Диагностика и нейтрализация синтаксических ошибок

Согласно заданию на курсовую работу, необходимо реализовать нейтрализацию синтаксических ошибок, используя метод Айронса.

Метод Айронса

Суть метода Айронса заключается в следующем:

При обнаружении ошибки (во входной цепочке в процессе разбора встречается символ, который не соответствует ни одному из ожидаемых символов), входная цепочка символов выглядит следующим образом: Tt, где T – следующий символ во входном потоке (ошибочный символ), t – оставшаяся во входном потоке цепочка символов после T. Алгоритм нейтрализации состоит из следующих шагов:

1. Определяются недостроенные кусты дерева разбора;

2. Формируется множество L – множество остаточных символов недостроенных кустов дерева разбора;

3. Из входной цепочки удаляется следующий символ до тех пор, пока цепочка не примет вид Tt, такой, что U => T, где U ∈ L, то есть до тех пор, пока следующий в цепочке символ T не сможет быть выведен из какого-нибудь из остаточных символов недостроенных кустов.

4. Определяется, какой из недостроенных кустов стал причиной появления символа U в множестве L (иначе говоря, частью какого из недостроенных кустов является символ U).

Таким образом, определяется, к какому кусту в дереве разбора можно «привязать» оставшуюся входную цепочку символов после удаления из текста ошибочного фрагмента.

Метод Айронса для контекстно-свободной грамматики

Разрабатываемый синтаксический анализатор построен на базе контекстно-свободной грамматики G[‹Function›]. Реализация алгоритма Айронса для данной грамматики имеет следующие особенности:

Дерево разбора для КС-грамматики функций JavaScript включает:

Корневой узел ‹Function›

Поддеревья для параметров, тела функции и выражений

Рекурсивные ветви для выражений (‹Expression› → ‹Term› + ‹Expression›)

Дерево разбора представлено на рисунке 2.

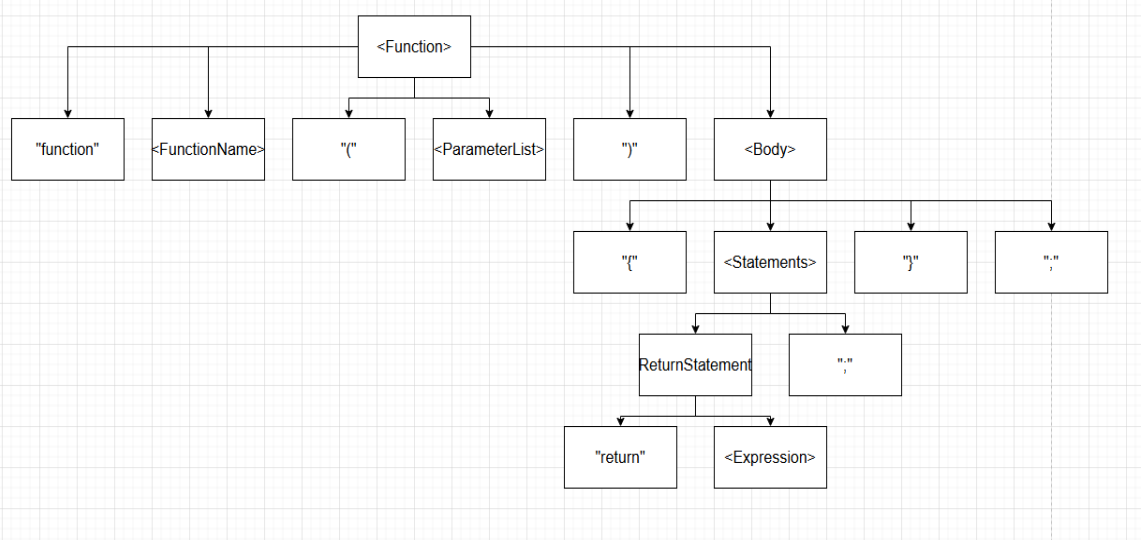

Рисунок 2 – Структура дерева разбора

При возникновении синтаксической ошибки аиксируется текущий недостроенный куст (например, ‹Expression› при отсутствии правого операнда после +). Так же формируется множество L ожидаемых терминалов:

Для ‹Expression›: {+, -,), ;}

Для ‹ParameterList›: {",",)}

Для ‹ReturnStatement›: {‹Expression›, ;}