- **Detail about Unsolvable problems:** Unsolvable problems are those that cannot be solved by any algorithm. One example is the "Halting Problem," where no general algorithm can decide if a given program will halt or continue to run indefinitely. Unsolvable problems typically involve undecidability in computational theory. These problems arise due to inherent limitations in the algorithms and computational models, such as Turing machines, indicating that certain questions will never have a definitive, algorithmic solution.

- **Discuss about Instantaneous Description in a PushDown Automata:** An Instantaneous Description (ID) of a PushDown Automata (PDA) captures the current state, the unread input, and the contents of the stack at any given moment during the computation process. It is usually represented as a tuple (q, w, s), where q is the current state, w is the remaining input string, and s represents the stack content. IDs are used to track the progression of the PDA as it processes the input, making it possible to describe how the automata behaves at any given step, particularly useful in language recognition tasks.

- **Explain the role of symbol table in compiler design:** In compiler design, the symbol table is a data structure used to store information about the identifiers (variables, functions, etc.) used in a program. It serves as a repository that tracks properties such as type, scope, and memory location of each identifier. The symbol table allows the compiler to efficiently perform semantic checks, resolve variable references, and generate correct machine code by ensuring proper variable declaration and usage.

- **What is a synthesised attribute?** A synthesized attribute is one that is computed from the attribute values of its children in a syntax tree. It is associated with a non-terminal symbol in a grammar, and the value is propagated upwards during the parse tree traversal. This is commonly used in semantic analysis, where the semantic information of a construct is derived based on its components, such as calculating the type of an expression based on the types of its sub-expressions.

- **Convert the given statement into three-address code:** The given expression `(((a-b)+(c*d))/((a-b)-(c*d)))` can be converted into three-address code as follows:

```
makefile
Copy code
t1 = a - b
t2 = c * d
t3 = t1 + t2
t4 = t1 - t2
t5 = t3 / t4
```

Here, each intermediate step breaks down the computation into individual operations with temporary variables (t1, t2, etc.), facilitating easier machine code generation and optimization.

- **Comment about unreachable code:** Unreachable code refers to portions of code that can never be executed during program runtime due to logical flaws, such as code written after a return statement. It may result from poor code structuring or logical errors in condition checks. Detecting and eliminating unreachable code is crucial for optimizing performance and preventing resource wastage, as well as improving code readability and maintainability.

- **What is a production rule?** A production rule in context-free grammar defines how a non-terminal symbol can be replaced with a sequence of terminal and/or non-terminal symbols. These rules govern how sentences in a language are derived. In formal grammar, production rules are used to generate strings belonging to the language, and they are key to constructing parsers for programming languages.

- **What is the relevance of buffer pairs and sentinels in lexical analysis?** Buffer pairs are used in lexical analyzers to store input data efficiently, allowing the system to process large amounts of text without frequent I/O operations. A sentinel is a special character placed at the end of the buffer to simplify boundary checks during scanning. The use of buffer pairs and sentinels helps in optimizing lexical analysis by reducing the need for constant boundary condition checks, improving the overall performance of the scanner.

- **For the given grammar, and an input a*b+c, construct a derivation tree:** The input expression `a*b+c` is derived using the provided grammar as follows:

```
mathematica
Copy code
E -> E + E
    -> E * E + E
    -> a * b + c
```

The derivation tree represents how the input string conforms to the grammar rules, showing how the operators and operands are hierarchically related based on precedence and associativity.

- **Describe the Regular Expression for a language Σ = {p,q}, which can contain strings whose length is divisible by 3:** The regular expression for this language consists of strings made up of symbols `p` and `q` such that the length is a multiple of 3. The regular expression would be: `((p|q)(p|q)(p|q))*`, where each group of three characters forms a valid substring. This pattern ensures that only strings with lengths divisible by 3 are accepted, as the expression is repeated in multiples of three characters.

- **Explain the functioning of a lexical analyser:** A lexical analyzer, or scanner, is the first phase of a compiler that processes the source code to divide it into tokens. These tokens are the smallest meaningful units like keywords, identifiers, and operators. The lexical analyzer reads the input character by character, groups them into tokens, and passes them to the parser for further syntactic and semantic analysis.

- **What is the relevance of code optimisation?** Code optimization is the process of transforming a program to improve its performance by reducing resource usage (such as CPU time and memory), without altering its output. This phase aims to produce more efficient code, improving execution speed and lowering power consumption. Optimization techniques include eliminating redundant computations, minimizing memory access, and reordering instructions for better hardware utilization. Efficient code leads to faster, more scalable software.

1. What is the relevance of sentinels in lexical analysis?

Sentinels in lexical analysis are special characters or symbols added to the end of input to simplify the lexical analyzer's code and improve its efficiency. They act as markers that signal the end of the input stream, allowing the analyzer to avoid constantly checking if it has reached the end of the input.

The main relevance of sentinels is that they help reduce the number of comparisons needed during lexical analysis. By adding a sentinel at the end of the input, the analyzer can focus on processing characters without repeatedly testing for the end of input condition. This can lead to cleaner, more efficient code and potentially faster lexical analysis, especially for large input streams.

2. Prove by Contradiction that $\sqrt{2}$ is irrational.

To prove that $\sqrt{2}$ is irrational by contradiction, we start by assuming the opposite - that $\sqrt{2}$ is rational. If $\sqrt{2}$ is rational, it can be expressed as a fraction a/b, where a and b are integers with no common factors. We can write this as: $\sqrt{2} = a/b$, or equivalently, $2 = a^2/b^2$.

Multiplying both sides by b^2, we get: $2b^2 = a^2$. This means a^2 is even, which implies that a must be even (since odd numbers squared are odd). If a is even, we can write it as $a = 2k$ for some integer k. Substituting this back, we get: $2b^2 = (2k)^2 = 4k^2$. Simplifying, we have $b^2 = 2k^2$. This means b^2 is even, which implies b is even. But if both a and b are even, they have a common factor of 2, contradicting our initial assumption that they had no common factors. This contradiction proves that our initial assumption was false, and therefore $\sqrt{2}$ must be irrational.

3. What is syntax directed translation?

Syntax directed translation is a method used in compiler design where the translation of the source language to the target language is guided by the parse tree or abstract syntax tree of the source program. In this approach, semantic actions are associated with each production rule of the grammar, and these actions are executed as the parser recognizes the corresponding syntactic constructs.

The key idea behind syntax directed translation is to perform the translation incrementally as the input is being parsed. This allows for a more streamlined and efficient translation process, as the semantic analysis and code generation can be integrated directly into the parsing phase. Syntax directed translation is often implemented using attribute grammars, where attributes are associated with grammar symbols and used to compute and propagate information throughout the parse tree.

4. Describe the Regular Expression for a language $\Sigma = \{p,q\}$, which can contain strings whose length is divisible by 3.

The regular expression for a language $\Sigma = \{p,q\}$ that contains strings whose length is divisible by 3 can be described as: (p|q)(p|q)(p|q))*

This regular expression uses the alternation operator '|' to allow either 'p' or 'q' in each position, and groups three of these choices together. The asterisk '*' at the end allows this group to be repeated zero or more times, ensuring that the total length of the string is always a multiple of 3.

To break it down further: (p|q) represents a single character that can be either 'p' or 'q'. By grouping this three times (p|q)(p|q)(p|q), we create a pattern that matches exactly three characters. The asterisk after this group allows it to be repeated any number of times, including zero. This ensures that the language includes the empty string (which has length 0, divisible by 3) as well as strings of length 3, 6, 9, and so on.

5. Explain the DAG representation of a basic block.

A Directed Acyclic Graph (DAG) representation of a basic block is a compact way to represent the computations and data dependencies within a sequence of instructions that have no branches except at the end. In a DAG, nodes represent operations or variables, and edges represent the flow of data between these nodes.

The DAG representation offers several advantages in compiler optimization. It eliminates redundant computations by identifying common subexpressions, as identical subexpressions will be represented by the same node in the DAG. It also clearly shows the data dependencies between operations, which is useful for instruction scheduling and parallelization. Leaf nodes in the DAG typically represent input values or constants, while interior nodes represent operations. The DAG can be used to generate optimized code for the basic block by traversing it in a way that respects the dependencies while potentially reordering operations for efficiency.

6. Discuss on handle pruning.

Handle pruning is a technique used in bottom-up parsing, particularly in LR parsing methods, to improve the efficiency of the parsing process. A handle is a substring of the right-hand side of a production that can be reduced to the left-hand side nonterminal in the next step of a rightmost derivation. Handle pruning involves identifying and removing handles that are not viable for the current parse.

The main purpose of handle pruning is to reduce the number of possible reductions that need to be considered at each step of the parsing process. By eliminating handles that cannot lead to a valid parse, the parser can work more efficiently. This is particularly useful in more complex grammars where multiple potential handles might exist at any given point. Handle pruning can be implemented using lookahead symbols and state information in LR parsers, allowing the parser to make more informed decisions about which reductions to perform.

7. Differentiate between decidable and undecidable problems.

Decidable problems are computational problems for which an algorithm exists that can always provide a correct yes-or-no answer in a finite amount of time. For any input to a decidable problem, the algorithm will terminate and produce the correct output. Examples of decidable problems include determining if a number is prime, checking if a string is accepted by a finite automaton, or determining if a propositional logic formula is satisfiable.

Undecidable problems, on the other hand, are problems for which no algorithm exists that can always provide a correct yes-or-no answer in a finite amount of time for all possible inputs. These problems are fundamentally unsolvable by computational means. The most famous example is the halting problem, which asks whether a given program will eventually halt or run forever on a given input. Other examples include determining if two context-free

grammars generate the same language or if a given Turing machine will accept any input. The existence of undecidable problems is a fundamental result in computability theory and has important implications for the limits of computation.

8. What is the role of a lexical analyser?

A lexical analyser, also known as a lexer or scanner, is the first phase of a compiler or interpreter. Its primary role is to read the input source code character by character and group these characters into meaningful units called tokens. These tokens represent the basic elements of the programming language, such as keywords, identifiers, literals, operators, and punctuation symbols.

The lexical analyser performs several important functions in the compilation process. It removes whitespace and comments from the source code, as these are not needed for further processing. It also helps in error detection by identifying lexical errors such as invalid characters or malformed tokens. Additionally, the lexical analyser often maintains a symbol table to keep track of identifiers and their attributes. By breaking down the source code into tokens, the lexical analyser simplifies the work of subsequent phases of the compiler, particularly the parser, which can then work with these higher-level constructs instead of individual characters.

9. How is Non-Determinism introduced in an Automaton?

Non-determinism in an automaton is introduced when the automaton has multiple possible transitions for a given input symbol in a particular state. This means that for the same input and state, the automaton can choose between different next states or actions, as opposed to a deterministic automaton where each (state, input) pair uniquely determines the next state.

Non-determinism can be introduced in several ways. One common method is through ε-transitions (epsilon-transitions), which allow the automaton to change states without consuming an input symbol. Another way is by having multiple transitions with the same input symbol from a single state. In a non-deterministic finite automaton (NFA), for example, a state might have two or more arrows labeled with the same symbol pointing to different states. This non-determinism allows for more compact and intuitive representations of certain languages, although it's important to note that any NFA can be converted to an equivalent deterministic finite automaton (DFA).

10. Explain the role of interchanging statements in the compiler.

Interchanging statements in a compiler refers to the process of reordering program statements as part of code optimization. This technique is used to improve the efficiency of the generated code without altering its semantic meaning. The primary goals of statement interchanging are to enhance instruction-level parallelism, improve cache utilization, and reduce pipeline stalls.

The compiler analyzes the dependencies between statements and determines if their order can be safely changed without affecting the program's output. Common applications include loop optimizations like loop interchange, where nested loops are reordered to improve memory access patterns, and instruction scheduling, where individual instructions are reordered to maximize the use of processor resources. However, the compiler must be careful to preserve

the original program semantics and not violate any data dependencies or control flow constraints when performing these interchanges.

11. Describe FIRST() function in parsers.

The FIRST() function is a crucial component in predictive parsing, particularly for LL(1) parsers. For a given grammar symbol X (terminal or non-terminal), FIRST(X) is the set of terminals that can appear as the first symbol of any string derived from X. If X can derive an empty string (ε), then ε is also included in FIRST(X).

The FIRST() function is used to construct parsing tables and guide the parsing process. It helps the parser decide which production to use when faced with a non-terminal symbol. By looking at the next input token and comparing it with the FIRST set of each production for that non-terminal, the parser can choose the correct production to expand. This allows for efficient top-down parsing without backtracking. The FIRST() function is typically computed recursively for all grammar symbols as part of the parser construction process.

12. Construct a State Transition Diagram over $\Sigma = \{0,1\}$ which accepts set of all strings ending with '0'.

To construct a State Transition Diagram that accepts all strings over $\Sigma = \{0,1\}$ ending with '0', we need two states: a non-accepting state and an accepting state. Let's call them q0 (non-accepting) and q1 (accepting).

The State Transition Diagram would look like this:

- We start in state q0.
- From q0, if we read a '0', we move to q1 (accepting state).
- From q0, if we read a '1', we stay in q0.
- From q1, if we read a '0', we stay in q1.
- From q1, if we read a '1', we move back to q0.

This diagram ensures that the automaton is in the accepting state q1 if and only if the last symbol read was a '0'. It can handle strings of any length, including the single-character string "0".

CopyRetry