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, 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)

$k - \mathrm{MFA}\ \mathscr{A}$ имеет функцию перехода из $Q \times \Sigma \cup \{\varepsilon\} \cup \{1, \ldots, k\}$ в подмножество $Q \times \langle o, c, \diamond \rangle^k$, где 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 $\delta(q, b) \rightarrow \rangle q', s_1, \ldots, s_k \rangle$ 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 $r_i' = c$.

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 флаги управления памятью o, c, $\diamond$ означают:

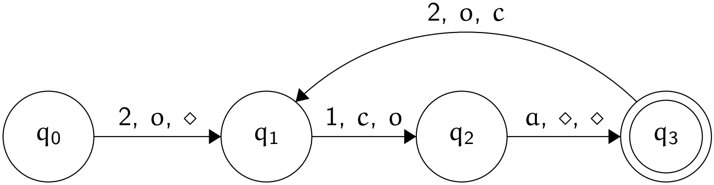
- c — «закрыть» ячейку памяти;
- o — «открыть» ячейку памяти;
- $\diamond$ — не менять состояние ячейки.

Начальная конфигурация памяти: $\langle\langle\varepsilon, c\rangle, \ldots, \langle\varepsilon, c\rangle\rangle$. 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 $\{a^{n^2}\}$

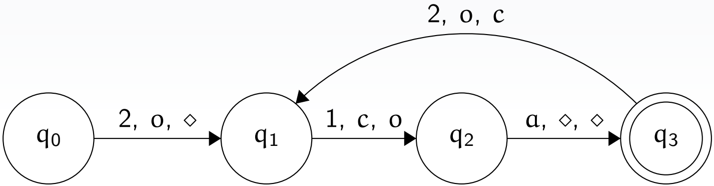
Backref-REGEX для этого языка: $([_1 x_2]_1 [_2 x_1 a]_2)^+$. Здесь важно, что неинициализированная переменная (первое вхождение x_2) получает значение ε.

Посмотрим на 2-MFA, построенный по этому выражению:



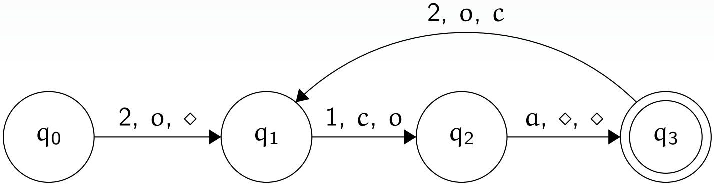
Поскольку открытие и закрытие памяти происходит перед чтением с ленты (и, соответственно, записью в память), то на первом переходе в открытую память первой ячейки записывается пустое слово и оно же читается с ленты (т.к. начальная конфигурация второй ячейки памяти есть ε).

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: $\langle\langle\varepsilon,\ c\rangle, \langle a,\ o\rangle\rangle$
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 — $\mathcal{MFA}$ 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

$k - \mathrm{MFA}$ 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 $\forall m \exists n, p_n, v_n$ 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 $u)$.

Пример применения JL

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

- Пересечём $\mathscr{L}$ с $\mathtt{a}^*\mathtt{b}^+$. Получим язык $\mathscr{L}' = \{\mathtt{a}^n\mathtt{b}^n\}$.
- Предположим, что выполнены условия JL. Рассмотрим возможные значения v_n.
  - Первое v_n, для которого выполняется требуемое условие, обязано иметь вид $\mathtt{a}^n$. Действительно, в противном случае при чтении префикса, состоящего из букв $\mathtt{a}$, автомат мог бы принимать лишь конечное число состояний, однако классов эквивалентности по Майхиллу-Нероуду относительно языка $\mathtt{a}^n\mathtt{b}^n$ в языке $\mathtt{a}^*$ бесконечно много.
  - $v_n = \mathtt{a}^n$. Тогда $p_n = \mathtt{a}^{n+k}$. Слово $\mathtt{a}^{n+k}\mathtt{b}^{n+k} \in \mathscr{L}'$, но его суффикс $\mathtt{b}^{n+k}$ не начинается с v_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 $A_0 \to A_1 \dots A_n$ — 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 $\mathsf{N} \to \mathsf{N} - \mathsf{N}$, 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 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 $\mathrm{inh\_iter}$, означающий то же самое, но наследуемый сверху вниз по дереву разбора, а не снизу вверх. Здесь $==$ — предикат; $:=$ — операция присваивания.

Синтетический вариант:

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

Вариант с наследованием:

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

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 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]) $\to$ [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 | otherwise = t where t = lookup(vtable, **id**)
$\text{Exp}_1+\text{Exp}_2$	| $t_1 \neq$ 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

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

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

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

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

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

Пробное задание на РК-2

1. Язык, описывающийся следующей атрибутной грамматикой:

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

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

Продолжение (третье задание)

3. Язык, описывающийся следующей атрибутной грамматикой:

$$\begin{array}{lll} S \to SbS & ; & S_0.\mathit{iter} := 2 \cdot S_1.\mathit{iter},\ S_1.\mathit{iter} == S_2.\mathit{iter} \\ S \to a & ; & S.\mathit{iter} := 1 \end{array}$$

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 подслова, пытаемся выявить структуру подслов, 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 подслов).

Языки SRS

1. $\mathtt{bb} \to \mathtt{aa}$, $\mathtt{ab} \to \mathtt{aba}$, носитель $\mathtt{b}^*$. Регулярен: правило $\mathtt{ab} \to \mathtt{aba}$ влечёт правило $\mathtt{ab} \to \mathtt{aba}^+$. Далее разбиваем на подходящие подслова и итерируем их чередование.
2. $\mathtt{ab} \to \mathtt{baa}$, носитель $\mathtt{ab}^+$. Не КС: пересечём с $\mathtt{b}^+\mathtt{a}^+$.
   - Кратчайшее слово: $\mathtt{baa}$. Поскольку все $\mathtt{b}$ только перемещаются влево, можно предположить, что каждое очередное слово получается максимальным передвижением $\mathtt{b}$ с самой правой позиции на самую левую.
   - Рассмотрим цепочку превращений $\mathtt{baab}$. Получаем $\mathtt{baab} \to \mathtt{babaa} \to \mathtt{bbaaaa}$.
   - Аналогично из слова $\mathtt{ba}^n\mathtt{b}$ получается слово $\mathtt{bba}^{2\cdot n}$.
   - Полученный язык $\{\mathtt{b}^n\mathtt{a}^{2^n}\}$ известен, как не КС.

Языки SRS

1. $ab \to baa$, носитель ab^+. Не КС: пересечём 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 $baab \to babaa \to bbaaaa$.
    - Аналогично из слова ba^nb получается слово $bba^{2\cdot n}$.
    - 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 $\{w_1 w_2 \mid w_i = z_{i,1} a z_{i,2} \;\&\; |z_{i,1}| = |z_{i,2}|\}$.
- 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 $a \to bab$, $ba \to ab$ над базисным словом aa (единственным). Подсказка: сначала решить задачу над базисным словом a.
2. Язык $\{w_1 u u^R w_2 \mid |u| > 1 \;\&\; w_i \neq z_1 u z_2\}$. 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 $[\mathsf{Name}].\mathsf{id}$ такое $[\mathsf{Val}].val$, что $([\mathsf{Name}].\mathsf{id} = [\mathsf{Val}].val) \in \mathsf{Table}$):

$[\mathsf{S}] \to \{[\mathsf{Decl}]\}[\mathsf{Exp}]$	;	$[\mathsf{Exp}].vars \subseteq [\mathsf{Decl}].vars$, $[\mathsf{Exp}].val == 1$, $[\mathsf{Exp}].\mathsf{inh\_table} := [\mathsf{Decl}].\mathsf{table}$
$[\mathsf{Decl}] \to ([\mathsf{Name}] = [\mathsf{Val}])[\mathsf{Decl}]$	;	$[\mathsf{Name}].\mathsf{id} \notin [\mathsf{Decl}]_1.vars$, $[\mathsf{Decl}]_0.\mathsf{table} := [\mathsf{Decl}]_1.\mathsf{table} \cup \{[\mathsf{Name}].\mathsf{id} = [\mathsf{Val}].val\}$, $[\mathsf{Decl}]_0.vars := [\mathsf{Decl}]_1.vars \cup \{[\mathsf{Name}].\mathsf{id}\}$
$[\mathsf{Decl}] \to \varepsilon$	;	$[\mathsf{Decl}].vars := \varnothing$, $[\mathsf{Decl}].\mathsf{table} := \varnothing$
$[\mathsf{Exp}] \to [\mathsf{Name}]$	;	$[\mathsf{Exp}].val := \mathsf{lookup}([\mathsf{Name}].\mathsf{id}, [\mathsf{Exp}].\mathsf{inh\_table})$
$[\mathsf{Exp}] \to [\mathsf{Exp}]\ \&\ [\mathsf{Exp}]$	;	$[\mathsf{Exp}]_0.val := \mathsf{min}([\mathsf{Exp}]_1.val, [\mathsf{Exp}]_2.val)$, $[\mathsf{Exp}]_1.\mathsf{inh\_table} := [\mathsf{Exp}]_0.\mathsf{inh\_table}$, $[\mathsf{Exp}]_2.\mathsf{inh\_table} := [\mathsf{Exp}]_0.\mathsf{inh\_table}$
$[\mathsf{Name}] \to \mathsf{a}[\mathsf{Name}]$	;	$[\mathsf{Name}]_0.\mathsf{id} := \mathsf{a} \mathbin{++} [\mathsf{Name}]_1.\mathsf{id}$
$[\mathsf{Name}] \to \varepsilon$	;	$[\mathsf{Name}].\mathsf{id} := \varepsilon$
$[\mathsf{Val}] \to 0$	;	$[\mathsf{Val}].val := 0$
$[\mathsf{Val}] \to 1$	;	$[\mathsf{Val}].val := 1$