

КЗ-свойства языков. MFA, атрибутивные 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 — $[_i,]_i, \chi_i$. Вхождения χ_i и скобки с индексом i не могут встречаться внутри $[_i \dots]_i$, однако разные скобочные блоки могут быть перепутаны:

$$[_1 a [_2 b]_1 \chi_1]_2 \chi_2$$



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:

- 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 $r'_i = c$.



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.

- $[_x(a|b)^*]_x s x$ 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 — MFA детерминированный, если

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

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

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



Пример применения JL

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

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



Отделение семантики и синтаксиса

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



Атрибутные грамматики

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

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

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

Неповторные нетерминалы в уравнениях на атрибуты обычно не индексируются.



Пример АГ для $\{a^n b^n c^n\}$

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

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

Синтетический вариант:

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

Вариант с наследованием:

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



Определение типа

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

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

Б.Пирс



Определение типа

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

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.

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

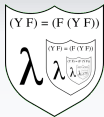


Таблица связывания

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

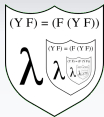
- $\text{bind} :: ([\text{таблица}], [\text{имя}], [\text{тип}]) \rightarrow [\text{таблица}];$
- $\text{lookup} :: ([\text{таблица}], [\text{имя}]) \rightarrow [\text{тип}].$



Пример правил типизации

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

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



Пример правил типизации

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



Пример правил типизации

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

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

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



Пример правил типизации

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

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



Правила типизации в форме вывода

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

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

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

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

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



Пробное задание на РК-2

- 1 Язык, описывающийся следующей атрибутивной грамматикой:

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

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

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

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

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

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

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

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

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

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



Продолжение (третье задание)

- 3 Язык, описывающийся следующей атрибутивной грамматикой:
- $$\begin{aligned} S &\rightarrow SbS && ; && S_0.iter := 2 \cdot S_1.iter, S_1.iter == S_2.iter \\ S &\rightarrow a && ; && S.iter := 1 \end{aligned}$$