Yehor Zakupin, Valery Pavlov

CREATING TRANSLATORS OF HIGH-LEVEL PROGRAMMING LANGUAGES

Єгор Закупін, Валерій Павлов

СТВОРЕННЯ ПЕРЕКЛАДАЧІВ МОВ ПРОГРАМУВАННЯ ВИСОКОГО РІВНЯ

The paper deals with the creation of a program-translator for high-level programming languages. Analyzed the principles of building such applications and existing solutions. The program takes as a basis the structure of the translator, using as the input and output language - a high-level language. The program also allows you to choose the input and output languages.

Key words: programming language, translator.
Fig.: 4. Tabl.: 0. Bibl.: 8.

У статті розглядається питання створення програми-перекладача для мов програмування високого рівня. Проаналізовані принципи побудови подібних програм та існуючі рішення. Програма бере за основу структуру транслятору, використовуючи в якості вхідної та вихідної мови – мову високого рівня. Також програма надає змогу вибору вхідної та вихідної мови.

Ключові слова: мова програмування, перекладач.
Рис.: 4. Табл.:0. Бібл.: 8.

Relevance of research topic. A steady increase in the number of high-level programming languages can create significant problems when creating unified software. This necessitates the creation of an interpreter to facilitate and accelerate the development of software through the use of existing modules, regardless of the programming language in which they were written.

Target setting. The lack of programs that allow you to flexibly select incoming and outgoing languages, as well as the unreliability of existing solutions due to the lack of analysis of input text, which greatly complicates the use of such software.

Actual scientific researches and issues analysis. Typically, the translation of programs from one programming language to another is based on the theory of constructing compilers [1, 2, 3], but for the case when both the input and the output languages are high-level languages only cross-compilation is considered. There is a fairly small number of publications devoted to the problem of translation of programming languages. Whether they are solving local problems [4], or are based on outdated versions of programming languages [5], or are purely commercial projects,

where only demo versions are offered on a free basis [6, 7]. For example, you can specify "Java to C ++ Converter" as well as "ANSI / Turbo Pascal to C / C ++ converter", the main disadvantage of which is their strict attachment to the input and output languages. Also, in some such programs there is a lack of semantic verification systems.

Uninvestigated parts of general matters defining. The article proposes an analysis of the application of a new approach for translating between high-level programming languages. It is based on the use of pre-analysis of the text of the input language, as well as the use of external data to provide information on programming languages.

The research objective. The objective is to create a software application that allows you to transfer incoming text written in a high-level language, also at another high-level language. In this case, the program should conduct a preliminary analysis of the input text, and information on the structure of programming languages to take from external files.

The statement of basic materials. The solution method can be divided into two stages. In the first stage, the syntax of the programming language will be written, since the work of the translator depends on the form in which the syntax will be written. At the second stage, is developing a translator.

Method of writing language syntax. It is recommended to use the Backus - Naur form for language descriptions. In contrast to the meta-language of Chomsky or Chomsky-Schutzenberger, which were used in mathematical literature in the description of simple abstract languages, this meta-language was first used to describe the syntax of the real programming language Algol 60 [8]. Along with the new symbols of metacharacters, it used meaningful designations of non-terminals. This made the description of the language more vivid and allowed to continue to widely use this universal notation to describe the real languages of programming.

Developing a translator. Common properties and patterns are inherent in different programming languages, as well as translators from these languages. They have similar processes of converting the source text. In spite of the fact that the interaction of these processes can be organized in different ways, one can distinguish the functions, implementation of which leads to the same results. We call these functions the phases of the translation process. They determine the overall structure of the compiler, shown in Fig. 1.

It stands out:

1. The phase of lexical analysis.
2. The phase of syntax analysis, consisting of:

• recognition of syntactic structure;

• semantic parsing, in the process of which the work with tables is performed, the generation of an intermediate semantic representation or an object model of language.

3. The code generation phase, implementing:

• semantic analysis of the component of the intermediate representation or the object model of the language;

• Intermediate representation or object model translation into object code.

Along with the main phases of the translation process, additional phases are possible:

2a Phase of research and optimization of the interim report, consisting of:

2a.1. analysis of the correctness of the interim presentation;

2a.2. optimization of the interim presentation.

3a The phase of optimization of the object code.

In addition, we can select a single process for all phases to analyze and correct the errors that exist in the original source code of the program.

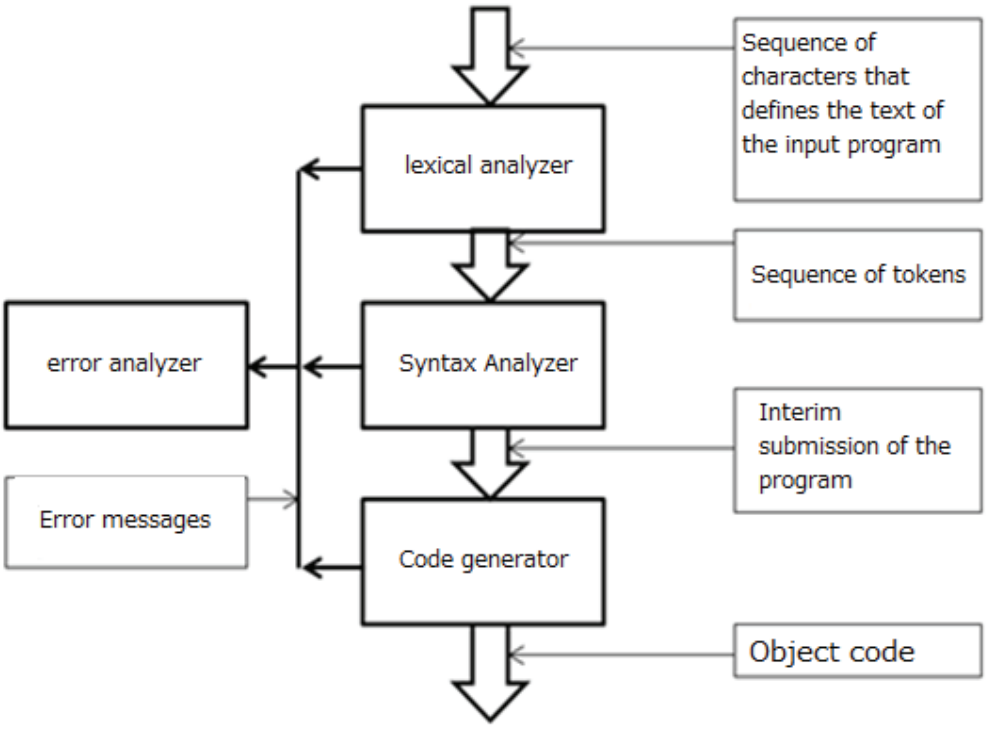

Fig. 1. Structure of the compiler

The syntax analyzer (Fig. 2.) performs the analysis of the input program using the received tokens, the construction of the syntactic structure of the program and semantic analysis with the formation of the object model of language. The object model represents a syntactic structure complemented by semantic links between existing concepts. These connections can be:

• references to variables, types of data, and procedure names that are placed in the names tables;

• links defining sequence of execution of commands;

• links defining the attachment of elements of the object model of language and others.

Thus, the parser is a fairly complex block of the translator. Therefore, it can be divided into the following components:

• recognizer;

• semantic analysis unit;

• an object model, or an intermediate representation, consisting of a table of names and a syntactic structure.

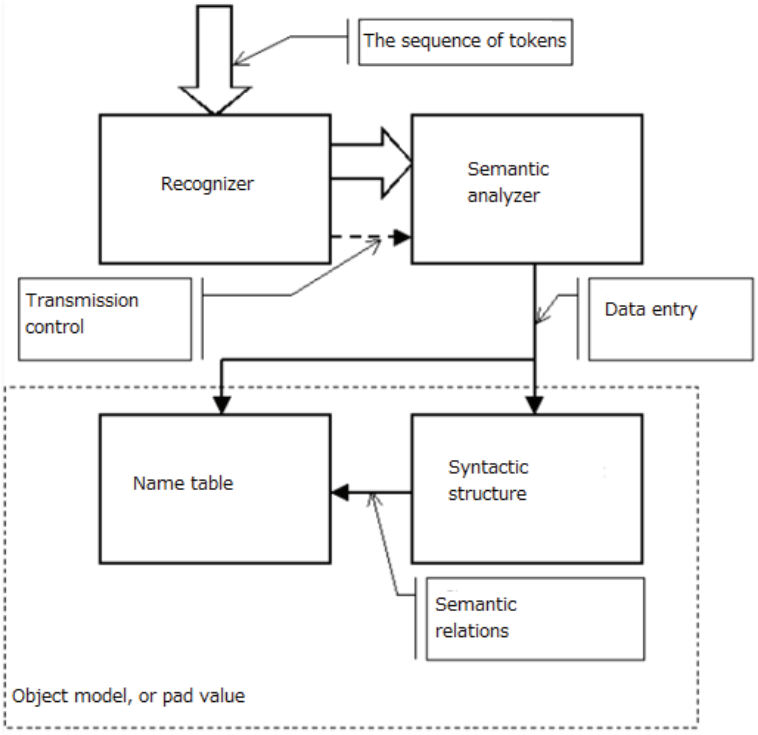

Fig. 2. The general scheme of syntax analyzer

Recognizer receives a chain of tokens and on its basis performs analysis in accordance with the rules used. Lexems, with successful parsing of rules, are transmitted to a semantic analyzer, which builds a names table and captures fragments of the syntactic structure. In addition, between the table name and syntactic structure recorded additional semantic relationships. As a result, the object model of the program is formed, freed from the binding to the syntax of the programming language. Quite often instead of a syntactic structure, it completely copies the hierarchy of objects of the language, creating its simplified analog, called intermediate representation.

Error analyzer receives error information that occurs in different blocks of the translator. Using the information he receives, he generates a message to the user. In addition, this unit may try to correct the error to continue the analysis further. It also relies on actions related to the correct completion of the program in the event that it is not possible to continue the subsequent translation.

The code generator builds the code based on an analysis of the object model or intermediate representation. The construction of the code is accompanied by additional

semantic analysis. At the stage of this analysis, the possibility of conversion is finally determined and effective options are selected. Code generation itself is the re-coding of some commands to others.

An important role in the algorithm is played by the code optimizer, which in this implementation is proposed to embed in the structure of the code generator. The role of the optimizer - before building the code, degrade it, to improve the compatibility of languages, and after the construction - optimize for a better result.

The importance of the previous decomposition of complex structures, on simpler (degradation), is the irreversibility of some transformations. For an example, let's take C ++, Java and Pascal. At first glance, the structure of these operators is very similar (Fig. 3), but if the Pascal language checks and increments (decrements) occur exclusively with the cycle variable in the beginning, in C ++ and Java, the cycle variable, verification and transformation may has nothing in common. Therefore, it makes sense to schedule such a cycle in languages C ++ and Java in the form:

for (<set initial conditions >; <condition>; <variable transform>)
{<body of the cycle}
in the form of:
<set initial conditions>;
while (<condition>) {
<body of the cycle >;
<variable transform>;
}

In this form, any C ++ and Java language cycle can be translated into Pascal without causing errors.

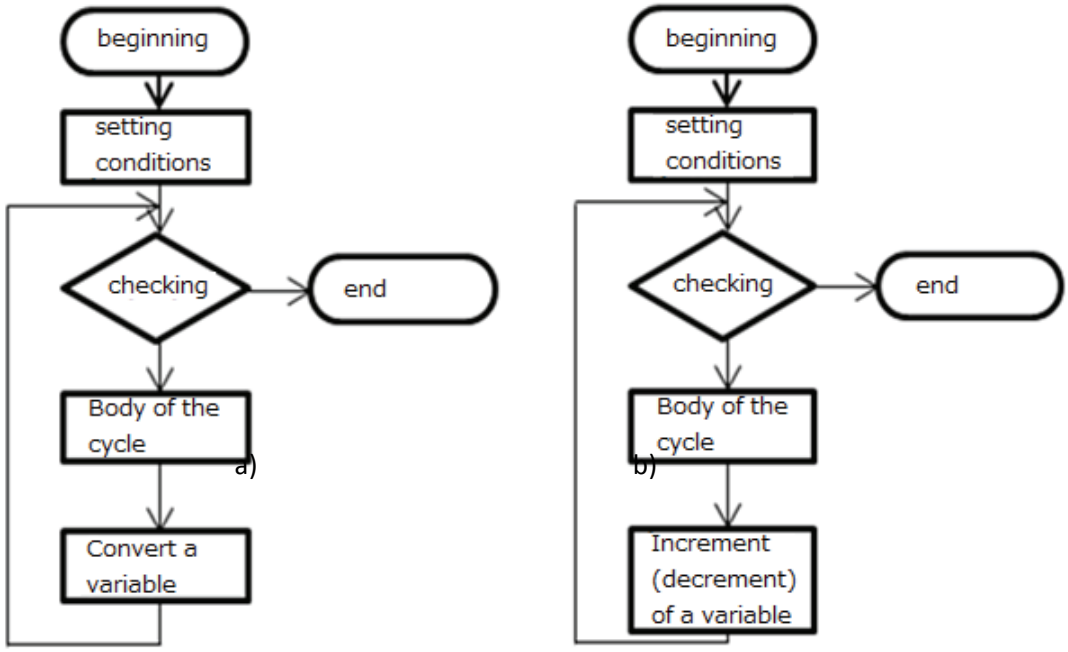

Fig. 3. The structure of cycle “for” a) in Java and C ++ languages; b) in Pascal

Due to such differences, there is a problem of the irreversibility of some transformations. As we see from Fig. 4. If there are only three languages, and one operator, the big part of the transformations can not be inverse. That is why the equally important part is the correct formation of relations between operators, and their structure at the stage of writing syntax languages.

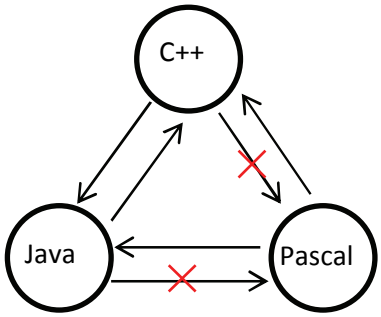

Fig. 4. Ability to convert operator for to other languages

Conclusions. The paper shows an approach to solving the problems of creating a high-level programming language translator. This is achieved through the use of a full-fledged translator model that includes an error detection system, as well as the use of language syntax descriptions and code optimization, which allows for the creation of correct relationships between similar high-level structures.

References

1. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. - М.: Мир, 1978.

2. Кауфман В. Ш. Языки программирования. Концепции и принципы. - М.: Радио и связь, 1993. - 432 с.

3. Льюис Ф., Розенкранц Д., Стринз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979.

4. Brian Alliet. Complete Translation of Unsafe Native Code to Safe Bytecode. Rochester Institute of Technology. URL: http://www.megacz.com/berkeley/research/papers/nestedvm.ivme04.pdf. (дата звернення: 09.08.2009).

5. C2J Converter. URL: http://tech.novosoft-us.com/jsps/downloads.jsp. (дата звернення: 11.11.2001).

6. C to Java Translation. Migration Technology Systems. URL: https://www.mtsystems.com. (дата звернення: 01.02.2018)

7. Our Source Code Converters. Tangible Software Solutions Inc. URL: https://www.tangiblesoftwaresolutions.com/index.html. (дата звернення: 02.05.2019).

8. J. W. Backus. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. Proceedings of the International Conference on Information Processing, UNESCO, 1959, pp.125-132.

Authors

Pavlov Valery - associate professor, Ph.D., Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: pavlovvg@ukr.net.

Павлов Валерій Георгійович – доцент, к.т.н, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Zakupin Yehor – student, Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: egoza343@gmail.com.

Закупін Єгор Олександрович – студент, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

РОЗШИРЕНА АНОТАЦІЯ

Єгор Закупін,
Валерій Павлов

СТВОРЕННЯ ПЕРЕКЛАДАЧІВ МОВ ПРОГРАМУВАННЯ ВИСОКОГО РІВНЯ

Актуальність теми дослідження. Постійне збільшення кількості мов програмування високого рівня може створювати суттєві проблеми при створенні уніфікованого програмного забезпечення. Таким чином виникає необхідність створення перекладача для полегшення та прискорення розробки програмного забезпечення за рахунок використання вже існуючих модулів в незалежності від мови програмування, на якій вони були написані.

Постановка проблеми. Відсутність програм, що дозволяли гнучко вибирати вхідні та вихідні мови, а також ненадійність існуючих рішень через відсутність аналізу вхідного тексту, що значно ускладнює процес використання подібного програмного забезпечення.

Аналіз останніх досліджень та публікацій. Зазвичай, переклад програм з однієї мови програмування на іншу базується на теорії побудови компіляторів, але для випадку, коли й вхідна, і вихідна мови є мовами високого рівня розглядається лише крос-компіляція. Існує досить невелика кількість публікацій, присвячених проблемі перекладу мов програмування. Існуючи рішення або

стосуються вирішання локальних задач, або спираються на застаріли версії мов програмування, або є суто комерційними проектами, де на безкоштовній основі пропонуються лише демоверсії.

Виділення недосліджених частин загальної проблеми. У статті пропонується аналіз застосування нового підходу для перекладу між мовами програмування високого рівня. Він ґрунтується на використанні попереднього аналізу тексту вхідної мови, а також використанні зовнішніх даних для надання інформації, щодо мов програмування.

Постановка завдання. Завданням є створити програмний додаток, що надає можливість перекладу вхідного тексту, написаного на однієї мові високого рівня, також на іншу мову високого рівня. При цьому програма має проводити попередній аналіз вхідного тексту, а інформацію щодо структури мов програмування брати з зовнішніх файлів.

Викладення основного матеріалу. Метод рішення можна розподілити на два етапи. На першому етапі буде проводиться запис синтаксису мови програмування, оскільки від того, в якому вигляді буде записаний синтаксис, залежить робота перекладача. На другому етапі проводиться розробка транслятору.

Висновки. В роботі показаний підхід до вирішення задачі створення перекладача мов програмування високого рівня. Це досягається за рахунок використання повноцінної моделі транслятора, що включає в себе систему виявлення помилок, а також використанням описів синтаксису мов та оптимізатору коду, що дозволяє створювати коректні співвідношення між подібними структурами мов високого рівня.

Ключові слова: мова програмування, перекладач.