DOI 10.34185/1562-9945-6-137-2021-12
УДК 004.4

І.О. Сокіл, О.С. Волковський

СИСТЕМА КОНВЕРТАЦИИ ИСХОДНЫХ ПРОГРАММНЫХ КОДОВ

Аннотация. Нарастающий объем технологий, прекращение поддержки активно используемых средств разработки, устаревшее API и т. д., вызывают потребность конвертации программных кодов. В IT компаниях и не только, часто возникает вопрос поддержки устаревшего ПО, которым продолжают пользоваться клиенты, либо перевода нынешнего ПО на актуальные технологии. Для программистов рациональнее воспользоваться конвертацией, сохранив большую часть существующей кодовой базы, чем переписывать все ПО вручную, даже если понадобится ручная корректировка. На данный момент качественных систем конвертации кодов существует немного. В большинстве своем, системы конвертации работают неплохо только со схожими языками программирования. Задача конвертации программных кодов является глубокой и сложной темой. Программисты пытаются улучшить технологии перевода, сталкиваясь со многими трудностями. В данной статье рассматриваются базовые принципы построения системы конвертации программных кодов и некоторые особенности при её практической реализации.
Ключевые слова: конвертация программных кодов, компилятор, лексический анализатор, лексемы, синтаксический анализатор, абстрактное синтаксическое дерево, дерево полного разбора, семантический анализ, конечные автоматы.

Вступление. Нарастающий объем технологий, прекращение поддержки активно используемых средств разработки, устаревшее API и т. д., вызывает потребность конвертации программных кодов.

У технологий конвертации программных кодов есть множество критиков. Основной причиной их критики является тот факт, что код, генерируемый большинством систем перевода, является плохо читаемым и требует ручной доработки.

Однако, на практике, в IT компаниях и не только, часто возникает вопрос поддержки устаревшего ПО, которым продолжают пользоваться клиенты, либо перевода нынешнего ПО на актуальные технологии. Для программистов рациональнее воспользоваться конвертацией, сохранив большую часть существующей кодовой базы, чем переписывать все ПО вручную [4], даже если пона-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

добится ручная корректировка. С точки зрения компании есть возможность сэкономить огромное количество времени и материальных средств.

На данный момент качественных систем конвертации кодов существует немного. Из систем в открытом доступе можно вспомнить GWT (Google Web Toolkit) — это набор инструментов для разработки веб-приложений. В него входит компилятор, который преобразовывает код написанный на Java в код на JavaScript [9]. Некоторые, IDE имеют подобную возможность. Например, IDE IntelliJ IDEA имеет возможность конвертации кода языка Java в код языка Kotlin [10]. Так же есть набор некоммерческих проектов от энтузиастов [11]. Некоторые системы конвертации разрабатываются в компаниях для внутренних целей и закрыты от глаз посторонних.

В целом системы перевода кода можно разделить на два типа – это системы, в которых переведенный код не похож на исходный, и системы, которые пытаются сохранить структуру как можно ближе к исходному коду, для облегчения разработки и отладки.

Первый тип систем обеспечивает подготовку кода к развертыванию на целевой платформе, при этом, поддержка подобного кода практически невозможна. К таким системам относится GWT. Свою работу она выполняет качественно и приносит пользу программистам, желающим разрабатывать на привычном для них языке. Однако, выходной код на JavaScript является «кашей», в которой очень сложно разобраться.

Системы второго типа являются наиболее сложными. Им необходимо перевести исходный код одного языка в код другого, при этом сохранив исходною структуру, что позволило бы программистам перейти на новый уровень технологий и выполнять поддержку переведенного кода.

Задача конвертации программных кодов не тривиальная. На данный момент, в большинстве своем, системы конвертации работают неплохо только со схожими языками программирования [5].

Многие программисты пытаются улучшить технологии перевода. В эпоху активного развития искусственного интеллекта, который уже используется во многих системах обработки и анализа кода, направление разработки систем конвертации кодов выходит на новый уровень. С возросшей популярностью высокоуровневых языков программирования общего назначения с динамической типизацией задача конвертации становится актуальной.

Проблема дословного перевода исходного кода.Рассмотрим, для примера, естественные языки, которые используются людьми для общения друг с другом. Существует множество автоматических переводчиков естественных языков. И мы понимаем, что качество перевода сильно зависит от грамматической, лексической и синтаксической схожести языков. Если прямолинейно выполнять дословный перевод предложения, мы можем столкнуться с полной потерей смысла исходной фразы. Язык программирования так же используется для общения, только общения программиста с ЭВМ. Сложность перевода, так же, зависит от степени схожести лексических, синтаксических и семантических правил языков. Мы должны понимать контекст переводимой фразы и особенности синтаксической расстановки отдельных её элементов. Тем более современные языки имеют огромное количество различных конструкций, которые не получится однозначно интерпретировать для целевого языка, что приводит к практической невозможности дословного перевода. Качественная конвертация программных кодов имеет сложную структуру.

Этапы конвертации.В данной статье рассмотрим классический подход к конвертации программных кодов, который используется, например, в компиляторах исходного кода.

В целом этапы конвертации выглядят следующим образом [3]:

1. Лексический анализ.
2. Синтаксический анализ.
3. Семантический анализ.
4. Процесс перевода.

Укрупнённая схема системы конвертации программных кодов приведена на рис. 1.

Рассмотрим каждый этап в отдельности.

Лексический анализ. Целью лексического анализа является разбор исходного текста программы на значимые единицы, согласно лексике языка. Данные значимые единицы называются лексемами. Проводя параллель с естественными языками, лексемы – это слова. В результате лексического анализа формируется набор, так называемых, токенов – объектов или структур, в которых хранится информация о найденной лексеме [6]. Список токенов необходим для последующей обработки исходного кода.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

X	Исходный язык программирования
Y	Целевой язык программирования
Z	Промежуточный язык программирования

Рисунок 1 – Укрупнённая схема системы конвертации программных кодов

Помимо генерации токенов, проведя лексический анализ, мы можем определить соответствие кода лексике языка, т. е. определить лексические ошибки программиста, поскольку не имеет смысла анализировать и переводить лексически некорректный текст.

Синтаксический анализ. Целью синтаксического анализа является выявление и проверка синтаксических конструкций исходного кода, согласно синтаксису языка. В естественных языках синтаксис – это правила сочетания слов в рамках предложения. Например, следующая строка в языке программирования Си является синтаксически допустимой:

int num = 5;

А эта строка лексически допустима [7], но синтаксически неверна:

int = 5;

Синтаксический анализ использует результаты предыдущего этапа, т. е. набор токенов, сформированный лексическим анализатором.

Чаще всего, на практике, лексический и синтаксический анализатор взаимосвязаны и выполняются последовательно. Синтаксический анализатор, выполнив обработку очередной конструкции, обращается к лексическому анализатору за следующей лексемой. При этом синтаксический анализатор может затребовать «откатиться назад» для обработки конструкции по другому правилу. При возникновении неоднозначности определения типа и границы очередной лексемы анализаторы начинают функционировать параллельно для решения данной проблемы [3].

Параллельное взаимодействие имеет высокую сложность при реализации и требует большего времени анализа исходного кода [3]. Возможный способ избегания параллельной работы анализаторов будет описан далее в статье.

В результате работы синтаксического анализатора строится абстрактное синтаксическое дерево и дерево разбора. Абстрактное синтаксическое дерево (АСД) по сути является иерархией операторов и операндов, при этом в АСД отбрасываются синтаксические правила, не влияющие на общую семантику программы. Дерево разбора учитывает весь синтаксис исходного кода, к примеру, группирующие правила [12]. В результате работы синтаксического анализатора мы можем приступить к формированию общей структуры данных исходного кода.

Семантический анализ. В естественных языках семантический анализ позволяет донести смысл исходной фразы или текста [1]. Так же, для языков программирования, задача семантического анализа состоит в том, чтобы извлечь из формального описания информацию о структуре программы [8].

При формальном описании возникает проблема нахождения соответствия между произвольными грамматиками [8]. Языки программирования имеют различное назначение и сферы применения. Как описывалось раньше, в со-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

временных языках существует множество синтаксических конструкций и, так называемого, «синтаксического сахара», которые семантически могут выполнять одну и ту же работу. Из примеров это множество конструкций обозначения блока кода в PHP, либо реализация «геттеров» и «сеттеров» в одну строчку в C# и т. д. Плюс ко всему, языки, которые более ориентированы на архитектуру компьютера, могут совсем не иметь синтаксической конструкции, которая соответствует конструкции из высокоуровневого языка. В данном случае также необходим семантический анализ и формирование набора конструкций, которые могут решить данную проблему.

Процесс перевода. Используя информацию и наработки предыдущих этапов выполняется перевод кода исходного языка в код целевого. Но, при построении архитектуры системы, возникает необходимость разделить данный этап на два:

1. Перевод кода исходного языка в промежуточный язык.
2. Перевод кода промежуточно языка в целевой.

Использование промежуточного языка. Причина использования промежуточно языка состоит в вопросе масштабирования системы. Из-за отсутствия универсальных моделей сопоставления произвольных грамматик многие языки будут обладают уникальной семантикой [8]. Это приводит к необходимости создавать уникальную реализацию модели для перевода «Код в Код», что приводит к проблеме масштабируемости подобной системы.

Возьмем, для примера, четыре языка программирования, совершенно условно: Java, Python, C++ и C#. На рис. 2 стрелками показаны отдельные реализации этапа перевода языка. Как видно на рисунке, для каждого языка из списка необходимо реализовать связь с остальными языками.

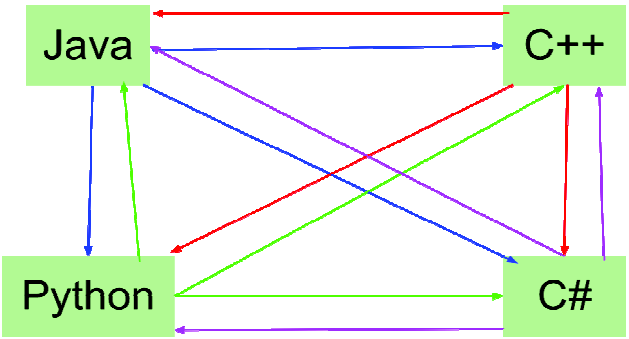

Рисунок 2 – Количество реализаций этапа преобразования кода без промежуточного язика

Теперь возьмем все те же языки, только используем прослойку в виде промежуточного языка (рис. 3).

Как видим необходимое количество реализаций уменьшилось. В схеме, приведенной на рис. 3 причина наличия у языков двух связей с промежуточным языком состоит в том, что реализация однозначного алгоритма преобразования, который смог бы работать в «обе стороны», является большой проблемой. В данном алгоритме корректность обратного перевода зависит от выбранного промежуточно языка и степени различий его с переводимыми языками.

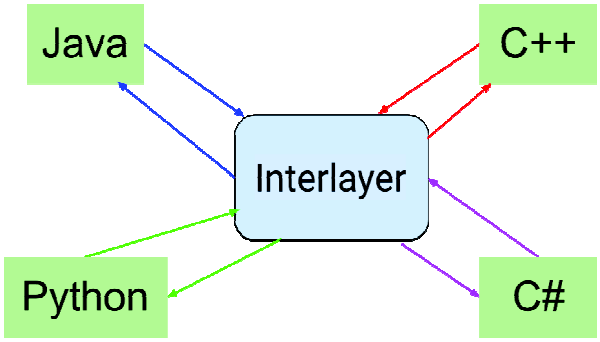

Рисунок 3 – Количество реализаций этапа преобразования кода с использованием промежуточного языка

Но даже в случае реализации двойной связи, результаты введения промежуточного языка очевидны и их можно хорошо продемонстрировать с помощью Табл.1.

Из Табл.1 видно, что при поддержке двух языков программирования количество необходимых реализаций без промежуточного языка меньше, чем во втором варианте. При трёх языках – они сравнялись. А при четырёх и более, преимущество варианта с использованием промежуточного языка явно выделяется. Для системы без промежуточного языка формула для определения количества реализаций – $x * (x - 1)$, где x – количество поддерживаемых языков.

Для системы с использованием промежуточного языка – количество реализаций определяется как: $x * 2$.

В целом использование промежуточного языка снижает сложность системы конвертации программных кодов при её масштабировании. Однако, от выбора промежуточного языка во многом зависит успех реализации системы.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Таблица 1

Количество языков	Без промежуточного языка	С промежуточным языком
2	**2**	**4**
3	**6**	**6**
4	**12**	**8**
5	**20**	**10**
6	**30**	**12**
7	**42**	**14**

Особенности реализации системы конвертации программных кодов.

На основе рассмотренных данных можно сделать несколько выводов:

1. Модель лексического и синтаксического анализатора можно сделать универсальной, заменяя в ней только правила, актуальные для анализируемого языка.

2. Так как процесс семантического анализа невозможно привести к универсальной математической модели и формальным средствам описания, он, совместно с этапом перевода, должен реализовываться вручную для отдельного языка программирования [2].

Рассмотрим вопрос решения параллельной работы синтаксического и лексического анализатора. Часто разработчики языков программирования и компиляторов идут на некие ограничения синтаксиса. Одним из таких ограничений является соглашение о том, что при возникновении проблемы с определением границы лексемы выбирается лексема максимальной длины [3]. Например, в языке Си следующая операция

n = i+++++j;

лексическим анализатором будет разбита таким образом:

n = i++ ++ +j;

и синтаксический анализатор выдаст ошибку.

После проведения практических экспериментов с реализацией лексического анализатора, можно сделать весомый вывод:

Использование регулярных выражений для выделения лексем вызывает множество проблем. Если составить список регулярных выражений и исполь-

зовать их последовательно, то очередная лексема будет отнесена к правилу, которое описано раньше в этом списке. В этом случае конструкции, которые явно синтаксически допустимы будут определяться неверными из-за ошибок лексического анализа.

Например, для языка Си возьмем два простых регулярных выражения. Первое – регулярное выражение ключевого слова “int”, второе – выражение идентификатора “[a-zA-Z_]+\w*”. Рассмотрим следующую строку:

int intnum = 5;

Синтаксически данное выражение абсолютно верно. Но, если первым в списке регулярных выражений будет выражение “int”, то лексический анализатор выделит лексемы следующим образом (представим, что правила для выделения отступов, чисел и операторов уже описаны):

[int][][int][num][][=][][5][;]

Синтаксический анализатор увидит, что после указания типа идет не идентификатор или функция, а снова указание типа, и выдаст ошибку. Естественно, можно добавить правило, что перед и после ключевого слова должен идти разделитель. Но что делать в случае, когда ключевое слово написано в начале текста, либо после него нет ни одного символа. Все эти моменты должны быть учтены в регулярном выражении, что увеличивает его сложность и размер.

Чаще всего в литературе рекомендуют использовать конечные автоматы [3][6]. При этом алгоритм не пытается выделить определенное правило целиком и сами правила проще в реализации. В конечный автомат передается очередной символ исходной последовательности, при этом автомат изменяет свое внутренне состояние и выдает это состояние на наружу. Тогда, алгоритм разбора получается несложным. Необходимо провести каждый символ входящей строки через набор конечных автоматов - автомат определяет уникальный токен. На каждой итерации проверяется, остался ли какой-либо автомат в активном состоянии. Если таковых нет, значит, часть строки до данного символа является лексемой. А для определения типа токена будет использоваться автомат, который последним перешел в неактивное состояние [6].

Вывод. В процессе постоянного появления новых языков программирования, а также спада востребованности существующих, задача конвертации программных кодов становится все более актуальной. Она является глубокой и сложной темой. Программисты пытаются улучшить качество процесса перевода, сталкиваясь со многими трудностями.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Данная статья направленна на пояснение базовых принципов системы конвертации кода и некоторых нюансов при её реализации.

ЛИТЕРАТУРА/ЛІТЕРАТУРА

1. Volkovskyi O. S., Kovylin Y. R. Mathematical model for automatic creation the semantic thesaurus for the scientific text. Системні технології. Регіональний збірник міжвузівських наукових праць. 2019. Вип.6, № 125. С. 82–88.

2. Volkovskyi O. S., Kovylin Y. R. Matematical model for constructing the semantic network of a scientific text. Modern engineering and innovative technologies. // International periodic scientific journal, Karlsruhe, Germany. – 2020. – Issue №11, Part №2. – P. 128–133.

3. Молдованова О. В. Языки программирования и методы трансляции: Учебное пособие. – Новосибирск/СибГУТИ, 2012. – 134с.

4. Транскомпилируемые языки: проекты конвертации код-в-код [Электронный ресурс] – Режим доступа к ресурсу: https://habr.com/ru/company/vk/blog/480724/.

5. The Most Accurate and Reliable Source Code Converters [Electronic resource] – Resource access mode: https://www.tangiblesoftwaresolutions.com/.

6. Лексический анализатор на JavaScript [Электронный ресурс] – Режим доступа к ресурсу: https://bit.ly/3FcIBfw.

7. Алфавит языка Си. Лексемы [Электронный ресурс] – Режим доступа к ресурсу: https://studfile.net/preview/3675473/page:3/.

8. Теория языков программирования и методы трансляции [Электронный ресурс] – Режим доступа к ресурсу: http://ermak.cs.nstu.ru/trans/.

9. GWT — Краткое руководство [Электронный ресурс] – Режим доступа к ресурсу:https://coderlessons.com/tutorials/veb-razrabotka/izuchite-google-web-toolkit/gwt-kratkoe-rukovodstvo.

10. Get started with Kotlin | IntelliJ IDEA - JetBrains [Электронный ресурс] – Режим доступа к ресурсу: https://www.jetbrains.com/help/idea/get-started-with-kotlin.html.

11. Awesome Transpilers by target language [Электронный ресурс] – Режим доступа к ресурсу: https://github.com/transpiler/awesome-transpiler.

12. Abstract syntax tree [Электронный ресурс] – Режим доступа к ресурсу: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

REFERENCES

1. Volkovskyi O. S., Kovylin Y. R. Mathematical model for automatic creation the semantic thesaurus for the scientific text. System technologies. Regional interuniversity compendium of scientific works. 2019. Vol.6, No. 125. P. 82–88.

2. Volkovskyi O. S., Kovylin Y. R. Matematical model for constructing the semantic network of a scientific text. Modern engineering and innovative technologies. // International periodic scientific journal, Karlsruhe, Germany. – 2020. – Issue №11, Part №2. – P. 128–133.

3. Moldovanova O. V. Programming languages and translation methods: Study guide. – Novosibirsk/SibSUTIS, 2012. – 134p.

4. Transcompiled languages: code-to-code conversion projects [Electronic resource] – Resource access mode: https://habr.com/ru/company/vk/blog/480724/.

5. The Most Accurate and Reliable Source Code Converters [Electronic resource] – Resource access mode: https://www.tangiblesoftwaresolutions.com/.

6. Lexical analyzer in JavaScript [Electronic resource] – Resource access mode https://bit.ly/3FcIBfw.

7. C language alphabet. Lexemes [Electronic resource] – Resource access mode: https://studfile.net/preview/3675473/page:3/.

8. Programming languages theory and translation methods [Electronic resource] – Resource access mode: http://ermak.cs.nstu.ru/trans/.

9. GWT — Quick guide [Electronic resource] – Resource access mode: https://coderlessons.com/tutorials/veb-razrabotka/izuchite-google-web-toolkit/gwt-kratkoe-rukovodstvo.

10. Get started with Kotlin | IntelliJ IDEA - JetBrains [Electronic resource] – Resource access mode: https://www.jetbrains.com/help/idea/get-started-with-kotlin.html.

11. Awesome Transpilers by target language [Electronic resource] – Resource access mode: https://github.com/transpiler/awesome-transpiler.

12. Abstract syntax tree [Electronic resource] – Resource access mode: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

Received 06.10.2021.
Accepted 09.10.2021.

Program source codes conversion system

The growing volume of technologies, the end of actively used development tools support, outdated API etc., entails the need of program codes conversion. In IT companies and not only, often begged the question of deprecated software support, which customers continue to use, or translation of current software to actual technologies. It is more rational for programmers to use the conversion and save most of code base, than rewriting all software by hand, even if manual

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

adjustment is needed. At this moment, there are few high-quality code conversion systems. Largely, conversion systems work well only with similar programming languages. The task of program codes conversion is a deep and complex topic. Programmers are trying to improve translation technologies and facing with many challenges. This article discusses the basic principles of building a system for program codes conversion and some features of its practical implementation.

Система конвертації вихідних програмних кодів

Наростаючий обсяг технологій, припинення підтримки засобів розробки, що активно використовуються, застаріле API і т. д., викликають потребу конвертації програмних кодів. У IT компаніях і не тільки часто виникає питання підтримки застарілого ПЗ, яким продовжують користуватися клієнти, або перекладу нинішнього ПЗ на актуальні технології. Для програмістів раціональніше користуватися конвертацією, зберігши більшу частину існуючої кодової бази, ніж переписувати все ПЗ вручну, навіть якщо знадобиться ручне коригування. На даний момент якісних систем конвертації кодів існує небагато. Здебільшого, системи конвертації працюють непогано лише з подібними мовами програмування. Завдання конвертації програмних кодів є глибокою та складною темою. Програмісти намагаються покращити технології перекладу, стикаючись із багатьма труднощами. У цій статті розглядаються базові принципи побудови системи конвертації програмних кодів та деякі особливості під час її практичної реалізації.

Сокіл Іван Олександрович – аспірант, кафедра комп'ютерних наук та інформаційних технологій, Дніпровський національний університет імені О. Гончара.

Волковський Олег Степанович – к.т.н., доцент кафедри КНДТ, Дніпровський національний університет імені О.Гончара.

Сокол Иван Александрович – аспирант, кафедра компьютерных наук и информационных технологий, Днепровский национальный университет имени О. Гончара.

Волковский Олег Степанович – к.т.н., доцент кафедры КНИТ, Днепровский национальный университет имени О. Гончара.

Sokol Ivan – graduate student, Department of computer science and information technologies, Oles Honchar Dnipro National University.

Volkovskyi Oleg – PhD in Technical Science, associate professor of CSIT Department, Oles Honchar Dnipro National University.