**Текст**

**2 COOL**

COOL is an Object-Oriented language and shares some design and syntax with Java. Fig. 1 shows a small COOL program that prints HelloWorld (lines 1-3) and a simple loop. The COOL manual, called CoolAid [2], provides the formal syntax and semantics of the language, which we summarize below.

*Syntax.* The syntax of COOL is designed to be simple. This allows students to write regular expressions (regexes) for lexer and parser to recognize syntactically correct COOL programs. The standard COOL specification does not have advanced features such as list/array structures, threading/multi-processing supports, or exception handling (supports for these are often used as homework or extra credit assignments).

*Tab. 1* displays the syntax and context-free grammar of COOL. A COOL program is a set of COOL classes, and each class consistsof features which are attributes (variables) and methods. Each class defines a type and thus programmer defines new types and associated data and methods by creating new classes (similar to an OO language such as Java). COOL is an expression language and thus most COOL constructs are expressions. Expressions take up a large portion of the COOL syntax, but in general, are relatively straightforward and similar to expressions in traditional languages. Note that the let expression that declares a new variable is similar to the one used in a functional language such as OCaml.

*Type Checking and Semantics*. COOL is a type-safe language and thus its compiler type-checks the input program to ensure no typing errors at run time. The typing rules for COOL, defined in the CoolAid manual, provide deduction rules for COOL expressions (e.g., if x is an integer and y is an integer then the expression x + y results in an integer).

The evaluation of a COOL program is provided using operational semantics rules. Similar to type checking rules that deduce the types of COOL expressions, these operational semantic rules deduce values for COOL expressions (e.g., if x is 3 and y is 7 then the expression x + y results in 10). These rules are also specified in the CoolAid manual. Fig. 2 shows the type checking and operational semantics rules for applying binary operators ∗, +, −, / to expressionss.

*Reference Compiler*. Students taking compiler courses using COOL often implement a full COOL compiler, which consists of main phases including lexing, parsing, type checking, semantic evaluation, and ASM transforming. To help students debug and make progress, the instructor often provides a complete “reference” compiler, e.g., an executable binary compiled from a COOL compiler implementation written in C or OCaml. Students then use the reference compiler to check each step of their implementation in both output results and error messages. COOLIO relies on the reference compiler to return error messages to the user.

**3 COOLIO**

The COOL Language Server Extension (COOLIO) can be found through the VSCode’s extension window or its Github repository given in §1. It is automatically activated when the user opens a COOL file with a .cl extension and adds language support to the COOL program opened in the editor. COOLIO is composed of four components shown in Fig. 3.

○1 The editor provides a GUI interface to the user, and is the host of many editing features such as the syntax highlight and autocomplete. It also interfaces with the language client (○2 ) through its API to display information such as error messages from the COOL compiler.

○2 The language client interacts with the editor and the language server by sending and receiving information, e.g., getting the content of an opened file, cursor position, or display error messages. Whenever the client is activated, it creates a language server and sends data to the server.

○3 The language server is a proxy to our COOL compiler. When a request from the client is received, the server retrieves the source code and sends it to the COOL compiler for analysis. The analysis results are composed into the LSP response format and returned to the client for display.

○4 The COOL reference compiler performs standard compilation phases such as parsing, type checking, and communicating results (e.g., warning and error messages) to the language server).

**3.1 Syntax Highlight**

Syntax highlight in COOLIO is handled by the VSCode editor. The editor tokenizes the source code using the regex rules specified in a TextMate configuration file. After that, the editor assigns colors to the tokens displayed in the IDE based on their defined roles and the colors specified in IDEs current theme (e.g., comments are blue in some themes while red in others).

For COOLIO, we create a TextMate[3] JSON file consisting of regular expressions (regexes) for all of COOL’s keywords, tokens, etc. Fig. 4 shows a snippet of a TextMate configuration having regexes for line comments and class identifiers. Tab. 2 shows syntax highlighting regexes for several main components in COOL.

When the user opens a COOL (.cl) file, the COOLIO extension is activated and uses these rules for syntax highlighting. While regexes can be complicated, the COOL reference compiler already contains similar regexes for lexing and parsing. Thus, we adapt and reuse these regexes. Example. Fig. 1 demonstrates syntax highlight in COOLIO. In this default VSCode Light theme, keywords are black, control flow structures are purple, class names are cyan, integer constants are green, and so on.

**3.2 Auto Completion**

COOLIO supports standard autocompletion (also called "Intellisense" in VSCode[4]). This allows the user to type a few keywords and the IDE can suggest and autocomplete common code structures. For example, when COOLIO detects that the user is typing the keyword if, it will ask if the user wants to replace that with the COOL snippet if condition then expression else expression fi. This autocompletion is straightforward yet useful in code development in an IDE as they help accomplish the goal of reducing typing effort and syntax errors from the developer.

To offer syntactically and structurally correct snippets, COOLIO’s autocompletion has knowledge about the syntax and grammar of COOL. Similar to syntax highlighting (§3.1), this is achieved at the editor level using a user-supplied TextMate configuration file. This TextMate file consists of rules mapping prefix strings (e.g., let) with code snippets (e.g., let var ... in ..). The code snippets can have multiple placeholders for any sub-expressions, and the user can switch between them using the “Tab” key. Once the configuration file is loaded with the extension, the editor will automatically try to match the prefix with the string the user entered. If the input matched (can be fully or partially) the prefix of any snippets, the name and description of the snippet will be listed in a drop-down manual for the user to select.

Fig. 5 shows an example of COOLIO’s TextMate configuration. Here, "body" defines the code snippet line by line, and the "${...}" marks the placeholders. This snippet defines the snippet for a class definition in COOL (i.e., when the user types class, the COOLIO can fill in the skeleton for defining a class in COOL).

Example. Fig. 6 shows an example when COOLIO senses that the user wants to declare a new variable through the let keyword and generates the snippet for new variable declaration with appropriate placeholders for specifying the type and initial value.

**Перевод**

**2 COOL**

COOL - это объектно-ориентированный язык, который имеет схожий дизайн и синтаксис с Java. На рис. 1 показан небольшой программный код на COOL, который выводит HelloWorld (строки 1-3) и простой цикл. Руководство по COOL, называемое CoolAid [2], предоставляет формальный синтаксис и семантику языка, которые мы резюмируем ниже.

*Синтаксис.*

Синтаксис COOL разработан с учётом простоты, что позволяет студентам писать регулярные выражения для лексера и парсера для распознавания синтаксически правильных программ на COOL. Стандартная спецификация COOL не имеет продвинутых функций, таких как структуры списков/массивов, поддержку потоков/многопроцессорности или обработку исключений (поддержка для этих функций часто используется в качестве заданий на дом или дополнительных баллов).

Таблица 1 отображает синтаксис и контекстно-свободную грамматику COOL. Программа на COOL представляет собой набор классов на COOL, и каждый класс состоит из атрибутов (переменных) и методов. Каждый класс определяет тип, и программисты определяют новые типы и связанные данные и методы, создавая новые классы (аналогично объектно-ориентированному языку, такому как Java). COOL - это язык выражений, поэтому большинство конструкций на COOL являются выражениями. Выражения занимают большую часть синтаксиса COOL, но в целом они относительно просты и аналогичны выражениям в традиционных языках. Обратите внимание, что выражение let, которое объявляет новую переменную, аналогично используемому в функциональном языке, таком как OCaml.

Проверка типов и семантика. COOL - это язык, обеспечивающий безопасность типов, и поэтому его компилятор проверяет типы входной программы, чтобы гарантировать отсутствие ошибок типизации во время выполнения. Правила типизации для COOL, определенные в руководстве CoolAid, предоставляют правила вывода для выражений на COOL (например, если x - целое число, а y - целое число, то выражение x + y дает целое число).

Выполнение программы на COOL осуществляется с помощью правил операционной семантики. Аналогично правилам проверки типов, которые выводят типы выражений на COOL, эти правила операционной семантики выводят значения для выражений на COOL (например, если x равно 3, а y равно 7, то выражение x + y даёт 10). Эти правила также указаны в руководстве CoolAid. На рис. 2 показаны правила проверки типов и операционной семантики для применения бинарных операторов ∗, +, −, / к выражениям.

*Справочный компилятор.*

Студенты, изучающие курсы по компиляторам с использованием COOL, часто реализуют полный компилятор на COOL, который состоит из основных фаз, включая лексирование, парсинг, проверку типов, семантическую оценку и преобразование ASM. Чтобы помочь студентам отлаживать и продвигаться вперед, преподаватель часто предоставляет полный "справочный" компилятор, например, исполняемый бинарный файл, скомпилированный из реализации компилятора на COOL, написанного на C или OCaml. Затем студенты используют справочный компилятор для проверки каждого шага своей реализации в выводимых результатах и сообщениях об ошибках. COOLIO использует справочный компилятор для возврата сообщений об ошибках пользователю.

**3 COOLIO**

Расширение сервера языка COOL (COOLIO) можно найти через окно расширений VSCode или его репозиторий на Github, указанный в §1. Оно автоматически активируется при открытии пользователем файла COOL с расширением .cl и добавляет поддержку языка для программы на COOL, открытой в редакторе. COOLIO состоит из четырех компонентов, показанных на рис. 3.

○1 Редактор предоставляет пользователю графический интерфейс и является хостом для многих функций редактирования, таких как подсветка синтаксиса и автозаполнение. Он также взаимодействует с языковым клиентом (○2) через его API для отображения информации, такой как сообщения об ошибках от компилятора COOL.

○2 Языковой клиент взаимодействует с редактором и сервером языка, отправляя и получая информацию, например, получая содержимое открытого файла, позицию курсора или отображая сообщения об ошибках. Когда клиент активируется, он создаёт сервер языка и отправляет данные на сервер.

○3 Языковой сервер является прокси для нашего компилятора COOL. При получении запроса от клиента сервер извлекает исходный код и отправляет его на анализ в COOL-компилятор. Результаты анализа компонуются в формат ответа LSP и возвращаются клиенту для отображения.

○4 Справочный компилятор COOL выполняет стандартные этапы компиляции, такие как разбор синтаксиса, проверку типов и передачу результатов (например, предупреждений и сообщений об ошибках) на языковой сервер.

**3.1 Подсветка синтаксиса**

Подсветка синтаксиса в COOLIO обрабатывается редактором VSCode. Редактор токенизирует исходный код, используя правила регулярных выражений, указанные в файле конфигурации TextMate. Затем редактор назначает цвета токенам, отображаемым в IDE, основываясь на их определенных ролях и цветах, указанных в текущей теме IDE (например, комментарии в некоторых темах синие, а в других - красные).

Для COOLIO мы создаем файл JSON TextMate[3], содержащий регулярные выражения (regexes) для всех ключевых слов, токенов и т.д. COOL. Рис. 4 показывает фрагмент конфигурации TextMate с regexes для строковых комментариев и идентификаторов классов. Табл. 2 показывает regexes подсветки синтаксиса для нескольких основных компонентов в COOL.

Когда пользователь открывает файл COOL (.cl), расширение COOLIO активируется и использует эти правила для подсветки синтаксиса. Хотя regexes могут быть сложными, справочный компилятор COOL уже содержит похожие regexes для лексического и синтаксического анализа. Таким образом, мы адаптируем и повторно используем эти regexes. Пример. Рис. 1 демонстрирует подсветку синтаксиса в COOLIO. В этой стандартной теме VSCode Light ключевые слова черные, структуры управления потоком - фиолетовые, имена классов - голубые, целочисленные константы - зеленые и т.д.

**3.2 Автодополнение**

COOLIO поддерживает стандартное автодополнение (также называемое "Intellisense" в VSCode[4]). Это позволяет пользователю вводить несколько ключевых слов, и IDE может предложить и автоматически завершить распространенные структуры кода. Например, когда COOLIO обнаруживает, что пользователь пишет ключевое слово if, он спросит, хочет ли пользователь заменить это на COOL-фрагмент if condition then expression else expression fi. Это автодополнение простое, но полезное в разработке кода в IDE, так как оно помогает достигнуть цели сокращения усилий набора и синтаксических ошибок от разработчика.

Чтобы предложить синтаксически и структурно правильные фрагменты, автодополнение COOLIO имеет знания о синтаксисе и грамматике COOL. Аналогично подсветке синтаксиса (§3.1), это достигается на уровне редактора с использованием предоставленного пользователем файла конфигурации TextMate. Этот файл TextMate состоит из правил, сопоставляющих строковые префиксы (например, let) с фрагментами кода (например, let var ... in ..). Фрагменты кода могут иметь несколько заполнителей для любых подвыражений, и пользователь может переключаться между ними, используя клавишу "Tab". Как только файл конфигурации загружен с расширением, редактор автоматически пытается сопоставить префикс со строкой, введенной пользователем. Если ввод соответствует (может быть полностью или частично) префиксу любого фрагмента, имя и описание фрагмента будут перечислены в выпадающем меню для выбора пользователем.

Рис. 5 показывает пример конфигурации TextMate для COOLIO. Здесь "body" определяет фрагмент кода построчно, а "${...}" обозначает заполнители. Этот фрагмент определяет фрагмент для определения класса в COOL (т.е. когда пользователь вводит class, COOLIO может заполнить каркас для определения класса в COOL).

Пример. Рис. 6 показывает пример, когда COOLIO обнаруживает, что пользователь хочет объявить новую переменную через ключевое слово let, и генерирует фрагмент для объявления новой переменной с соответствующими заполнителями для указания типа и начального значения.