SDD and SDT

10 April 2025
Thursday

Syntax Directed Definition (SDD):

Syntax Directed Definition (SDD) are a generalization of context Free Grammar in which,

1. Grammar symbols have an associated set of attributes.
2. Productions are associated with semantic rules for computing the values of the attributes.

attribute → val

SDD: CFG + Semantic Rules.

example:

$$E \rightarrow E + T \; \{ E.val = E.val + T.val \}$$

\# For simplicity, we are considering only one attribute, right now.

\# প্রত্যেকটা production rule এর সাথে তার val attribute associate থাকবে,

\# For each production rule, there can be multiple attributes as well.

Syntax Directed Translation (SDT):

Syntax Directed Translation (SDT) is done by attaching rules or program fragments to production in grammar.

$$E \rightarrow E + T \; \{ \text{print "1"} \}$$

(print "1" → program fragment.)

\# There can be semantic rules or program fragments or mix ture of both.

SDD → CFG + Semantic Rules.

$$\begin{cases} S \rightarrow E & \{ S.val = E.val \} \\ E \rightarrow E + T & \{ E.val = E.val + T.val \} \\ E \rightarrow T & \{ E.val = T.val \} \\ T \rightarrow T * F & \{ T.val = T.val * F.val \} \\ T \rightarrow F & \{ T.val = F.val \} \\ F \rightarrow digit & \{ F.val = digit.lexval \} \end{cases}$$

↳ digit terminal টার actual lexeme এর যে value আছে সেটা সে ধরবে।

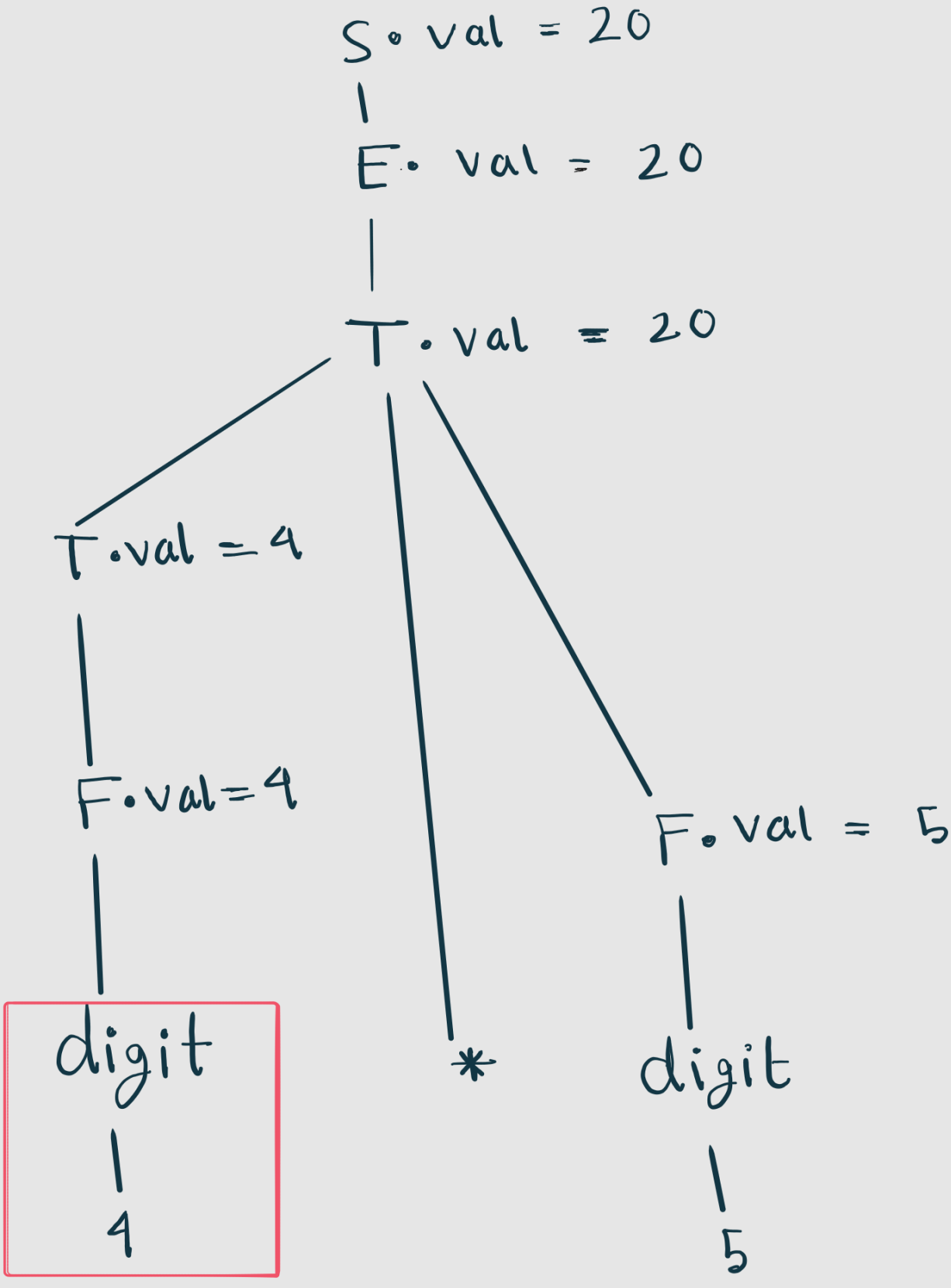
\# Theory concepts easily বুঝার জন্য যদি production rule এর head and body তে যদি same symbol থাকে তাহলে body এর ঐ symbol এ 1, 2, 3 ...... reference দিতে পারি.

\# Code করার সময় object reference হবে,

input string : 4 * 5 = 20
digit * digit

▨ based on the given SDD generate the parse tree for the given input string also find out the attributes.

[P.T.O]



↳ digit → actual lexeme value 4.

Types of Attributes

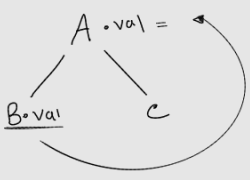
There are two types of Attributes.

- Synthesized Attributes
- Inherited Attributes.

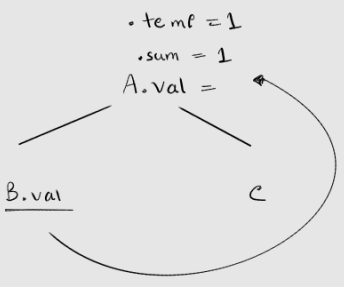
Synthesized Attributes: A synthesized attribute at node N is defined only in terms of attribute values at the children of N and N itself.

> Value, either child থেকে আসবে অথবা ঐ নিজের node থেকেই আসবে,

$A \rightarrow BC \quad \{A.val = B.val\}$



$A \rightarrow BC \quad \{A.val = B.val\}$ # child থেকে value
$\{A.sum = 1\}$
$\{A.temp = A.sum\}$ # নিজের থেকে নিজে value



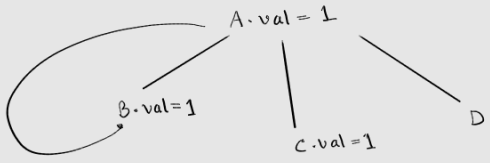
child থেকে এবং নিজের থেকে নিজে help নিচ্ছে তাই synthesized attribute

Inherited Attribute:

An inherited attributed at node N is defined only in terms of all attribute values at N's parent, N itself and N's siblings.

> • Parent থেকে
> • নিজের থেকে
> • Siblings থেকে.

$A \rightarrow BCD \quad \{A.val = 1\}$
$\{B.val = A.val\}$
$\{c.val = 1\}$
$\{B.val = c.val\}$
$\{c.sum = 1\}$



[P.T.O]

Evaluation Orders for SDD

3 things are needed for maintaining the order

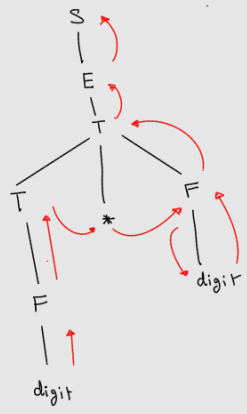
→ Dependency Graph.

→ S - attributed SDD

→ L - attributed SDD

Dependency Graph:

A dependency graph helps us to determine how the values will be computed. It tells us about which order we should evaluate the tree.



কোনটা থেকে কোনটায় যাবে তা computer কে বলতে হবে।

Tips: Same Tree তেই evaluate করব এবং dependency graph show করব

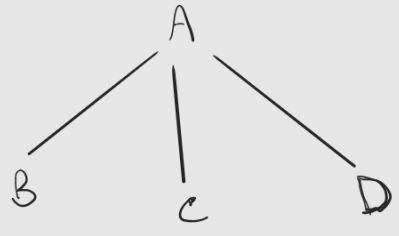
S - Attributed SDD:

A SDD is S Attributed if every attribute is synthesized.

L - Attributed SDD:

A SDD is L Attributed if it uses synthesized and inherited attribute with a restriction that inherited attribute can inherit values from the left sibling only.

$A \rightarrow BCD$



B can't access from anyone.

C can access from B only

D can access from B and C only.

L - attributed SDD

Question Types:

1. ☐ Definition of SDD and SDT ; The difference between SDD and SDT.
2. ☐ What is Synthesized Attribute and what is inherited attribute, difference between them.
3. ☐ From a given parse tree find out the synthesized and inherited attributes
4. ☐ What is dependency graph?
5. ☐ What is L_attributed SDD and what is S-attributed SDD.; and difference between these two.

SDD vs SDT

Aspect	Syntax Directed Definition (SDD)	Syntax Directed Translation (SDT)
Nature	Declarative	Procedural
Components	Grammar, attributes, semantic rules	Grammar, embedded actions, possibly attributes
Focus	What is computed (attribute values)	What and how to translate
Evaluation Order	Not specified (determined by dependencies)	Explicitly defined by action placement
Implementation	Requires separate attribute evaluation algorithm	Directly integrated with parsing
Abstraction Level	Higher-level specification	Closer to implementation
Parse Tree	Decorates parse tree with attributes	May not build explicit parse tree
Flexibility	More flexible for theoretical analysis	More practical for implementation
Readability	Generally cleaner for semantic specifications	Can become complex with many actions
Common Applications	Academic analysis, formal specifications	Practical compiler implementation

Synthesized vs Inherited Attributes

Aspect	Synthesized Attributes	Inherited Attributes
Information Flow	Bottom-up (child to parent)	Top-down or left-to-right (parent to child or left sibling to right sibling)
Dependencies	Children's attributes and own terminal value	Parent's attributes or left siblings' attributes or own's attributes.
Parsing Compatibility ✓	Natural fit for bottom-up parsing	Better with top-down parsing
Evaluation ✗	Single bottom-up pass sufficient	May require multiple passes or specialized algorithms
Implementation Complexity	Simpler to implement	More complex handling required
Common Uses	Expression evaluation, aggregating information	Passing context, environment information
Example Applications	Type checking expressions, code generation	Symbol table management, scope handling
Pass Requirements	Single post-order traversal	May require pre-order or multiple traversals
Dependency Direction	Depends only on subtree	Depends on context outside subtree
Definition Location	At the production's left-hand side	At symbols on production's right-hand side

L-attributed vs S-attributed SDDs

Aspect	S-attributed SDD	L-attributed SDD
Attribute Types	Only synthesized attributes	Both synthesized and inherited (with restrictions)
Information Flow	Bottom-up only	Both bottom-up and restricted top-down/ left-to-right
Parsing Technique	Works well with bottom-up parsing	Works well with top-down parsing
Evaluation Complexity	Simple bottom-up evaluation	Left-to-right evaluation with dependency management
Expressive Power	Limited	Greater than S-attributed, practical for most tasks
Pass Requirements	Single bottom-up pass	Single left-to-right pass possible
Dependency Restrictions	Natural bottom-up dependencies	Inherited attributes have left-to-right restrictions
Implementation Ease	Easier to implement	Moderate complexity
Common Applications	Expression evaluation, simple type checking	Type checking, symbol table management, code generation
Context Handling	Limited context handling	Good context handling capabilities