ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА МЕТОДАМИ СУПЕРКОМПИЛЯЦИИ

Автор: Сергей Головань

Руководитель: Александр Коновалов

МГТУ им. Н.Э. Баумана

2018

Постановка задачи

Для заданной входной грамматики требуется построить два набора тестов: позитивные и негативные, проверяющие всю логику работы синтаксического анализатора.

В качестве входных рассматриваются **LL(1)-грамматики**.

Пример 1. Арифметические выражения | 3

Пример. Грамматика арифметических выражений:

$$E \quad ::= TE';$$

$$E' \quad ::= '+' \; TE' \mid \varepsilon \; ;$$

$$T \quad ::= FT';$$

$$T' \quad ::= \; '*' \; FT' \mid \varepsilon \; ;$$

$$F \; ::= n \mid '(' \; E \; ')';$$

Позитивные тесты:

«n + n * n»,
«(n + (n * (n)))»

Негативные тесты:

«n + +», «n n», «(n *)», ...

LL(1)-грамматика

КС-грамматика $\boldsymbol{G}$ является $\boldsymbol{LL(1)}$,

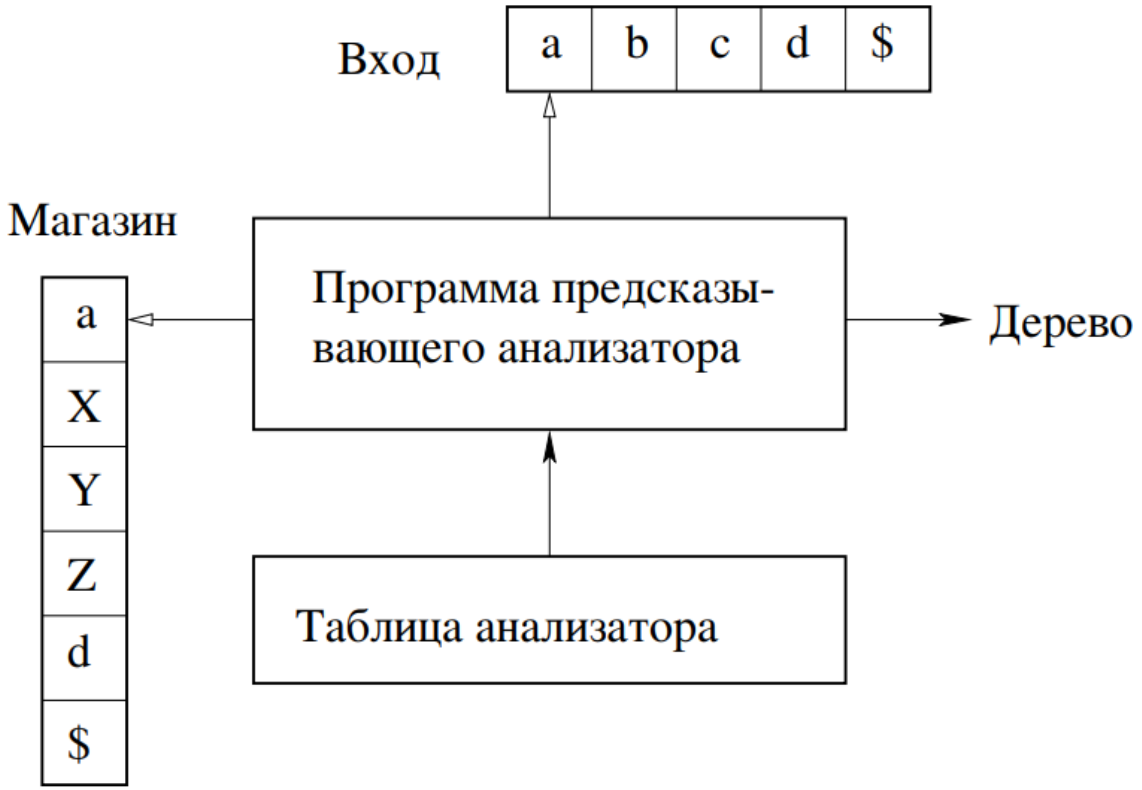
если для любого правила $\boldsymbol{A \Rightarrow u \mid v}$ выполняется:

1. $FIRST(u) \cap FIRST(v) = \emptyset$
2. *Если* $v \Rightarrow* \varepsilon$, *то* $FIRST(u) \cap FOLLOW(A) = \emptyset$

Для таких грамматик возможно построить таблицу предсказывающего анализатора

LL(1)-грамматика

LL(1)-грамматики распознаются предсказывающим анализатором – автоматом с магазинной памятью.



LL(1)-грамматика

$ERROR$ – недопустимый переход, **синтаксическая ошибка**.

Пример. Грамматика арифметических выражений:

	+	*	n	(	)	$
E	$error$	$error$	$\boldsymbol{T\ E'}$	$\boldsymbol{T\ E'}$	$error$	$error$
E'	$+\ \boldsymbol{T\ E'}$	$error$	$error$	$error$	$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$
T	$error$	$error$	$\boldsymbol{F\ T'}$	$\boldsymbol{F\ T'}$	$error$	$error$
T'	$\boldsymbol{\varepsilon}$	$*\ \boldsymbol{F\ T'}$	$error$	$error$	$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$
F	$error$	$error$	$\boldsymbol{n}$	$(\ \boldsymbol{E}\ )$	$error$	$error$

LL(1)-грамматика

Расширенное восстановление при ошибках:

$a \in FOLLOW(N)$: $N \to \varepsilon$, иначе $N \to aN$ (кроме \$)

Пример. Грамматика арифметических выражений:

	+	*	n	(	)	\$
E	$+\,E$	$*\,E$	$T\,E'$	$T\,E'$	ε	ε
E'	$+\,T\,E'$	$*\,E'$	$n\,E'$	$(\,E'$	ε	ε
T	ε	$*\,T$	$F\,T'$	$F\,T'$	ε	ε
T'	ε	$*\,F\,T'$	$n\,T'$	$(\,T'$	ε	ε
F	ε	ε	n	$(\,E\,)$	ε	ε

Критерий полноты тестирования

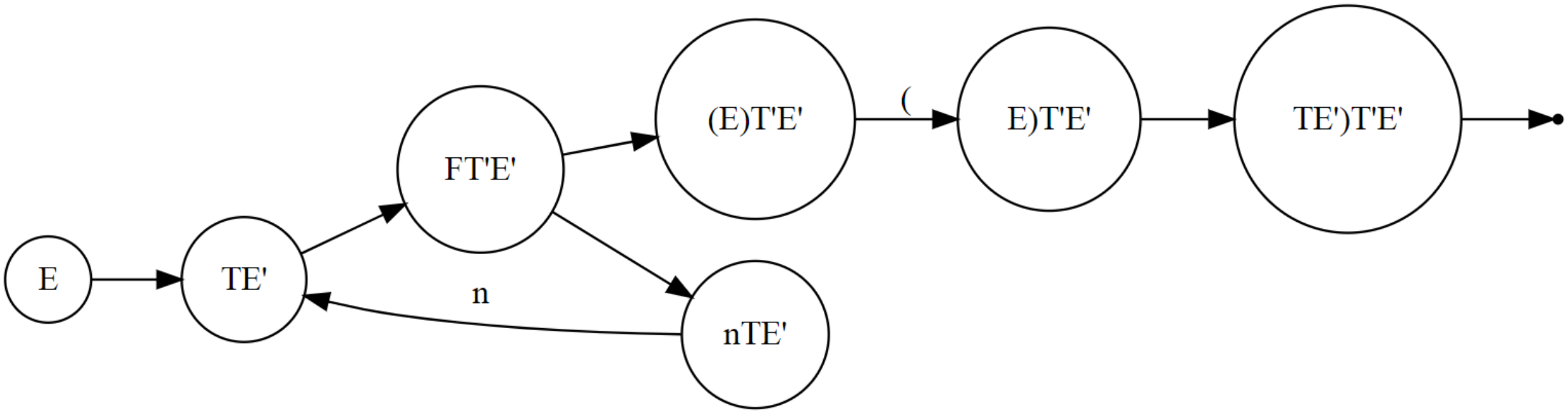
Критерии, по которым проводится классификация всех возможных вариантов выполнения программы с точки зрения проверки правильности программы – **критерии полноты тестирования**.

Критерий полноты тестирования (первое приближение):

Тестирование синтаксического анализатора является полным тогда и только тогда, когда в процессе вывода слов из тестового набора окажутся посещены все ячейки таблицы предсказывающего разбора.

Граф состояний

Рассматривая процесс работы анализатора, можно представить его в виде графа (в общем случае бесконечного):



Граф состояний. Пример

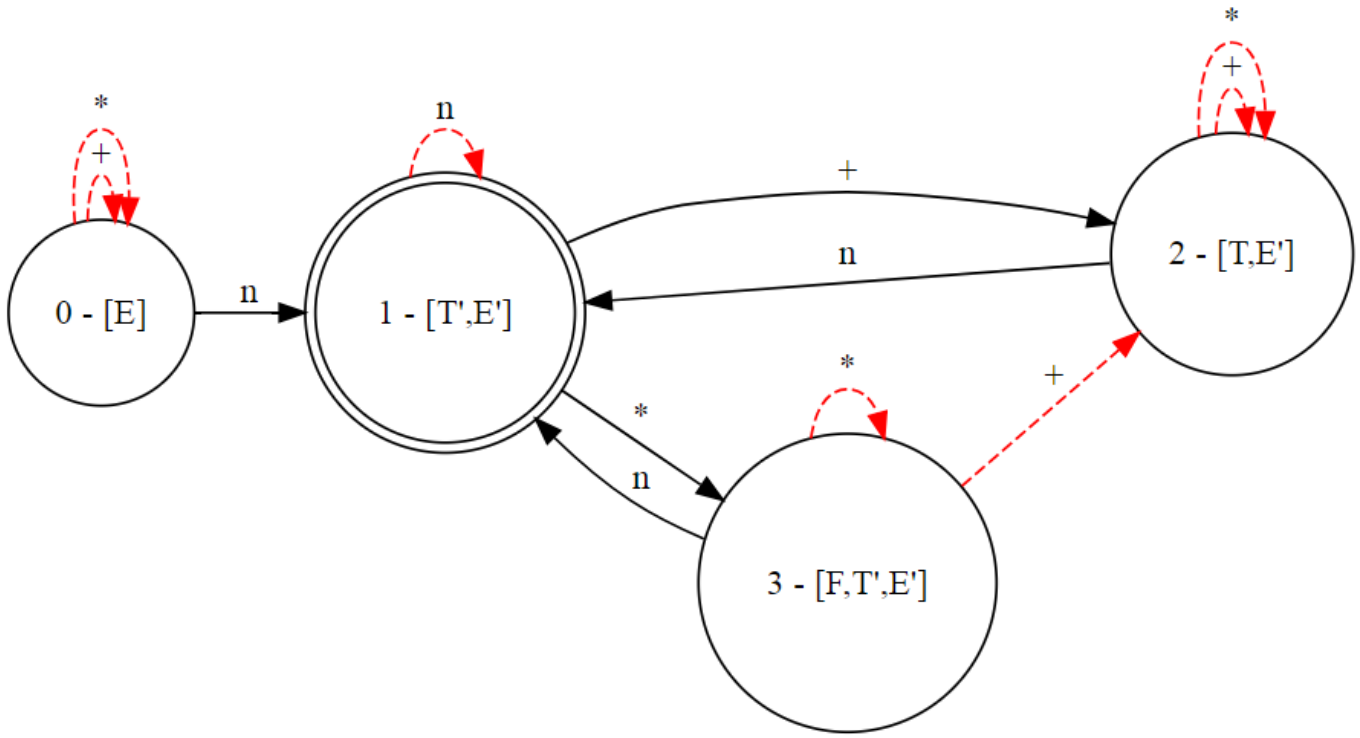
$$E \quad ::= TE';$$

$$E' \quad ::= '+' \; TE' \mid \varepsilon \; ;$$

$$T \quad ::= FT';$$

$$T' \quad ::= '*' \; FT' \mid \varepsilon \; ;$$

$$F \; ::= n$$



Суперкомпиляция

Процесс анализа и преобразования программ, основанный на следующих действиях:

1. Построение дерева конфигураций
2. Свертка дерева в граф конфигураций (**вложение**, **обобщение**)
3. Построение остаточной программы

Применительно к синтаксическому анализу, **граф конфигураций** есть **свернутый граф состояний** автомата с магазинной памятью**.**

Суперкомпиляция

Свертка графа конфигураций.

Вложение – выделение уже вычисленной ранее части данной конфигурации (элементов стека).

Обобщение – сведение к «более общей» конфигурации.

Let-вершина – вершина графа конфигураций, из которой исходят два ребра, каждое из которых развивается согласно вложению или обобщению.

Свертка графа конфигураций. **Вложение.**

Пусть на некотором шаге построения в некоторой вершине $\boldsymbol{U}$ имеется непустой стек $\boldsymbol{K} = [\boldsymbol{A}, \boldsymbol{B}]$, где $\boldsymbol{A}, \boldsymbol{B} \in (N \cup T)^+$

Если среди предков $\boldsymbol{U}$ имеется вершина $\boldsymbol{W}$ со стеком $[\boldsymbol{A}]$, то имеется **вложение**, а вершина $\boldsymbol{U}$ заменяется **let-вершиной**, одна ветвь которых ссылается на $\boldsymbol{W}$, а вторая - на отдельное развитие стека $[\boldsymbol{B}]$.

Суперкомпиляция | 14

Свертка графа конфигураций. **Обобщение.**

Пусть на некотором шаге построения в некоторой вершине $\boldsymbol{U}$ имеется непустой стек $\boldsymbol{K} = [\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]$, где $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in (N \cup T)^+$

Если среди предков $\boldsymbol{U}$ имеется вершина $\boldsymbol{W}$ со стеком $[\boldsymbol{A}, \boldsymbol{C}]$, то требуется **обобщение**, то есть вычисленный подграф с корнем в $\boldsymbol{W}$ заменяется на **let-вершину**, развитие которой начинается с вершин со стеками $[\boldsymbol{A}]$ и $[\boldsymbol{C}]$.

С учетом сказанного ранее, переформулируем **критерий полноты тестирования**:

Тестирование синтаксического анализатора является полным тогда и только тогда, когда в процессе вывода слов из тестового набора окажутся посещены все ребра графа конфигураций.

Пример 1. Арифметические операторы | 16

$$S \quad ::= E\ ';';$$

$$E \quad ::= TE';$$
$$E' \quad ::= '+'\ TE' \mid \varepsilon\ ;$$

$$T \quad ::= FT';$$
$$T' \quad ::= '*'\ FT' \mid \varepsilon\ ;$$

$$F \ ::= n \mid '('\ E\ ')';$$

Сгенерированный граф в файле:
oper.svg

Позитивные тесты:
$n + n\ * n\ ;$
$(n + n\ * n);$
$(\ (\ n + (\ n\ * (n) + (n) * (n))\ )\ );$

Пример 2. JSON


```
object ::= '{' pair (',' pair)* '}';
json ::= object | array;
pair ::= STRING ':' value ;
array ::= '[' value (',' value)* ']';
value ::= STRING |
        NUMBER |
        object |
        array |
        ‘true’|  ‘false'|
        null';
```

Сгенерированный граф в файле:
json.svg

Заключение

Результатом выполнения данной работы является генератор тестов для синтаксических анализаторов LL(1)-грамматик.

В дальнейшем возможно применить использованные техники для более широких классов грамматик, накладывая дополнительные начальные условия и корректируя построение и обход графа конфигураций.

Заключение

Благодарю за внимание!