СУПЕРКОМПИЛЯЦИЯ LL(1)-ГРАММАТИК

Автор: Сергей Головань

Руководитель: Александр Коновалов

МГТУ им. Н.Э. Баумана

Совместное рабочее совещание
ИПС имени А. К. Айламазяна РАН
и
МГТУ имени Н. Э. Баумана

5 июня 2018 г.

Постановка задачи

Необходимо построить **конечный набор тестов** (строк языка) для заданной **КС-грамматики**, проверяющей **всю** логику работы синтаксического анализатора.

В частности, далее будут рассмотрены **LL(1)-грамматики**.

КС-грамматика

Контекстно свободная грамматика $\boldsymbol{G}$ – кортеж $< \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{S} >$, где:

$\boldsymbol{N}$ – множество нетерминальных символов

$\boldsymbol{T}$ – множество терминальных символов

$\boldsymbol{P}$ – набор правил вывода: $\boldsymbol{A} \rightarrow \boldsymbol{u}$, *где* $\boldsymbol{A} \in \boldsymbol{N}, \boldsymbol{u} \in (\boldsymbol{N} \cup \boldsymbol{T})^*$

$\boldsymbol{S}$ – стартовое правило вывода (аксиома)

КС-грамматика

Пример. Грамматика арифметических выражений:

$$E \quad ::= TE';$$
$$E' \quad ::= '+' \; TE' \mid \varepsilon \; ;$$
$$T \quad ::= FT';$$
$$T' \quad ::= '*' \; FT' \mid \varepsilon \; ;$$
$$F \; ::= n \mid '(' \; E \; ')';$$

Пример выводимой цепочки:

$$\boldsymbol{n} + (\boldsymbol{n} \; * \boldsymbol{n} \; * \boldsymbol{n}) \; * \boldsymbol{n}$$

КС-грамматика

Множества $\boldsymbol{FIRST}$ и $\boldsymbol{FOLLOW}$ связаны с грамматикой языка и позволяют построить **таблицу предсказывающего разбора**.

$\boldsymbol{FIRST}(\boldsymbol{u})$ - множество терминалов, с которых начинаются цепочки, выводимые из $\boldsymbol{u}$.

$\boldsymbol{FIRST}(\boldsymbol{X})$ - множество терминалов таких, что существует вывод вида $\boldsymbol{S} \Rightarrow * \boldsymbol{uXav}$.

LL(1)-грамматика

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* $FIRST(u) \cap FOLLOW(A) = \emptyset$

Предварительно в грамматике устраняют **левую рекурсию** и выполняют **левую факторизацию**.

LL(1)-грамматика

Используя множества $\boldsymbol{FIRST}$ и $\boldsymbol{FOLLOW}$, строим таблицу предсказывающего разбора.

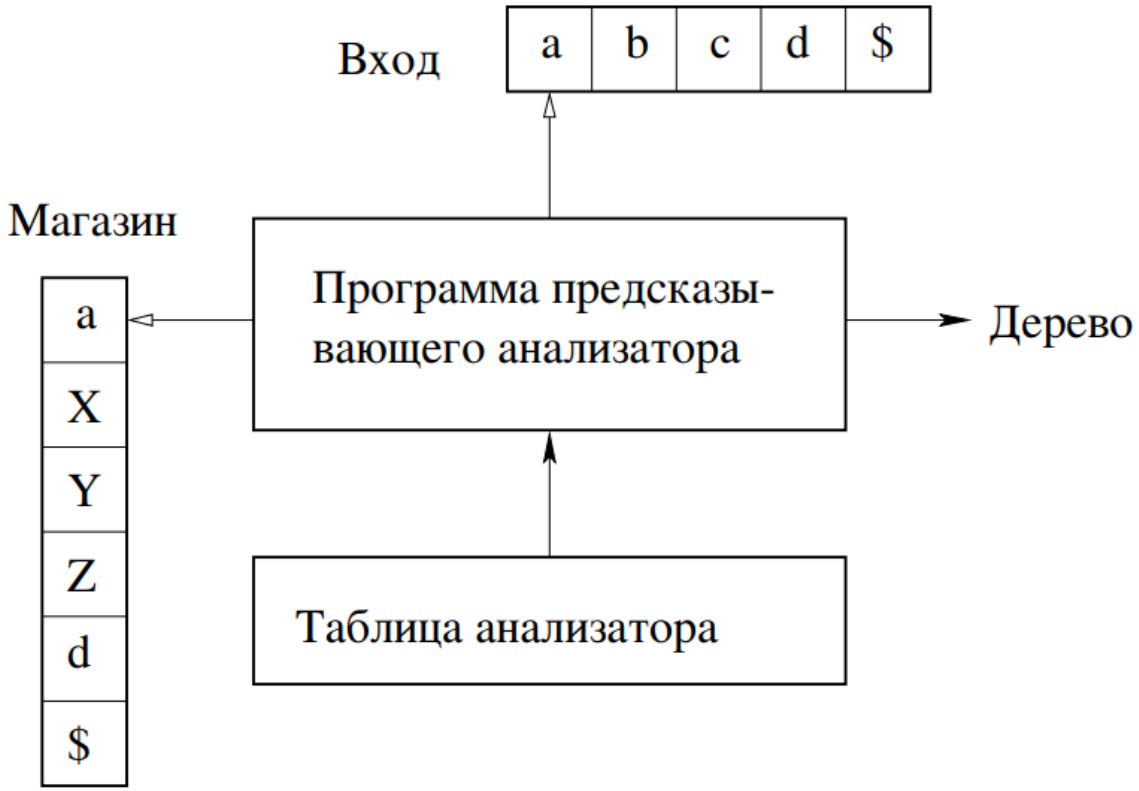
$\boldsymbol{ERROR}$ – недопустимый переход, **синтаксическая ошибка**.

Пример. Грамматика арифметических выражений:

	'+'	'*'	n	'('	')'	$
E	ERROR	ERROR	T E′	T E′	ERROR	ERROR
E′	'+' T E′	ERROR	ERROR	ERROR	ε	ε
T	ERROR	ERROR	F T′	F T′	ERROR	ERROR
T′	ε	'*' F T′	ERROR	ERROR	ε	ε
F	ERROR	ERROR	n	'(' E ')'	ERROR	ERROR

LL(1)-грамматика

Структура предсказывающего анализатора:



Критерий полноты тестирования | 9

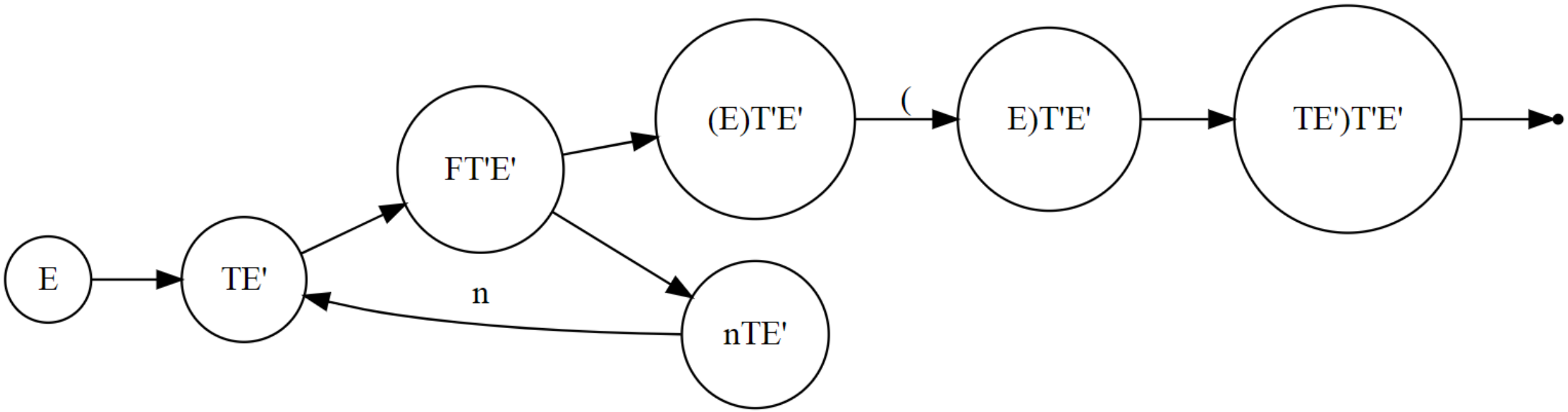
Критерии, по которым проводится классификация всех возможных вариантов выполнения программы с точки зрения проверки правильности программы – **критерии полноты тестирования**.

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

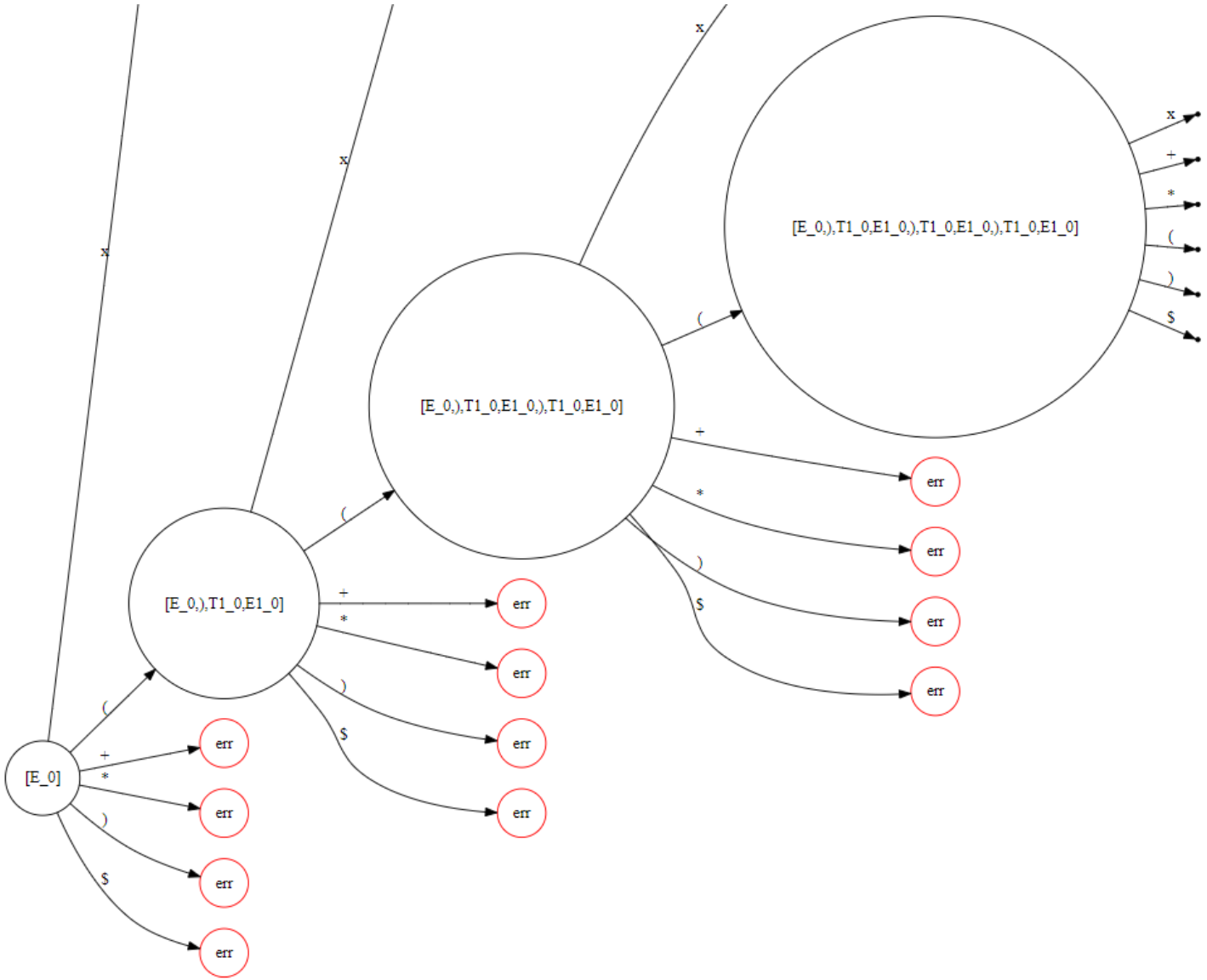
Состояние – вершина в графе, описываемая состоянием магазина.

Состояния **A** и **B эквивалентны,** если их магазины совпадают.

Транзитный переход – переход магазина из состояния **A** в состояние **B**, при котором входной символ не потребляется.

Выполняя удаление транзитных узлов в графе, а также образуя циклы к эквивалентным вершинам, получаем **граф состояний**.

Граф состояний



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 $[\boldsymbol{B}]$.

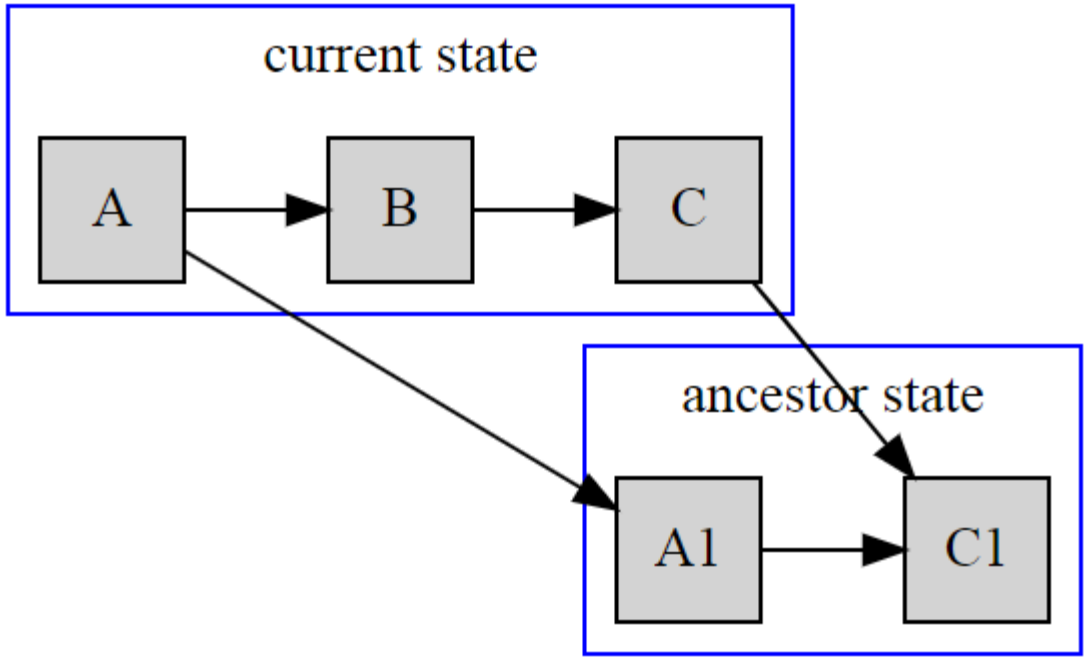
Суперкомпиляция | 16

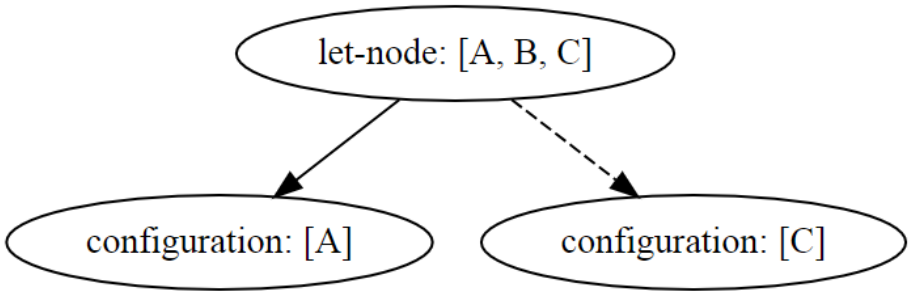
Свертка графа конфигураций. **Обобщение.**

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 $[\boldsymbol{C}]$.

Свертка графа конфигураций. **Обобщение.**





Суперкомпиляция

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

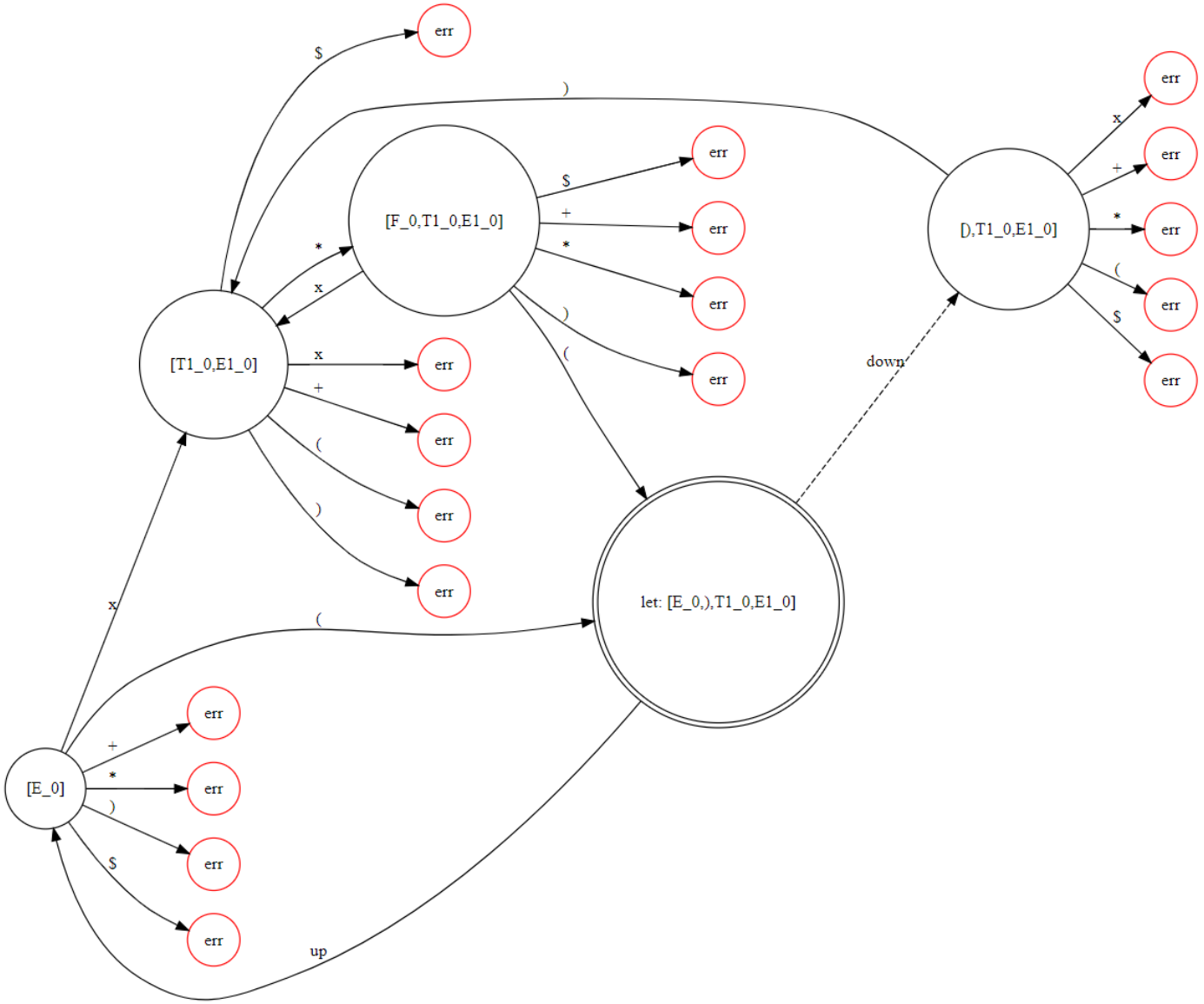
Тестирование синтаксического анализатора является полным тогда и только тогда, когда в процессе вывода слов из тестового набора окажутся посещены все ребра графа конфигураций.

Пример 1. Арифметические выражения | 19

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

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

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



Пример 2. Арифметические операторы | 20

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

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

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

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

Сгенерированный граф в файле:
operator.svg

Пример 3. 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

Дальнейшее развитие:

1. Восстановление при ошибках
2. Генерация остаточной грамматики
3. Поиск минимального набора тестов

Заключение

Благодарю за внимание!