SMVS
Page No.
Date : / /

Typing

* Explain Motivating Typing with Dynamic and Static Typing.

=> Motivating Typing :

In XML data is represent using a hierarchical structure of element and attributes.

XML itself does not force data types on the values contained within the element and Attribute.

-> Motivating XML use :

1 Data Integrity : Defining types ensure that data conforms to the expected such as integer, dates or any other specific pattern.

2 Vaildation : XML Schema Allows You to validate XML documents against predefined rule.

3 Interoperability : System that exchange XML data can rely on well-defined data type to interupt data correctly.

Date : / /

4 Documentation : XML Schema servers as documentation for data structure which help to understand type of data.

5 User defined Syntax : XML does not follow pre-defined tag syntax. So, User can write its own undertable tag in XML Document.

=> Dynamic Typing :

Dynamic Typed Language are checked at runtime not complie time.

This means that variable can hold value of any type that can change when program is run

In Dynamic Typing, Type Declaration does not needed.

Type-related error are detect when program is executed

Example : Python, JavaScript

SMVS
Page No.
Date : / /

=> Static Typing :

Static Typing Language are checked at complie time rather than runtime.

Variable must be explicitly declared with a type and Type is fixed when it declared.

In Static Typing, Type Declaration is required and gives better performance

Type-related error are detect when program is complie.

Ex. Java, C++.

* Typing Graph Data :

Graph data in XML involves representing and managing data structures that are more complex than simple trees.

XML can handle graph data through mechanism like ID and IDRF in DTD.

Page No. SMVS
Date : / /

ID : An Attributes type that uniquely identifiers an element within XML Document.

IDRF : An Attribute type that refers to an element with corresponding ID.

=> Example :

- DTD :

```
<root>
    <node id="1"> Root Node </node>
    <node id="2" ref="1">
        Child Node </node>
</root>
```

-

- Schema

```
<xs:schema xmlns:xs="  ">
<xs:element name="node">
  <xs:complexType>

    <xs:attribute name="id"
      type="xs:ID"
        use="required" />

    <xs:attribute name="ref"
      type="xs:IDRF"
        use="Optional" />
```

SMVS
Page No.
Date : / /

```
    </xs:complexType>
  </xs:element>

  <xs:key name="nodeID">
    <xs:selector xpath="node" />
    <xs:field xpath="@id" />
  </xs:key>

  <xs:keyref name="nodeRef"
      refer="nodeID">
    <xs:selector xpath="node" />
    <xs:field xpath="@ref" />
  </xs:keyref>

</xs:schema>
```


* ~~Gap~~

* Graph Semistructured Data

=> Object Exchange Model (OEM):

An OEM is a model used to describe graph data and its characterized by:

1) Nodes (N) : Set of All Nodes

2) Edges (E) : Each Edge has a label start From L.

3) Root (r): A specific node in a graph that serves as Root.

The Object Exchanges Model is represented as a labeled, rooted Graph (N, E, r) where,

N = Nodes
E = Edges
r = Root

* Graph Bisimulation:

Graph Bisimulation is a concept used to classify and compare graph structures based on their structural properties.

A Graph can be considered as having a type if it is ~~di~~ bisimilar to a simpler or more abstract graph.

Bisimulation is a method to abstract or simplify complex graph by creating a similar graph.

SMVS
Page No.
Date : / /

=> Definition :

1 Simulation :

A Relation s between the nodes of two graph (E, r) and (E', r') is called simulation, if

(i) S(r, r') => Roots of Both Graph are related.

(ii) For every edge E(s, t, l) in (E, r) if S(s, s') and E(s, t, l) then there exist t' such that S(t, t') and E'(s', t', l) in (E', r').

This means,
If You can move node s to node t in Graph E
than, You should able to move node s' to node t' in Graph E'

2 Bisimulation :

A Relation S is a bisimulation if :

- S is Simulation of (E, r) with (E', r')

- The Inverse of S is simulation of (E, r) a with (E', r')

S Being Bisimulation means that two graph (E, r) and (E', r') can simulate each other's structure.

* Graph : Data Guide

Data Guide are a method used to type and navigate graph data by focusing on the paths from the root.

They Provide way to represent and explore the structure of path data.

A Data Guide focuses on path within the graph.

It represents sequences of nodes and edges starting From root.

Data Guide is used to provides structured way to ~~m~~ navigate

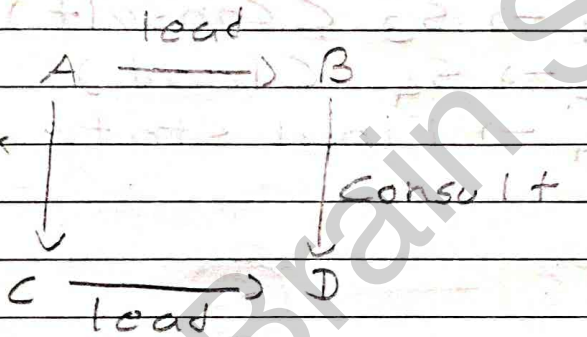
SMVS
Page No.
Date : / /

through the graph.

Data Guide is used to generate deterministic automata for efficient navigation and quering.

A Deterministic Finite Automaton can be used to recognize the regular Language defined by the path in graph.

Ex.



(Path Based Representation)

Root Node A

Path 1: A lead B Consult D
A → B → D

Path 2: A work C
A → C

Path 3: A work C lead D
A → C → D

Path 4: A lead B
A → B

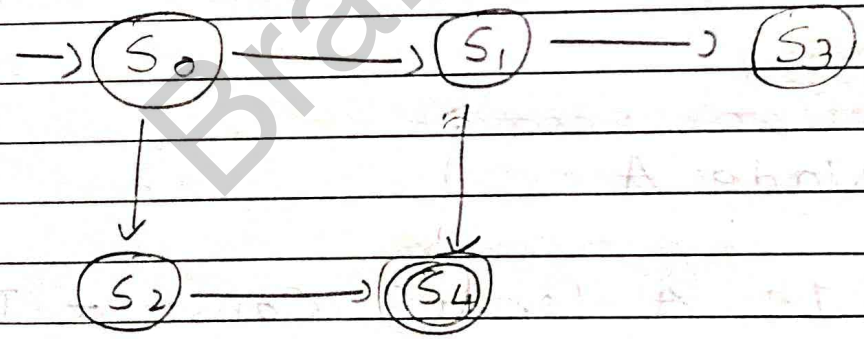
-> Regular Expression

A (lead B consult D | lead B | work C consult D | work C)

-> DFA:

Start : S_0 State

Transition: $S_0 \rightarrow S_1$ (lead)
$S_0 \rightarrow S_2$ (work)
$S_1 \rightarrow S_3$ (consult)
$S_2 \rightarrow S_4$ (lead)
$S_4 \rightarrow$ Final state



=> Limitation:

1 Non-Determinism:

The Automation derived From Graph is highly Non-Deterministic

SMVS
Page No.
Date : / /

2 State Explosion:

To Convert NDA → DFA Gives more state

3 Limited Information: Different Graph can give same Data Guide.