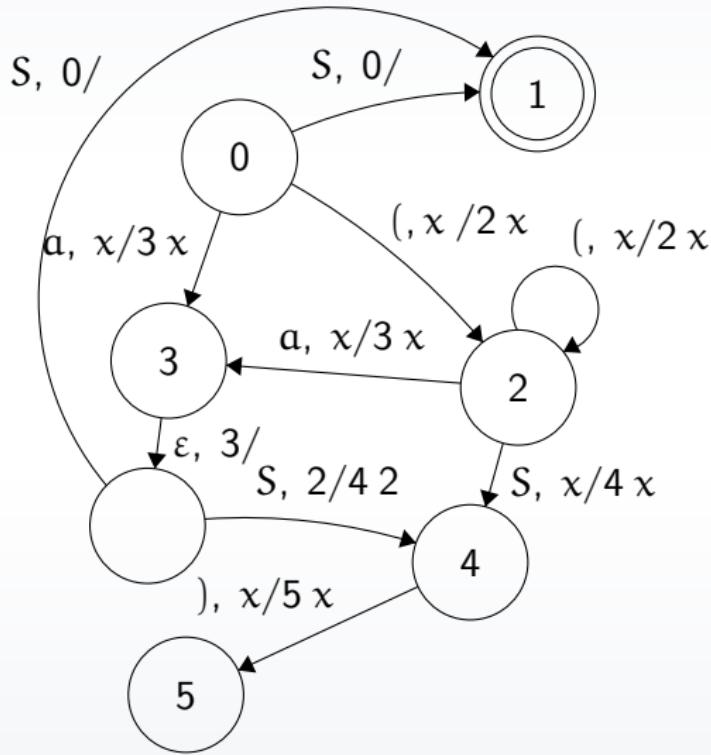
Пример построения PDA



0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S \bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a \bullet$	$a \rightarrow S$
4	$S \rightarrow (S \bullet)$	
5	$S \rightarrow (S) \bullet$	$(S) \rightarrow S$



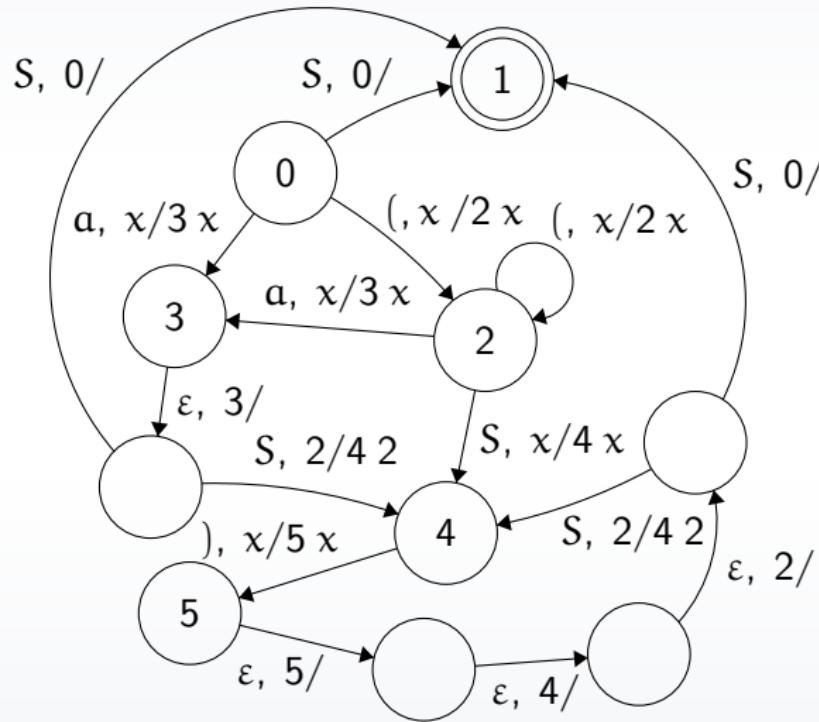
Пример построения PDA



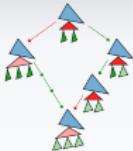
0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S\bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a\bullet$	$a \rightarrow S$
4	$S \rightarrow (S\bullet)$	
5	$S \rightarrow (S)\bullet$	$(S) \rightarrow S$



Пример построения PDA



0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S\bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a\bullet$	$a \rightarrow S$
4	$S \rightarrow (S\bullet)$	
5	$S \rightarrow (S)\bullet$	$(S) \rightarrow S$

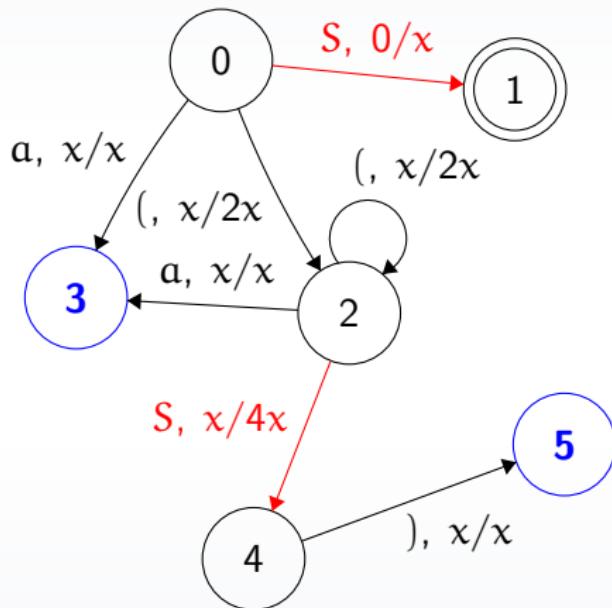


Промежуточный PDA-распознаватель

- Стековые символы, ведущие в состояния свёртки, не являющиеся финальными (у нас это 3 и 5), бесполезны, потому что сразу же безальтернативно извлекаются из стека.
- Распознаватель ещё не может быть использован как парсер, потому что он «читает» нетерминалы с ленты. Этого можно избежать, если принять, что нетерминал обязан быть считанным сразу после свёртки, и объединить свёртку (порождение нетерминала) и его считывание в один ϵ -переход.
- После добавления таких ϵ -переходов исходные переходы по нетерминалам можно удалять.



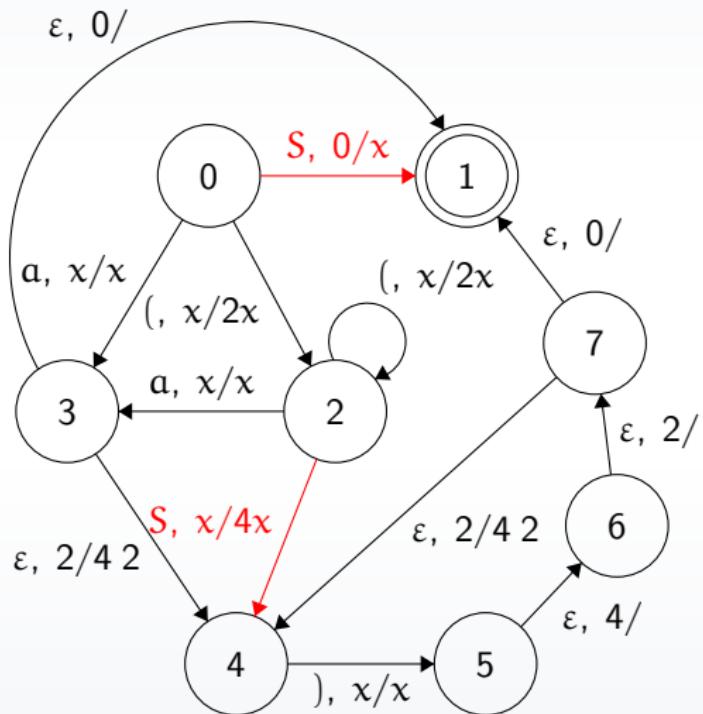
Избавление от переходов по нетерминалам



0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S\bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a\bullet$	$a \rightarrow S$
4	$S \rightarrow (S\bullet)$	
5	$S \rightarrow (S)\bullet$	$(S) \rightarrow S$



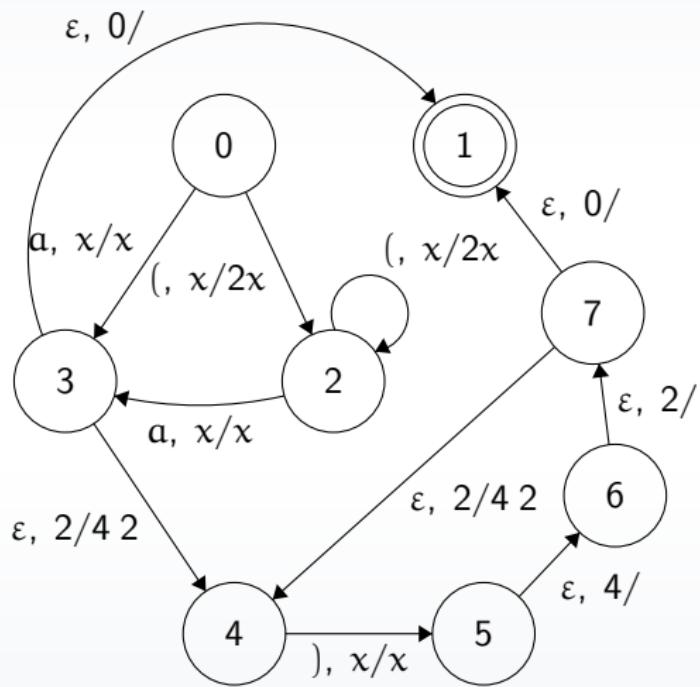
Избавление от переходов по нетермина- лам



0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S\bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a\bullet$	$a \rightarrow S$
4	$S \rightarrow (S\bullet)$	
5	$S \rightarrow (S)\bullet$	$(S) \rightarrow S$



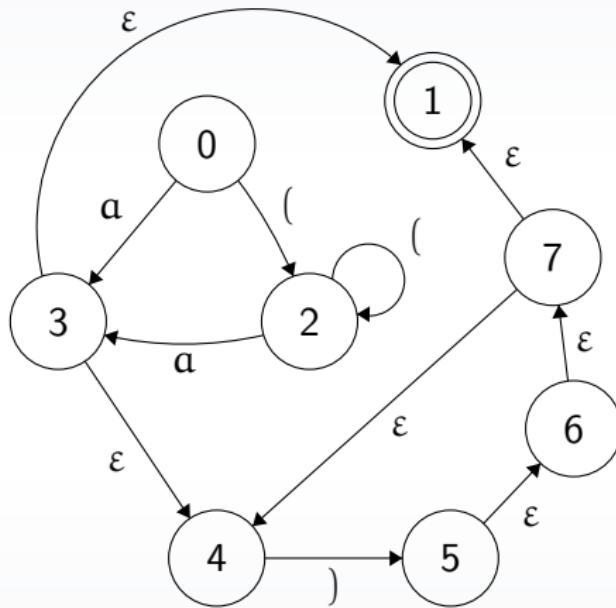
Избавление от переходов по нетермина- лам



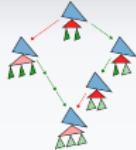
0	$S' \rightarrow \bullet S$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
1	$S' \rightarrow S \bullet$	$S \rightarrow S'$
2	$S \rightarrow (\bullet S)$ $S \rightarrow \bullet(S)$ $S \rightarrow \bullet a$	
3	$S \rightarrow a \bullet$	$a \rightarrow S$
4	$S \rightarrow (S \bullet)$	
5	$S \rightarrow (S) \bullet$	$(S) \rightarrow S$



Бонус — регулярная аппроксимация

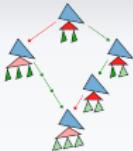


Аппроксимацией исходного языка $(^n a)^n$, построенной по LR(0)-автомату (Pereira–Wright), является язык $(* a)^*$.



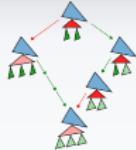
PDA или DPDA?

- Если есть ϵ -переходы, то нет никаких других.
- Если есть ϵ -переход, то он единственный из данного состояния.



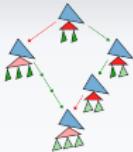
PDA или DPDA?

- Если есть ϵ -переходы, то нет никаких других. Если делается свёртка, то нельзя сделать сдвиг.
- Если есть ϵ -переход, то он единственный из данного состояния. Если делается свёртка одного типа, то нельзя сделать свёртку другого типа.
- Допуск — по пустому стеку \Rightarrow DPDA для языков с префикс-свойством.
- DPDA с допуском по пустому стеку распознают те же языки, что и LR(0)-разбор.
- В конструкции LR(0)-автомата часто навязывается эндмаркер \Rightarrow изначальная грамматика может описывать не LR(0)-язык!



Отказ от эндмаркера и SLR

- Используем ту же конструкцию автомата.
- Разрешим при возможности сделать свёртку вида $\beta \rightarrow A$ заглянуть в множество $\text{FOLLOW}(A)$, чтобы понять, какую свёртку делать (и делать ли).

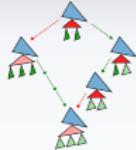


Отказ от эндмаркера и SLR

- Используем ту же конструкцию автомата.
- Разрешим при возможности сделать свёртку вида $\beta \rightarrow A$ заглянуть в множество $\text{FOLLOW}(A)$, чтобы понять, какую свёртку делать (и делать ли).

$$\begin{array}{lll} S' \rightarrow E & E \rightarrow E + T & E \rightarrow T \\ E \rightarrow V = E & T \rightarrow (E) & T \rightarrow \text{id} \\ & V \rightarrow \text{id} & \end{array}$$

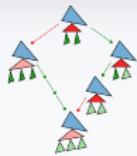
Здесь есть конфликт свёрток для S' (по $V \rightarrow \text{id}\bullet$ и $T \rightarrow \text{id}\bullet$), но $\text{FOLLOW}_1(V) \cap \text{FOLLOW}_1(T) = \emptyset \Rightarrow$ эта грамматика — SLR(1).



Коллапс линейных парсеров

Теорема

Для всякого языка из класса DCFL существует распознающая его SLR(1)-грамматика.



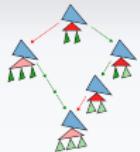
Теоретический коллапс линейных парсеров

Теорема

Для всякого языка из класса DCFL существует распознающая его SLR(1)-грамматика.

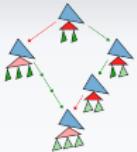
Следует из теоремы:

Для всякого языка из класса DCFL существует распознающая его LR(k)-грамматика.



LR(k)-распознаватели

Грамматика G — $LR(k)$, тогда и только тогда, когда для всех пар сентенциальных форм xy , xy' , порождаемых правосторонним разбором, где $y, y' \in \Sigma^+$, таких что xy допускает правую свёртку в префикс y по правилу ξ_1 , а xy' — свёртку где угодно по правилу ξ_2 , и первые k символов y и y' совпадают, $\xi_1 = \xi_2$.

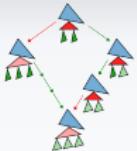


LR(k)-распознаватели

Грамматика G — LR(k), тогда и только тогда, когда для всех пар сентенциальных форм xy, xy' , порождаемых правосторонним разбором, где $y, y' \in \Sigma^+$, таких что xy допускает правую свёртку в префикс y по правилу ξ_1 , а xy' — свёртку где угодно по правилу ξ_2 , и первые k символов y и y' совпадают, $\xi_1 = \xi_2$.

$$\begin{array}{lll} S' \rightarrow S & S \rightarrow L = R; & S \rightarrow R; \\ L \rightarrow id & L \rightarrow *R & R \rightarrow L \end{array}$$

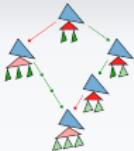
Поскольку $= \in FOLLOW_1(R)$, возникает конфликт вида сдвиг–свёртка при попытке анализа с.ф. L . Но lookahead у L , порождённой посредством $S \rightarrow L = R$, и посредством $S \rightarrow R; \rightarrow L;$, будет разный.



LR(k)-распознаватели

Грамматика G — $LR(k)$, тогда и только тогда, когда для всех пар сентенциальных форм xu , xu' , порождаемых правосторонним разбором, где $u, u' \in \Sigma^+$, таких что xu допускает правую свёртку в префикс u по правилу ξ_1 , а xu' — свёртку где угодно по правилу ξ_2 , и первые k символов u и u' совпадают, $\xi_1 = \xi_2$.

Любая $LR(k)$ -грамматика по определению гарантирует однозначный разбор при определённой длине lookahead-строки, поэтому ни одна грамматика с неоднозначным разбором не является $LR(k)$ ни для какого значения k .

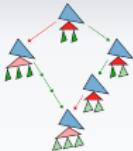


LR(k) \rightarrow LR(1), Mickunas–Lancaster–Shneider

$$\begin{array}{lll} S' \rightarrow S & S \rightarrow Abb & S \rightarrow Bbc \\ A \rightarrow aA & A \rightarrow a & B \rightarrow aB \\ & & B \rightarrow a \end{array}$$

Не LR(1), из-за свёрток $A \rightarrow a$, $B \rightarrow a$. Используем трансформацию присоединения правого контекста:

$$\begin{array}{lll} S' \rightarrow S & S \rightarrow [Ab]b & S \rightarrow [Bb]c \\ [Ab] \rightarrow a[Ab] & [Ab] \rightarrow ab & [Bb] \rightarrow a[Bb] \\ & [Bb] \rightarrow ab & \end{array}$$



$\text{LR}(k) \rightarrow \text{LR}(1)$, Mickunas–Lancaster–Shneider

$$\begin{array}{l} S' \rightarrow S \\ S \rightarrow bSS \\ S \rightarrow a \\ S \rightarrow aac \end{array}$$

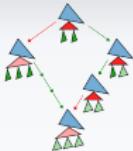
Не $\text{LR}(1)$, конфликт свёртки на префикссе ba с контекстом a .

Используем трансформацию уточнения правого контекста:

$$\begin{array}{llll} S \rightarrow bSa[a/S] & S \rightarrow bSb[b/S] & S \rightarrow a & S \rightarrow aac \\ [a/S] \rightarrow \epsilon & [a/S] \rightarrow ac & [b/S] \rightarrow Sa[a/S] & [b/S] \rightarrow Sb[b/S] \end{array}$$

Теперь присоединим правые контексты:

$S \rightarrow b[Sa][a/S]$	$ b[Sb][b/S]$	$ a$	$ aac$
$[Sa] \rightarrow b[Sa][[a/S]a]$	$ b[Sb][[b/S]a]$	$ aa$	$ aaca$
$[Sb] \rightarrow b[Sa][[a/S]b]$	$ b[Sb][[b/S]b]$	$ ab$	$ aacb$
$[a/S] \rightarrow \epsilon$	$ ac$		
$[[a/S]a] \rightarrow a$	$ aca$		
$[[a/S]b] \rightarrow b$	$ acb$		
$[[b/S]a] \rightarrow [Sa][[a/S]a]$	$ [Sb][[b/S]a]$		
$[[b/S]b] \rightarrow [Sa][[a/S]b]$	$ [Sb][[b/S]b]$		



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

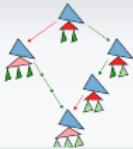
Видно, что если язык L распознаётся DPDA (т.е. является LR(1)-языком), то он также является LR(0)-языком, поскольку удовлетворяет префикс-свойству. Действительно, любое слово этого языка содержит единственную букву c , причём она расположена точно в середине слова.

Построим пробную КС-грамматику для языка L :

$$S \rightarrow aSb \mid aCa \mid bCb \mid c$$

$$C \rightarrow aCa \mid bCb \mid c$$

Проверим, является ли она LR(0)-грамматикой. Для этого построим LR(0)-автомат и проанализируем его на конфликты.



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

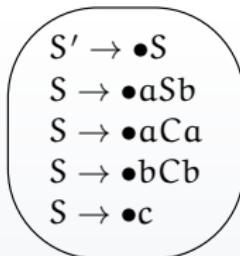
Пробная грамматика для L :

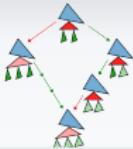
$$\begin{array}{rcl} S & \rightarrow & aSb \mid aCa \mid bCb \mid c \\ C & \rightarrow & aCa \mid bCb \mid c \end{array}$$

Начинаем строить LR(0)-автомат. Для этого вводим новое стартовое состояние S' (состояние окончательной свёртки) и начинаем разбор правила $S' \rightarrow \bullet S$.

Поскольку отмеченная позиция в правиле находится перед нетерминалом S , добавляем в состояние все ситуации вида $S \rightarrow \bullet \alpha$.

Переходы по нетерминалу S и терминалу c ведут к бесконфликтным свёрткам, поэтому малоинтересны. Разберёмся с переходом по a .





Применение MLS-подгонки

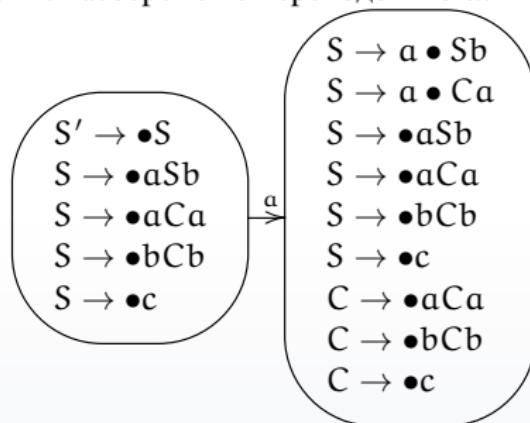
Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

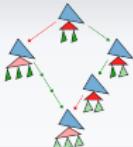
Пробная грамматика для L :

S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Переходы по нетерминалу S и терминалу c ведут к бесконфликтным свёрткам, поэтому малоинтересны. Разберёмся с переходом по a .



Похоже, что есть потенциальный конфликт (даже два) по свёрткам в S и C . Построим конфликтное состояние явно.



Применение MLS-подгонки

Исследовать на детерминированность язык

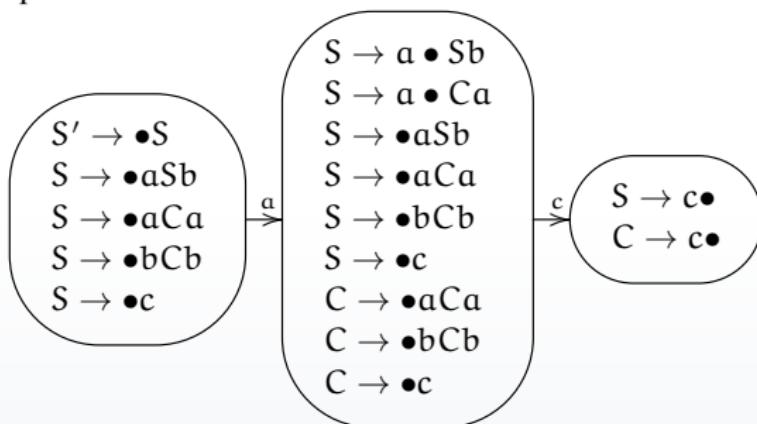
$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

Пробная грамматика для L :

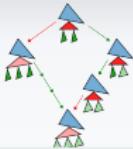
S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Похоже, что есть потенциальный конфликт (даже два) по свёрткам в S и C .

Построим конфликтное состояние явно.



Присоединим к конфликтующим S и C -нетерминалам их правые контексты.



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

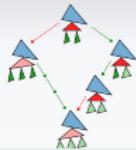
Пробная грамматика для L :

S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Грамматика для L после присоединения правых контекстов к нетерминалам S и C методом MLS (новые нетерминалы выделены красным):

S	\rightarrow	$a[Sb] \mid a[Ca] \mid b[Cb] \mid c$
$[Sb]$	\rightarrow	$a[Sb]b \mid a[Ca]b \mid b[Cb]b \mid cb$
$[Ca]$	\rightarrow	$a[Ca]a \mid b[Cb]a \mid ca$
$[Cb]$	\rightarrow	$a[Ca]b \mid b[Cb]b \mid cb$

Можно построить LR(0)-автомат для этой грамматики и убедиться, что он не содержит конфликтов. Значит язык L — детерминированный (более того, LR(0)).

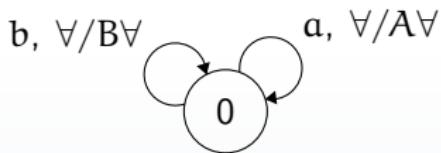


Другой подход к анализу КС-языков

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

Можно сразу попробовать построить DPDA для L . Заметим, что до прочтения буквы с стек обязательно заполняется (иначе потеряется информация либо о структуре палиндрома, либо о количестве букв a в начале слова), причём, поскольку неизвестно, когда именно префикс a^n переходит в палиндром, придётся запоминать, какие конкретные буквы были прочитаны: считаем, что символ стека A соответствует a , символ B — терминалу b .



Для экономии места символ \emptyset использован в роли параметра, пробегающего значения A , B и Z_0 : на детерминированность это не влияет, поскольку переходы с его участием делаются по разным терминалам.

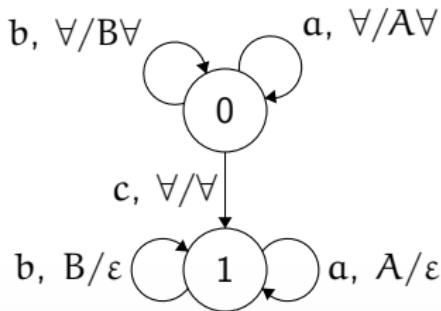


Другой подход к анализу КС-языков

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

После прочтения буквы с стек только опустошается: структура оставшейся части слова определяется уже прочитанной его частью.



Единственная тонкость — это переход от чтения w^R к чтению b^n . Он происходит, если на вершине стека лежит A , а читается буква b , и это не приводит к неопределённости, поскольку при чтении буквы b из палиндромной части мы обязаны всегда иметь на вершине стека символ B .

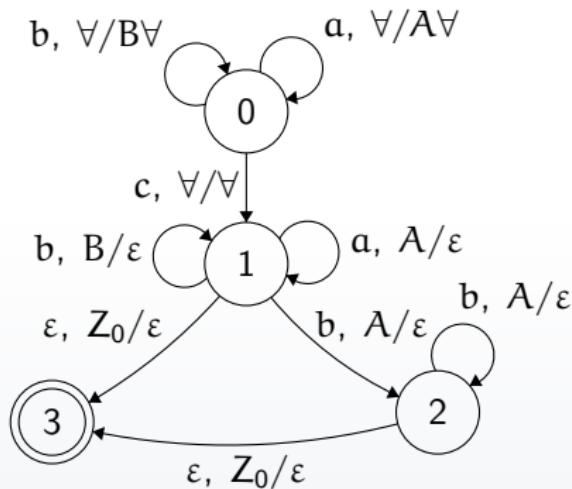


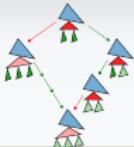
Другой подход к анализу КС-языков

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

Добавляем состояние чтения суффикса b^n (в нём на вершине стека должны быть всегда лишь символы A) и финальное состояние. Легко убедиться, что итоговый стековый автомат — DPDA.



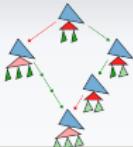


Лемма о накачке для DCFL

Теорема (S. Yu)

Пусть L — DCFL. Тогда существует такая длина накачки p , что для всех пар слов $w, w' \in L$, таких что $w = xy$ & $w' = xz$, $|x| > p$ и первые буквы y, z совпадают, выполнено одно из двух:

- ① существует накачка только префикса x (в привычном смысле);
- ② существует разбиение $x = x_1x_2x_3$, $y = y_1y_2y_3$, $z = z_1z_2z_3$ такое, что $|x_2x_3| \leq p$, $|x_2| > 0$, и $\forall i (x_1x_2^i x_3 y_1 y_2^i y_3 \in L \text{ & } x_1x_2^i x_3 z_1 z_2^i z_3 \in L)$.



Лемма о накачке для DCFL

Теорема (S. Yu)

Пусть L — DCFL. Тогда существует такая длина накачки p , что для всех пар слов $w, w' \in L$, таких что $w = xy$ & $w' = xz$, $|x| > p$ и первые буквы y, z совпадают, выполнено одно из двух:

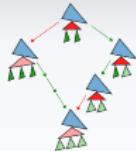
- ① существует накачка только префикса x (в привычном смысле);
- ② существует разбиение $x = x_1x_2x_3$, $y = y_1y_2y_3$, $z = z_1z_2z_3$ такое, что $|x_2x_3| \leq p$, $|x_2| > 0$, и $\forall i (x_1x_2^i x_3 y_1 y_2^i y_3 \in L \text{ & } x_1x_2^i x_3 z_1 z_2^i z_3 \in L)$.

Рассмотрим язык $\{a^n b^n\} \cup \{a^n b^{2n}\}$, положим $x = a^n b^{n-1}$, $y = b$, $z = b^{2n-1}$, где $n - 1 > p$. Тогда в случае 2 придётся накачивать в x только b , а в случае 1 нет подходящей накачки.



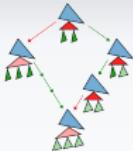
Замыкания DCFL

- Замкнуты относительно дополнения (смена конечных состояний в DPDA).
- Замкнуты относительно пересечения с регулярным языком.
- Не замкнуты относительно объединения (см. $\{a^n b^n\} \cup \{a^n b^{2n}\}$).
- Не замкнуты относительно пересечения.



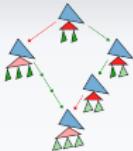
Замыкания DCFL

- Замкнуты относительно дополнения (смена конечных состояний в DPDA).
- Замкнуты относительно пересечения с регулярным языком.
- Не замкнуты относительно объединения (см. $\{a^n b^n\} \cup \{a^n b^{2n}\}$).
- Не замкнуты относительно пересечения.
- Не замкнуты относительно гомоморфизмов. См. $\{ca^n b^n\} \cup \{a^n b^{2n}\}$.
- Не замкнуты относительно конкатенации. См. $L_1 = \{ca^n b^n\} \cup \{a^n b^{2n}\}, L_2 = c^*$.



Метод подмены vs накачка для DCFL

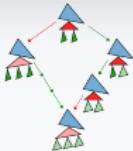
- Так же, как и в лемме о накачке для DCFL, нужно подобрать два слова xy , xz с длинными одинаковыми префиксами и различными суффиксами y , z , принадлежащие языку L .
- В лемме о накачке суффиксы y и z должны иметь существенно разное происхождение с точки зрения их распознавания PDA (разное поведение стека на префикссе x в слове xy и в слове xz), а в методе подмены часто достаточно, если стек на x только накапливается, а на y и z читается по-разному.
- В обоих случаях x лучше выбирать так, чтобы от поведения стека на нём максимально сильно зависел успех распознавания суффиксов y и z .



Метод подмены vs накачка для DCFL

Исследовать LL(k)-свойства уже известного нам DCFL
 $L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}$.

- Подберём два слова с одинаковым поведением стека до буквы с и разными суффиксами. Проще всего это сделать, если положить, что до с встречаются только буквы а. Тогда $xz = a^{n+k}ca^{n+k}$, $yz = a^{n+k}cb^{n+k}$, где n так велико, что после прочтения префикса $x = a^n$ в стеке точно есть минимум $k + 3$ символа, где k — предполагаемый lookahead.

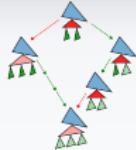


Метод подмены vs накачка для DCFL

Исследовать LL(k)-свойства уже известного нам DCFL

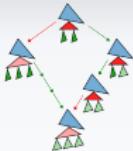
$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}$.

- $xz = a^{n+k}ca^{n+k}$, $yz = a^{n+k}cb^{n+k}$, где n так велико, что после прочтения префикса $x = a^n$ в стеке точно есть минимум $k+3$ символа, где k — предполагаемый lookahead.
- Пусть последний символ стека после чтения a^n — T_z . В слове $a^{n+k}ca^{n+k}$ при чтении символа T_z анализатору будет видно $k_1 \leq k$ букв a (если $k_1 < k$, то за ними будет конец слова), и начиная с этого состояния анализатор распознает суффикс a^i , $i \geq k_1$. В слове $a^{n+k}cb^{n+k}$ при чтении символа T_z анализатор увидит $k_2 \leq k$ букв b и распознает суффикс b^j , $j \geq k_2$.
- Если заменить в слове $a^{n+k}cb^{n+k}$ суффикс b^j на a^i , то анализатор прочитает T_z с lookahead'ом, равным a^{k_1} . Ситуация ничем не будет отличаться от той, где он видел a^{k_1} букв в суффиксе слова $a^{n+k}ca^{n+k}$, и анализатор определит, что слово $a^{n+k}cb^{n+k-j}a^i \in L$, что неверно. Значит, L — не LL(k).



Степень недетерминизма языка

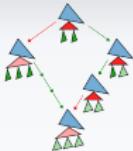
Если PDA \mathcal{A} допускает декомпозицию на DPDA, между которыми есть максимум k недетерминированных переходов, но не допускает такую декомпозицию при $i < k$ переходов, скажем, что \mathcal{A} задаёт КС-язык с k -недетерминированностью.



Степень недетерминизма языка

Если PDA \mathcal{A} допускает декомпозицию на DPDA, между которыми есть максимум k недетерминированных переходов, но не допускает такую декомпозицию при $i < k$ переходов, скажем, что \mathcal{A} задаёт КС-язык с k -недетерминированностью.

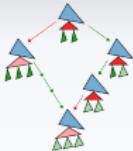
- ① Степень недетерминированности языка $\{a^n b^n\} \cup \{a^n b^{2n}\}$?



Степень недетерминизма языка

Если PDA \mathcal{A} допускает декомпозицию на DPDA, между которыми есть максимум k недетерминированных переходов, но не допускает такую декомпозицию при $i < k$ переходов, скажем, что \mathcal{A} задаёт КС-язык с k -недетерминированностью.

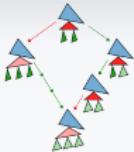
- ① Степень недетерминированности языка $\{a^n b^n\} \cup \{a^n b^{2n}\}$?
Ответ: 1
- ② Степень недетерминированности языка $\{a^n b^n\} \cup \dots \cup \{a^n b^{k*n}\}$?



Степень недетерминизма языка

Если PDA \mathcal{A} допускает декомпозицию на DPDA, между которыми есть максимум k недетерминированных переходов, но не допускает такую декомпозицию при $i < k$ переходов, скажем, что \mathcal{A} задаёт КС-язык с k -недетерминированностью.

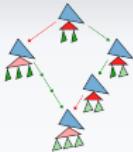
- ① Степень недетерминированности языка $\{a^n b^n\} \cup \{a^n b^{2n}\}$?
Ответ: 1
- ② Степень недетерминированности языка $\{a^n b^n\} \cup \dots \cup \{a^n b^{k*n}\}$? Ответ: тоже 1 (см. критерий исправляемости)
- ③ Степень недетерминированности языка $\{ww^R\}$ — также 1.
- ④ Степень недетерминированности языка $\{ww^Rvv^R\}$ равна 2.



Исправление недетерминированности

Пусть L — недетерминированный КС-язык и $k > 0$. Язык L — k -исправляемый, если существует алфавит Δ , $\Delta \cap \Sigma = \emptyset$ и DCFL $L(k) \subseteq (\Sigma \cup \Delta)^*$ такой, что для $h(\Delta) = \varepsilon$, $h(L(k)) = L$ и все слова языка $L(k)$ содержат не больше k букв из Δ .

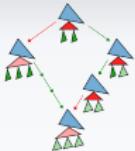
Язык L имеет k -ую степень недетерминизма $\Leftrightarrow L$ k -исправляемый, но не $k - 1$ -исправляемый.



Исправляемость и анализ на DCFL

Техника использования леммы о накачке для DCFL

- анализируем позиции в словах языка L , в которых может произойти смена наполнения стека на его опустошение, а может не произойти. Такие позиции считаем подозрительными на исправляемость.
- подбираем два слова из L , xyz_1 , xyz_2 такие, что исправляемая позиция находится в подсловае y , причём в подсловае y слова xyz_1 происходит наполнение стека, а в слове xyz_2 стек опустошается либо игнорируется.
- убеждаемся, что отдельно x накачать нельзя, после чего рассматриваем накачки yz_1 и yz_2 . Из-за разного поведения стека на их префиксах, скорее всего, эти накачки будут выводить из языка L .

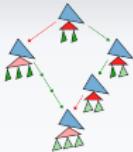


Пример применения

Проанализировать контекстно-свободный язык

$$L = \{w a^n c^n w^R \mid w \in \{a, b\}^*\}.$$

- В словах языка есть произвольные под слова из $\{a, b\}^*$, что усложняет анализ. К тому же есть блок c^n , который на первый взгляд однозначно указывает на детерминизм, однако нет условия $n \geq 1$, поэтому в некоторых случаях на его существование нельзя положиться.
Воспользуемся замкнутостью DCFL относительно пересечений с регулярными языками, избавимся от c^n и сузим область накачек.
- Простейший язык, с которым мы можем пересечь L для этой цели: $a^* b^* a^*$, после чего взять $xy = a^m$, $z_1 = a^{n_1}$, $z_2 = a^{n_2} b^{2*n_3} a^{m+n_2}$.

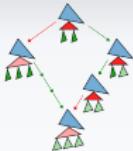


Пример применения

Проанализировать контекстно-свободный язык

$$L = \{w a^n c^n w^R \mid w \in \{a, b\}^*\}.$$

- Нужно избавиться от под слова с буквами c и сузить область накачек.
- Простейший язык, с которым мы можем пересечь L для этой цели:
 $a^* b^* a^*$, после чего взять $xy = a^m$, $z_1 = a^{n_1}$, $z_2 = a^{n_2} b^{2*n_3} a^{m+n_2}$.
- Хотя поведение стека на этих фрагментах слов соответствует рекомендуемому, анализ ни к чему не приводит: мы без проблем можем накачивать в этих словах одновременно суффикс послов xy и элементы z_1 и z_2 , а всё потому, что слова в языке a^* , являющиеся палиндромами, описываются регулярными выражениями. Искомое пересечение языков неудачное, выберем то, которое чётче обозначит нерегулярную структуру палиндрома.

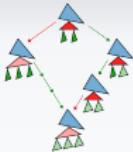


Пример применения

Проанализировать контекстно-свободный язык

$$L = \{w a^n c^n w^R \mid w \in \{a, b\}^*\}.$$

- Рассмотрим пересечение L с языком $a^*b^*a^*b^*a^*$. В нём уже будут два типа палиндромов, не распознаваемые регулярками (с одним или двумя подсловами, состоящими из букв b).
- Абеляр (т.е. антагонист) выбирает длину накачки p .
- Элоиза (т.е. мы) выбирает слова $a^{p+1}ba^{p+1}$ и $a^{p+1}ba^{p+1}a^{p+1}ba^{p+1}$ и $xy = a^{p+1}ba^p$, $z_1 = a$, $z_2 = a^{p+2}ba^{p+1}$.
- Абеляр не может накачивать только $a^{p+1}ba^p$: при накачке только второго a^p произойдёт рассинхронизация с суффиксом z_1 , а при любой накачке с участием первого a^{p+1} — рассинхронизация с суффиксом z_2 .
- Значит, Абеляру остаётся только накачивать под слово суффикса a^p синхронно с под словом z_1 (т.е. a) (и некоторым под словом z_2 , но это уже не важно), что также приводит к выходу из языка палиндромов.

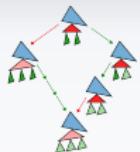


Пример применения

Проанализировать контекстно-свободный язык

$$L = \{w a^n c^n w^R \mid w \in \{a, b\}^*\}.$$

- Рассмотрим пересечение L с языком $a^* b^* a^* b^* a^*$.
- Абеляр (т.е. антагонист) выбирает длину накачки p .
- Элоиза (т.е. мы) выбирает слова $a^{p+1}ba^{p+1}$ и $a^{p+1}ba^{p+1}a^{p+1}ba^{p+1}$ и $xy = a^{p+1}ba^p$, $z_1 = a$, $z_2 = a^{p+2}ba^{p+1}$.
- Абеляр не может накачивать только $a^{p+1}ba^p$: при накачке только второго a^p произойдёт рассинхронизация с суффиксом z_1 , а при любой накачке с участием первого a^{p+1} — рассинхронизация с суффиксом z_2 .
- Значит, Абеляру остаётся только накачивать подслово суффикса a^p синхронно с подсловом z_1 (т.е. a) (и некоторым подсловом z_2 , но это уже не важно), что также приводит к выходу из языка палиндромов.
- Заметим, что если взять слова a^pba^p и $a^pba^pa^pba^p$ и $xy = a^pba^{p-1}$, тогда синхронную накачку придумать можно: накачивать в xy букву b (она ещё в пределах длины накачки), в z_2 её же, а в z_1 ничего.
- Мы показали, что язык пересечения — NCFL, значит, язык L — NCFL.



Иерархия недетерминированных КС-языков

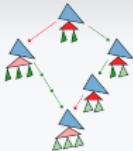
Семейство языков $w_1w_1^R\$ \dots w_kw_k^R\$$ ($\$ \notin \Sigma$) задаёт бесконечную иерархию недетерминированных языков с k -недетерминизмом.



Иерархия недетерминированных КС-языков

Семейство языков $w_1w_1^R\$ \dots w_kw_k^R\$$ ($\$ \notin \Sigma$) задаёт бесконечную иерархию недетерминированных языков с k -недетерминизмом.

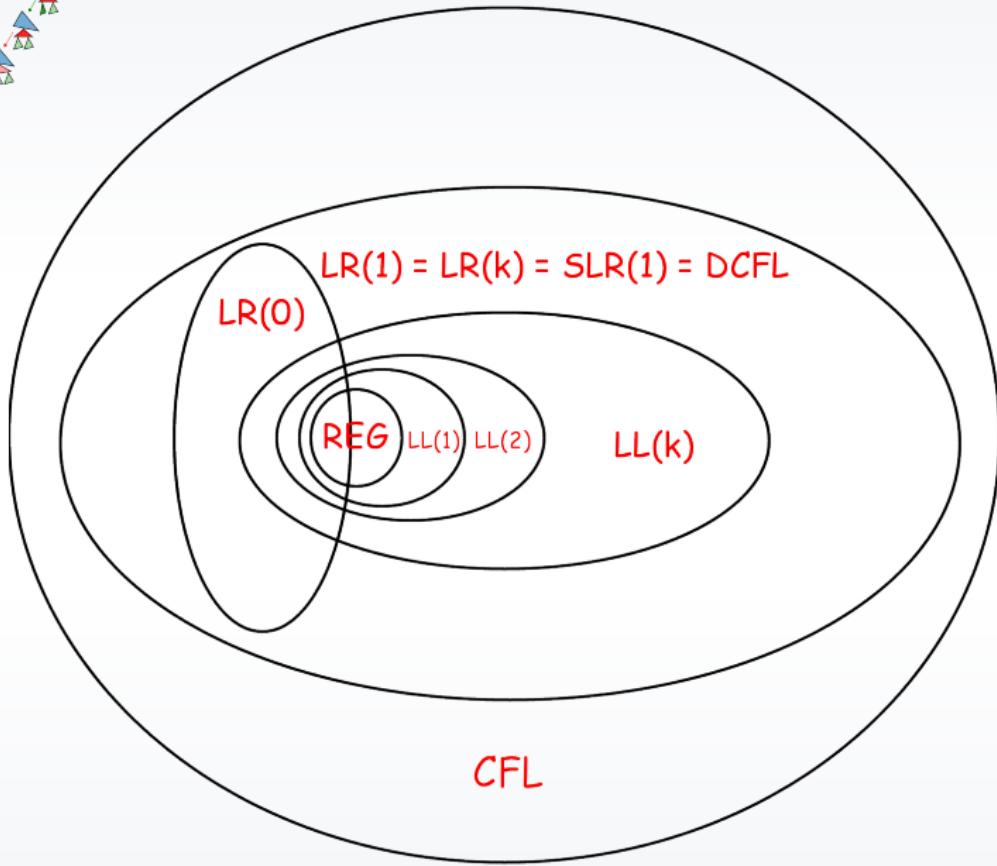
- Введение вложенных структур с совпадающими маркерами начала и конца приводит к неограниченному недетерминизму.



Иерархия Хомского revisited

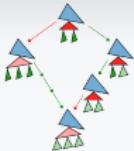
Утверждения ниже касаются только языков (не грамматик)!

- $\text{RegL} \subset \text{CFL}$;
- $\text{RegL} \subset \text{DCFL}$;
- $\text{DCFL} \subset \text{CFL}$;
- $\text{RegL} \subset \text{LL}(1)$;
- $\text{LR}(0)$ не сравним с RegL ;
- $\text{LR}(0)$ не сравним с $\text{LL}(k)$;
- $\text{LL}(k) \subset \text{LL}(k + 1)$;
- $\text{LL}(k) \subset \text{LR}(1)$;
- $\text{LR}(k) = \text{SLR}(1) = \text{DCFL}$.



Трансформации и AST. Обработка ошибок. Проблема соответствия Поста

Теория формальных языков
2022 г.

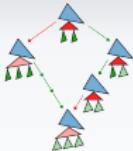


LR(k) \rightarrow LR(1), Mickunas–Lancaster–Shneider

$$\begin{array}{lll} S' \rightarrow S & S \rightarrow Abb & S \rightarrow Bbc \\ A \rightarrow aA & A \rightarrow a & B \rightarrow aB \\ & & B \rightarrow a \end{array}$$

Не LR(1), из-за свёрток $A \rightarrow a$, $B \rightarrow a$. Используем трансформацию присоединения правого контекста:

$$\begin{array}{lll} S' \rightarrow S & S \rightarrow [Ab]b & S \rightarrow [Bb]c \\ [Ab] \rightarrow a[Ab] & [Ab] \rightarrow ab & [Bb] \rightarrow a[Bb] \\ & [Bb] \rightarrow ab & \end{array}$$



$\text{LR}(k) \rightarrow \text{LR}(1)$, Mickunas–Lancaster–Shneider

$$\begin{array}{l} S' \rightarrow S \\ S \rightarrow bSS \\ S \rightarrow a \\ S \rightarrow aac \end{array}$$

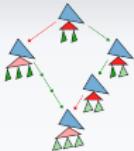
Не $\text{LR}(1)$, конфликт свёртки на префикссе ba с контекстом a .

Используем трансформацию уточнения правого контекста:

$$\begin{array}{llll} S \rightarrow bSa[a/S] & S \rightarrow bSb[b/S] & S \rightarrow a & S \rightarrow aac \\ [a/S] \rightarrow \epsilon & [a/S] \rightarrow ac & [b/S] \rightarrow Sa[a/S] & [b/S] \rightarrow Sb[b/S] \end{array}$$

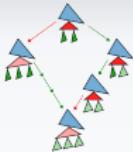
Теперь присоединим правые контексты:

$S \rightarrow b[Sa][a/S]$	$ b[Sb][b/S]$	$ a$	$ aac$
$[Sa] \rightarrow b[Sa][[a/S]a]$	$ b[Sb][[b/S]a]$	$ aa$	$ aaca$
$[Sb] \rightarrow b[Sa][[a/S]b]$	$ b[Sb][[b/S]b]$	$ ab$	$ aacb$
$[a/S] \rightarrow \epsilon$	$ ac$		
$[[a/S]a] \rightarrow a$	$ aca$		
$[[a/S]b] \rightarrow b$	$ acb$		
$[[b/S]a] \rightarrow [Sa][[a/S]a]$	$ [Sb][[b/S]a]$		
$[[b/S]b] \rightarrow [Sa][[a/S]b]$	$ [Sb][[b/S]b]$		



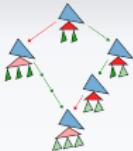
Присоединение правого контекста

- Пусть нужно присоединить правые контексты к нетерминалу A . Для всех правил вида $C \rightarrow \gamma_1 A t \gamma_2$, где t — терминал, порождаем нетерминал $[At]$ и заменяем им часть At данного правила.
- Для всех правил вида $A \rightarrow \delta$ добавляем правило $[At] \rightarrow \delta t$.
- Данное преобразование не может быть применено к правилу вида $C \rightarrow \gamma_1 A B \gamma_2$. Поэтому, если нужно присоединять контекст в таком правиле, необходимо воспользоваться алгоритмом уточнения правого контекста.



Уточнение правого контекста

- Пусть нужно уточнить правый контекст у A по правилу $C \rightarrow \gamma_1 A \gamma_2$. Положим, что $\text{FIRST}(B)$ не содержит ϵ .
- Для каждого элемента $c \in \text{FIRST}(B)$ строим нетерминал $[c/B]$ и правило $C \rightarrow \gamma_1 A c [c/B] \gamma_2$.
- Для всех правил вида $B \rightarrow c\delta$ строим правила $[c/B] \rightarrow \delta$.
- Для всех правил вида $B \rightarrow D\delta$ таких, что $c \in \text{FIRST}(D)$, строим правила вида $[c/B] \rightarrow [c/D]\delta$. Рекурсивно замыкаем процедуру (до неподвижной точки).
- Если нужно уточнить контекст A по правилу $C \rightarrow \Phi A$, тогда ищем все правила $C' \rightarrow \Psi_1 C \Psi_2$, которые порождают C , получаем правила $C' \rightarrow \Psi_1 \Phi A \Psi_2$ и действуем с ними так же, как при обычном уточнении правого контекста.
- В полученной грамматике могут появиться ϵ -правила (кодировку для ϵ можно выбрать произвольно). Поэтому их придётся в дальнейшем устраниć.



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

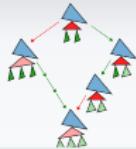
Видно, что если язык L распознаётся DPDA (т.е. является LR(1)-языком), то он также является LR(0)-языком, поскольку удовлетворяет префикс-свойству. Действительно, любое слово этого языка содержит единственную букву c , причём она расположена точно в середине слова.

Построим пробную КС-грамматику для языка L :

$$S \rightarrow aSb \mid aCa \mid bCb \mid c$$

$$C \rightarrow aCa \mid bCb \mid c$$

Проверим, является ли она LR(0)-грамматикой. Для этого построим LR(0)-автомат и проанализируем его на конфликты.



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

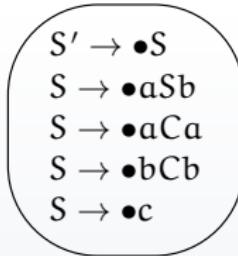
Пробная грамматика для L :

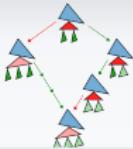
S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Начинаем строить LR(0)-автомат. Для этого вводим новое стартовое состояние S' (состояние окончательной свёртки) и начинаем разбор правила $S' \rightarrow \bullet S$.

Поскольку отмеченная позиция в правиле находится перед нетерминалом S , добавляем в состояние все ситуации вида $S \rightarrow \bullet \alpha$.

Переходы по нетерминалу S и терминалу c ведут к бесконфликтным свёрткам, поэтому малоинтересны. Разберёмся с переходом по a .





Применение MLS-подгонки

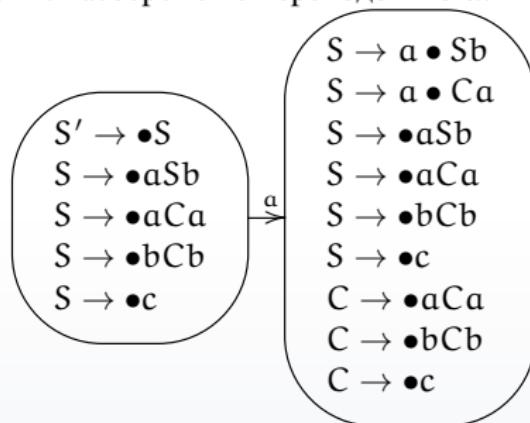
Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

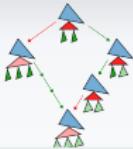
Пробная грамматика для L :

S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Переходы по нетерминалу S и терминалу c ведут к бесконфликтным свёрткам, поэтому малоинтересны. Разберёмся с переходом по a .



Похоже, что есть потенциальный конфликт (даже два) по свёрткам в S и C . Построим конфликтное состояние явно.



Применение MLS-подгонки

Исследовать на детерминированность язык

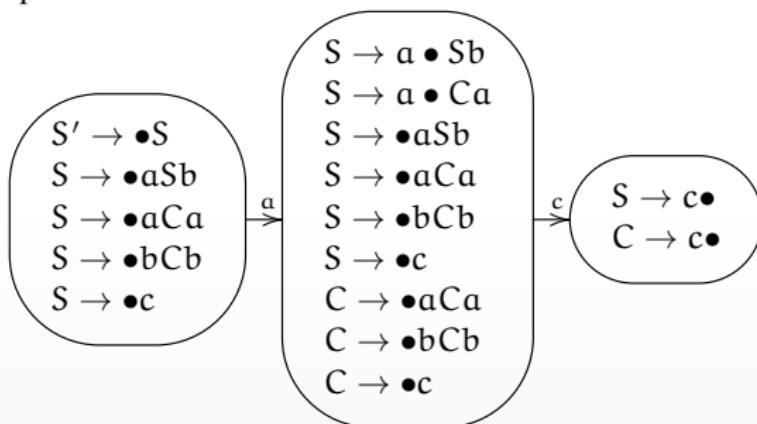
$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

Пробная грамматика для L :

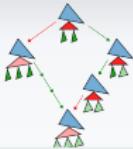
S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Похоже, что есть потенциальный конфликт (даже два) по свёрткам в S и C .

Построим конфликтное состояние явно.



Присоединим к конфликтующим S и C -нетерминалам их правые контексты.



Применение MLS-подгонки

Исследовать на детерминированность язык

$$L = \{a^n w c w^R b^n \mid w \in \{a, b\}^*\}.$$

Пробная грамматика для L :

S	\rightarrow	$aSb \mid aCa \mid bCb \mid c$
C	\rightarrow	$aCa \mid bCb \mid c$

Грамматика для L после присоединения правых контекстов к нетерминалам S и C методом MLS (новые нетерминалы выделены красным):

S	\rightarrow	$a[Sb] \mid a[Ca] \mid b[Cb] \mid c$
$[Sb]$	\rightarrow	$a[Sb]b \mid a[Ca]b \mid b[Cb]b \mid cb$
$[Ca]$	\rightarrow	$a[Ca]a \mid b[Cb]a \mid ca$
$[Cb]$	\rightarrow	$a[Ca]b \mid b[Cb]b \mid cb$

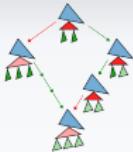
Можно построить LR(0)-автомат для этой грамматики и убедиться, что он не содержит конфликтов. Значит язык L — детерминированный (более того, LR(0)).



LL-подгонка

- Устранение левой рекурсии;
- Извлечение левого контекста:

Если даны $A \rightarrow \Phi\gamma_1$, $A \rightarrow \Phi\gamma_2$, тогда можно построить эквивалентные правила $A \rightarrow \Phi A'$, $A' \rightarrow \gamma_1 | \gamma_2$.



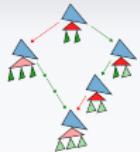
Абстрактное синтаксическое дерево

Переход от конкретного дерева разбора к дереву разбора, содержащему только значащие нетерминалы, называется переходом к AST.

- Можно сливать транзитные узлы;
- Можно стирать ветви дерева разбора.

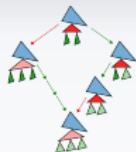
При применении подгонок и упрощений дерево разбора тоже меняется:

- устранение ϵ -правил — добавление новой абстрактной структуры;
- извлечение левого контекста — слияние сиблингов;
- присоединение и извлечение правого контекста — зависит от лексера и синхронизирующих токенов;
- устранение левой рекурсии — полностью перестраивает структуру дерева



Подходы к восстановлению после ошибок в PDA

- Множество к.э. по Майхиллу–Нероуду бесконечно \Rightarrow синхронизация учитывает стек.



Подходы к восстановлению после ошибок в PDA

- Множество к.э. по Майхиллу–Нероуду бесконечно \Rightarrow синхронизация учитывает стек.
- Стандартный подход: множество синхронизирующих терминалов строится для каждого нетерминала отдельно.



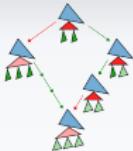
Подходы к восстановлению после ошибок в PDA

- Множество к.э. по Майхиллу–Нероуду бесконечно \Rightarrow синхронизация учитывает стек.
- Стандартный подход: множество синхронизирующих терминалов строится для каждого нетерминала отдельно.
- (режим паники) При восстановлении после ошибки отбрасывается не только префикс ошибочного входа, но и вершина стека.



Подходы к восстановлению после ошибок в PDA

- Множество к.э. по Майхиллу–Нероуду бесконечно \Rightarrow синхронизация учитывает стек.
- Стандартный подход: множество синхронизирующих терминалов строится для каждого нетерминала отдельно.
- (режим паники) При восстановлении после ошибки отбрасывается не только префикс ошибочного входа, но и вершина стека.
- (режим починки) При восстановлении после ошибки стек не отбрасывается, а вход подгоняется под стек. Набор действий может зависеть от ячейки таблицы, содержащей ошибку.

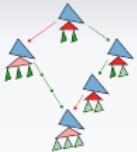


Panic mode для LL-разбора

Ошибочная ситуация

Терминал в стеке не совпадает с терминалом на ленте, либо переход по таблице правил приводит к ошибке.

- Отбрасываем вершину стека до синхронизирующего токена и входные символы до успеха перехода по нему.
- Возможное удаление \Rightarrow для токена A синхронизирующими могут предполагаться элементы $\text{FOLLOW}(A)$.
- Возможная вставка \Rightarrow синхронизирующие — $\text{FIRST}(A)$. Если конфликт терминалов — интерпретируем как возможную вставку.



Panic mode для LR-разбора

Ошибочная ситуация

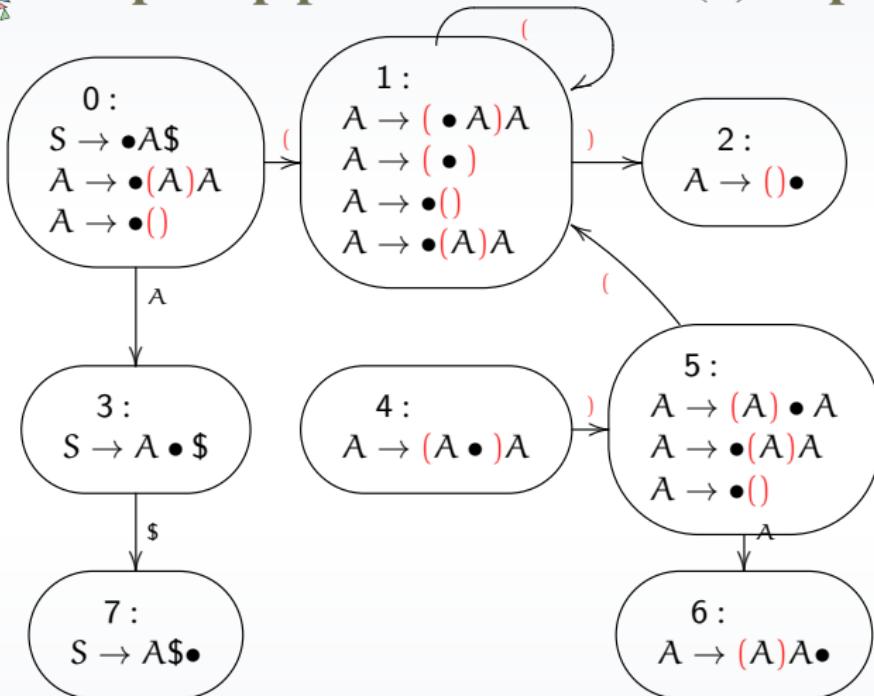
Переход по таблице правил приводит к ошибке.

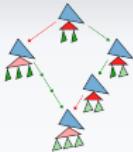
- Вводим специальный токен «ошибки» в правилае
 $A \rightarrow \beta \bullet \alpha$, на котором она произошла.
- Отбрасываем вершину стека до свёртки по правилу
 $A \rightarrow \text{«ошибка»} \alpha \bullet$, не добавляя ничего в стек (если есть lookahead, то до совпадения с lookahead-ом).
Продолжаем разбор дальше.

Альтернатива: поиск «починки» — минимального количества действий, позволяющего возобновить парсинг.

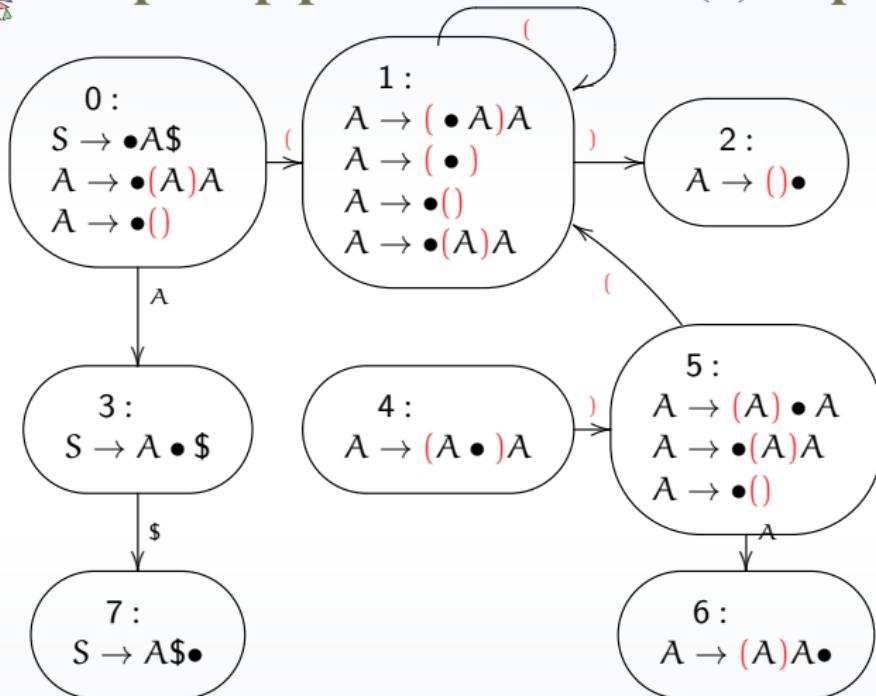


Пример panic mode в LR(0)-парсере



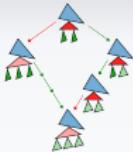


Пример panic mode в LR(0)-парсере



Разбор строки $()()\$$: $([0], ()()\$) \rightarrow ([1, 0], ()()\$) \rightarrow ([2, 1, 0], ()\$) \rightarrow ([3, 0], ()\$)$

На этом шаге происходит ошибка. Строим $S \rightarrow \bullet \text{«ошибка»} \$$, отбрасываем $()$ и редуцируемся в S .

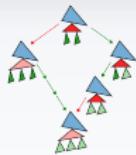


Бурке–Фишер и его вариации

Идея алгоритма

При заранее заданном k и ошибке на i -ом терминале входа рассмотреть возможные последовательности терминалов от i -ого до $i + k - 1$ -ого, продолжающие парсинг, и выбрать в качестве «починки» ту из них, расстояние Левенштейна до которой от реального входа наименьшее.

- (Corchuello et al) Также разрешается делать операции сдвига по lookahead-у.
- (Diekmann et al) Ищутся все возможные варианты «починки» и выбирается тот из них, который позволяет продолжить разбор на наибольшую глубину.



Разрешимость проблем в КС-грамматиках

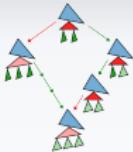
Разрешимые проблемы

- Пустота языка
- Вхождение слова в язык
- Бесконечность языка

Неразрешимые проблемы

...большинство остальных.

Подход к доказательству неразрешимости: машины Тьюринга «с историей».



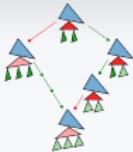
«История» вычислений

Плоская конфигурация машины Тьюринга — это $P_1 q_i p_j P_2$, где P_1 — это лента слева от головки МТ, q_i — состояние МТ, p_j — ячейка ленты, на которой стоит головка, P_2 — лента справа от головки.

Тогда шаг МТ описывается как SRS на конфигурациях.

- Пусть в состоянии q_i , прочитав символ p_i , МТ записывает p'_i и сдвигает головку вправо, переходя в состояние q_j . Тогда правило переписывания имеет вид $q_i p_i \rightarrow p'_i q_j$.
- Пусть в состоянии q_i , прочитав символ p_i , МТ записывает p'_i и сдвигает головку влево, переходя в состояние q_j , причём слева от ячейки стоит символ p_{i-1} . Тогда правило переписывания имеет вид $p_{i-1} q_i p_i \rightarrow q_j p_{i-1} p'_i$.

История вычисления МТ — это $w_0 \# w_1 \# \dots \# w_F$, где w_0 — стартовая, w_F — финальная конфигурации, и w_{i+1} получается из w_i применением SRS, описывающей шаги МТ.



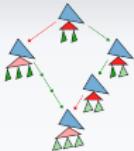
Пересечение КС-грамматик

Рассмотрим следующие истории:

$$w_0 \# w_1^R \# w_2 \# w_3^R \dots \# w_F^G$$

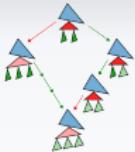
Где w_i — конфигурации, начиная со стартовой и кончая какой-нибудь финальной (но не обязательно согласующиеся с правилами МТ); $w_{2 \cdot i}^G$ — это просто $w_{2 \cdot i}$; $w_{2 \cdot i + 1}^G$ — это $w_{2 \cdot i}^R$ (реверсированная конфигурация).

- Язык $\mathcal{L}_1 = \{w_0 \# w_1^R \# w_2 \# w_3^R \dots \# w_F^G \mid \# w_{2 \cdot i} \text{ согласована с } \# w_{2 \cdot i + 1} \text{ относительно правил МТ}\}$ является КС (достаточно попарно разобрать конфигурации как слова, получающиеся из палиндромов конечным числом правил).
- Язык $\mathcal{L}_2 = \{w_0 \# w_1^R \# w_2 \# w_3^R \dots \# w_F^G \mid \# w_{2 \cdot i + 1} \text{ согласована с } \# w_{2 \cdot i + 2} \text{ относительно правил МТ}\}$ является КС (аналогично).



Пересечение КС-грамматик

- Язык $\mathcal{L}_1 = \{w_0 \# w_1^R \# w_2 \# w_3^R \dots \# w_F^G \mid \#w_{2 \cdot i} \text{ согласована с } \#w_{2 \cdot i + 1} \text{ относительно правил МТ}\}$ является КС (достаточно попарно разобрать конфигурации как слова, получающиеся из палиндромов конечным числом правил).
- Язык $\mathcal{L}_2 = \{w_0 \# w_1^R \# w_2 \# w_3^R \dots \# w_F^G \mid \#w_{2 \cdot i + 1} \text{ согласована с } \#w_{2 \cdot i + 2} \text{ относительно правил МТ}\}$ является КС (аналогично).
- $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \emptyset \Leftrightarrow$ язык, порождаемый МТ, не пуст.
- Следовательно, проблема $\mathcal{L}_1 \cap \mathcal{L}_2 \stackrel{?}{=} \emptyset$ неразрешима для КС-языков.

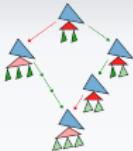


Неразрешимость всеобщности

Язык $\mathcal{L}_{\text{fail}} = \Sigma^* \setminus (\mathcal{L}_1 \cap \mathcal{L}_2)$ является КС. КС-грамматика — объединение грамматики для языка, где хотя бы один переход с чётного шага истории на нечётный не согласуется с правилами МТ, грамматики для языка, где несогласование есть при переходе с нечётного шага на чётный, и грамматики для языка слов, имеющих неправильную лексическую структуру.

Следствие

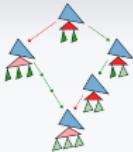
Вопрос $\mathcal{L} \stackrel{?}{=} \Sigma^*$ неразрешим для КС-грамматик (т.к. если есть способ разрешать $\mathcal{L}_{\text{fail}} \stackrel{?}{=} \Sigma^*$, тогда есть способ и разрешить проблему пустоты языка МТ).



Проблема соответствия Поста

Рассмотрим «домино» из пар $\langle u, w \rangle \in \langle \Sigma^*, \Sigma^* \rangle$. Пусть имеется n таких пар вида $\langle u_i, w_i \rangle$. Существует ли последовательность индексов, такая что $u_{i_1} \dots u_{i_k} = w_{i_1} \dots w_{i_k}$?

- Неразрешима — рассмотрим пошаговые «истории» МТ.
- Следствие — вопрос о неоднозначности КС-грамматики тоже неразрешим; вопрос о вхождении палиндрома в КС-язык неразрешим; вопрос о вхождении квадрата в КС-язык неразрешим.



Теорема Грейбах

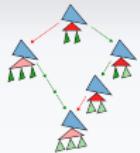
Пусть C — семейство языков, содержащее все регулярные языки, для которого неразрешима проблема всеобщности. Если это семейство замкнуто относительно объединения и приписывания регулярных языков, то для него неразрешимо никакое свойство, выполняющееся для всех регулярных языков и замкнутое относительно производных.

Visibly Pushdown Languages.

Конъюнктивные языки.

Древесные языки

Теория формальных языков
2022 г.

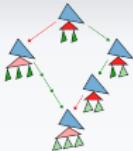


Скобочные языки

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.



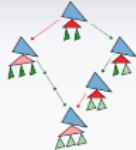
Visibly Pushdown Languages

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).

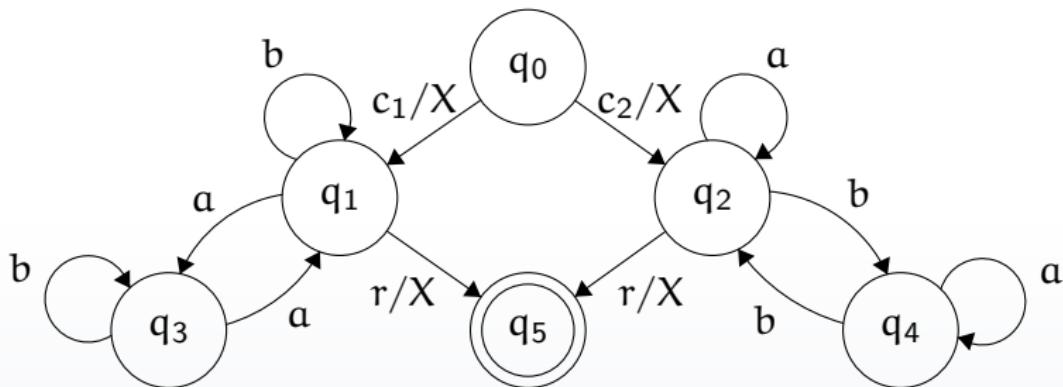
Дополнительные допущения:

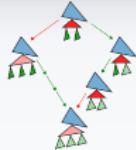
- завершение по финальному состоянию;
- если достигли дна стека (символ \perp), то символы из Σ_r не меняют его (т.е. дно стека вытолкнуть из него нельзя).



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).

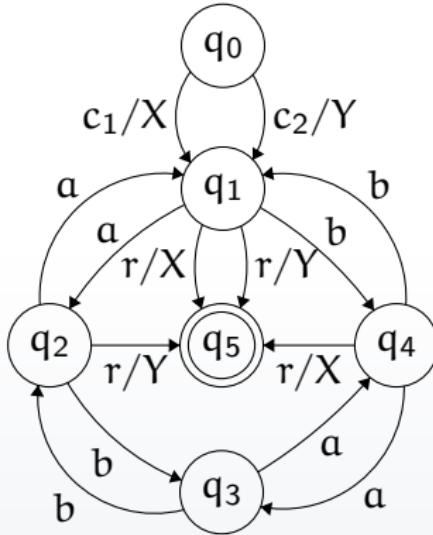


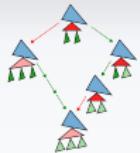


Примеры: два эквивалентных VPDA

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.



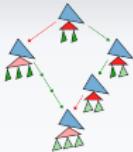


Теорема Майхилла–Нероуда

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:

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



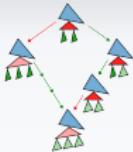
Расширения КС-грамматик?

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

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

Примеры задач:

<pre>int f(int,...,int) { ... } int g(int,...,int) { ... } int main() { x1 = f(0,...,0); x2 = g(0,...,0); }</pre>	<pre>int f(int,...,int) { ... } int main() { x = f(f(0,...,0), ..., f(0,...,0)) }</pre>
---	---



Конъюнктивные грамматики

Конъюнктивная грамматика G — грамматика, правила которой имеют вид

$$A_i \rightarrow \Phi_1 \ \& \ \dots \ \& \ \Phi_2$$

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

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

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

$$A \rightarrow aA \mid ab$$

$$C \rightarrow aCa \mid B$$

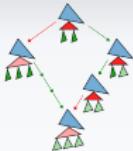
$$B \rightarrow BA \mid b$$



Язык равенства

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

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



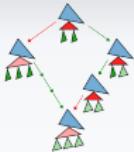
Вопросы парсинга

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:

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

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

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



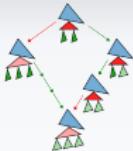
Древесные автоматы

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)



Real-time клеточные автоматы

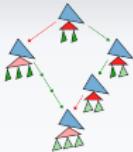
Если переписывание стартует с «листьев», но структурой является сеть, то получается т.н. «автомат Треллиса».

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):

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

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

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



Грамматики 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:

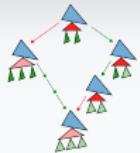
- $A[\circ \circ \eta] \rightarrow \Phi_1 A'[\circ \circ \eta'] \Phi_2$
- или $A[\eta] \rightarrow \Phi$

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

К3-свойства языков. MFA, атрибутные грамматики и типизация



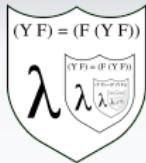
Теория формальных языков
2022 г.



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).

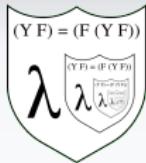


Языки с backref (Shmidt, 2014)

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 $ababbab$.



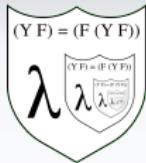
Memory Finite Automata (MFA)

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.

Из состояния $\langle q, v\omega, \langle u_i, r_i \rangle \rangle$ в состояние $\langle q', \omega, \langle u'_i, r'_i \rangle \rangle$ ($u_i \in \Sigma^*$, $r_i \in \{o, c\}$) переходит по правилу $\delta(q, b) \rightarrow \langle q', s_1, \dots, s_k \rangle$ следующим образом:

- 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.

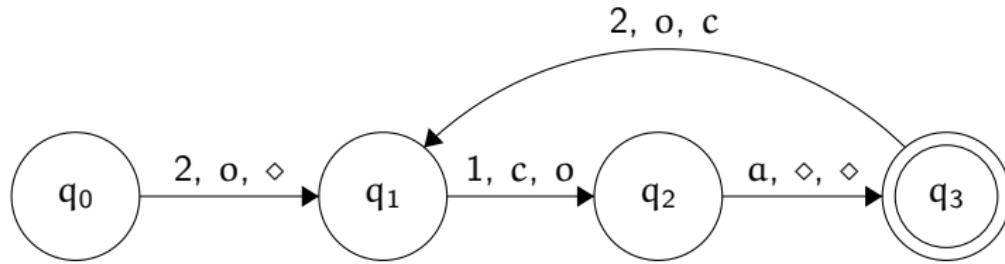
Заметим, что запись в ячейку слова, считанного с ленты по переходу, осуществляется не исходя из флагов памяти в предыдущем состоянии, а исходя из флагов памяти в состоянии, куда осуществляется переход. То есть мы сначала открываем (или закрываем) память, а уже потом читаем с ленты и пишем в открытые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

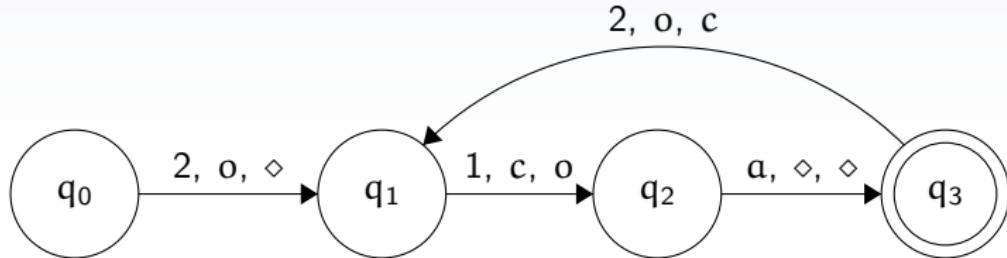
Посмотрим на 2-MFA, построенный по этому выражению:



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 ϵ).



MFA для $\{a^{n^2}\}$

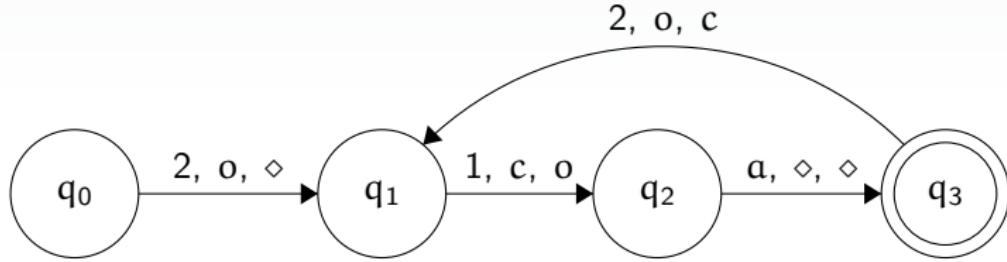


Поскольку открытие и закрытие памяти происходит перед чтением с ленты (и, соответственно, записью в память), то на первом переходе в открытую память первой ячейки записывается пустое слово и оно же читается с ленты (т.к. начальная конфигурация второй ячейки памяти есть ε). Затем память первой ячейки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

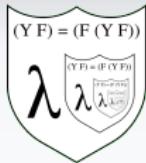


MFA для $\{a^{n^2}\}$



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

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.

- $[_x(a|b)]_x c x$ 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 дополнения и пересечения с
регулярным языком;
- Не замкнуты относительно объединения (и даже —
объединения с регулярными языками), и относительно
пересечения друг с другом.



DMFA и Jumping Lemma

$k - \text{MFA}$ 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

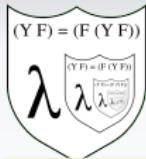
Язык $\mathcal{L} = \{a^n w b^n h(w) \mid w \in a, b^* \text{ & } h(a) = aa \text{ & } h(b) = ab\}$ не является DMFL.

- 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 атрибутных 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.

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

Синтетический вариант: $A \rightarrow \epsilon$; $A.\text{iter} := 0$

$$\begin{array}{lcl} T \rightarrow bTc & ; & T_0.\text{iter} := T_1.\text{iter} + 1 \\ T \rightarrow \epsilon & ; & T.\text{iter} := 0 \end{array}$$

Вариант с наследованием:

$$\begin{array}{lcl} S \rightarrow AT & ; & B.\text{inh_iter} := A.\text{iter} \\ A \rightarrow aA & ; & A_0.\text{iter} := A_1.\text{iter} + 1 \\ A \rightarrow \epsilon & ; & A.\text{iter} := 0 \\ T \rightarrow bTc & ; & T_1.\text{inh_iter} := T_0.\text{inh_iter} - 1 \\ T \rightarrow \epsilon & ; & T.\text{inh_iter} == 0 \end{array}$$

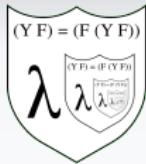


Определение типа

Понятие типа ограничивает возможные операции над его сущностями ⇒ 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 Γ — это перечисление $x_i : \tau_i$ — aka контекст.

Читается: «в контексте Γ терм M имеет тип σ ».

Понимается: «если придать переменным x_i типы τ_i , тогда можно установить, что тип выражения M есть σ ».

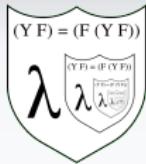
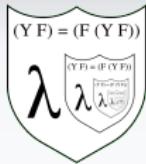


Таблица связывания

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

- bind :: ([таблица], [имя], [тип]) → [таблица];
- lookup :: ([таблица], [имя]) → [тип].



Пример правил типизации

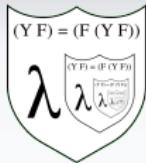
- Сорта (простые типы): Bool, Int.
- Операторы: $=$, $+$, условный, вызов функции.
- Синтаксис:

[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

```
[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

[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]

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

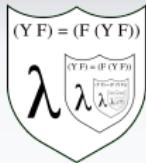
num	int
id	t == undef = err; int otherwise = t where t = lookup(vtable, id)
Exp ₁ +Exp ₂	t ₁ ≠ int t ₂ ≠ int = err; int otherwise = int where t ₁ =tchExp(Exp ₁ ,vtable,ftable), t ₂ =tchExp(Exp ₂ ,vtable,ftable)



Пример правил типизации

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

num	int
id	$\begin{cases} \text{t == undef} = \text{err; int} \\ \text{otherwise} = \text{t} \\ \quad \text{where t = lookup(vtable, id)} \end{cases}$
$\text{Exp}_1 + \text{Exp}_2$	$\begin{cases} \text{t}_1 \neq \text{int} \parallel \text{t}_2 \neq \text{int} = \text{err; int} \\ \text{otherwise} = \text{int} \\ \quad \text{where } \text{t}_1 = \text{tchExp}(\text{Exp}_1, \text{vtable}, \text{ftable}), \\ \quad \text{t}_2 = \text{tchExp}(\text{Exp}_2, \text{vtable}, \text{ftable}) \end{cases}$
$\text{Exp}_1 = \text{Exp}_2$	$\begin{cases} \text{t}_1 == \text{t}_2 = \text{bool} \\ \text{otherwise} = \text{err; bool} \\ \quad \text{where } \text{t}_1 = \text{tchExp}(\text{Exp}_1, \text{vtable}, \text{ftable}), \\ \quad \text{t}_2 = \text{tchExp}(\text{Exp}_2, \text{vtable}, \text{ftable}) \end{cases}$



Правила типизации в форме вывода

$$\frac{}{\Gamma \vdash \mathbf{num} : \text{int}}$$

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

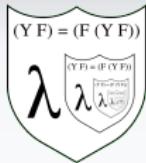
$$\frac{}{\Gamma, \mathbf{id} : \tau \vdash \mathbf{id} : \tau}$$

$$\frac{\Gamma \vdash t_1 : \sigma, \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 = t_2 : \text{bool}}$$

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

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

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



Пробное задание на РК-2

- ① Язык, описывающийся следующей атриутной грамматикой:

$S \rightarrow AT ; T.\text{rng} > A.\text{iter}$

$A \rightarrow aA ; A_0.\text{iter} := A_1.\text{iter} + 1$

$A \rightarrow \epsilon ; A.\text{iter} := 0$

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

$T \rightarrow K ; T.\text{rng} := K.\text{rng}$

$K \rightarrow aK ; K_0.\text{rng} := K_1.\text{rng} + 1$

$K \rightarrow bK ; K_0.\text{rng} := 0$

$K \rightarrow \epsilon ; K.\text{rng} := 0$

- ② Язык $\{wcvw_{\text{pref}}zw_{\text{suff}} \mid w, z \in \{a, b\}^* \text{ & } v \in \{a, b, c\}^*\}$.
Здесь w_{pref} — непустой префикс слова w ; w_{suff} — непустой суффикс слова w .



Продолжение (третье задание)

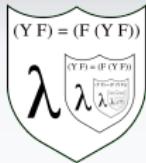
- ③ Язык, описывающийся следующей атриутной грамматикой:

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



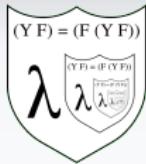
Классы задач

- 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 под слова, пытаемся выявить структуру под слов, 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 под слов).



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: пересечён 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 $bba^{2 \cdot n}$.
 - Полученный язык $\{b^n a^{2^n}\}$ известен, как не КС.



Языки SRS

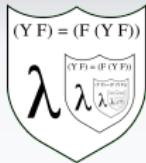
❶ $ab \rightarrow baa$, носитель ab^+ . Не КС: пересечён с b^+a^+ .

- 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. Также не DFML: достаточно взять $v_n = b^n$, $p_n = b^{n+k}$.
 $b^{n+k} b^n a^{2^{2n+k}}$ 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 $\{w_1 w_2 \mid w_i = z_{i,1} a z_{i,2} \text{ & } |z_{i,1}| = |z_{i,2}|\}$.
- 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 $\{w_1uu^Rw_2 \mid |u| > 1 \text{ & } w_i \neq z_1uz_2\}$. Т.е. подслово u не содержится нигде больше в слове, при любом выборе u таком, что uu^R входит в слово языка и притом $|u| > 1$.
Подсказка: сначала рассмотреть $|u| > 0$.



Продолжение (третье задание)

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

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

- Язык, описывающийся следующей атрибутной грамматикой:

```

 $S \rightarrow AT ; T.rng > A.iter$ 
 $A \rightarrow aA ; A_0.iter := A_1.iter + 1$ 
 $A \rightarrow \epsilon ; A.iter := 0$ 
 $T \rightarrow TcT ; T_0.rng := \max(T_1.rng, T_2.rng)$ 
 $T \rightarrow K ; T.rng := K.rng$ 
 $K \rightarrow aK ; K_0.rng := K_1.rng + 1$ 
 $K \rightarrow bK ; K_0.rng := 0$ 
 $K \rightarrow \epsilon ; K.rng := 0$ 

```

- Язык $\{wcvw_{pref}zw_{suff} \mid w, z \in \{a, b\}^* \text{ & } v \in \{a, b, c\}^*\}$. Здесь w_{pref} — непустой префикс слова w ; w_{suff} — непустой суффикс слова w .

- Язык, описывающийся следующей атрибутной грамматикой:

```

 $S \rightarrow SbS ; S_0.iter := 2 \cdot S_1.iter, S_1.iter == S_2.iter$ 
 $S \rightarrow a ; S.iter := 1$ 

```

Решение задачи III

Посмотрим на несколько первых итераций, порождающих слова языка \mathcal{L} :

```

 $S \rightarrow SbS ; S_0.iter := 2 \cdot S_1.iter, S_1.iter == S_2.iter$ 
 $S \rightarrow a ; S.iter := 1$ 

```

Слово aba семантическому критерию удовлетворяет, т.к. оно получается разбором двух нетерминалов с атрибутом $iter = 1$. При этом для aba соответствующий атрибут равен 2.

Из этих базовых слов мы не можем получить слово $a\color{blue}{b}aba$, т.к. атрибуты его левых и правых частей не равны. То же самое верно и про $a\color{red}{b}aba$. Слово $\color{red}{a}\color{blue}{b}ababa$ построить можно, и его атрибут $iter$ будет равен 4.

Мы видим, что на каждой новой итерации мы получаем возможность скомбинировать только два слова с предыдущей итерации, при этом атрибут $iter$ полученного слова будет не равен атрибутам $iter$ никаких ранее построенных слов. Если на i -й итерации было построено слово $(ab)^k a$, то на $i + 1$ -й итерации будет построено слово $(ab)^k ab(ab)^k a = (ab)^{2k+1} a$. Отсюда следует, что множество слов языка в свёрточной форме выглядит так: $\{(ab)^{2^{i-1}} a\}$.

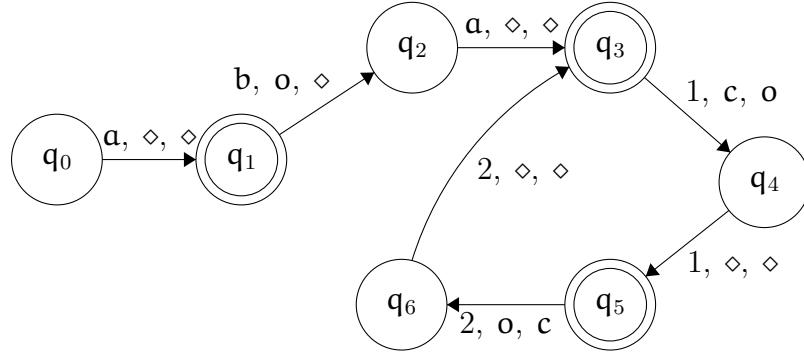
Проще всего обосновать не КС-свойство для данного языка с помощью теоремы Париха. Действительно, кратности букв b (впрочем, как и a) в словах языка описываются экспоненциально растущей функцией, которая не

может быть представлена как объединение линейных. Доказательство не КС-свойства языка \mathcal{L} с помощью леммы о накачке предоставляется читателю.

Построим backref-регулярку для данного языка, пользуясь наблюдением, что $2^n - 1 = 1 + 2 + 2^2 + \dots + 2^{n-1}$.

$$a([_1 b a]_1 ([_2 x_1 x_1]_2 [1 x_2 x_2]_1)^* [_2 x_1 x_1]_2)?$$

Заметим, что эту backref-регулярку можно преобразовать в детерминированный МФА. Единственной проблемой, которая в ней может возникнуть, является переход от итерации Клини к последнему чтению $x_1 x_1$, который решается тем, что промежуточное состояние внутри итерации перед чтением $[1 x_2 x_2]_1$ нужно сделать финальным.



Решение задачи II

$$\mathcal{L} = \{wcvw_{\text{pref}}zw_{\text{suff}} \mid w, z \in \{a, b\}^* \text{ & } v \in \{a, b, c\}^*\}$$

Можно заметить, что для любого слова в \mathcal{L} при рассмотрении $w_{\text{pref}} = s_{\text{fst}}w'_{\text{pref}}$ и $w_{\text{suff}} = w'_{\text{suff}}s_{\text{last}}$, где s_{fst} и s_{last} — первая и последняя буквы соответственно, получается слово вида $wcv s_{\text{fst}} w'_{\text{pref}} z w'_{\text{suff}} s_{\text{last}}$, в котором можно положить $z' = w'_{\text{pref}} z w'_{\text{suff}}$, а новыми префиксом и суффиксом — только первую и последнюю буквы. И это слово также принадлежит \mathcal{L} , однако его структура подходит под большее число случаев w , чем исходная. Поэтому язык \mathcal{L} по факту представляет собой следующий:

$$\mathcal{L} = \{s_{\text{fst}}ws_{\text{last}}cvs_{\text{fst}}zs_{\text{last}} \mid s_{\text{fst}}ws_{\text{last}}, z \in \{a, b\}^* \text{ & } v \in \{a, b, c\}^*\}$$

Поскольку w, v, z здесь можно заменить регулярками $(a|b)^*$ и $(a|b|c)^*$, то данный язык регулярен и является объединением следующих языков:

- $ac(a|b|c)^*a(a|b)^*a$
- $bc(a|b|c)^*b(a|b)^*b$
- $a(a|b)^*ac(a|b|c)^*a(a|b)^*a$
- $a(a|b)^*bc(a|b|c)^*a(a|b)^*b$
- $b(a|b)^*ac(a|b|c)^*b(a|b)^*a$
- $b(a|b)^*bc(a|b|c)^*b(a|b)^*b$

Префикс-свойство для него не выполняется, потому что внутри подвыражения $(a|b|c)^*$ или последнего подвыражения $(a|b)^*$ может встретиться повтор последующего суффикса. В качестве контрпримера, подтверждающего это, достаточно взять слова $aacaa$ и слово $aaacaa$, принадлежащие \mathcal{L} .

Всякий регулярный язык является LL(1). Соответствующую грамматику можно построить, тупо взяв ДКА для него и преобразовав его в праволинейную грамматику с эпилон-правилами для самой последней буквы (иначе будет LL(1)-конфликт; нигде, кроме как с финальными состояниями, конфликта быть не может в силу того, что автомат детерминирован).

Решение задачи I

Сначала посмотрим на атрибутную грамматику.

$$\begin{aligned}
 S &\rightarrow AT \quad ; \quad T.\text{rng} > A.\text{iter} \\
 A &\rightarrow aA \quad ; \quad A_0.\text{iter} := A_1.\text{iter} + 1 \\
 A &\rightarrow \epsilon \quad ; \quad A.\text{iter} := 0 \\
 T &\rightarrow TcT \quad ; \quad T_0.\text{rng} := \max(T_1.\text{rng}, T_2.\text{rng}) \\
 T &\rightarrow K \quad ; \quad T.\text{rng} := K.\text{rng} \\
 K &\rightarrow aK \quad ; \quad K_0.\text{rng} := K_1.\text{rng} + 1 \\
 K &\rightarrow bK \quad ; \quad K_0.\text{rng} := 0 \\
 K &\rightarrow \epsilon \quad ; \quad K.\text{rng} := 0
 \end{aligned}$$

Все атрибуты синтетические, кроме того, язык K явно регулярен, и атрибут rng в нём изменяется, только пока в префиксе встречаются исключительно буквы a . То есть если $K \rightarrow a^n(b(a|b))^?$, то $K.\text{rng} = n$.

Теперь посмотрим на язык T . Заметим, что его можно переписать в форме $(Kc)^*K$, устранив левую рекурсию (поскольку язык K не содержит

букв c , а разделители между K -развёртками не содержат a и b , то можно K представить единственным токеном и далее, например, применить метод Блюма–Коха). При этом семантическое свойство будет выглядеть так: $T.rng = \max(K.rng)$.

Осталось разобраться с атрибутом A . Здесь, очевидно, если $A \rightarrow a^m$, то $A.iter = m$. Но именно в случае A есть неоднозначность границы разбора: если $S \rightarrow AKcT$, причём $K \rightarrow a^n(b(a|b))^?$, $A \rightarrow a^m$, то мы можем сделать какие угодно разборы префикса a^{n+m} начиная от $K \rightarrow a^{n+m}(b(a|b))^?$, $A \rightarrow \epsilon$ и кончая $K \rightarrow (b(a|b))^?$, $A \rightarrow a^{n+m}$. С точки зрения языка, наиболее выгодно положить $K \rightarrow a^{n+m}(b(a|b))^?$, $A \rightarrow \epsilon$ — это даст возможность породить наиболее широкий класс слов.

Таким образом, условие на атрибуты вырождается в $T.rng > 0$, а грамматика для данного языка может быть преобразована в следующую форму:

```

 $S \rightarrow T ; T.rng > 0$ 
 $T \rightarrow KcT ; T_0.rng := \max(K.rng, T_1.rng)$ 
 $T \rightarrow K ; T.rng := K.rng$ 
 $K \rightarrow aK ; K_0.rng := K_1.rng + 1$ 
 $K \rightarrow bK ; K_0.rng := 0$ 
 $K \rightarrow \epsilon ; K.rng := 0$ 

```

Эта грамматика определяет регулярный язык $((a|b)^*c)^*a^+(a|b)^*(c(a|b)^*)$.

Если бы условие на атрибуты в корне было $T.rng < A.iter$, тогда наиболее выгодное с точки зрения языка решение по неоднозначности было бы $K \rightarrow (b(a|b))^?$, $A \rightarrow a^{n+m}$. То есть требовалось бы сравнить по длине максимальные подстроки из букв a в префиксах, идущих сразу после буквы c , с префиксом a^{n+m} (и если ни одной буквы c в слове нет, то это условие тривиально выполняется, если только в начале слова есть хотя бы одна буква a). Будем рассматривать только нетривиальную ситуацию — буквы c присутствуют. Упрощенная атрибутная грамматика станет такая:

```

 $S \rightarrow ABcT ; T.rng < A.iter$ 
 $A \rightarrow aA ; A_0.iter := A_1.iter + 1$ 
 $A \rightarrow \epsilon ; A.iter := 0$ 
 $T \rightarrow KcT ; T_0.rng := \max(T_1.rng, T_2.rng)$ 
 $T \rightarrow K ; T.rng := K.rng$ 
 $K \rightarrow aK ; K_0.rng := K_1.rng + 1$ 
 $K \rightarrow B ; K.rng := 0$ 
 $B \rightarrow b(a|b)^* | \epsilon ; \text{атрибутов нет}$ 

```

В свёрточной форме язык переписывается как $\{a^n(b(a|b))^?(ca^i(b(a|b))^?)^+ | \forall i(n > i)\}$.

Похоже, что есть более чем двойственная взаимосвязь между блоками a^i (т.к. каждый из них нужно сравнивать с самым первым, что означало бы, что из первого блока нужно читать в стек несколько раз). Попробуем опровергнуть КС-свойство для полученного языка путём лёммы о накачке. Для этого сначала пересечём наш язык с $a^+ca^+ca^+$, чтобы избавиться от лишних подстрок, определяемых конечноавтоматным поведением, и от накачек, наращивающих число К-фрагментов.

Пусть p — длина накачки. Рассмотрим слово $a^{p+1}ca^pca^p$. Отдельно фрагмент a^{p+1} накачивать мы не можем — отрицательная накачка сразу же выведет из языка. Накачивать его вместе с фрагментом a^p тоже не получится — отрицательная накачка даст слово, в котором первый фрагмент не больше по длине, чем фрагмент a^p . Однако накачивать только a^pca^p (в любых вариациях) тоже нельзя — это сразу же приведёт к выходу из языка при повторной положительной накачке.

Можно заметить, что язык конъюнктивен. Действительно, для крайней пары нетерминалов A и K можно КС-свободно описать слова вида

$$a^{n+m+1}(b(a|b)^*|\epsilon)c(a|b|c)^*ca^n(b(a|b)^*|\epsilon)$$

Например, такой грамматикой:

$$\begin{aligned} S' &\rightarrow S'a \mid S'b \mid Cb \mid C \\ C &\rightarrow aCa \mid aAc \mid aAcWc \\ A &\rightarrow aA \mid bW \mid \epsilon \\ W &\rightarrow (a|b|c)W \mid \epsilon \end{aligned}$$

Конъюнктивная грамматика готова:

$$\begin{aligned} S &\rightarrow S' \& ScK \\ S &\rightarrow aK \\ K &\rightarrow (a|b)K \mid \epsilon \end{aligned}$$

Заметим, что этот язык — не DMFA. Используем замкнутость DMFA относительно пересечения с регулярными языками и пересечём наш язык с a^+ca^+ . Получим язык слов вида $a^{n+m}ca^n$. Его префиксы a^* определяют бесконечное число классов эквивалентности по Майхиллу–Нероуду, поэтому для них можно применить JL (то есть положить v_n подсловом префикса a^+). Положим $v_n = a^n$. Для p_n есть два варианта: $p_n = a^{n+k}$ и $p_n = a^{n+k}ca^i$, где $i < 2n + k$. Существуют такие z_i , что слова $p_nv_nz_i$ входят в язык, а именно: $z_i = c$ в первом случае и $z_i = a$ во втором случае. Однако слова $a^{n+k}c$ и $a^{n+k}ca^i a$ также входят в язык, а их суффиксы (после p_n) не начинаются с v_n . Поэтому ни с какой позиции слова запомненный префикс v_n детерминированно прочитано быть не может.