Tyler McCluskey

CMPSC 470 Section 001

Nelson

29 April 2024

**Car Interpreter Language Translator Architecture**

**1. Introduction**

The Car Interpreter Language (CIL) is a domain-specific language designed for controlling simulated car functions. This document outlines the architecture for translating CIL code into executable commands for simulated car control functions.

**2. Components**

**2.1. Interpreter**

The Interpreter component is responsible for parsing and executing CIL code. It consists of the following sub-components:

* Tokenizer: Splits the input code into individual tokens for processing.
* Parser: Analyzes the structure of the code and determines the appropriate actions to take.
* Executor: Executes the parsed commands by invoking the corresponding car control functions.

**2.2. Car Control Functions**

These are the simulated car control functions that CIL commands are translated into. They include:

* **drive()**: Initiates forward motion of the car.
* **stop()**: Stops the car's motion.
* **reverse()**: Engages the car in reverse motion.
* **horn()**: Activates the car's horn.
* **read\_sensor()**: Simulates reading data from sensors, such as obstacle detection.

**2.3. Input Processor**

Responsible for obtaining CIL code input from the user or external source.

**3. Translation Process**

**3.1. Input Processing**

* CIL code is obtained from the user or an external source.
* The code is passed to the Interpreter component for translation.

**3.2. Tokenization**

* The input code is split into individual tokens based on whitespace and syntax.
* Tokens are analyzed to determine their type and meaning.

**3.3. Parsing**

* The tokens are parsed to identify commands and their associated parameters.
* Conditional statements, loops, and arithmetic operations are identified and processed.

**3.4. Execution**

* Parsed commands are executed by invoking the corresponding car control functions.
* Conditional statements are evaluated, and appropriate code blocks are executed based on the conditions.
* Loops are iterated, and the contained code blocks are executed repeatedly until loop conditions are no longer satisfied.
* Arithmetic operations are performed, and results are stored or used as needed.

**3.5. Error Handling**

* Errors such as unknown tokens, syntax errors, and potential infinite loops are detected and reported during the translation process.

**4. Conclusion**

The Translator Architecture described above provides a framework for efficiently translating CIL code into commands for controlling simulated car functions. By leveraging the components and processes outlined in this document, developers can build robust and reliable systems for interpreting and executing CIL commands.