1. **Go Backend (Analysis)**: The Go backend reads the Monke source code from a file, scans it to generate tokens, and parses these tokens into an **Abstract Syntax Tree (AST)** or **Parse Tree**. This process involves **traversing the tree in pre/postorder** and possibly **multiple passes** for different analysis stages.
2. **Intermediate Representation (IR)**: The Go backend then traverses the AST or Parse Tree to generate an **IR** (Intermediate Representation). This IR is a series of Variable, Expression, and Statement objects (as defined in the .proto file) that represent the semantics of the original Monke code.
3. **gRPC or JSON Service**: The Go frontend then uses gRPC or JSON to send this IR to the Python backend. In the case of gRPC, the Go backend would call the Translate method of the MonkeTranslator service, passing the IR as an argument. In the case of JSON, Monke would serialize the IR to a JSON string and send this string to Paw over HTTP or some other protocol.
4. **Python (Code Generation)**: The Paw receives the IR and deserializes it back into Variable, Expression, and Statement objects. It then traverses these objects to generate corresponding Python code. This could involve multiple passes and different traversal orders for optimization purposes.
5. **Python Code**: Paw returns the generated Python code as a PythonCode object (as defined in the .proto file). This code can then be written to a file, executed, or further processed as needed.