Hanami: A Novel Programming Language for Modern Computing Challenges

Author1
Department of Computer Science
University Name
City, Country
email@domain.edu

Author2
Department of Computer Science
University Name
City, Country
email@domain.edu

Author3
Research Laboratory
Organization Name
City, Country
email@domain.edu

Author4
Research Laboratory
Organization Name
City, Country
email@domain.edu

***Abstract*—Abstract**

This paper introduces Hanami, a transpiled programming language developed as an educational exploration by students in CS3370 - Nature of Programming Languages. Hanami serves as a practical investigation into programming language design principles, incorporating features such as a gradient type system, contextual execution models, and effect handling mechanisms. Rather than compiling to machine code, Hanami transpiles to established languages including C++, Java, JavaScript, and Python, enabling cross-platform execution through these host languages. We present our design philosophy, implementation challenges, and insights gained throughout the development process. This project demonstrates how academic exploration of language concepts can materialize into a functional prototype while providing valuable learning experiences about language paradigms, type systems, and transpiler design. Through Hanami, we've created a practical framework for understanding the theoretical underpinnings of programming languages while applying these concepts in a tangible educational context.

***Index Terms*—programming languages, type systems, effect systems, concurrent programming, distributed systems, formal verification**

CONTENTS

I. Introduction

Programming languages play a crucial role in software development, each designed with specific goals and trade-offs. As students in CS3370 - Nature of Programming Languages, we've been exploring these concepts through the development of Hanami, a transpiled programming language created as an educational project.

Modern programming involves working with various languages, each with distinct characteristics: Java offers portability, C++ prioritizes performance, Rust focuses on memory safety, and Python emphasizes readability. Through our coursework, we recognized how these trade-offs impact development choices and wondered if a student project could explore alternative approaches.

A. Motivation and Educational Objectives

Our motivation for creating Hanami stems from three key observations in our programming language studies:

- Type systems typically follow either static or dynamic approaches, with few options in between
- Programming languages often make fundamental design choices that limit their flexibility across different use cases
- Learning compiler/transpiler design principles requires hands-on experience with language implementation

Hanami addresses these educational objectives through a unique approach - instead of compiling to machine code, it transpiles to established languages including C++, Java, JavaScript, and Python. This design decision allows us to focus on language design principles while leveraging existing compilation infrastructure.

The name "Hanami", inspired by the Japanese tradition of flower viewing, reflects our design philosophy. Just as hanami celebrates the beauty of cherry blossoms, our language aims to showcase the elegant aspects of programming language design we've studied in this course.

B. Educational Impact and Academic Significance

The development of Hanami provides several significant educational benefits:

- **Practical Application of Theory**: Students gain first-hand experience implementing theoretical concepts from programming language design.
- **Cross-language Understanding**: The transpilation approach deepens understanding of how different languages implement similar concepts.
- **Compiler Construction Knowledge**: Each phase of development reinforces compiler construction principles in a concrete, accessible way.
- **Design Trade-off Analysis**: Students make deliberate design choices, weighing the consequences of each decision.

This project bridges the gap between theoretical knowledge and practical implementation, allowing students to experience the challenges and complexities of language design that are difficult to appreciate through traditional coursework alone.

TABLE I
Hanami Keyword Examples

Hanami	C++ Equivalent	Meaning
garden <Name>	namespace <Name>	Declares a namespace
species <Name>	class <Name>	Declares a class
grow <Name>() -> <type>	<type> <Name>()	Declares a function
bloom << x;	std::cout << x;	Prints to console

C. Technical Overview and Implementation Approach

Key features of our implementation include:

- A lexical analyzer (lexer) that converts source code into tokens
- A parser that builds an abstract syntax tree (AST) from these tokens
- A semantic analyzer that verifies program correctness
- A transpiler that converts Hanami code to target languages

Hanami's syntax draws inspiration from multiple languages while introducing nature-themed keywords aligned with its name. For instance, "garden" replaces "namespace," "species" defines classes, and functions are declared using "grow." This design choice makes the language both distinctive and memorable while serving as a practical demonstration of syntax design principles.

D. Innovative Features and Distinguishing Characteristics

Hanami incorporates several innovative features that distinguish it from typical student projects:

- **Multi-target Transpilation**: Unlike most educational language projects that target a single platform, Hanami transpiles to multiple languages, enabling cross-platform compatibility.
- **Nature-Inspired Lexical Design**: The consistent nature theme provides a cohesive and memorable programming experience while demonstrating naming convention principles.
- **Graduated Type System**: Rather than adopting a purely static or dynamic approach, Hanami implements a type system with configurable strictness levels, allowing students to explore the spectrum between these paradigms.
- **Effect Handling**: Inspired by modern research in programming languages, Hanami includes basic effect handling mechanisms that allow for more controlled side effects.

These features were carefully selected to balance educational value with implementation feasibility while showcasing important concepts in modern programming language design.

E. Paper Organization

Through this project, we've gained practical insights into compiler construction stages including lexical analysis, parsing, semantic analysis, and code generation. Our implementation demonstrates these concepts while creating a functional transpiler that converts Hanami code to multiple target languages.

TABLE II
HANAMI KEYWORD EXAMPLES

Hanami	C++ Equivalent	Meaning
garden	namespace	Declares namespace
species	class	Declares class
grow	function	Declares function
bloom	cout	Prints to console

The remainder of this paper is organized as follows: Section II details Hanami's language design and features. Section III describes our transpiler implementation, including lexical analysis, parsing, and code generation. Section **??** presents sample programs and their transpiled outputs. Section **??** discusses challenges encountered and lessons learned. Section **??** summarizes our findings and suggests potential improvements.

This project represents our practical exploration of programming language concepts and compiler design principles learned throughout the CS3370 course.

II. Language Design and Core Features

Hanami is a programming language designed as an educational project in the CS3370 course. Our design philosophy centers on creating a language that students can use to gain a deeper understanding of how programming languages work.

A. Language Syntax and Semantics

Hanami employs a syntax designed to balance readability with expressiveness. The core syntax draws inspiration from multiple paradigms, combining the clarity of Python, the type expressiveness of TypeScript, and some features of Rust.

Hanami's keywords are inspired by nature, creating a unique experience for users and distinguishing it from other languages. For example:

B. Core Language Features

Although Hanami is a learning project, we focused on implementing several core features to better understand programming language design:

1) Transpilation to Multiple Target Languages: One of Hanami's key features is its ability to transpile source code to various target languages. Our transpiler system currently supports generating code for C++, Java, JavaScript, and Python. This allows us to learn about how different languages handle similar concepts and how to implement them effectively.

2) Type System: Hanami's type system is designed to be simple yet powerful enough to support basic data types, arrays, and user-defined structures. During development, we learned about the complexity of mapping between data types in different target languages.

3) Function System: Hanami supports both regular functions and class methods. The function declaration syntax is inspired by Rust and TypeScript, with return types specified after an arrow:

```
grow add(x: int, y: int) -> int {
 return x + y;
}
```

4) Object-Oriented Programming: To better understand object-oriented programming models, Hanami supports basic concepts like classes (species), inheritance, and encapsulation. This implementation has helped us understand how different languages handle OOP.

III. Implementation

Developing Hanami has helped us understand the stages involved in designing and building a programming language. Our transpiler includes the following components:

- **Lexer**: Converts source code into tokens
- **Parser**: Builds an abstract syntax tree (AST) from tokens
- **Semantic Analyzer**: Analyzes program structure and verifies consistency
- **Code Generator**: Produces code for different target languages

Each component was a valuable learning opportunity about compiler design and programming language principles.

IV. Evaluation

During Hanami's development, we evaluated the language's performance and usability through several simple program examples:

- Basic algorithms (sorting, searching, string processing)
- Small programs demonstrating OOP (student management system, simple games)
- Basic file handling and

Although Hanami is not designed to compete with commercial programming languages, this evaluation provided valuable insights into factors affecting programming language performance and usability.

A. Limitations and Future Work

As a student project, Hanami has several limitations that could be addressed in future work:

- Limited support for advanced features (generics, lambda expressions)
- Performance of generated code could be further optimized
- Error handling and reporting needs improvement
- Standard library expansion

These limitations also present learning opportunities for future projects in the CS3370 class.

References

[1] A. Author, "Title of reference 1," *Journal Name*, vol. 10, pp. 1–10, 2020.
[2] B. Author, "Title of reference 2," *Journal Name*, vol. 11, pp. 11–20, 2021.
[3] C. Author, "Title of reference 3," *Journal Name*, vol. 12, pp. 21–30, 2022.
[4] D. Author, "Title of reference 4," *Journal Name*, vol. 13, pp. 31–40, 2023.
[5] E. Author, "Title of reference 5," *Journal Name*, vol. 14, pp. 41–50, 2023.
[6] F. Author, "Title of reference 6," *Journal Name*, vol. 15, pp. 51–60, 2023.
[7] G. Author, "Title of reference 7," *Journal Name*, vol. 16, pp. 61–70, 2023.
[8] H. Author, "Title of reference 8," *Journal Name*, vol. 17, pp. 71–80, 2023.
[9] I. Author, "Title of reference 9," *Journal Name*, vol. 18, pp. 81–90, 2023.
[10] J. Author, "Title of reference 10," *Journal Name*, vol. 19, pp. 91–100, 2023.
[11] K. Author, "Title of reference 11," *Journal Name*, vol. 20, pp. 101–110, 2023.