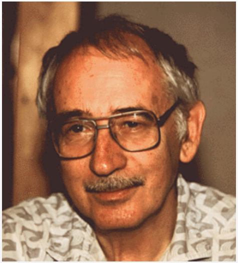
# Краткое вступление



**В.Ф. Турчин**
(1931–2010)

- Функциональный язык Рефал
- Суперкомпиляция
- . . .

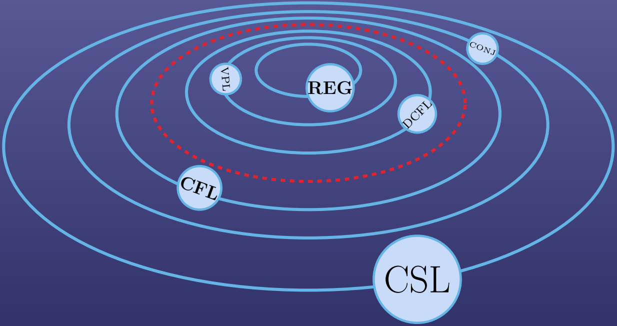
---

- Семинары **МЕТА**: 2008–2016 в Переславле-Залесском
- Приглашённые докладчики: Neil D. Jones, Simon Peyton-Jones

## Образовательный семинар META

ИПС им. А.К. Айламазяна РАН, 1 июля, 2025

# Абстрактная алгебра в проектировании распознавателей



# Распознаватели для формальных языков

### *Проблема слова*

Пусть $\mathscr{L}$ — формальный язык над алфавитом $T$, $\omega$ — терм в алфавите $T$. Проверить, верно ли, что $\omega \in \mathscr{L}$.

- Автоматы-распознаватели:
  - \+ Естественная реализация, эффективность
  - \- Неявные ограничения на свойства языка, слабая модифицируемость
- Алгебраические методы:
  - \+ Расширяемость (правила переписывания), семантика — база алгоритма
  - \- Неочевидная реализация, необходимость оптимизаций

# Алгебраический язык как опора проектирования

Простейшие аналогии:

- ассоциативность операции — возможность асинхронного разбора
- коммутативность операции — возможность сортировки
- ACI (ассоциативность, коммутативность, идемпотентность) — возможность представления в форме множества

# Регулярные языки

## *Академические регулярные выражения*

- *Альтернатива (сложение, объединение)* $|$ — *ассоциативна, коммутативна, идемпотентна* .
- *Конкатенация (умножение) — ассоциативна.*
- *Унарная операция итерации* $^*$.
- *Нет нетривиальных соотношений для нуля* ($\underbrace{\varnothing}_{\textit{пустой регекс}}$).
- *Нетривиальные соотношения для единицы* ($\underbrace{\varepsilon}_{\textit{пустое слово}}$):

  $\varepsilon \in x^*$, *даже если* $\varepsilon \notin x$.

# Регулярные языки и их распознаватели

**Регекс**

Экспоненциален относительно состояний ДКА

Устранение состояний

$\Longleftarrow$

$\Longrightarrow$

Детерминизация

**ДКА**

Экспоненциален относительно длины регекса

## *(Conway* & *Krob)*

Устранение состояний сохраняет регекс по модулю правил:

- $x(yx)^* = (xy)^*x$
- $x^*(yx^*)^* = (x \mid y)^*$
- $\varepsilon \mid xx^* = x^*, \varepsilon \mid x^*x = x^*$
- $x(y \mid z) = xy \mid xz, (y \mid z)x = yx \mid zx$

# Расширенные регексы — приоритеты и якори

*Альтернатива и конкатенация ассоциативны* — условно, при переходе к именованным группам захвата.

*Альтернатива коммутативна* — неверно:
Регекс $((\,)\mid \mathtt{a})(\mathtt{a}\mid \mathtt{b})^*$ не эквивалентен $(\mathtt{a}\mid(\,))(\mathtt{a}\mid \mathtt{b})^*$ из-за групп захвата.

- $\underbrace{\varepsilon}_{((\,)|\mathtt{a})}\overbrace{\mathtt{aa}}^{(\mathtt{a}|\mathtt{b})^*}$ — группы 1 и 2 примут значение $\varepsilon$, группа примет значение $\mathtt{a}$ (последнее захваченное регексом $\mathtt{a}|\mathtt{b}$ значение).
- $\underbrace{\mathtt{a}}_{(\mathtt{a}|(\,))}\overbrace{\mathtt{a}}^{(\mathtt{a}|\mathtt{b})^*}$ — группы 1 и 2 примут значение $\mathtt{a}$, группа примет значение $\mathtt{a}$.

*Нет нетривиальных соотношений для нуля* $(\varnothing)$ — неверно:
Регекс $\mathtt{a}\backslash\mathtt{A}$ эквивалентен $\varnothing$: $\backslash\mathtt{A}$ может лишь начинать выражение.

# Производные Бржозовски

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

---

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

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

С помощью последовательного взятия производных можно свести задачу $\omega \in \mathscr{L}(R)$ к задаче $\varepsilon \in \omega^{-1}R$. Используя ACI-эквивалентности, можно построить ДКА, помеченный производными языками, выполняющий такое распознавание.

# Расширенные производные Бржозовски

Движок .NET:

- Отказ от перехода к автомату, производные используются непосредственно для разбора.
- Дополнительные условия для отслеживания единицы (синтаксически корректные якори слов).
- Дополнительные условия для отслеживания нуля (синтаксически некорректные якори слов).

*Итог: алгоритм распознавания $\omega$, линейный по длине $\omega$.*

---

- Возможные расширения на опережающие проверки.

D.Moseley et al: *Derivative Based Nonbacktracking Real-World Regex Matching with Backtracking Semantics*, Proceedings of the ACM on Programming Languages, Volume 7, Issue PLDI, Article No.: 148, pp. 1026–1049, 2023.

# Расширенные регексы — обратные ссылки

Только регексы с именованными группами захвата.

Backref-регексы (слова со ссылками, Shmid):

$$\begin{cases} [_k \tau]_k & \text{(захват в память)} \\ \&_k & \text{(чтение памяти)} \end{cases}$$

Пример: $[_1\mathtt{a}^*]_1\mathtt{a}^+\mathtt{b}\&\mathtt{1}$ определяет язык $\{a^m b a^n \mid m > n\}$

- $\varepsilon$-семантика (Shmid) — неинициализированная ссылка распознаёт $\{\varepsilon\}$;
- $\varnothing$-семантика (практические реализации) — неинициализированная ссылка распознаёт $\varnothing$.

Выбор не влияет на язык.

# Проблема: циклическая память

- Выразительная сила и сложность анализа возрастает по сравнению с практическими регулярными выражениями.

Среди 3000 расширенных регексов со StackOverflow не нашлось ни одного циклического.

- Неочевидное синтаксическое свойство:
  - слово $([_1\&\mathtt{2}]_1[_2\mathtt{a}\&\mathtt{1}]_2[_1\mathtt{a}^*]_1)^*$ циклическое,
  - слово $([_1\&\mathtt{2}]_1[_1\mathtt{a}^*]_1[_2\mathtt{a}\&\mathtt{1}]_2)^*$ нециклическое.

# Семантически корректная переименовка

| | $[_1\mathtt{a}^*]_1 \mapsto [_3\mathtt{a}^*]_3$ | $[_1\&\mathtt{2}]_1 \mapsto [_3\&\mathtt{2}]_3$ | Циклы после подст. |
|---|---|---|---|
| $([_1\&\mathtt{2}]_1[_2\mathtt{a}\&\mathtt{1}]_2[_1\mathtt{a}^*]_1)^*$ | ✓ | ✗ | ✓ |
| $([_1\&\mathtt{2}]_1[_1\mathtt{a}^*]_1[_2\mathtt{a}\&\mathtt{1}]_2)^*$ | ✗ | ✓ | ✗ |

# Класс $ACREG$

### *Ациклические регексы*

- Ограниченная переинициализация зависимых ячеек памяти
- Скобки групп захвата — операции над памятью
- Могут быть корректно переименованы в регексы без переинициализаций

$ACREG$ — идемпотентное полукольцо, удовлетворяющее теоремам Конвея – Кроба

# Автоматы с памятью

## Конечный автомат с памятью

Пятерка $\langle Q, \Sigma, \delta, q_0, F \rangle$:

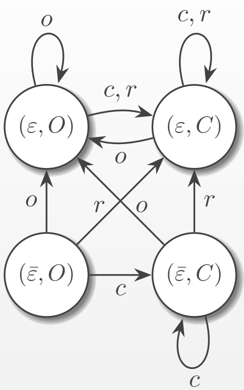
- $Q$ — множество состояний автомата;
- $\Sigma$ — входной (терминальный) алфавит автомата;
- $\delta$ — множество правил перехода. Переходы помечаются действиями над памятью: $o$ — открытие ячейки, $c$ — закрытие ячейки, $r$ — сброс до $\varepsilon$;
- $q_0 \in Q$ — начальное состояние, $F \subseteq Q$ — множество конечных состояний.

Операция сброса не применяется в оригинальной статье Шмида и необходима, чтобы сделать алгебру действий над памятью композиционно замкнутой.

# MFA: состояния памяти

- текущее состояние;
- содержимое памяти;
- конфигурации ячеек:

$$\begin{cases} O & \text{(open)} \\ C & \text{(closed)} \end{cases}$$

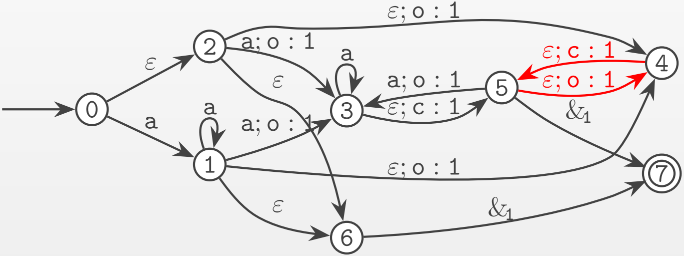



- Начальная конфигурация памяти: $\big(q_0, w, (\varepsilon, C), \ldots, (\varepsilon, C)\big)$.
- Действия над ячейками памяти происходят до чтения с ленты.

# Необходимость откатов и пустые действия

Расширенное регулярное выражение: $\mathtt{a}^*[\mathtt{a}^*] : \mathtt{1}^* \&_1$

Без сбросов памяти необходимо возникают циклы по пустому слову.



# Необходимость откатов и пустые действия

Расширенное регулярное выражение: $\mathtt{a}^*[\mathtt{a}^*] : 1^* \&_1$

Без переходов по пустому слову с операцией сброса распознавание эффективно.