OPEN ACCESS

Journal of Creativity, Media, and Communication Studies | Vol. 1 No. 2 (2025)

Research Article

Translation of Natural Language to Source Code in the Case of Repetition in Pascal Language

Adi Yusuf[1*], Aditya Kusuma Setyanegara[2], Muhammad Raafi Febrian Tara[3]
[1]Departement of Informatics Engineering, Faculty of Engineering and Computer Science, Universitas Komputer Indonesia, Indonesia
[2]Departement of Computer Science, Faculty of Mathematics and Natural Sciences, IPB University, Indonesia
[3]Departement of Islamic Criminal Law, Faculty of Sharia, State Islamic University of Raden Mas Said Surakarta, Indonesia
Email: [1)] adiyusuf25@gmail.com, [2)] adityakusuma365@gmail.com, [3)] raafi.febriantara@gmail.com

Received:	Revised:	Accepted:	Online:
February 26, 2025	March 20, 2025	April 22, 2025	April 23, 2025

Abstract
Source code in computer science is a set of commands used to solve problems written in languages that can be understood by computers. Programming languages use concise syntax, and due to this conciseness, they avoid redundancy and ambiguity. However, these languages are often difficult to learn because they require strict adherence to specific syntactic rules. One approach to minimize the difficulty for programmers is to allow program writing using natural language, which is more flexible and easier for humans to understand. Therefore, this research aims to translate natural language, specifically Indonesian, into Pascal source code. The method applied is a rule-based method consisting of three stages: preprocessing (case folding and filtering), analysis (scanning and parsing), and translation. The preprocessing stage removes noise, the analysis stage checks conformity with the defined grammar, and the translation stage converts Indonesian commands into Pascal syntax. Testing was conducted using 60 Indonesian command texts across three types of loop structures (for do, while do, and repeat until). The system achieved an accuracy of approximately 95%. The results show that the rule-based approach is effective in translating repetitive structures. However, improvements to the grammar and parsing rules are recommended to address translation errors and further enhance system accuracy.

Keywords: Natural Language Processing, Pascal Language, Programming Languages, Source Code, Translation

1. Introduction

Machine translation is the ability of computers to understand human language and translate it into other languages (Budiharto & Suhartono, 2014). To be able to respond to a language in the form of text and then automatically translate it into another language according to human wishes, the translation machine must implement NLP (Natural Language Processing) or Natural Language Processing. The translation engine has been implemented in research in the field of NLP like Google Translate. In the programming language, repetition or loop can be done a number of times or until the loop stop condition is reached. With repetition, a programmer no longer writes the same code or process repeatedly. Source code has writing rules that have been determined by the particular programming language used, to be able to reduce the difficulty of programmers in writing programs without having to understand in detail the rules of writing from certain programming languages is to do program writing in natural languages, because of their flexibility and ease of use by humans, reducing the need for extensive training, and possibly using speech recognizers for input data (Biermann et al., 1983).

The study of translating natural language into source code is not a new study but has been previously studied (Biermann et al., 1983; Dirgahayu et al., 2017), research conducted by A. Biermann, B. Ballard and A. Sigmon translates natural language into source code. In the research of A. Biermann, B. Ballard and A. Sigmon have been able to detect 81% of English sentences correctly and the overall success rate is 73.9% (Biermann et al., 1983). And research conducted by Dirgahayu, Huda, Zukhri, and Ratnasari (Dirgahayu et al., 2017), can handle cases of collision, branching, and repetition. It's just that the research still uses input text with the pseudocode format in Indonesian (Dirgahayu et al., 2017). The similar research conducted by Kohar (2019) is the translation of natural language which is converted into the Pascal programming language. However, in that study, Kohar (2019) it was only able to handle the case of the buildup and could not yet handle the case of repetition. Therefore, this research will build a translation system from natural language in Indonesian to source code.

Based on the description above, this researcher will add complexity to the grammar both in Indonesian and in the programming language and add translation features to be able to solve problems in previous studies that have not been handled by previous studies (Kohar, 2019). The significance of this research lies in its contribution to improving code generation systems from natural language, especially in Bahasa Indonesia, by making programming more accessible and supporting a wider range of cases. This advancement is expected to aid novice programmers and open opportunities for further development of educational tools and voice-assisted coding systems.

2. Literature Review

2.1. Natural Language Processing

Natural language processing is part of AI (Artificial Intelligence). Natural language processing examines communication between humans and computers with natural language intermediaries that humans have. In doing that, natural language sent to a computer must be processed first so that it can be understood by the computer (Budiharto & Suhartono, 2014). One of the challenges in natural language processing is the choice of meanings of words that have more than one meaning, such as the word 'critical', the word 'critical' can mean 'roof' and can also mean 'critical' according to the sentence form (Desiani & Arhami, 2006). Research that can handle cases of words that have more than one meaning is by means of the Part of Speech Tagger, as conducted by (Purnamasari & Suwardi, 2018). System answering questions, summarization, speech recognition, document classification, and machine translation are research in the field of natural language processing. Machine translation is research that aims to make the computer understand the natural language entered and translate it into another desired language.

2.2. Pascal Programming Language

Pascal programming language is a high-level programming language and its orientation is on all purposes (Kadir, 1991). Pascal programming language is widely used in the world of education. Because Pascal is relatively easy to read and the notation is similar to standard English (Rinaldi, 2011).

2.3. Scanning

Scanning is the stage of sorting input text into tokens based on the class. In this study, the scanning phase receives word stream input which then sorts the input text into lexic units or tokens (Utdirartatmo, 2005). The scanner also reads the input characters one by one to determine the primitive word units, or "tokens", for each command. Then the scanning tokens will become input data at the Parsing stage.

2.4. Parsing

Parsing is the process of determining how a terminal string can be generated by a grammar (Aho et al., 2007). The parsing method is divided into two namely top down and bottom up.

3. Methods

In this study, a system built using the rule-base method is determined based on the case raised.

3.1. System Overview

In this study, the system built will be able to translate Indonesian into source code in Pascal. The system built has three main processes, namely preprocessing (case folding and filtering), the analysis process (scanning and parsing), and the last is the translation process.

At the preprocessing stage, the system performs case folding to convert text to lowercase, followed by filtering to remove unnecessary characters, leaving only valid input such as letters, numbers, punctuation, and spaces.

Next, in the analysis stage, the scanning process breaks the input into tokens and categorizes them using a grammar dictionary. These tokens are then passed to the parsing process, which checks whether the token structure matches the defined grammar rules.

In the final translation stage, matched tokens are converted into Pascal source code based on rule-based mappings, producing output that corresponds to the original Indonesian commands.

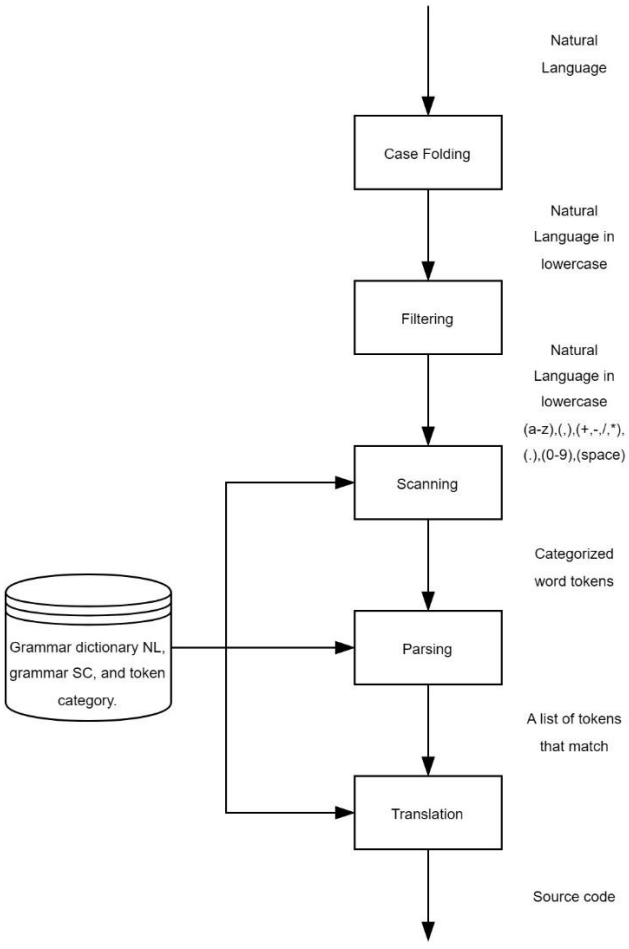

Figure 1. Entire system diagram blocks

4. Results and Discussion

In this chapter we will discuss the research conducted and the results obtained from this study.

4.1. Problem Analysis

The problem in previous research conducted by Kohar (2019), has not been able to deal with repetitive translation and at the scanning stage has not been able to classify two or more words so that more than one word will automatically be classified as a string.

Table 1. Table Prior Research Processes

No	Kohar (2019) Research	Dirgahayu et al. (2018) Research
1	Case folding	Case folding
2	Filtering	
3	Scanning	Pseudocode Translation
4	Parsing	Validation
5	Translation	Source code Translation

In this research, system will translate natural language in Indonesian into a source code in Pascal will be built. The system built focuses on translating the repetition of the Pascal programming language. Iterations in Pascal language are divided into several types, namely: for to do, for down to do, while do, repeat until, nested repetition.

4.2. Data Input Analysis

In this study, input data that can be received by the system in the form of natural language text in Indonesian that is structured in solving problems, follows the rules of writing Pascal language that starts from making program titles, variable declarations, and program content sections. Examples of input data are shown in Table 2.

Table 2. Input Data Sample

No	Data Input
1	Buat aplikasi looping. Buat variabel i dengan tipe data bilangan bulat. selama i lebih kecil dari 10 lakukan Tampilkan “benar” dan i tambah 1 masukkan ke i.
2	Buat program hitung. buat variabel x dengan tipe data bilangan bulat. ulangi tampilkan menulias bahasa pemrograman pertama lalu x ditambahkan 1 dimasukkan ke x.

4.3. Preprocessing Process

Preprocessing aims to get clean input data so that it is ready to be used in the analysis and translation process. Preprocessing in this study consists of three stages, namely case folding and filtering.

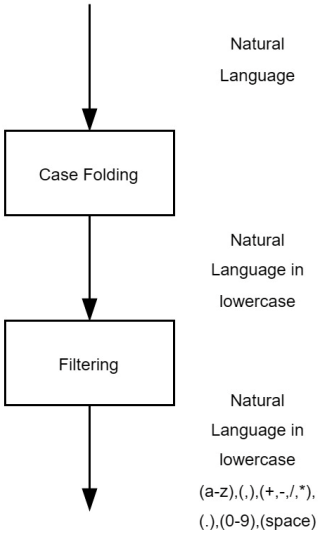

Figure 2. Block Preprocessing Process Diagram

4.3.1. Case folding

In this study all input data is converted to lowercase letters. Data input which has become lowercase letters will be used for the filtering process.

Table 3. Example of Case Folding

Before
Selama i lebih kecil dari 10 lakukan Tampilkan “benar” dan i tambah 1 masukkan ke i.
After
selama i lebih kecil dari 10 lakukan tampilkan “benar” dan i tambah 1 masukkan ke i.

4.3.2. Filter

In this study the only characters allowed to enter the analysis stage are the characters a-z, 0-9, ‘_’, ‘-‘, comma (‘,’), dot (‘.’), And space.

Table 4. Example of Filtering

Before
selama i lebih kecil dari 10 lakukan tampilkan “benar” dan i tambah 1 masukkan ke i.
After
selama i lebih kecil dari 10 lakukan tampilkan benar dan i tambah 1 masukkan ke i.

4.4. Scanning Process

Scanning is the process by which the system reads the input characters one by one to determine the word token unit.

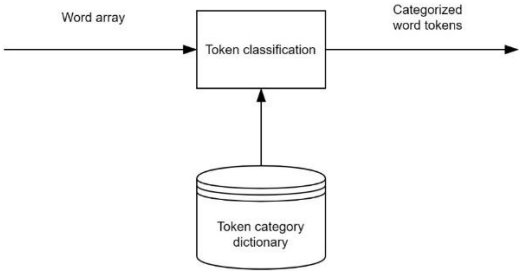

Figure 3. Block Scanning Process Diagram

In Figure 3, it has two stages, namely checking the word class and checking the next word class. Word class checking is looking for the first token class. If the first token class is detected, then the token class will be saved. Then the first word is combined with the next word. If the combination of the first word and the next word results in a new token class, then the new token class will be saved, but if not, the old token class will be saved.

Table 5. Token List and Class

No	Data Input
Arithmetic Operator	tambah, ditambah, tambahkan, ditambahkan, kurang, kurangi, dikurangi, kali, dikali, kalikan, dikalikan, bagi, dibagi, dikurang.
Keyword	program, aplikasi, variabel, var, peubah, tipe, integer, string, masuk ke, masukan, dimasukan, isi, isikan, diisikan, tampil, tampilkan, ditampilkan, baca.
Additional Token	buat, buatkan, buatlah, kemudian, lalu, dan, dengan, data, nilai, nilainya, hasil, hasilnya, datanya, desimal, dari.
IdentApp	[a..z, 0..9, '_']
IdentVar	[a..z, 0..9, '_']
String	[a..z, 0..9, '_']
Number	[0..9, ',']
Delimiter	',', '.'

In this study, the addition of tokens was carried out in order to handle previous research problems.

Table 6. List of Additional Tokens

No	Data Input
Arithmetic Operator	sisa bagi, hasil bagi, sisa bagi, lebih besar, lebih kecil, lebih besar sama dengan, lebih kecil sama dengan, bernilai sama, tidak sama dengan, sampai, lebih dari, lebih besar dari, kurang dari, lebih kecil dari, lebih dari sama dengan, lebih besar sama dengan dari, kurang dari sama dengan, lebih kecil sama dengan dari.
Keyword	selama, maka, untuk, bernilai, pada, ketika, ulangi, sehingga, lakukan, sama dengan, ulang, kerjakan, bilangan bulat.
Additional Token	perulangan, iterasi, terus, selanjutnya, berikutnya.

Examples of scanning results from the filtering data contained in Table 4 can be seen in Table 7.

Table 7. Example of Scanning

Before	
selama i lebih kecil dari 10 lakukan tampilkan "benar" dan i tambah 1 masukkan ke i.	
After	
Token	Class
selama	Keyword
i	IdentVar
lebih kecil dari	ArithmeticOperator
10	Number
lakukan	Keyword
tampilkan	Keyword
Benar	String
dan	AdditionalToken
i	IdentVar
tambah	ArithmeticOperator
1	Number
masukkan	Keyword

Ke	AdditionalToken
I	IdentVar
.	Delimiter

4.5. Parsing Process

The parsing stage is the stage of checking the order in which Indonesian text input appears from the scanning process.

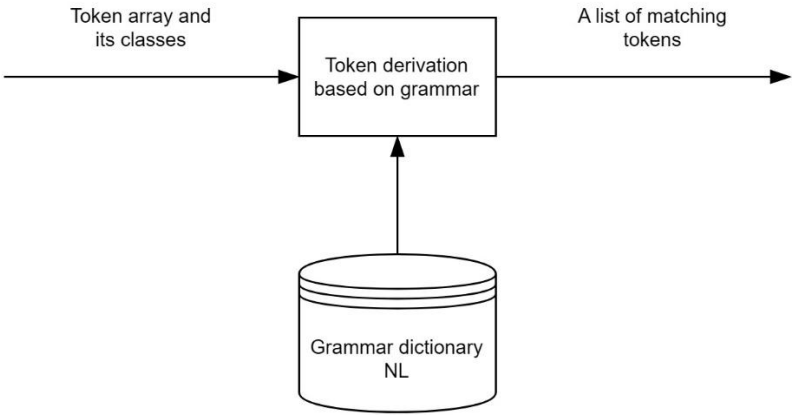

Figure 4. Block Parsing Process Diagram

In Figure 4, the system will lower the token based on the specified grammar.

Table 8. Example of the parsing

<LOOPING>
<LOOPING_FOR> <LOOPING_WHILE> <LOOPING_REPEAT>
<KEYWORD_WHILE> <EKSPRESI> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama <EKSPRESI> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama <FACTOR> <MATH_OPR> <EKSPRESI_1> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama <FACTOR> <MATH_OPR> <FACTOR> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i <MATH_OPR> <FACTOR> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari <FACTOR> <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 <LOOP_OPR> <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan <IO_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan <OUTPUT_STATEMENT> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan <KEYWORD_OUTPUT> <EKSPRESI> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan tampilkan <EKSPRESI> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan tampilkan <FACTOR> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan tampilkan <STRING> <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>

selama i lebih besar dari 10 lakukan tampilkan benar <ORDINARY_CONJUNCTION> <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan tampilkan benar dan <CHANGE_CONDITION>
selama i lebih besar dari 10 lakukan tampilkan benar dan <IDENT_VAR> <MATH_OPR> <FACTOR> <INPUT_OPR> <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i <MATH_OPR> <FACTOR> <INPUT_OPR> <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i tambah <FACTOR> <INPUT_OPR> <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i tambah <NUMBER> <INPUT_OPR> <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i tambah 1 <INPUT_OPR> <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i tambah 1 masukkan ke <IDENT_VAR>
selama i lebih besar dari 10 lakukan tampilkan benar dan i tambah 1 masukkan ke i

4.6. Translation Process

This translation process is the stage of changing the input data into the Pascal programming language. This translation phase will be carried out when the parsing process is received. The translation process has five stages, namely the removal of additional tokens, changing tokens, sorting tokens, syntactic adjustments in the Pascal language, and tidying the code.

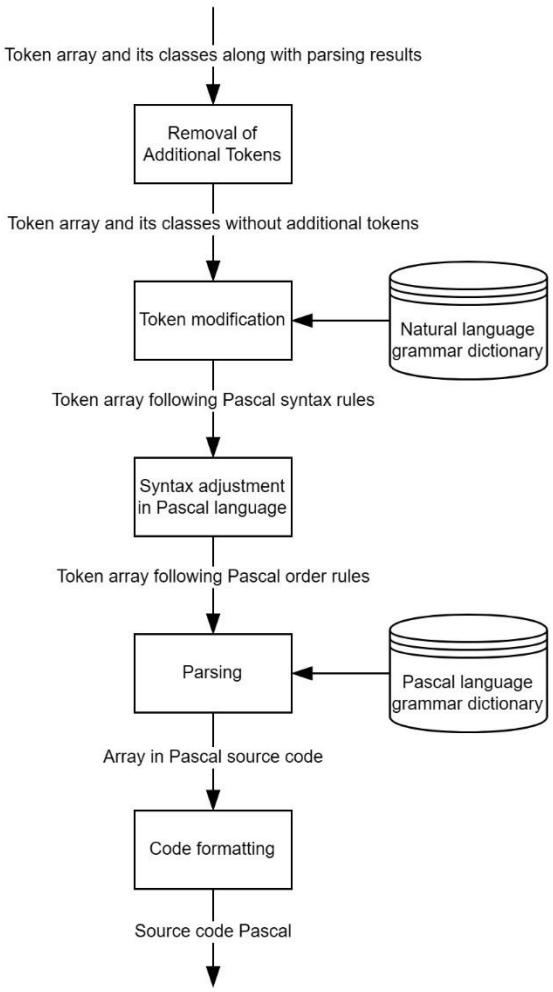

Figure 5. Block Translation Process Diagram

4.6.1. Additional Tokens Removal

This additional token removal step is the stage to remove tokens that will not be used, Tokens that have the 'AdditionalToken' class will be deleted by the system.

Table 9. Example of Additional Token Removal

Before	
Token	Class
selama	Keyword
i	IdentVar
lebih kecil dari	ArithmeticOperator
10	Number
lakukan	Keyword
tampilkan	Keyword
Benar	String
dan	AdditionalToken
i	IdentVar
tambah	ArithmeticOperator
1	Number
masukkan	Keyword
ke	AdditionalToken
i	IdentVar
.	Delimiter
After	
Token	Class
selama	Keyword
i	IdentVar
lebih kecil dari	ArithmeticOperator
10	Number
lakukan	Keyword
tampilkan	Keyword
benar	String
i	IdentVar
tambah	ArithmeticOperator
1	Number
masukkan	Keyword
i	IdentVar
.	Delimiter

4.6.2. Changing Token

The token change stage is the stage to change tokens other than tokens that have the AdditionalToken class into the pascal programming language.

Table 10. Example of Token Changing

Before	After
selama	while
i	i
lebih kecil dari	<
10	10
lakukan	do
tampilkan	writeln
benar	benar
i	i
tambah	+
1	1

masukkan	:=
i	i
.	;

4.6.3. Tokens Position Mapping

The token position mapping stage is the stage of adjusting the order in which tokens appear according to Pascal rules.

Table 11. Token Position Mapping

Before	After
while	while
i	i
<	<
10	10
do	do
writeln	writeln
benar	benar
i	i
+	:=
1	i
:=	+
i	1
;	;

4.6.4. Syntax Adjustment in Pascal Language

The syntax adjustment stage in Pascal language is the stage to adjust the syntax according to the rules in Pascal language but not in natural language writing, for example 'begin', '(', ')', '" ',';',' end '.

Table 12. Syntax Adjustments in Pascal Language

Before	After
while	while
i	i
<	<
10	10
do	do
writeln	begin
benar	writeln
i	(
:=	'
i	benar
+	'
1	)
;	;
	i
	:=
	i
	+
	1
	;
	end

4.6.5. Code Tidying

The code tidiness stage is the stage where tokens that are in accordance with the rules in the Pascal language will be tidied up so that they are easy to understand.

Table 13. Code Tidying

Before	After

while
i
<
10
do
begin
writeln
(
‘
benar
‘
)
;
i
:=
i
+
1
;
end
;
	while i < 10 do
	begin
	writeln (‘benar‘);
	i := i + 1;
	end;

4.7. Analysis of Testing Result

Accuracy testing is done by comparing the translation results obtained from the system with the expected results. Tests are carried out on commands for do, call while do, and repeat until. The test is carried out on 60 test data which are divided into 3 combinations of orders. The results of translational testing from Indonesian to source code in Pascal can be seen in Table 14.

Table 14. Example of Accuracy Testing Results

Command Combinations	Number of Test Data	Translation Result	
		Correct	False
The ‘for do’ function	20	19	1
The ‘while do’ function	20	19	1
The ‘repeat until’ function	20	19	1
Total	60	57	3

The overall accuracy achieved by the system is 95%. This high accuracy indicates that the system is effective in translating Indonesian natural language commands into Pascal source code for loop structures.

The results show consistent translation performance across all three types of loop commands, each achieving 95% individual accuracy. This consistency suggests that the grammar rules and parsing mechanisms implemented in the system are generally robust across different syntactic structures. However, the small number of errors encountered also highlights specific weaknesses in the system’s current implementation. Notably, the errors occurred when the parser failed to distinguish between

decimal values written with a comma (e.g., "2,5") and a list of values, or when it misidentified variables as string literals.

These findings reinforce the need for improved lexical analysis and semantic understanding in the system. Enhancing the system's ability to contextually differentiate between variable names and data types could reduce such misinterpretations.

This study is limited to the translation of loop structures and does not yet address other programming constructs such as conditional statements, functions, or nested loops. Additionally, the system's handling of ambiguous or colloquial expressions in natural language is still limited, which may affect usability in broader contexts. The use of manually defined rules also restricts the scalability and flexibility of the system when dealing with more complex or diverse language inputs.

5. Conclusion

Based on the testing that has been conducted, the system successfully translates Indonesian-language instructions into Pascal source code, particularly in the context of repetition structures. The model achieved an accuracy of 95%, indicating that the approach is effective and reliable for the intended use case. This result demonstrates that the use of rule-based translation combined with syntactic analysis can produce accurate code generation in structured programming languages.

These findings suggest that the proposed system can serve as a foundational tool for assisting beginners in programming, especially in understanding how natural language instructions are transformed into actual code. Furthermore, the success of this system opens up opportunities for expansion into other programming constructs such as conditional statements, functions, or even object-oriented paradigms.

As a suggestion for future work, it is recommended to enhance the system's capabilities to cover more diverse programming structures and to support multiple programming languages. Additionally, integrating machine learning techniques may improve flexibility and reduce reliance on manually crafted rules.

5.1. Author Contributions

This research was conducted with equal contributions from all authors. The first author was responsible for developing the translation module into the Pascal programming language, including algorithm implementation and system testing. The second author played a role in collecting and analysing the data used in the research, including processing test data and validating results. The third author was responsible for composing the scientific journal, improving the writing structure, and making revisions based on feedback from readers and reviewers.

5.2. Conflicts of Interest

The authors declare that there is no conflict of interest in this research. This study was conducted as a contribution to the development of knowledge in the field of Natural Language Processing (NLP) and the translation of natural language into source code in Indonesia. The purpose of this research is purely for academic development and to enhance understanding of automatic translation techniques in the programming world.

6. References

Aho, A. V, Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, techniques and tools, 2nd editio*. United States of America: Pearson Education Limited.

Biermann, A. W., Ballard, B. W., & Sigmon, A. H. (1983). An experimental study of natural language programming. *International Journal of Man-Machine Studies*, *18*(1), 71–87.

Budiharto, W., & Suhartono, D. (2014). Artificial Intelligence Konsep dan Penerapannya. *Yogyakarta: Andi*.

Desiani, A., & Arhami, M. (2006). Konsep kecerdasan buatan. *Penerbit Andi, Yogyakarta*.

Dirgahayu, T., Huda, S. N., Zukhri, Z., & Ratnasari, C. I. (2017). Automatic translation from pseudocode to source code: A conceptual-metamodel approach. *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 122–128.

Kadir, A. (1991). Pemrograman Turbo Pascal untuk IBM PC Menggunakan Versi 5.5. *Jakarta: Elex Media Komputindo*.

Kohar, M. (2019). *Penerjemah Bahasa Alami Dalam Bahasa Indonesia Ke Source Code Dalam Bahasa Pascal*. Universitas Komputer Indonesia.

Purnamasari, K. K., & Suwardi, I. S. (2018). Rule-based Part of Speech Tagger for Indonesian Language. *IOP Conference Series: Materials Science and Engineering*, *407*(1), 12151.

Rinaldi, M. (2011). Algoritma dan Pemrograman. *Bandung: Informatika*.

Utdirartatmo, F. (2005). Teknik Kompilasi. *Yogyakarta: Graha Ilmu*.