The analysis and synthesis model of compilation helps bridge the gap between high-level programming languages and machine-level execution, enabling the development of efficient and portable software applications.

Analysis Phase:

The analysis phase focuses on understanding the structure and meaning of the source code, ensuring its correctness and adherence to syntax and semantics.

1. Lexical Analysis: The compiler breaks down the source code into individual tokens, such as keywords, identifiers, operators, and literals. It removes unnecessary elements like white spaces and comments.

2. Syntax Analysis: The compiler verifies the syntax of the code by checking the arrangement of tokens according to the language's grammar rules. It builds a parse tree or abstract syntax tree (AST) that represents the hierarchical structure of the code.

3. Semantic Analysis: The compiler checks the meaning and context of the code. It ensures that expressions, statements, and declarations adhere to the language's semantic rules. Type checking and symbol table construction are performed to catch any semantic errors.

Synthesis Phase:

The synthesis phase involves generating more efficient representations of the code, optimizing it, and finally producing the target code that can be executed by the computer.

1. Intermediate Code Generation: The compiler may generate an intermediate representation of the source code, which is often platform-independent and provides a more optimized

representation for further processing.

2. Optimization: The compiler applies various optimization techniques to the intermediate code. These optimizations improve the efficiency and performance of the resulting executable. Examples include constant folding, loop unrolling, and dead code elimination.

3. Code Generation: The compiler generates the target code, which can be machine code specific to the target hardware or assembly language closely resembling the machine code. This output code is executable on the target system without the need for further translation.

Related Posts:

1. Introduction to Compiler
2. Bootstrapping and Porting
3. Lexical Analyzer: Input Buffering
4. Storage Allocation Strategies
5. Type Checking
6. Specification & Recognition of Tokens
7. Front end and back end of the compiler
8. LEX
9. Analysis synthesis model of compilation
10. Data structure in CD
11. Register allocation and assignment
12. Loops in flow graphs
13. Dead code elimination
14. Syntax analysis CFGs
15. L-attribute definition

16. Operator precedence parsing
17. Analysis of syntax directed definition
18. Recursive descent parser
19. Function and operator overloading
20. Storage allocation strategies
21. Equivalence of expression in type checking
22. Storage organization
23. Parameter passing
24. Run time environment
25. Type checking
26. Code generation issue in design of code generator
27. Boolean expression
28. Declaration and assignment in intermediate code generation
29. Code optimization
30. Sources of optimization of basic blocks
31. Loop optimization
32. Global data flow analysis
33. Data flow analysis of structure flow graph (SFG)