Теория автоматов и формальных языков

Синтаксически управляемая трансляция

Автор: Екатерина Вербицкая

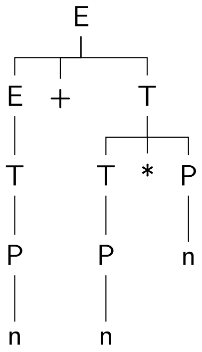
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

22 ноября 2016г.

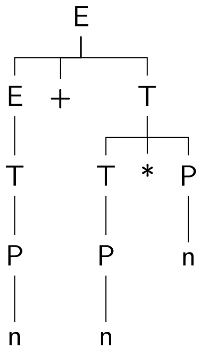
В предыдущей серии

- Что такое язык, когда предложение принадлежит языку
- Классы языков
  - Регулярные
  - Контекстно-свободные
    - LL(k)
    - LR(k), LALR(k)
  - Задаваемые PEG
- Как задать язык
  - Конечный автомат
  - Магазинный автомат
  - PEG
- Синтаксический анализ
  - Определение, принадлежит ли цепочка языку
  - Построение дерева разбора

Дерево разбора — лишь цепочка в некотором языке



Дерево разбора — лишь цепочка в некотором языке



$$[.E[.E[.T[.P[.n]]]][.+][.T[.T[.P[.n]]][.*][.P[.n]]]]$$

Трансляция (перевод)

- **Трансляция** — преобразование некоторой входной строки в некоторую выходную
  - Σ — входной алфавит, Π — выходной алфавит. Трансляцией с языка $L_i \subseteq \Sigma^*$ на язык $L_o \subseteq \Pi^*$ называется отображение $\tau : L_i \to L_o$
- Построение дерева разбора — простейший пример трансляции
- Другие примеры трансляции
  - Вычисление значения арифметического выражения
  - Преобразование арифметического выражения из инфиксной записи в постфиксную
  - Преобразование программы на языке Java в байт-код
  - Компиляция программ
  - ...
- Фактически синтаксический анализ нужен для трансляции

Схемы синтаксически управляемой трансляции

Схема синтаксически управляемой трансляции — пятерка (N, Σ, Π, P, S)

- N — конечное множество нетерминальных символов
- Σ — конечный входной алфавит
- Π — конечный выходной алфавит
- $S \in N$ — стартовый нетерминал
- P — конечное множество правил трансляции вида $A \to \alpha, \beta$, где $\alpha \in (N \cup \Sigma)^*, \beta \in (N \cup \Pi)^*$
  - Вхождения нетерминалов в цепочку β образуют перестановку нетерминалов из цепочки α
  - Если нетерминалы повторяются больше одного раза, то их различают по индексам: $E \to E^l + E^r, E^r + E^l$

Выводимая пара в СУ-схеме

- Если $A \to (\alpha, \beta) \in P$, то $(\gamma A_i \delta,\ \gamma' A_i \delta') \Rightarrow (\gamma\alpha\delta,\ \gamma'\beta\delta')$
- Рефлексивно-транзитивное замыкание отношения $\Rightarrow$ называется отношением выводимости в СУ-схеме, обозначается $\overset{*}{\Rightarrow}$
- Трансляцией назовем множество пар $\{(\alpha, \beta) \,|\, (S, S) \overset{*}{\Rightarrow} (\alpha, \beta), \alpha \in \Sigma, \beta \in \Pi\}$
- СУ-схема называется простой, если во всех правилах $A \to (\alpha, \beta)$, нетерминалы в α и β встречаются в одном и том же порядке

Пример СУ-схемы

$$
\begin{array}{lcll}
E & \rightarrow & E+T & , \quad ET+ \\
 & | & T & , \quad T \\
T & \rightarrow & T*F & , \quad TF* \\
 & | & F & , \quad F \\
F & \rightarrow & n & , \quad n \\
 & | & (E) & , \quad E
\end{array}
$$

Пример СУ-схемы

$$
\begin{array}{lcll}
E & \to & E + T & , \quad ET+ \\
& | & T & , \quad T \\
T & \to & T * F & , \quad TF* \\
& | & F & , \quad F \\
F & \to & n & , \quad n \\
& | & (E) & , \quad E
\end{array}
$$

$(E, E) \Rightarrow (T, T) \Rightarrow (T * F, T\ F*) \Rightarrow (F * F, F\ F*) \Rightarrow (id * F, id\ F*) \Rightarrow$
$(id*(E), id\ E*) \Rightarrow (id*(E+T), id\ E\ T+*) \Rightarrow (id*(T+T), id\ T\ T+*) \Rightarrow$
$(id * (F + T), id\ F\ T + *) \Rightarrow (id * (id + T), id\ id\ T + *) \Rightarrow$
$(id * (id + F), id\ id\ F + *) \Rightarrow (id * (id + id), id\ id\ id + *)$

Обобщенные схемы синтаксически управляемой трансляции

Обобщенная схема синтаксически управляемой трансляции — шестерка $(N, \Sigma, \Pi, \Gamma, P, S)$

- Γ — конечное множество символов перевода вида $A_i, A \in N; i \in \mathbb{Z}$
- P — конечное множество правил трансляции вида $A \to \alpha, A_1 = \beta_1, \dots, A_n = \beta_n$, где $\alpha \in (N \cup \Sigma)^*$
  - $A_i \in \Gamma, 1 \leq i \leq n$
  - Каждый символ x, входящий в β_i, либо $x \in \Pi$, либо $x = B_k in \Gamma$, где $B \in \alpha$
  - Если α имеет более одного вхождения символа B, то каждый символ B_k во всех β соотнесен (верхним индексом) с конкретным вхождением B

Входной грамматикой назовем четверку (N, Σ, P', S), где $P = \{A \to \alpha \,|\, A \to \alpha, A_1 = \beta_1, \dots, A_n = \beta_n \in P\}$

Выход обобщенной СУ-схемы

- Для каждой внутренней вершины дерева, соответствующей нетерминалу A, с каждым A_i связывается одна цепочка
  - Такую цепочку назовем значением (трансляцией) символа A_i
- Каждое значение определяется подстановкой значений символов трансляции данного элемента $A_i = \beta_i$, определенных в прямых потомках вершины
- **Трансляцией**, определяемой данной схемой, назовем множество $\{(\alpha, \beta)\}$
  - α имеет дерево разбора в данной входной грамматике
  - β — значение выделенного символа S_k

Пример обобщенной СУ-схемы: дифференцирование

$$
\begin{array}{lcll}
E & \to & E + T & , \quad E_1 = E_1 + T_1 \\
 & & & , \quad E_2 = E_2 + T_2 \\
 & | & T & , \quad E_1 = T_1 \\
 & & & , \quad E_2 = T_2 \\
T & \to & T * F & , \quad T_1 = T_1 * F_1 \\
 & & & , \quad T_2 = T_1 * F_2 + T_2 * F_1 \\
 & | & F & , \quad T_1 = F_1 \\
 & & & , \quad T_2 = F_2 \\
F & \to & (E) & , \quad F_1 = (E_1) \\
 & & & , \quad F_2 = (E_2) \\
 & | & sin(E) & , \quad F_1 = sin(E_1) \\
 & & & , \quad F_2 = cos(E_1) * E_2 \\
 & | & cos(E) & , \quad F_1 = cos(E_1) \\
 & & & , \quad F_2 = -sin(E_1) * E_2 \\
 & | & x & , \quad F_1 = x \\
 & & & , \quad F_2 = 1
\end{array}
$$

Транслирующие грамматики

- КС-грамматика, терминальный алфавит которой разбит на два множество: входных и выходных символов
- **Транслирующая грамматика** — пятерка $(N, \Sigma_i, \Sigma_o, P, S)$
  - N — алфавит нетерминалов
  - Σ_i — алфавит входных терминалов
  - Σ_o — алфавит выходных терминалов
  - $S \in N$ — стартовый нетерминал
  - $P = \{A \to \alpha\}, \alpha \in (\Sigma_i \cup \Sigma_o \cup N)^*$ — множество правил вывода

Пример транслирующей грамматики

$$
\begin{array}{lcl}
E & \to & E + T\,\{+\} \\
  & | & T \\
T & \to & T * F\,\{*\} \\
  & | & F \\
F & \to & n\,\{n\} \\
  & | & (E)
\end{array}
$$

Пример транслирующей грамматики

$$
\begin{array}{lcl}
E & \rightarrow & E + T\,\{+\} \\
 & | & T \\
T & \rightarrow & T * F\,\{*\} \\
 & | & F \\
F & \rightarrow & n\,\{n\} \\
 & | & (E)
\end{array}
$$

$E \Rightarrow E + T\,\{+\} \Rightarrow T + T\,\{+\} \Rightarrow P + T\,\{+\} \Rightarrow n\,\{n\} + T\,\{+\} \Rightarrow$
$n\,\{n\} + T * P\,\{*\}\,\{+\} \Rightarrow n\,\{n\} + P * P\,\{*\}\,\{+\} \Rightarrow$
$n\,\{n\} + n\,\{n\} * P\,\{*\}\,\{+\} \Rightarrow n\,\{n\} + n\,\{n\} * n\,\{n\}\,\{*\}\,\{+\}$

Пример транслирующей грамматики

$$
\begin{array}{lcl}
E & \to & E + T\,\{+\} \\
 & | & T \\
T & \to & T * F\,\{*\} \\
 & | & F \\
F & \to & n\,\{n\} \\
 & | & (E)
\end{array}
$$

$E \Rightarrow E + T\,\{+\} \Rightarrow T + T\,\{+\} \Rightarrow P + T\,\{+\} \Rightarrow n\,\{n\} + T\,\{+\} \Rightarrow$
$n\,\{n\} + T * P\,\{*\}\,\{+\} \Rightarrow n\,\{n\} + P * P\,\{*\}\,\{+\} \Rightarrow$
$n\,\{n\} + n\,\{n\} * P\,\{*\}\,\{+\} \Rightarrow n\,\{n\} + n\,\{n\} * n\,\{n\}\,\{*\}\,\{+\}$

- Если вычеркнуть все выходные символы, получим $n + n * n$
- Если вычеркнуть все входные символы, получим $n\,n\,n + *$ — постфиксная запись выражения

Постфиксная транслирующая грамматика

- Если выходные символы встречаются только в конце правил, транслирующая грамматика называется постфиксной
- Это требование формально не выдвигается: транслирующие грамматики могут быть не постфиксными
- На практике постфиксные транслирующие грамматики удобнее

Атрибутная транслирующая грамматика

- Входной алфавит — алфавит лексем
  - Лексема характеризуется **типом** и **значением**
- Транслирующая грамматика описывает перевод только **типа** лексемы
  - Это существенно снижает выразительность формализма
- Для борьбы с этим недостатком предложены атрибутные грамматики
  - Модификация транслирующих грамматик, снабженная атрибутами
  - Выходные символы транслирующих грамматик — транслирующие символы
    - Нетерминалы, которые раскрываются в ε, и в момент раскрытия выполняют связанные с ними действие

Атрибут

Атрибут — дополнительные данные, ассоциированные с грамматическими символами

- Если X — символ, а a — его атрибут, то значение a в узле дерева, помеченном X, записывается как $X.a$
- Узлы дерева могут реализовываться как записи или объекты, а атрибуты — как поля
- Атрибуты могут быть любого типа
- Если в каждом узле дерева атрибуты уже вычислены, оно называется **аннотированным**
- Процесс вычисления этих атрибутов называется **аннотированием** дерева разбора.

Вычисление атрибутов не всегда возможно

$$A \to B \qquad \begin{aligned} A_s &= B_i \\ B_i &= A_s + 1 \end{aligned}$$

Синтезируемый атрибут, S-атрибутная грамматика

- Атрибут, значение которого зависит от значений атрибутов детей данного узла или от других атрибутов этого узла, называется **синтезируемым**
- Если в транслирующей грамматике используются только синтезируемые атрибуты, она называется **S-атрибутной грамматикой**
- Аннотирование дерева разбора S-атрибутной грамматики возможно путем выполнения семантических правил снизу вверх (от листьев к корню)

Пример S-атрибутной грамматики

$$
\begin{array}{lcllr}
S & \rightarrow & E & & S.val = E.val \\
\\
E_0 & \rightarrow & E_1 + T & \{ADD\ res = op_1 + op_2\} & ADD.op_1 = E_1.val \\
 & & & & ADD.op_2 = T.val \\
 & & & & E_0.val = ADD.res \\
\\
E & \rightarrow & T & & E.val = T.val \\
\\
T_0 & \rightarrow & T_1 + F & \{MUL\ res = op_1 * op_2\} & MUL.op_1 = T_1.val \\
 & & & & MUL.op_2 = F.val \\
 & & & & E_0.val = ADD.res \\
\\
T & \rightarrow & F & & T.val = F.val \\
F & \rightarrow & n & & F.val = n.val \\
F & \rightarrow & (E) & & F.val = E.val
\end{array}
$$

Наследуемый атрибут, L-атрибутная грамматика

- Атрибут, значение которого зависит только от атрибутов братьев узла или атрибутов родителей, называется **наследуемым**
- Если в транслирующей грамматике атрибуты узла зависят только от атрибутов родителей или братьев **слева**, она называется **L-атрибутной грамматикой**

Пример L-атрибутной грамматики

$$
\begin{array}{lcllr}
D & \to & TL & & L.inh = T.type \\
T & \to & int & & T.type = integer \\
T & \to & real & & T.type = real \\
\\
L_0 & \to & L_1, id & \{ENTRY\ type = (k, v)\} & L_1.inh = L_0.inh \\
 & & & & ENTRY.k = id.text \\
 & & & & ENTRY.v = L_0.inh \\
\\
L & \to & id & \{ENTRY\ type = (k, v)\} & ENTRY.k = id.text \\
 & & & & ENTRY.v = L.inh
\end{array}
$$