ПРОГРАММНАЯ СИСТЕМА ПРЕОБРАЗОВАНИЯ МОДЕЛЕЙ АЛГОРИТМОВ

Воронич М. С., Лисицына И. Н., Трубина Н. Ф.
Одесский национальный университет имени И. И. Мечникова

Ключевые слова: регулярные языки, регулярные выражения, конечные автоматы, модели алгоритмов.

В теории и практике программирования на протяжении десятилетий используются такие модели алгоритмов как регулярные выражения и конечные автоматы [1].

Анализ текстовой информации, фильтрация огромных массивов данных, разработка компонентов компилятора [3, 5], валидация пользовательских данных, проверка ответов в поисковых системах, системах тестирования знаний [4] и т.д. — это прикладные задачи, решаемые регулярными выражениями.

В действительности, например, утилита `grep` в UNIX-подобной системе и приложения (текстовые редакторы ed, sed, vim и Notepad++ и т. д.) компилируют регулярное выражение в детерминированный или недетерминированный конечный автоматы, которые впоследствии видоизменяются для распознавания паттернов в обрабатываемом тексте. Генераторы лексических анализаторов, такие как Lex и Flex, будучи компонентами компилятора, получают формальные описания лексем, по существу — регулярных выражений, и генерируют детерминированный конечный автомат, распознающий, какая из лексем появляется на его входе. Причиной такого решения являлось то, что в случае, когда возникает необходимость отредактировать лексический анализатор, то гораздо проще и безопаснее внести изменения в регулярное выражение, чем тратить время на отладку кода, чтобы исправить возникший дефект [6].

Все популярные языки программирования включают библиотеки поддержки этого инструмента или даже имеют реализацию этой поддержки, встроенной непосредственно в сам язык. В качестве известных примеров выступают языки Perl, Java, JavaScript, C#, Ruby и многие другие [2].

Несмотря на то, что регулярные выражения алгебраически задают такие же языки, что и конечные автоматы — регулярные языки — существует явная разница между этими двумя моделями, состоящая в том, что регулярные

выражения определяют допустимые последовательности символов декларативным способом [6].

Построение регулярного выражения является нетривиальной задачей. Выражение может быть громоздким и неоправданно сложным. При этом распознавание текста, соответствующего этому выражению, бывает неэффективным. Оптимизация, то есть получение упрощенного выражения, определяющее те же языки, что и более сложное выражение, возможна при помощи другой модели алгоритма, полученной преобразованием первоначальной. Например, конечный автомат, эквивалентный некоторому регулярному выражению, минимизируется, после чего выполняется обратное преобразование в регулярное выражение.

Главной целью проекта является разработка инструмента, помогающего автору регулярного выражения наглядно отобразить это выражение в двух различных формах: таблицы и/или диаграммы переходов, благодаря которой удобно представлять конечный автомат в виде графа [3, 6]. Также в приложении предоставляется возможность оптимизации регулярного выражения за счет минимизации его описания в виде детерминированного конечного автомата. Достижение указанной цели требует преобразование входного регулярного выражения следующими методами [3-6]:

1) Алгоритм Томпсона: преобразование регулярного выражения в недетерминированный конечный автомат (НКА);
2) Алгоритм построения подмножества: преобразование недетерминированного конечного автомата в детерминированный (ДКА);
3) Алгоритм Хопкрофта, алгоритм Бжозовского: минимизация ДКА;
4) Алгоритм Клини: преобразование минимального ДКА в регулярное выражение.

Выбор инструментов разработки остановлен на среде программирования Microsoft Visual Studio 2019 Community Edition, языке программирования C#, генераторе лексических анализаторов C# Flex и генераторе синтаксических анализаторов C# Bison.

Приложение представляется полезным в качестве поддерживающего программного обеспечения в учебных дисциплинах, связанных с разработкой компиляторов, теорией алгоритмов, разработкой информационно-поисковых систем.

Преобразования моделей алгоритмов вышеописанными методами находит своё применение в проектировании программного обеспечения. Такой подход сродни конструктивному доказательству теорем. Следовательно, спроектированные по этим моделям программы не требуют верификации, так

как их получение одновременно является доказательством существования алгоритма.

Литература

1. В. И. Поляков, В. И. Скорубский. Преобразование моделей алгоритмов. 2012. Т. 55, № 10, 2012, 41-46 с.
2. М. Фицджеральд. Регулярные выражения: основы.: Пер. с англ. - М.: ООО “И.Д. Вильямс”: 2015. - 144 с.
3. А. Ахо. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. - М. ООО “И.д. Вильямс”, 2008. - 1184 с.
4. О.А. Сычев, Г.В. Терехов. Инструменты помощи автору регулярных выражений для тестовых вопросов в СДО Moodle. Открытое образование - Т. 20. No 3. 2016
5. Keith D. Cooper, Linda Torczon. Engineering a Compiler. Second Edition. 2012 Elsevier, Inc. – P. 800
6. Хопкрофт, Джон, Э., Мотвани, Раджив, Ульман, Джеффри, Д. Введение в теорию автоматов, языков и вычислений, 2-е изд.: Пер. с англ. - М.: Издательский дом “Вильямс”, 2008. - 528 с.