**Definition of Syntax-Directed Translation**

**Syntax-directed translation (SDT)** is a technique in compiler design where the translation of source code (e.g., into intermediate code, object code, or other representations) is directly guided by the syntactic structure of the input, as determined by the parser during syntax analysis.

In essence, it integrates semantic actions with the parsing process, allowing the compiler to perform tasks like type checking, code generation, or building data structures (e.g., abstract syntax trees) while parsing the source code. The parser drives the entire translation process, making it "syntax-directed."

**Key Concepts and Components**

SDT is built on **context-free grammars (CFGs)** augmented with additional mechanisms:

1. **Attributes**: Values associated with grammar symbols (terminals or non-terminals). These carry semantic information.

    - ✓ **Synthesized attributes**: Computed from the attributes of children nodes in the parse tree (bottom-up flow). Common for values like expression results or types.
    - ✓ **Inherited attributes**: Passed down from parent or left sibling nodes (top-down or left-to-right flow). Used for context like type information propagated to children.

2. **Semantic Rules/Actions**: Code fragments or rules attached to grammar productions that compute attribute values or perform actions (e.g., emitting code).

**Types of SDT**

- **S-Attributed SDT**: Uses **only synthesized attributes**. Easy to implement with bottom-up parsers (e.g., LR parsers). Attributes flow bottom-up.
    - ✓ Suitable for simple translations like evaluating expressions or generating postfix code.
    - ✓ Example: Translating infix to postfix without needing context from above.
- **L-Attributed SDT**: Allows both synthesized and inherited attributes, but inherited ones can only come from the parent or left siblings (left-to-right dependency). Compatible with top-down parsers (e.g., LL parsers).
    - ✓ More powerful for handling context-sensitive info, like type propagation in declarations.
    - ✓ Every S-attributed SDT is also L-attributed.

**How SDT Works in Practice**

1. The parser constructs a parse tree (explicitly or implicitly).
2. Semantic actions are triggered during parsing:

- ✓ In bottom-up parsing: Actions often at reduction (end of production).
- ✓ In top-down: Actions interleaved during descent.

3. Attributes are evaluated in a depth-first, left-to-right manner (or bottom-up for S-attributed).
4. Output: Annotated parse tree, intermediate code, symbol table entries, etc.

**Applications in Compilers**

- Building **Abstract Syntax Trees (ASTs)** (condensed parse trees omitting syntactic details).
- Type checking (propagate types via attributes).
- Generating intermediate code (e.g., three-address code).
- Symbol table management.
- Error reporting and recovery.

SDT bridges syntax analysis with semantic analysis and early code generation, making compilers modular and efficient.

**Advantages**

- Structured and modular: Translation logic tied directly to grammar.
- Efficient memory use: Can translate without full parse tree.
- Supports both top-down and bottom-up parsing.