ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ  
И ИНФОРМАТИКИ»

**Расчетно-графическая работа**

по дисциплине  
«Компиляторные технологии»

на тему

**Разработка семантического анализатора для языка программирования Cool**

|  |  |
| --- | --- |
| Выполнил студент | Пухов Максим Станиславович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИС-242 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | профессор д.т.н. М.Г. Курносов |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2025

ВВЕДЕНИЕ

Семантический анализ является важным этапом процесса компиляции, обеспечивающим корректность программы с точки зрения типов данных, областей видимости и других семантических правил. В данной работе представлена реализация семантического анализатора для учебного объектно-ориентированного языка Cool (Classroom Object-Oriented Language), который используется для изучения принципов разработки компиляторов.

1 Язык программирования Cool

* 1. Грамматика языка: формальное описание и синтаксические конструкции

Язык программирования Cool (Classroom Object-Oriented Language) представляет собой объектно-ориентированный язык, разработанный для учебных целей. Его грамматика строго формализована и включает следующие ключевые элементы:

- Программа (program)

Программа на Cool представляет собой последовательность классов, разделенных точкой с запятой. Каждый класс является независимой единицей, содержащей описание полей и методов.

Формальное определение:

program ::= [class;]\*

- Класс (class)

Класс является основной структурной единицей языка. Он может наследовать свойства и методы другого класса (указанного после ключевого слова inherits) или быть самостоятельным. Тело класса заключено в фигурные скобки и содержит объявления полей и методов.

Формальное определение:

class ::= class TYPEID [inherits TYPEID] { [feature;]\* }

- Методы и поля (feature)

Методы определяют поведение класса, а поля — его состояние. Методы могут принимать параметры (формальные аргументы) и возвращать значение указанного типа.

Формальное определение:

feature ::= ID( [formal [, formal]\*] ) : TYPEID { expr }

| ID : TYPEID [ <- expr ]

- Выражения (expr)

Выражения в Cool включают:

* Операции присваивания (<-).
* Вызовы методов (expr@TYPEID.ID( [expr [, expr]\*]).
* Условные конструкции (if-then-else).
* Циклы (while-loop-pool).
* Блоки кода ({ [expr;]+ })
* Локальные переменные (let).
* Арифметические и логические операции.

Примеры выражений:

expr ::= ID <- expr

| if expr then expr else expr fi

| while expr loop expr pool

| { expr; expr; }

| let ID : TYPEID [ <- expr ] in expr

1.2 Семантика языка: правила и ограничения

Семантика Cool определяет, как конструкции языка должны интерпретироваться и проверяться на корректность. Основные аспекты семантики включают:

* Классы и наследование

- Каждый класс имеет уникальное имя.

- Наследование формирует иерархию классов, где подклассы наследуют поля и методы родительских классов.

- Запрещено циклическое наследование (например, класс A наследует B, а B наследует A).

- Базовые классы (Int, Bool, String) не могут быть родительскими.

- Обязательно наличие корневого класса Object и класса Main с методом main().

* Методы

- Сигнатура метода включает имя, список параметров и возвращаемый тип.

- Параметры метода должны иметь уникальные имена.

- При переопределении метода в подклассе его сигнатура должна совпадать с родительской версией.

* Типизация

- Все выражения имеют строго определенный тип.

- Проверяется совместимость типов в операциях (например, сложение допустимо только для числовых типов).

- Поддерживается полиморфизм: объект подкласса может использоваться везде, где ожидается объект родительского класса.

* Области видимости

- Переменные должны быть объявлены до их использования.

- Локальные переменные, объявленные в блоках let, видны только внутри этого блока.

- Ключевое слово self ссылается на текущий объект и не может быть переопределено.

2. Структура компилятора: архитектура и компоненты

Компилятор Cool реализован как многоэтапный транслятор, преобразующий исходный код в промежуточное представление. Его архитектура включает три основных модуля:

2.1 Лексический анализатор (Lexer)

Лексический анализатор выполняет первичную обработку исходного кода, разбивая его на токены — минимальные значимые элементы языка.

Функции лексического анализатора:

1. Чтение исходного кода:

- Анализатор обрабатывает текст посимвольно, игнорируя пробелы, табуляции и комментарии.

2. Идентификация токенов:

- Ключевые слова (class, inherits, if, while и др.).

- Идентификаторы (имена классов, методов, переменных).

- Литералы (числа, строки, булевы значения).

- Операторы (+, -, =, <- и др.).

3. Обработка ошибок:

- Обнаружение недопустимых символов или некорректных лексем (например, идентификаторов, начинающихся с цифры).

Пример работы:

Для строки class Main { main() : Int { 1 + 2; } } лексический анализатор сгенерирует токены:

- class, Main, {, main, (, ), :, Int, {, 1, +, 2, ;, }, }.

#### 2.2 Синтаксический анализатор (Parser)

Синтаксический анализатор проверяет, соответствует ли последовательность токенов правилам грамматики Cool, и строит абстрактное синтаксическое дерево (AST).

Функции синтаксического анализатора:

1. Проверка структуры программы:

- Соответствие токенов правилам грамматики (например, после class должно следовать имя класса).

2. Построение AST:

- Дерево отражает иерархию конструкций языка (классы, методы, выражения).

3. Обработка ошибок:

- Несоответствие грамматике (пропущенные скобки, некорректные выражения).

Пример AST для класса Main:

Program

└── Class(Main)

└── Method(main)

└── Block

└── BinaryOp(+)

├── Literal(1)

└── Literal(2)

2.3 Семантический анализатор

Семантический анализатор проверяет корректность программы с точки зрения правил языка, включая типизацию, области видимости и наследование.

**Основные проверки:**

* Уникальность названий классов
* Существования класса с методом main
* Уникальность названия полей, методов в классе
* Отсутствие циклов в графе наследования
* Проверка выражений на совместимость типов

Проверка на существование класса с методом main:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4 | if (std::string(class\_name->get\_string()) == "Main" &&  features\_names.find("main") == features\_names.end()) {  std::cerr << "Semantic Error! No method 'main' in class 'Main'\n";  } | |

Проверка на уникальность названий классов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5 | auto result = classes\_names.insert(class\_name);  if (!result.second) {  std::cerr << "Semantic Error! class '" << class\_name->get\_string()  << "' redeclared.\n";  } | |  | |

Проверка на уникальность названия полей, методов в классе

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7 | std::unordered\_set<std::string> features\_names;  ...  auto res = features\_names.insert(feature\_name);  if (!res.second) {  std::cerr << "Semantic Error! feature '" << feature\_name << "' in '"  << class\_name << "' already exists!" << '\n';  } | |  | |

Проверка на отсутствие циклов в графе наследования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | **if (semantic::detect\_cycle(classes\_hierarchy)) {**  **std::cerr << "Semantic Error! loop detected in classes inheritance hierarchy\n";**  **std::cerr << "\\ program classes' hierarchy (child : parent)\n";**  **for (auto p : classes\_hierarchy) {**  **std::cerr << '\t' << p.first << " : " << p.second << "\n";**  **}**  **}** | |  | |

Проверка выражений на совместимость типов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  8 | semantic::check\_builtin\_type(  attr->get\_type()->get\_string(), attr->get\_expr()); | |  | |

Проверка вызова метода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6 | **if (!semantic::check\_signatures(cur\_method, parent\_method)) {**  **std::cerr << "Semantic Error! '" << feature\_name**  **<< "' method from class '" << parent\_name->get\_string()**  **<< "' doesn't match override version of it in class '"**  **<< class\_name << "'";**  **}** | |  | |

Проверка на существование переменной при использовании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3 | **if (formal\_name == "self") {**  **std::cerr << "Semantic Error! can't use 'self' as formal name\n";**  **}** | |  | |

Проверка на повторное объявление переменной в текущей области видимости:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | auto f\_result = formals\_names.insert(formal\_name);  if (!f\_result.second) {  std::cerr << "Semantic Error! formal '" << formal\_name  << "' in method '" << feature\_name  << "' already exists!\n";  }  ...  f\_result = formals\_names.insert(formal\_name);  if (!f\_result.second) {  std::cerr << "Semantic Error! formal '" << formal\_name  << "' in method '" << feature\_name << "' from '"  << class\_name->get\_string()  << "' already exists!\n";  } | |  | |

3 Тестирование компилятора

3.1 Набор тестов

Classes.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | class Int {  };  class Bool {  };  class Main {  }; | |  | |

Duplicates.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class Main {  a : Int;  a : Int;  main(b : Int): Int {  {  let b: Int <- 4 in false;  10;  }  };  main(): Int {  10  };  };  class Main {  main(): Int {  10  };  }; | |  | |

Type\_compatibility.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12 | class Main {  main(): Int {  2.fun()  };  };  class B  {  main(): Int {  1 + "Hello"  };  }; | |  | |

Unique\_var.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class Main {  a : Int;  a : Int;  main(b : Int): Int {  {  let b: Int <- 4 in false;  10;  }  };  main(): Int {  10  };  };  class Main {  main(): Int {  10  };  }; | |  | |

Scope.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7 | class Main {  main(): Int {  {  x;  }  };  }; | |  | |

Inherits.cl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | class A inherits B {  x : Int <- 0;  };  class B inherits C {  y : Int <- 1;  };  class C inherits D {  z : Int <- 2;  };  class D inherits E {  w : Int <- 3;  };  class E inherits B {  v : Int <- 4;  };  class Main {  main(): Int {  10  };  }; | |  | |

3.2 Результаты

Classes.cl

|  |
| --- |
| Semantic Error! class 'Int' redeclared.  Semantic Error! class 'Bool' redeclared.  Semantic Error! No class contains method 'main' |

Duplicates.cl

|  |
| --- |
| Semantic Error! feature 'a' in 'Main' already exists!  Semantic Error! Variable 'b' is already defined in the current scope.  Semantic Error! formal 'b' in method 'main' from 'Main' already exists!  Semantic Error! feature 'main' in 'Main' already exists!  Semantic Error! class 'Main' redeclared. |

type\_compatibility.cl

|  |
| --- |
| Semantic Error! Class 'Int' has no method 'fun'.  Semantic Error! '+' operator applied to non-integer types. |

Unique\_var.cl

|  |
| --- |
| Semantic Error! feature 'a' in 'Main' already exists!  Semantic Error! Variable 'b' is already defined in the current scope.  Semantic Error! formal 'b' in method 'main' from 'Main' already exists!  Semantic Error! feature 'main' in 'Main' already exists!  Semantic Error! class 'Main' redeclared. |

Scope.cl

|  |
| --- |
| Semantic Error! Undefined variable 'x' used. |

Inherits.cl

|  |
| --- |
| Semantic Error! Unknown parent of class 'A' - 'B'  Semantic Error! Unknown parent of class 'B' - 'C'  Semantic Error! Unknown parent of class 'C' - 'D'  Semantic Error! Unknown parent of class 'D' - 'E'  Semantic Error! loop detected in classes inheritance hierarchy  \ program classes' hierarchy (child : parent)  E : B  B : C  Main : Object  D : E  C : D  A : B |

ЗАКЛЮЧЕНИЕ

В ходе работы был разработан семантический анализатор для языка Cool, что позволило углубить понимание принципов семантического анализа и их реализации в компиляторах. Проведенные тесты подтвердили корректность работы анализатора и его способность выявлять семантические ошибки в исходном коде.