C4: Key Algorithms and Concepts

Introduction

C4 is a remarkably concise C compiler designed to be self-hosting. Its architecture, while simple, demonstrates fundamental compiler principles. This report focuses on the core algorithms employed in C4, specifically lexical analysis, parsing, virtual machine implementation, and memory management.

1. Lexical Analysis: Tokenizing the Input

C4's lexical analysis, or tokenization, is performed by the next() function. It operates on a character-by-character basis, scanning the source code and grouping characters into meaningful tokens.

- **Character Classification:**
  - The process begins by reading a character from the input stream.
  - The character is then classified: is it a letter, a digit, an operator, whitespace, or a special character?
- **Token Recognition:**
  - **Identifiers:** Sequences of letters, digits, and underscores are recognized as identifiers. A simple hashing mechanism is used to store and retrieve identifiers from a symbol table.
  - **Numbers:** Digit sequences are converted into numerical values, handling decimal, hexadecimal, and octal representations.
  - **Operators:** Single and multi-character operators (e.g., +, -, ==, <=) are identified. A lookahead approach is used to distinguish between similar operators (e.g., = vs. ==).
  - **String/Character Literals:** Strings and characters enclosed in quotes are extracted. Escape sequences are handled within the literal.
  - **Comments:** Single-line comments (//) are skipped.
- **Symbol Table:**
  - The symbol table is a simple array that stores information about identifiers, including their type, class, and value.
  - When an identifier is found that is not in the symbol table, it is added.

The next() function maintains a global token variable (tk) and value variable (ival) to represent the current token.

2. Parsing: Constructing an Implicit AST

C4 uses a recursive descent parsing strategy, which is a top-down parsing method. The expr() and stmt() functions are the core components of the parser.

- **Expression Parsing:**
    - The expr() function implements operator precedence using a "precedence climbing" technique.
    - It recursively parses expressions, handling unary and binary operators, function calls, and variable access.
    - Instead of building a explicit AST, the parser directly emits virtual machine instructions as it parses the input, which is a very memory efficient approach.
- **Statement Parsing:**
    - The stmt() function handles control flow statements (if, while, return), compound statements, and expression statements.
    - It uses jump instructions (BZ, BNZ, JMP) to implement control flow.
    - Function definitions are parsed, and local variable allocation is handled.
- **Implicit AST:**
    - C4 doesn't build a traditional AST in memory. Instead, the parsing process directly generates virtual machine instructions, forming an implicit representation of the program's structure.

3. Virtual Machine Implementation: Executing Instructions

C4's virtual machine executes the compiled instructions.

- **Instruction Fetch and Decode:**
    - The virtual machine maintains a program counter (pc) that points to the current instruction.
    - Each instruction is fetched and decoded.
- **Instruction Execution:**

  - The virtual machine implements a set of opcodes (e.g., LEA, IMM, JMP, ADD, PSH).
  - It uses a stack (sp) and base pointer (bp) to manage function calls and local variables.
  - It also manages a data area for global variables and string literals.
- **System Calls:**
  - C4 provides system call instructions (e.g., OPEN, READ, PRTF, MALC) that interface with the operating system.

4. Memory Management: Stack and Heap

C4 uses a combination of stack and heap memory management.

- **Heap:**
  - The heap is used for memory pools that store the symbol table, generated code, data, stack, and source code.
  - malloc() and free() are used for dynamic memory allocation.
- **Stack:**
  - The stack is used for function call frames, local variables, and expression evaluation.
  - The stack pointer (sp) and base pointer (bp) are used to manage the stack.
- **Data Area:**
  - Global variables and string literals are stored in the data area.