

## Practical

This practical is to be done individually – it is not a group effort. The practical is to be done in stages, using the tools specified and each stage will be assessed. The purpose of the practical is to gain experience in using classic language translation tools as well as getting some basic familiarity with XML and DTDs.

### Introduction

The practical involves writing a translator for an XML specification of a state machine into runnable Java code. As mentioned in the lectures, this is an example of translating one high level language into another – and as a result is potentially difficult.

Using XML mark-up as the input language has the benefit of being relatively easy to parse with the consequent rapid development. XML, however, presents challenges for lexical analysis – either lexical analysis is almost redundant (but then syntactic analysis is more complicated) or you have to think carefully about what constitutes a token. You are required to implement a lexer and you are strongly urged to follow the hint in Stage 1.

### The Language

If this was a *compiler* construction course, we would give you a formal definition of the grammar of the source language and a formal specification of the semantics of that language. Indeed, the lecture course explains how to transform such descriptions into runnable code!

For this practical, however, we are not going to present the problem in this way. XML documents have an accompanying DTD (Document Type Definition) which specifies this. We provide a DTD and an example specification of a state machine that represents a light bulb and a switch.

As a consequence of the way we are presenting this, you will need to first figure out the grammar! It turns out this is pretty simple, and DTDs can be transformed into BNF grammars in a reasonably straight forward way.

Note well: we do *not* intend you to write a generic XML parser that is driven by any given DTD (although this is possible). We are asking you to write a parser for *this particular* XML/DTD specification.

### Stage 0 (Understand)

Read through the DTD/XML example. You need to have a clear idea of what is going to happen.

### Stage 1 (not assessed by us)

Transform the DTD into a suitable grammar. Be mindful of how you are going to parse it. One big hint/tip: “<FOO” should be treated as a single lexical element. Don't be tempted to try to do it the more general way: “<” followed by an identifier.

### Stage 2 (Lexical Analysis)

Build a lexer for the grammar using Jlex. We will ask you to provide a driver for this that does *precisely* the following:

reads a line of input

prints out a line of output for each token found: <start-char, end-char, characters-matched, token> (note that the brackets and the commas *are* required). This output to standard output. NO OTHER OUTPUT.

### Stage 3 (Parsing)

Build a parser for your language. For each sub-parser you must have a print statement at the start and the end of the sub-parser printing the name of the sub-parser and the current lexical token. This print statement must be under the control of an if statement testing a boolean constant called `debuggingParser` which should be set to true.

### Stage 4 (AST Generation)

Generate an AST. You must provide one treewalker for your AST which *pretty prints* the tree. This should re-produce the input XML (almost) exactly. You must indent each subtree exactly two spaces.

### Stage 5 (Translation)

Details in a following handout.