MIT 6.887 FRAP – Problem Set 3

Welcome to our first problem set where we turn you lose formalizing a system, instead of giving you the Coq-encoded theorem statements to prove! In this problem set, we give you an informal (not defined in Coq) description of a language very similar to XPath[1], and you will formalize it as an interpreter. We also ask you to (1) prove a particular theorem about the language and (2) choose another interesting property and prove it.

Introduction: XML and XPath

- XML[2], which stands for eXtensible Markup Language, is a popular language to store and transport data. It is both readable and easy to machine-parse.
- XPath, which stands for XML Path Langauge, provides easy tranformation of XML structures with various syntactic features. A term in the part of the language we focus on is like a "path" to identify and navigate to nodes in an XML structure.

Our Language Syntax

You are asked to formalize two languages, fXML and fXPATH, which are similar to XML and XPath, respectively. Below you can see the informal syntax for our target languages.

fxml. The biggest differences between fXML and XML are: (1) fXML does not have "attributes" and (2) we only allow maximum two children (subnodes) for each node. These two simplifications were chosen to avoid complex encodings in Coq and to help you focus in on the core challenges of defining and reasoning about interpreters.

Described in terms of what *is* present rather than what *isn't*, here is the syntax of fXML:

$$\begin{array}{rrcl} \text{Names} & n & \in & \mathsf{Strings} \\ \text{Values} & v & \in & \mathsf{Strings} \\ \mathsf{XmlTree} & t & ::= & \langle\rangle \mid \langle \mathsf{n} \rangle\ v\ \langle / \mathsf{n} \rangle \mid \langle \mathsf{n} \rangle\ t\ t\ \langle / \mathsf{n} \rangle \end{array}$$

Note that, as in the several examples we've seen in class, it's your job to encode this convenient syntax with inductive types in Coq. On the informal level, $\langle\rangle$ is a dummy empty node. $\langle \mathsf{n} \rangle\ v\ \langle / \mathsf{n} \rangle$ is a node with tag name n and string content v. Lastly, $\langle \mathsf{n} \rangle\ t_1\ t_2\ \langle / \mathsf{n} \rangle$ represents a node with subnodes, where that node's tag is n and its two children are t_1 and t_2.

[1] `https://en.wikipedia.org/wiki/XPath`

[2] `https://en.wikipedia.org/wiki/XML`

fxpath. Based on the fXML syntax described above, your next task is to define a formal syntax for fXPATH. Industrial-strength XPath has many syntactic features and over 200 built-in functions. Here we'll stick to a tiny core language fXPATH.

In common with XPath, we describe the execution of a query in terms of a series of operations changing a *selection*. We begin with a single selected tree: the overall XML (or the simpler fXML, in our case) document. Each operation in the fXPATH program transforms the selection into a new one. In general, a selection can encompass any number of trees, from zero to several. Here are the available operations.

/*nodename*	Selects all *children* of currently selected nodes with the tag name *nodename*.
/[*value*]	Keeps only those selected nodes whose values equal the provided string *value*.
/..	Selects the *parents* of the currently selected nodes. If the original, root document is in the selection, then it is merely dropped by this operation, since it has no parent.
//	Selects all *descendants* of currently selected nodes, including the nodes themselves. That is, we get the original selected nodes, their children, their children's children, and so on.

A full fXPATH program consists of a sequence of operations from this vocabulary. To illustrate further, we introduce a simple XML document, which we hope does a plausible job of illustrating a more general format for databases of MIT subjects.

$$
\begin{array}{l}
X = \langle \mathsf{subjects} \rangle \\
\quad \langle \mathsf{subject} \rangle \\
\quad\quad \langle \mathsf{number} \rangle\ 6.009\ \langle / \mathsf{number} \rangle \\
\quad\quad \langle \mathsf{name} \rangle\ \mathsf{Fundamentals\ of\ Programming}\ \langle / \mathsf{name} \rangle \\
\quad \langle / \mathsf{subject} \rangle \\
\quad \langle \mathsf{subject} \rangle \\
\quad\quad \langle \mathsf{number} \rangle\ 6.887\ \langle / \mathsf{number} \rangle \\
\quad\quad \langle \mathsf{name} \rangle\ \mathsf{Formal\ Reasoning\ About\ Programs}\ \langle / \mathsf{name} \rangle \\
\quad \langle / \mathsf{subject} \rangle \\
\langle / \mathsf{subjects} \rangle
\end{array}
$$

Here is an fXPATH program to *find all subject numbers in the database*. Like the other example programs we will give, it also works correctly on other fXML documents standing for different sets of subjects.

$$P_1 = /\mathsf{subject}/\mathsf{number}$$

The final selection for our example document is the set

$$\{\langle \mathsf{number} \rangle\ 6.009\ \langle / \mathsf{number} \rangle, \langle \mathsf{number} \rangle\ 6.887\ \langle / \mathsf{number} \rangle\}.$$

We can also imagine a more chaotic document-encoding convention, where subject nodes might appear at arbitrary levels of nesting. Simply switching in the "descendants of" operator would let us still return all subject numbers.

$$P_1' = ///\mathsf{subject}/\mathsf{number}$$

Next, here is an fXPATH program to find the name of the subject with number 6.887.

$$P_2 = /\mathsf{subject}/\mathsf{number}/[6.887]/../\mathsf{name}$$

The final selection for our example document is the set

$$\{\langle \mathsf{name}\rangle\ \mathsf{Formal\ Reasoning\ About\ Programs}\ \langle /\mathsf{name}\rangle\}.$$

Note that we aren't following standard XPath syntax, so don't get hung up consulting XPath documentation!

Beside encoding the syntax of fXML and fXPATH in Coq, the core of your task is to implement a *denotation function* $[\![\cdot]\!]$, in some ways similar to the one we saw in Lecture 4. You should define $[\![\cdot]\!]$ such that, for any program P, $[\![P]\!]$ is a function from trees to sets of trees. That is, given an overall document, the interpreter computes the selection resulting from running P.

That's a nice, simple way to think about the behavior that an XPath programmer expects to see. However, just as we often need to strengthen an induction hypothesis to complete a proof, you will probably want to define a variety of helper functions to use in your interpreter. In fact, the final interpreter function itself probably won't even be recursive. The reason is that effective interpretation will depend on maintaining more information, especially to implement that parent-of operation "..".

A theorem to prove

With your interpreter-based semantics defined, it is time to prove some theorems. We came up with one particular theorem that we require everyone to prove. It looks nice and straightforward with the LaTeX syntax we're using here, though of course it may be a bit more involved in your formalization.

THEOREM 1. $\forall n_1, n_2, t.\ [\![/n_1/n_2/..]\!](t) \subseteq [\![/n_1]\!](t)$

That is, when we start from a node, proceed to n_1 children, then to n_2 children, then back up to their parents, we don't generate any extra results beyond what we already had after the first step.

Your interesting property

In addition to the proof for the above theorem, we ask you to state and prove your own interesting property about fXML and fXPATH. We'll accept anything that seems "interesting"; ask the staff if you have doubts on whether your property is interesting enough. It probably isn't interesting enough if it's proved with a single line of proof script; likely some induction should be involved. It's also nice if a user of XPath would read your theorem and say, "yes, it really is important that this property holds."

If no ideas come to mind, here's a general suggestion. How about defining an encoding from some interesting Coq data type into fXML, along the lines of how we embedded a database of subjects in our earlier examples? Then important operations can be defined both on the original datatype and as fXPATH programs to run on the generated fXML documents. It should be possible to prove that both methods give the same answer.