# КЗ-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 с backref (Shmidt, 2014)

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

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

# Memory Finite Automata (MFA)

$k - \mathrm{MFA}\ \mathscr{A}$ имеет функцию перехода из $Q \times \Sigma \cup \{\varepsilon\} \cup \{1, \ldots, k\}$ в подмножество $Q \times \langle o, c, \diamond \rangle^k$, где:

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

Из состояния $\langle q, \nu\omega, \langle u_i, r_i \rangle\rangle$ в состояние $\langle q', \omega, \langle u_i', r_i' \rangle\rangle$ ($u_i \in \Sigma^*, r_i \in \{o, c\}$) переходит по правилу $\delta(q, b) \rightarrow \rangle q', s_1, \ldots, s_k\rangle$ следующим образом:

- если $b \in \Sigma \cup \{\varepsilon\}$, то $\nu = b$;
- если $b \in \{1, \ldots, k\}$ и $r_k = c$, то $\nu = u_b$;
- $r_i' = r_i$, если $s_i = \diamond$, и $s_i$ в противном случае;
- $u_i' = u_i\nu$, если $r_i' = r_i = o$; $u_i' = \nu$, если $r_i' = o$ и $r_i = c$; и не меняется, если $r_i' = c$.

# DMFA и Jumping Lemma

$k - MFA$ детерминированный, если в нём нет $\varepsilon$-переходов и $\forall q \in Q, b \in \Sigma(|\bigcup_{i=1}^{k} \delta(q, i)| + |\delta(q, b)| \leqslant 1)$. DMFL — такой язык, для которого существует DMFA.

- $[_x(a|b)^*]_x cx$ определяет DFML;
- $([_x y]_x [_y x a]_y)^*$ определяет DFML;
- $1^+[_x 0^*]_x (1^+ x)^* 1^+$ — тоже DFML (эквивалентен регулярке $1(1^+|0[_x 0^*]_x 1^+ (0x1^+)^*)$).

# DMFA и Jumping Lemma

$k$ — MFA детерминированный, если в нём нет $\varepsilon$-переходов и $\forall q \in Q, b \in \Sigma(|\bigcup_{i=1}^{k} \delta(q,i)| + |\delta(q,b)| \leqslant 1)$. DMFL — такой язык, для которого существует DMFA.

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

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

## 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 $A_0 \to A_1 \ldots A_n$ — правило КС-грамматики. Припишем к нему конечное число атрибутных свойств.

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

Повторные нетерминалы при присвоении атрибутов индексируются по вхождениям в правило слева направо. Т.е., например, если дано правило $\mathsf{N} \to \mathsf{N} - \mathsf{N}$, тогда уравнение на атрибуты $\mathsf{N}_0.\mathtt{attr} = \mathsf{N}_1.\mathtt{attr} - \mathsf{N}_2.\mathtt{attr}$ будет означать, что атрибут родителя есть атрибут левого потомка минус атрибут правого потомка, помеченных нетерминалами $\mathsf{N}$.

Неповторные нетерминалы в уравнениях на атрибуты обычно не индексируются.

## Пример АГ для $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n\}$

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

Синтетический вариант:

$$\begin{array}{lcl} S \to AT & ; & T.\mathsf{iter} == A.\mathsf{iter} \\ A \to aA & ; & A_0.\mathsf{iter} := A_1.\mathsf{iter} + 1 \\ A \to \varepsilon & ; & A.\mathsf{iter} := 0 \\ T \to bTc & ; & T_0.\mathsf{iter} := T_1.\mathsf{iter} + 1 \\ T \to \varepsilon & ; & T.\mathsf{iter} := 0 \end{array}$$

Вариант с наследованием:

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

## Определение типа

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

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

## Определение типа

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

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.
Записывается: $\Gamma \vdash M : \sigma$, где $\Gamma$ — это перечисление $x_i : \tau_i$ — aka контекст.
Читается: «в контексте $\Gamma$ терм $M$ имеет тип $\sigma$».
Понимается: «если придать переменным $x_i$ типы $\tau_i$, тогда можно установить, что тип выражения $M$ есть $\sigma$».

# Таблица связывания

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

- bind :: ([таблица], [имя], [тип]) $\to$ [таблица];
- lookup :: ([таблица], [имя]) $\to$ [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<br>\| otherwise = t<br>where t = lookup(vtable, **id**) |
| $Exp_1$+$Exp_2$ | \| $t_1 \neq$ int \|\| $t_2 \neq$ int = err; int<br>\| otherwise = int<br>where $t_1$=tchExp($Exp_1$,vtable,ftable),<br>$t_2$=tchExp($Exp_2$,vtable,ftable) |

# Пример правил типизации

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

| | |
|---|---|
| **num** | int |
| **id** | \| t == undef = err; int<br>\| otherwise = t<br>where t = lookup(vtable, **id**) |
| $Exp_1$+$Exp_2$ | \| $t_1 \neq$ int \|\| $t_2 \neq$ int = err; int<br>\| otherwise = int<br>where $t_1$=tchExp($Exp_1$,vtable,ftable),<br>$t_2$=tchExp($Exp_2$,vtable,ftable) |
| $Exp_1$=$Exp_2$ | \| $t_1 == t_2$ = bool<br>\| otherwise = err; bool<br>where $t_1$=tchExp($Exp_1$,vtable,ftable),<br>$t_2$=tchExp($Exp_2$,vtable,ftable) |
| | |

# Правила типизации в форме вывода

$$\frac{}{\Gamma \vdash \textbf{num} : \mathit{int}} \qquad \frac{\Gamma \vdash t_1 : \mathit{int}, \Gamma \vdash t_2 : \mathit{int}}{\Gamma \vdash t_1 + t_2 : \mathit{int}}$$

$$\frac{}{\Gamma, \textbf{id} : \tau \vdash \textbf{id} : \tau} \qquad \frac{\Gamma \vdash t_1 : \sigma, \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 = t_2 : \mathit{bool}}$$

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

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

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