Interactive Story Compiler: A Visual Approach to Context-Free Grammar Interpretation

Laura Beltrán, Santiago Sánchez
Dept. of Computer Engineering
Universidad Distrital Francisco José de Caldas
Email: lavbeltrans@udistrital.edu.co, sansanchezm@udistrital.edu.co

***Abstract*—Traditional compilers translate structured programming languages into machine code, but natural language inputs remain a challenge for computational models. This paper proposes a compiler that interprets structured natural language stories and generates interactive HTML narratives using concepts from automata theory and context-free grammars. Initial design and planning show the feasibility of mapping story branches to state transitions and implementing a visual interface driven by formal parsing.**

***Index Terms*—Interactive story, compiler, context-free grammar, finite-state machine, natural language processing**

I. Introduction

Interactive storytelling systems, particularly those inspired by the "choose-your-own-adventure" format, have gained popularity in digital education, entertainment, and user-driven narratives. Bridging this with formal methods from computer science offers a unique opportunity to apply theoretical constructs such as finite-state machine and grammars in a tangible, engaging way. Our project proposes a compiler that receives natural language stories with structured branching logic and converts them into interactive visual documents.

This initiative arises from the need to make formal language theory more applicable and visual, especially in educational contexts. Prior tools such as Twine and Ink allow similar interactivity, but rely on specific scripting languages. Our compiler, instead, starts from near-natural text and enforces structure using concepts from computation theory.

II. Objective

The main objective of this project is to design and implement a lightweight compiler capable of interpreting structured natural language stories and transforming them into interactive, visual outputs. By applying core principles of formal language theory—such as context-free grammars, lexical analysis, and finite-state machines—the compiler aims to bridge theoretical computer science with creative storytelling. Specifically, the project seeks to provide an educational tool that allows users to write branching narratives in a simple textual format and instantly visualize them through either HTML-based or graphical interfaces. This not only reinforces compiler design concepts but also fosters user creativity and engagement through tangible, interactive results.

III. Methods and Materials

The design of our solution begins with the definition of a simple compiler that can interpret structured natural language stories and generate interactive visual output in the form of an HTML page. The proposed compiler will accept a minimalistic story format that includes one initial scene, two decision options, and two corresponding outcome scenes. This structure is intentionally limited to ensure clarity, reduce complexity, and focus on demonstrating fundamental principles of compiler design and formal language theory.

To implement the compiler, we plan to use Python due to its ease of use, strong support for text processing, and the ability to generate files. The input will be a plain text file that follows a predefined syntax, using keywords such as `scene`, `text`, and `choice` to mark different parts of the narrative. For example, a line starting with `scene:` will indicate the beginning of a new visual state, while `choice:` will specify the user's decision and the resulting scene it leads to. This structure resembles a context-free grammar where each scene is a non-terminal and choices act as transitions between production rules.

Internally, the compiler will parse the input using simple string operations or regular expressions, storing the information in an intermediate data structure that mirrors a finite-state machine. Each state will represent a scene, and each transition will represent a user choice. This representation will then be used to generate a static HTML file, where each scene is embedded as a hidden section. JavaScript functions will control visibility, allowing the user to navigate between scenes by clicking on buttons, thus simulating state transitions interactively.

The output will be a standalone HTML file that can be opened in any web browser without the need for additional software or installation. We plan to use basic HTML, CSS, and JavaScript to build a clean and minimal interface. Each scene will display a background image and two buttons corresponding to the available choices. When a button is clicked, the interface will transition to the appropriate next scene.

Our choice to use a visual output rather than compiled machine code aligns with the educational objectives of this project. It allows us to explore compiler concepts—such as lexical analysis, parsing, and code generation—while keeping the final product tangible and easy to test. This also provides a way to demonstrate how theoretical models like context-free grammars and finite-state machines can be applied in a creative and interactive context.

IV. Expected Results

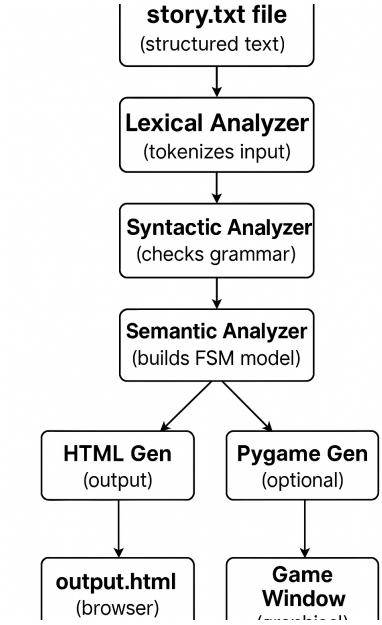
In future iterations, the compiler will:

- Parse a structured natural language story following a predefined format with scenes and choices.
- Validate the story's syntax using simple parsing rules inspired by context-free grammars.
- Generate a self-contained HTML file with embedded CSS and JavaScript to display the story and allow interactive navigation between scenes.

To validate the implementation, we will use manually crafted stories that follow the input specification. These test cases will help verify the parsing logic, correct rendering of scenes, and the functionality of choice-based transitions. Although the system is intentionally limited to a small number of scenes and options, it is expected to clearly demonstrate how compiler principles can drive user interactivity. The final output should offer a smooth, browser-based experience that showcases the connection between input structure and visual narrative.

V. Development

The implementation of the compiler followed a four-phase architecture inspired by traditional compilation pipelines: lexical analysis, syntactic parsing, semantic validation, and code generation. Each phase was implemented as an independent Python module to ensure modularity and separation of concerns.



The lexical analyzer (lexer.py) scans the input .txt file and tokenizes the content based on regular expressions. It recognizes reserved keywords (scene:, text:, choice:), arrows, identifiers, and quoted strings. These tokens are passed to the syntactic analyzer (syntactic.py), which checks whether the token sequence adheres to the predefined grammar. The grammar was designed as a minimal context-free grammar to maintain simplicity and determinism during parsing.

After syntax validation, the semantic analyzer (semantic.py) builds an internal dictionary-based structure that mirrors a finite-state machine. It verifies the consistency of scene references and ensures that each choice leads to a defined scene. This semantic check avoids broken links in the final story.

Finally, the code generation phase (compiler.py) outputs a self-contained HTML file. Each scene is rendered as a hidden ¡div¿ element that becomes visible when the user selects a corresponding choice. Simple JavaScript logic controls these transitions, allowing a smooth, browser-based exploration of the interactive narrative.

An optional extension includes a graphical version of the story experience using the a visual interface. In this mode, the same parsed data structure is rendered in a windowed interface, where scenes and choices are displayed with stylized fonts, colors, and buttons..

VI. Preliminary Results

Initial testing was conducted using a sample story file with four scenes and branching choices. The compiler successfully tokenized and parsed the input, generating a functional HTML file where scene transitions worked as expected. Each scene was displayed with its corresponding narrative text and choices, and clicking a button dynamically updated the content without refreshing the page.

The compiler also correctly flagged syntax errors in malformed input files, such as missing colons, unquoted strings, or undefined destination scenes. This helped refine the robustness of the parsing logic and improve error handling during development.

The Pygame output mode, while more visually engaging, requires more precise layout handling and event management. Preliminary builds demonstrate that the same story structure can be navigated through mouse clicks in a graphical window. Although more limited in styling compared to HTML/CSS, the Pygame interface adds a gamified dimension to the user experience.

These findings align closely with the expectations outlined earlier. The compiler successfully parsed structured input using the designed grammar, validated syntax and references, and generated a fully functional HTML file with scene transitions. Furthermore, the introduction of a new interface as a secondary output exceeded the original scope by offering a graphical alternative to the browser-based interface. The system's ability to handle incorrect inputs and display interactive narratives demonstrates that the original objectives were not only met but extended, validating the feasibility and educational value of the proposed approach.

Overall, these results confirm the feasibility of the proposed architecture and highlight the potential of this tool as a teaching aid for compiler design, parsing theory, and interactive storytelling.

VII. Conclusions

This project demonstrates the feasibility and educational value of applying compiler theory to structured natural lan-

guage through the development of an interactive story compiler. By integrating lexical, syntactic, and semantic analysis into a clean and modular Python-based pipeline, we successfully converted branching text narratives into fully functional, browser-based interactive stories and graphical game-like interfaces using Pygame.

Beyond the technical implementation, the project bridges abstract formal concepts—such as context-free grammars and finite-state machines—with creative expression. This duality highlights how computational theory can support not only traditional programming tasks but also storytelling, education, and human-centered design.

The compiler's architecture proves scalable, and the clear separation between parsing and code generation allows for multiple output formats, laying the groundwork for future extensions, including richer graphical interfaces, save/load systems, or narrative analytics. Ultimately, this work offers a novel way to visualize compiler pipelines and make theoretical computer science more engaging and tangible for learners.

REFERENCES

[1] Twine, "An open-source tool for telling interactive, nonlinear stories," [Online]. Available: https://twinery.org/

[2] Ink, "Inkle's narrative scripting language" [Online]. Available: https://www.inklestudios.com/ink/