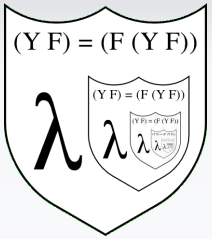
КЗ-свойства языков. MFA, атрибутные грамматики и типизация

Теория формальных языков

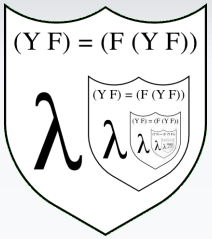
2022 г.



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.

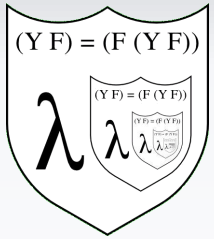
В отличие от кодов Хаффмана, не разбираются с помощью конечных автоматов. Необходимо понятие обратных ссылок (backreferences) — актуальное в современных REGEX библиотеках.



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:

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



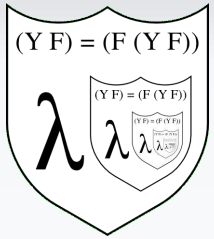
Memory Finite Automata (MFA)

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

- если $b \in \Sigma \cup \{\varepsilon\}$, то $v = b$;
- если $b \in \{1, \ldots, k\}$ 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 $r_i' = c$.

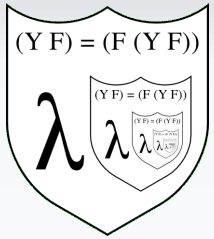


DMFA и Jumping Lemma

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

- $[_x(a|b)^*]_x cx$ определяет DMFL;
- $([_x y]_x [_y xa]_y)^*$ 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.

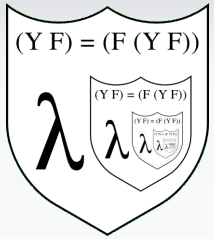


DMFA и Jumping Lemma

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

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

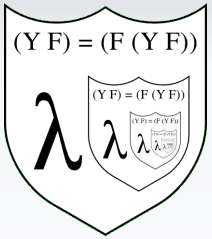
- $|\nu_n| = n$;
- ν_n — подслово p_n;
- $p_n\nu_n$ — префикс какого-то слова из $\mathscr{L}$;
- $\forall u \in \Sigma^+(p_n u \in \mathscr{L} \Rightarrow \nu_n$ — префикс $u)$.



Пример применения JL

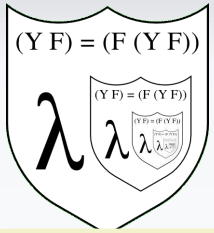
Язык $\mathscr{L} = \{a^n w b^n h(w) \mid w \in a, b^* \ \& \ h(a) = aa \ \& \ h(b) = ab\}$ не является DMFL.

- Пересечём $\mathscr{L}$ с $a^* b^+$. Получим язык $\mathscr{L}' = \{a^n b^n\}$.
- Предположим, что выполнены условия JL. Рассмотрим возможные значения ν_n.
  - Первое ν_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.
  - $\nu_n = a^n$. Тогда $p_n = a^{n+k}$. Слово $a^{n+k} b^{n+k} \in \mathscr{L}'$, но его суффикс b^{n+k} не начинается с ν_n. Что доказывает непринадлежность $\mathscr{L}'$ (а значит, и $\mathscr{L}$) 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.

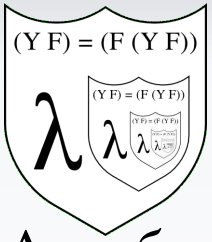


Атрибутные грамматики

Пусть $A_0 \to A_1 \ldots A_n$ — 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 $N \to N - N$, 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 $\{a^n b^n c^n\}$

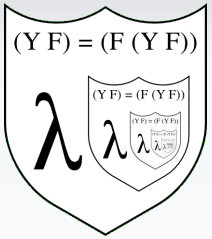
Атрибут нетерминала $iter$ семантически означает число итераций. Чтобы не смешивать синтетические и наследуемые атрибуты, введём также атрибут inh_iter, означающий то же самое, но наследуемый сверху вниз по дереву разбора, а не снизу вверх. Здесь $==$ — предикат; $:=$ — операция присваивания.

Синтетический вариант:

	$S \to AT$	;	$T.iter == A.iter$
	$A \to aA$	;	$A_0.iter := A_1.iter + 1$
	$A \to \varepsilon$	;	$A.iter := 0$
	$T \to bTc$	;	$T_0.iter := T_1.iter + 1$
	$T \to \varepsilon$	;	$T.iter := 0$

Вариант с наследованием:

$S \to AT$	;	$B.inh\_iter := A.iter$
$A \to aA$	;	$A_0.iter := A_1.iter + 1$
$A \to \varepsilon$	;	$A.iter := 0$
$T \to bTc$	;	$T_1.inh\_iter := T_0.inh\_iter - 1$
$T \to \varepsilon$	;	$T.inh\_iter == 0$

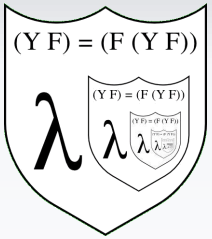


Определение типа

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 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».

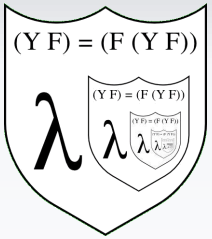
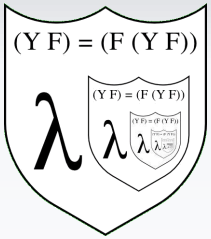


Таблица связывания

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]

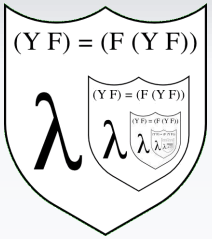
[F] ::= [TypeId] ([TIds]) = [Exp]

[Exps] ::= [Exp] | [Exp], [Exps]

[TypeId] ::= (Bool | Int) **id** [TIds] ::= [TypeId], [TIds] | [TypeId]

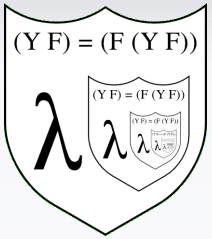
[Exp] ::= **num** | **id** | [Exp]+[Exp] | [Exp] = [Exp] | **id** ([Exps])

| 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]

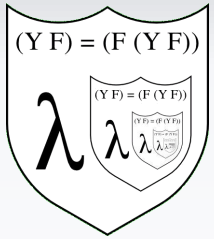


Пример правил типизации

[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

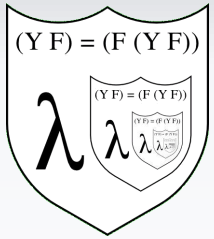
num	int
id	| t == undef = err; int | otherwise = t where t = lookup(vtable, **id**)
Exp_1+Exp_2	| $t_1 \neq$ int || $t_2 \neq$ int = err; int | otherwise = int where t_1=tchExp(Exp_1,vtable,ftable), t_2=tchExp(Exp_2,vtable,ftable)



Пример правил типизации

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	| $t_1 \neq$ int || $t_2 \neq$ int = err; int | otherwise = int where t_1=tchExp(Exp_1,vtable,ftable), t_2=tchExp(Exp_2,vtable,ftable)
Exp_1=Exp_2	| $t_1 ==$ t_2 = bool | otherwise = err; bool where t_1=tchExp(Exp_1,vtable,ftable), t_2=tchExp(Exp_2,vtable,ftable)



Правила типизации в форме вывода

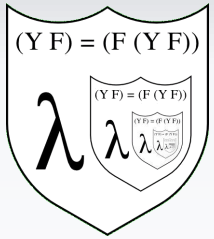
$$\frac{}{\Gamma \vdash \textbf{num} : int} \qquad \frac{\Gamma \vdash t_1 : int, \Gamma \vdash t_2 : int}{\Gamma \vdash t_1 + t_2 : int}$$

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

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

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

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

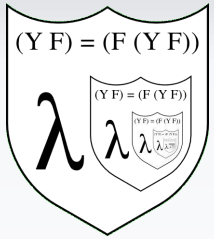


Пробное задание на РК-2

1. Язык, описывающийся следующей атрибутной грамматикой:

$$
\begin{array}{lcl}
S \to AT & ; & T.rng > A.iter \\
A \to aA & ; & A_0.iter := A_1.iter + 1 \\
A \to \varepsilon & ; & A.iter := 0 \\
T \to TcT & ; & T_0.rng := max(T_1.rng, T_2.rng) \\
T \to K & ; & T.rng := K.rng \\
K \to aK & ; & K_0.rng := K_1.rng + 1 \\
K \to bK & ; & K_0.rng := 0 \\
K \to \varepsilon & ; & K.rng := 0
\end{array}
$$

2. Язык $\{wcvw_{pref}zw_{suff} \mid w, z \in \{a, b\}^* \ \& \ v \in \{a, b, c\}^*\}$. Здесь w_{pref} — непустой префикс слова w; w_{suff} — непустой суффикс слова w.



Продолжение (третье задание)

3. Язык, описывающийся следующей атрибутной грамматикой:

$$S \to SbS \quad ; \quad S_0.iter := 2 \cdot S_1.iter,\ S_1.iter == S_2.iter$$
$$S \to a \quad ; \quad S.iter := 1$$