Fall 2005 Quals
Programming Languages

(open book , open notes)

Question 1 [65 pts]

We would like to invent a programming language that allows us to operate on XML documents (for simplicity, let's think of them as trees with labels attached to each node). Each program receives as input, at the beginning, one XML document and produces as output another XML document. The root of the input document is called INPUT. The root of the output document should be stored in the variable OUTPUT.
The language has the following syntax (terminals start with a lower-case letter, non-terminals start with an upper-case letter):

```
Program      ::=   program Command end
Command      ::=   Command ; Command
             |     Identifier := Expression
             |     Identifier.element := Expression
             |     for_each_child Identifier of Identifier do Command
             |     for_each_successor Identifier of Identifier do Command
IdentifierList    ::=   Identifier
             |     Identifier , IdentifierList
Expression   ::=   Identifier
             |     get_child ( Identifier , PosInteger )
             |     make_tree (Identifier, IdentifierList)
             |     Identifier.element
             |     " String "
```

Intuitively, the document is seen as a tree which has strings associated with each node; if the variable x contains a node of the tree, then we can access the string associated with such node by writing x.element, i.e., the string is the associated element. The commands include two types of assignment: the first stores a value in a variable, the other assumes that the variable contains a node, and it changes the element value of such node. We have two types of loops, both similar to the traditional for-loop. The first one assigns to the first Identifier the nodes which are the immediate children of the node stored in the second Identifier. The second for-loop instead explores the complete subtree. For example, if the node x has 3 immediate children (with elements "a", "b", "c"), and the subtree rooted in x contains the nodes with elements "a", "b", "c", "d", "e", "f", (e.g., as in Figure 1) then the loop

```
for_each_child A of x do command
```

will execute 'command' 3 times, the first time with A:=node with element "a", the second with A:=node with element "b", and the third with A:=node with element "c". The loop

```
for_each_successor A of x do command
```

will execute 'command' six times, with A:=node with element "a", A:=node with "b", A:=node with "c", A:=node with "d", A:=node with "e", and A:=node with "f". The nodes with "d", "e", and "f" are the children of one of the first three, or children of their children etc.

In the expression, the important cases to observe are the get_child(x,i) which returns the ith child of the node stored in x, and make_tree(x,y,z,..) which creates a new tree, having the node contained in x as root and the nodes y,z,... as immediate children. Note that, in the expression, Identifier denotes the name of a variable, while "String" represents a string constant, which is the element associated with the node.

Develop the semantic algebras and the valuation functions for this language. Make sure to clearly construct the semantic algebra to represent the XML documents (i.e., labeled trees).

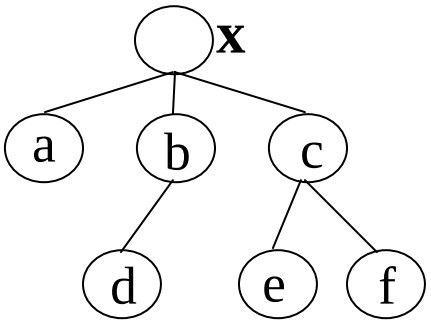

Figure 1. Sample Labeled Tree

Question 2 [35 pts]

Consider a standard imperative programming language, and let us assume that the language offers a case construct with the following syntax:

```
Command   ::=  case Variable of CASES endcase
CASES     ::=  case Number : Command
            |  case Number : Command CASES
```

The construct has the same meaning as the case-statement in Pascal: the value of the `Variable` is compared to the value of the `Number,` and the first matching case is executed; if a matching case is found, the successive cases are ignored.
Develop axiomatic semantics rules to describe the meaning of this construct. Provide one rule for the Command above and separate rules for the CASES.