Григорьев Семён. гр. №461
Научнай руководитель: Лукичёв Александр Сергеевич

Разработка прототипа генератора синтаксических анализаторов для произвольных КС грамматик
Прототип генератора синтаксических GLR анализаторов
Прототип генератора синтаксических GLR анализаторов для платформы .NET
GLR-анализ

Задачи автоматизированного реинжиниринга программ выдвигают особые требования к генераторам синтаксических анализаторов.

Для устаревшего языка сложно (а зачастую и невозможно) задать однозначную контекстно-свободную грамматику. Необходимо существенно преобразовать его спецификацию, приводимую в документации, чтобы получить такую грамматику, но при этом она перестает быть сопровождаемой. Поэтому устаревший язык обычно задается с помощью неоднозначной контекстно-свободной грамматики.

При решении задач реинжиниринга частого требуются преобразования уже существующей грамматики. При этом, зачастую, изменение одного правила приводит к появлению десятков конфликтов в грамматике, которые необходимо разрешать вручную. Это требует большого количества времени.

Как вариант решения этих задач предлагается использовать GLR грамматики и соответствующие инструменты построения анализаторов. Действительно, GLR-алгоритм разрешает неоднозначности в грамматике на уровне концепции. По этому задание спецификации трансляции становится проще, требует меньше времени. Получившийся код компактнее и сопровождаемее.

Главным достоинством GLR-алгоритма является обработка неоднозначных грамматик. Анализатор, построенный по неоднозначной грамматике с помощью данного алгоритма, в результате разбора строит не единственное дерево, а несколько деревьев - лес, дальнейшая работа с которым строится исходя из особенностей решаемой задачи.

Работу GLR-алгоритма можно рассматривать как параллельное исполнение набора LR-анализаторов. При этом данный набор дополняется процедурой управления стеками, оптимизирующей представление стеков путем их „склеивания“ и „расклеивания“, что позволяет хранить и строить параллельные выводы в рамках одного LR-анализатора, лишь в моменты их различия добавляя параллельный анализатор.

Стоит отметить, что по производительности такой анализатор, являясь некоторой "надстройкой" над LR-анализатором, незначительно ему уступает. На сегодняшний день в соотношении производительность/класс разбираемых языков GLR-алгоритм выглядит наиболее предпочтительно.

Удобным способом формального определения грамматики, элементов и атрибутов языка программирования является расширенная нормальная форма Бэкуса-Наура (Extended Backus–Naur form, EBNF). Поэтому инструмент должен работать с расширенными контекстно-свободными грамматиками.

При работе с инструментом пользователь ожидает получить результат описанный в терминах заданной им грамматики. Это выдвигает дополнительные требования к алгоритму. В случае, если входная грамматика была каким-либо образом преобразована, например с целью раскрыть конструкции EBNF, то появляется необходимость в построении "обратного" преобразования. Это преобразование должно "перевести" результат обратно в термины входной грамматики. Такие преобразования требуют дополнительных ресурсов и усложняют инструмент. Поэтому наиболее предпочтительными является алгоритмы, позволяющие обойтись без дополнительных преобразований грамматики.

Целью работы является разработка прототипа генератора синтаксических анализаторов, удовлетворяющего выдвинутым выше требованиям и обладающего следующими свойствами:

- Поддержка произвольных (в том числе неоднозначных) контекстно-свободных грамматик.
- Поддержка расширенных регулярных выражений в правых частях правил.
- Поддержка атрибутных грамматик.

Удобным для реализации на функциональном языке оказался рекурсивно-восходящий алгоритм. Он и был взят за основу. Основная его идея заключается в том, что поведение стека эмулируется стеком вызова рекурсивных функций. Алгоритм был модифицирован таким образом, чтобы позволить работать с EBNF конструкциями в правых частях правил без преобразования входной грамматики.

Для поддержки атрибутных грамматик будет применён алгоритм основанный на вычислении над лесом вывода. Основная его идея состоит в том, что сначала строится лес вывода, а потом над ним запускается вычисление соответствующих атрибутов (интерпретация леса вывода).

При применении данного алгоритма возникает ряд трудностей с поддержкой EBNF грамматик без преобразования. Сложность состоит в том, что при интерпретации дерева к каждому узлу применяется функция, аргументами которой являются непосредственные сыновья этого узла. В случае, когда EBNF конструкции не преобразовываются, непосредственные сыновья узла, соответствующего правилу грамматики, образуют строку, принадлежащую языку, заданному регулярным выражением в правой части правила. Для того чтобы правильно вычислить аргументы применяемой функции, необходимо обладать дополнительной информацией о выводе данной строки. Эту задачу так же предстоит решить.

В качестве основного языка реализации выбран F#, целевой язык так же F#. В качестве фронтенда, обладающего мощным и удобным языком спецификации трансляции выбран YARD, разрабатываемый Я.А. Кириленко.