Answer: Database Theory 2004 – Paper 8 Question 9 (GMB)

(a) [Bookwork] Semistructured data is often described as "schema-less" or "self-describing", in that unlike the relational or object models we do not insist on a schema. That is we have no schema language or data definition language. The primary reason for interest in the semistructured data model is the flexibility that it offers. In semistructured data the schema information as such is contained in the data, and can vary over time and within the database.

Semistructured data is best represented as a graph. Thus a database of semistructured data is a collection of node. Each node is either a leaf node or an interior node. Leaf nodes have data values associated with them, which is of some atomic type (integers, strings, etc.). Interior node have one or more arcs out. Each arc has a label. We do typically impose any restrictions on the labels. Thus it is valid to have more than one arc from the same node with the same label. Typically we will distinguish one node as being the root node. Thus, in conclusion, semistructured data is an edge-labelled graph.

We can define a simple linear syntax for semistructured data as follows:

```
ComplexValue ::= {label:SSDExp, ... , label:SSDExp}
SSDExp       ::= Value | oid<Value> | oid
Value        ::= atomic | ComplexValue
```

where `oid` ranges over a set of object identifiers, and `atomic` ranges over a set of constants.

(b) XML is a particular sort of semistructured data. Thus its quite easy to systematically convert ComplexValues into XML.

(c) Two main differences: (i) SSD expressions denote edge-labelled trees (this is actually given in the next part of the question) and XML values denote node-labelled trees. (ii) XML contains a schema description language.

(d) The main point of this question concerns the advantages that follow from ensuring that the SSD model is **deterministic**. By this we mean that given any node, the labels on the out arcs are **unique**. The grammar for deterministic SSDExpressions is pretty simple:

```
SSDExp       ::= atomic | ComplexValue

ComplexValue ::= {SSDExp:SSDExp,...,SSDExp:SSDExp}
```

(*e*) (*i*) `{0:"Doh",1:"Ray",2:"Me"}` (Some candidates may index from 1)

(*ii*) `{11:{},52:{},44:{}}`

(*iii*) `{10:3,13:2,42:1}`

(*f*) The key observation here is that as d-SSD is a deterministic model, the labels uniquely identify components. Thus a path, which is just a sequence of labels, matches a single point in the graph. Syntactically a path can be given as:

```
path ::= <SSDExp. ... . SSDExp>
```

Examples of valid paths are: `<0>` and `<0.{12:33,14:44}."Hello">`

(*f*) Given that a path denotes a unique node in the graph that can take the role of an oid in a SSD-expression. Hence we do not need to annotate the node and simply reference it using the path expression. We extend the grammar of `SSDExp` to:

```
SSDExp ::= atomic | ComplexValue | path
```