

6 Синтаксически управляемая трансляция

6.1 Введение

Сначала мы работали с задачей распознавания – да / нет. Потом нам понадобилось строить дерево разбора. Теперь нам и этого станет мало.

Заметим, что дерево разбора – это тоже цепочка в некотором языке (любое дерево кодируется как $root[child_1[...], child_2[...], ...]$).

Опр. 6.1 *Трансляция* - преобразование некоторой входной строчки в выходную. $\tau : L_i \Rightarrow L_o, L_i \in \Sigma_i^*, L_o \in \Sigma_o^*$

Примеры:

- Вычисление арифметического выражения
- Преобразование арифметического выражения
- Любое преобразование кода в компиляторе
- Восстановление дерева по коду Прюфера

То есть, фактически, синтаксический анализ – это трансляция¹¹.

Зачем же резать модели трансляции, если у нас есть ЯП общего назначения (Тьюринг-полный)? В теории, чтобы можно было гарантировать некоторые свойства транслятора.

Опр. 6.2 (Нестрогое) *Синтаксически управляемая трансляция* (англ. *Syntax-directed translation, SDT, CYT*) – преобразование текста в последовательность команд через добавление таких команд в правила грамматики

А почему бы не разобрать слово, а потом обойти полученное дерево разбора, и посчитать? Действительно, зачастую в алгоритмах преобразования различных графоструктурированных данных (например, в оптимизационных проходах компилятора) именно так и поступают. Однако, существует минимум 2 мотивации так не делать:

- Экономия памяти – как минимум, мы можем не хранить дерево разбора памяти. Проблема – больше историческая.
- Актуальная проблема: есть логика выражений, в которой мы что-то делаем с атрибутами; если мы запишем дерево, а потом сделаем visitor по дереву, нам снова придется описать всю логику работы внутри обходчика еще раз – получается дублирование функциональности

А СУТ позволяет логику и синтаксис описать в одном месте.

¹¹В задачах обобщения на графы это не всегда так – нас могут интересовать пересечения, пустота, etc

6.2 Атрибутные грамматики

Расширим понятие грамматики атрибутами и семантическими действиями.

- Пусть каждый символ в $X \in \Sigma \cup N$ в грамматике может иметь атрибуты, которые содержат данные¹². Это может быть *key : value* словарь, структура или union, не принципиально. Пусть, для определённости, для X с атрибутом t обращение к атрибуту может выглядеть как $X.t$, а ко всему атрибутам $X.attr$. Грамматика, содержащая такие «расширенные» символы, называется атрибутной грамматикой.
- Дополним атрибутную грамматику $G = (\Sigma, N, P, S)$ семантическими действиями – множеством функций $A - G = (\Sigma, N, P, S, A)$, где $\forall a \in A \exists p \in P : a(\{l.attr : l \in L\}, \{r.attr : r \in R\})$, l, r – всевозможные символы в соответственно левой и правой частях правила p , вызывается тогда и только тогда, когда применяется правило p . Говорят, что такая грамматика задаёт схему трансляции. Далее будем рассматривать только КС-грамматики, поэтому $|L| = 1$.

6.2.1 Типы атрибутов

Типы атрибутов вводятся с точки зрения действия над ними семантических операций в ходе разбора.

Опр. 6.3 *Синтезированные атрибуты – атрибуты, вычисляемые из правых частей правил.*

Синтезированные атрибуты содержат информацию, подтягиваемую вверх по ходу восходящего разбора (либо возврата из рекурсивного спуска, etc), в общем, вычисляются по мере восхождения от терминалов к корню дерева разбора: в момент сворачивания по некоторому правилу, мы знаем атрибуты правой части, но ещё не знаем атрибуты левой. Они-то и «синтезируются» на основе атрибутов правой части.¹³

Пример: вычисления на синтезируемых атрибутах:

```
E -> E+T { E.val = E.val + T.val then print (E.val)}  
E -> T   { E.val = T.val}  
T -> T*F { T.val = T.val * F.val}  
T -> F   { T.val = F.val}  
F -> Id  {F.val = id}
```

Другие примеры с синтезируемыми атрибутами были рассмотрены на паре про Flex/Bison.

Опр. 6.4 *Наследуемые атрибуты – атрибуты, вычисляемые из соседних либо родительских вершин дерева разбора.*

¹²Обычно такие атрибуты могут включать в себя тип переменной, значение выражения, и т.п.

¹³В этом месте становится понятно, почему Bison работает именно на синтезированных атрибутах, и вычисления происходят именно так

Пример: присвоение типа переменным при создании (`int a,b,c;`). Пример грамматики составить самостоятельно.

6.3 Более общая формулировка

Возьмем понятие трансляции из прошлого подраздела. Введем СУ схему как:

Опр. 6.5 *СУТ – это пятерка (Σ, N, P, S, Π) , где*

- Π – выходной алфавит
- P – конечное множество правил вида $A \rightarrow \alpha, \beta, \alpha \in (N \cup \Sigma)^*, \beta \in (N \cup \Pi)^*$,
- вхождения нетерминалов в цепочку β образуют перестановку нетерминалов из цепочки α
- Если нетерминалы повторяются более одного раза, их различают по индексам

В таком виде мы можем задавать, как преобразовывать цепочку. Получается, СУТ-схема задает синхронный вывод 2 цепочек.

- Если $A \rightarrow (\alpha, \beta) \in P$, то $(\gamma A^i \delta, \gamma' A^i \delta') \Rightarrow (\gamma \alpha^i \delta, \gamma' \beta^i \delta')$
- Рефлексивно-транзитивное замыкание отношения \Rightarrow называется отношением выводимости \Rightarrow^*
- Трансляцией называется множество пар $\{(\alpha, \beta) | (S, S) \Rightarrow^* (\alpha, \beta), \alpha \in \Sigma^*, \beta \in \Pi^*\}$
- Схема называется простой, если в любых правилах вида $A \rightarrow (x, y)$ нетерминалы x, y встречаются в одном и том же порядке.
- Схема называется однозначной, если не существует двух правил $A \rightarrow a, b, A \rightarrow a, c$, таких, что b, c – разные символы.

Т. 6.1 *Выходная цепочка однозначной СУТ-схемы может быть сгенерирована при одностороннем выводе.*

Также существует понятие обобщенной СУТ-схемы.

Там, фактически, параллельно строятся два дерева разбора:

Для каждой внутренней вершины дерева, соответствующей нетерминалу

A , с каждым A_j связывается цепочка (трансляция) символа A_j

TODO: дописать

$E \rightarrow E + T$, $E_1 = E_1 + T_1$
$ $	T , $E_2 = E_2 + T_2$
$T \rightarrow T * F$, $E_1 = T_1, E_2 = T_2$
$ $	F , $T_1 = T_1 * F_1$
$F \rightarrow (E)$, $T_2 = T_1 * F_2 + T_2 * F_1$
$ $	$\sin(E)$, $T_1 = F_1, T_2 = F_2$
$ $	$\cos(E)$, $F_1 = (E_1)$
$ $	x , $F_2 = (E_2)$
$ $	n , $F_1 = \sin(E_1)$
	, $F_2 = \cos(E_1) * E_2$
	, $F_1 = \cos(E_1)$
	, $F_2 = -\sin(E_1) * E_2$
	, $F_1 = x, F_2 = 1$
	, $F_1 = n, F_2 = 0$

Рис. 6: Обобщенная СУТ, позволяющая описать простейшее дифференцирование

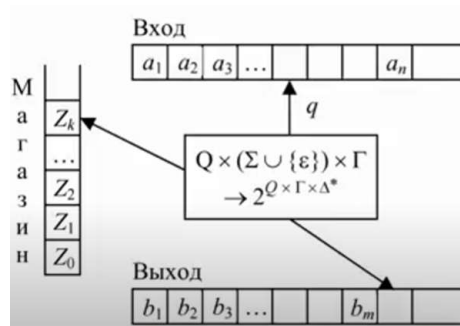


Рис. 7: МП-преобразователь

6.4 Магазинный преобразователь

Под этим лежит (может лежать) формальный вычислитель – магазинный преобразователь – выходная лента + МП автомат, который на каждый шаг на выходную ленту что-нибудь печатает.

Доказывается, что, так как МП-преобразователь не может что-нибудь переставить на своем стеке, то класс трансляций МП-автомата не шире класса простых СУ-трансляций.

Также доказывается, что по любой простой СУ-схеме можно построить МП-преобразователь, то есть классы совпадают.