Синтаксический анализ исходного кода, содержащего инструкции препроцессора

Савенко Сергей Андреевич

научный руководитель

Игнатов Сергей Сергеевич

Академический университет

2014 г.

Синтаксический анализ

Способы реализации парсера

- Вручную
- Парсер-генераторы (ANTLR, Bison, JavaCC и пр.)
- Парсер-комбинаторы

Текстовый препроцессор

Типичная функциональность

- Включение файлов
- Блоки условной компиляции
- Макроопределения

Сложности при обработке непрепроцессированного кода

- 2 языка
- Множество конфигураций препроцессора

Обработка непрепроцессированного кода

Существующие подходы

1. Обработка конкретных конфигураций препроцессора
2. Эвристическая обработка
3. Обработка всех возможных конфигураций препроцессора

Цели и задачи

Цель работы

Создать инструмент, позволяющий производить синтаксический анализ исходного кода, содержащего инструкции препроцессора, используя грамматику языка программирования и интерпретатор инструкций препроцессора

Задачи работы

1. Изучить существующие решения
2. Разработать и реализовать алгоритм синтаксического анализа непрепроцессированного кода
3. Проверить разработанную библиотеку, сравнив её с аналогами

Синтаксический анализ непрепроцессированного кода

Частичное препроцессирование

Используется подход применяемый в аналогах:

- Таблица макроопределений
- Задача достигающих определений
- Граф из последовательностей лексем для ветвей условной компиляции

Синтаксический анализ графа лексем

- Модифицированный алгоритм Earley
- Обработка условий наличия лексем
- Граф из узлов, соответствующих символам грамматики, и узлов ветвления

Особенности реализации

- Описание грамматики языка в БНФ
- Поддержка широкого класса языков (все КС-языки)
- Автоматическое ветвление и слияние ветвей разбора

Результаты

Для оценки результатов было описано подмножество языка Erlang для разработанной библиотеки и для аналогичного решения TypeChef

Свойства разработанного решения

- Простота в использовании
  - Отсутствие необходимости модифицировать грамматику (поддержка более широкого класса языков)
  - Отутствие необходимости описания ветвлений и слияний в грамматике
  - Лаконичность: ~100 строк на Java против ~200 строк на Scala
- Сравнимая скорость работы