**Topic : XLS Parsing Task**

**Author: Sharang Kulkarni. V-sharangk@usafacts.org**

**Point To Prove =>**

**extracting tables from XLS files with headers, column headers, table data, and footers is not solvable using traditional programming approaches and requires machine learning techniques.**

**Argument =>**

In automata theory, the classes of formal languages are organized into a hierarchy known as the Chomsky hierarchy. This hierarchy categorizes languages based on their complexity and the type of formal grammar or automaton required to recognize or generate them.

The problem of extracting tables from XLS files with varying structures and patterns can be viewed as a language recognition problem. If we consider the set of all possible XLS files as a formal language, we need to determine whether this language falls into a class of languages that can be recognized or generated by a finite automaton or a pushdown automaton (which corresponds to regular and context-free languages, respectively).

**Let me introduce you to the formal automata theory for reference =>**

**Deterministic Finite Automata (DFA)**:

**Definition:**

A Deterministic Finite Automaton (DFA) is a quintuple (*Q*,Σ,*δ*,*q*0​,*F*), where:

* *Q* is a finite set of states.
* Σ is a finite alphabet of input symbols.
* *δ*:*Q*×Σ→*Q* is the transition function.
* *q*0​ is the initial state.
* *F* is the set of accept states.

**Transition Function**:  
The transition function *δ* maps each state and input symbol pair to another state. Mathematically, it can be represented as:

*δ*:*Q*×Σ→*Q*

**State Diagram**: A DFA can be represented graphically using a state diagram. Each state is depicted as a node, and transitions between states are represented by labeled edges corresponding to input symbols. The initial state is denoted by an arrow pointing to it, and accept states are usually indicated by double circles.

**Example**:  
Consider a DFA recognizing strings over the alphabet{0,1} that contain an even number of 0s. The DFA can be defined as follows:

* *Q*={*q*0​,*q*1​}
* Σ={0,1}
* *q*0​ is the initial state.
* *F*={*q*0​}

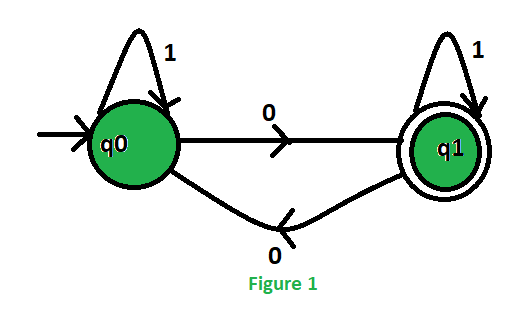
The transition function *δ* is defined as follows:

*δ*(*q*0​,0) = q1

*δ*(*q*0​,1) = q0

*δ*(*q*1​,0) = q0

*δ*(*q*1​,1)​=  *q*1​​



**Explanation**:

* If the current state is *q*0​ and the input symbol is 0, the DFA transitions to state *q*1​.
* If the current state is *q*1​ and the input symbol is 0, the DFA transitions back to state *q*0​.
* If the input symbol is 1, the DFA stays in the same state.

**Acceptance**:  
A string is accepted by the DFA if, after reading the entire input, it reaches an accept state. In the example DFA, a string containing an even number of 0s will lead to the DFA being in state *q*0​, which is an accept state.

**Regular Expressions**: A regular expression is a compact notation for specifying patterns in strings. It consists of symbols representing the alphabet, operators for concatenation (⋅⋅), union (++), and closure (∗∗), and parentheses for grouping.

**Equivalence with DFA**: One of the fundamental results in formal language theory is that regular languages are precisely those languages recognized by Deterministic Finite Automata (DFA). This equivalence is known as the **Regular Language Theorem**.

**Formal Statement**: Given a regular language *L* over an alphabet Σ, there exists a DFA *M* such that *L*(*M*)=*L*, where *L*(*M*) denotes the language accepted by DFA *M*.

Conversely, for any DFA *M*, there exists a regular language *L* such that *L*(*M*)=*L*.

**I am not delving in proof, Since its complex, but key takeaway is, regular languages are parsed through regular expression. But tables in xls are not part of regular languages.so task cannot be carried via use of regular expressions.**

Let’s further explore this topic with further advancing with other classes of languages

**Pushdown Automata (PDA)**:

**Definition**: A Pushdown Automaton (PDA) is a mathematical model of computation that extends the capabilities of a Deterministic Finite Automaton (DFA) by adding a stack. Formally, a PDA is defined as a 7-tuple *Q*,Σ,Γ,*δ*,*q*0​,*Z*0​,*F*), where:

* *Q* is a finite set of states.
* Σ is a finite input alphabet.
* Γ is a finite stack alphabet.
* Γ∗*δ*:*Q*×(Σ∪{*ϵ*})×Γ→2*Q*×Γ∗ is the transition function.
* *q*0​ is the initial state.
* *Z*0​ is the initial stack symbol.
* *F* is a set of accept states.

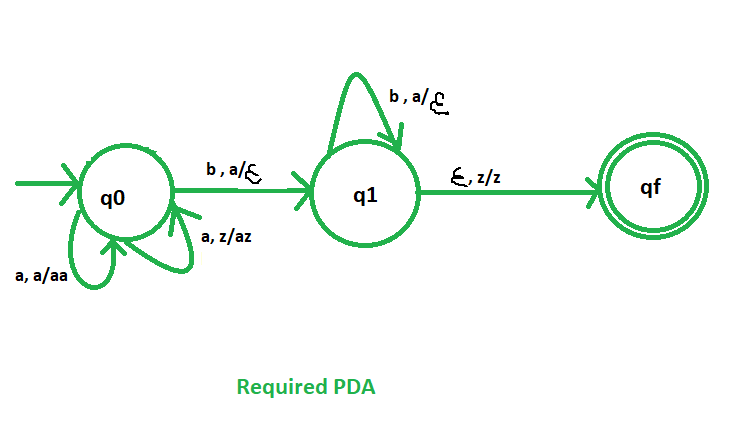
**Transition Function**: The transition function *δ* takes as input the current state, the current input symbol (or epsilon for empty input), and the symbol at the top of the stack. It returns a set of possible transitions, each consisting of a new state and a string of symbols to replace the top of the stack.

Consider a PDA recognizing the *L*={*anbn*:*n*≥0}, where each *a* is followed by the same number of *b*'s. The PDA operates as follows:

* The initial state pushes a special symbol *Z*0​ onto the stack.
* For each *a* encountered, it pushes *a* onto the stack.
* For each *b* encountered, it pops an *a* from the stack.
* If the stack is empty when the end of the input is reached, the PDA accepts.

**PDA i.e. push down automata can be simply thought as DFA with stack**

**I hope your are seeing, we are able to recognize, more complex language patterns, now we are able to “”count””**



**Context-Free Languages**: A Context-Free Language is a language that can be generated by a context-free grammar. It consists of strings of symbols that can be generated by applying production rules to non-terminal symbols. Context-Free Languages are more expressive than regular languages but less powerful than recursively enumerable languages and are recognized by PDA.

**Formal Statement**: Given a Context-Free Language *L*, there exists a Pushdown Automaton *P* such that *L*(*P*)=*L*, where *L*(*P*) denotes the language accepted by PDA *P*.

Conversely, for any Pushdown Automaton *P*, there exists a Context-Free Language *L* such that *L*(*P*)=*L*.

XLS tables cannot be described by context-free grammars, which are the formalism used to define context-free languages (CFLs). Context-free grammars operate based on rules that specify how symbols can be replaced by other symbols in a language, but they struggle to capture the intricate hierarchical structures and variable lengths commonly found in XLS tables.

XLS tables are not regular languages because regular languages are a subset of context-free languages, and XLS tables typically exhibit hierarchical or nested structures that cannot be captured by regular expressions or regular grammars. Let's explore why XLS tables do not fit neatly into the categories of regular or context-free languages:

**1. Nested Structures:**  
XLS tables often contain nested structures such as headers, subheaders, and nested tables. Regular languages cannot handle nested structures because they lack the ability to count or maintain state beyond a fixed number of transitions, which is insufficient to handle arbitrary nesting levels.

**2. Unbounded Stack Usage:**  
Context-free languages, which are recognized by pushdown automata (PDA), allow unbounded stack usage. However, the stack in a PDA is only capable of remembering information about the nesting level of symbols, not their hierarchical relationships. Therefore, PDAs are also inadequate for capturing the hierarchical structure of XLS tables.

**3. Variable Lengths:**  
XLS tables can have rows of variable lengths, and the number of columns can vary from row to row. Regular languages are incapable of handling variable-length strings or sequences, which are common in XLS tables.

**4. Semantic Constraints:**  
XLS tables often have semantic constraints that go beyond the capabilities of regular or context-free grammars. For example, a column might represent numerical data, and the cells within that column must conform to numeric formatting rules. Regular or context-free grammars cannot express such semantic constraints.

**5. Conditional Formatting:**  
XLS tables can incorporate conditional formatting, which alters the appearance of cells based on complex logical conditions. Regular expressions or context-free grammars are ill-suited for capturing the intricacies of conditional formatting rules.

**Conclusion:**  
In summary, XLS tables exhibit characteristics such as nested structures, unbounded stack usage, variable lengths, semantic constraints, and conditional formatting that exceed the expressive power of regular languages and context-free languages. As a result, more advanced techniques, possibly leveraging machine learning algorithms, are necessary for accurately extracting information from XLS tables.

**In simple terms, Not regular expression nor compiler could be written for recognizing xls**

Now Let’s move more powerful languages and computation model.

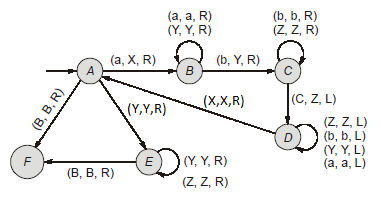
**Definition**: A Turing Machine (TM) is a theoretical model of computation introduced by Alan Turing in 1936. It consists of a tape of infinite length divided into cells, a read/write head that can move left or right along the tape, a finite set of states, and a transition function that specifies how the TM transitions between states based on the current symbol read and the current state.

**Formal Definition**: A Turing Machine is formally defined as a 7-tuple,Σ,Γ,*δ*,*q*0​,*q*accept​,*q*reject​), where:

* *Q* is a finite set of states.
* Σ is the input alphabet.
* Γ is the tape alphabet.
* *δ*:*Q*×Γ→*Q*×Γ×{*L*,*R*} is the transition function.
* *q*0​ is the initial state.
* *q*accept​ is the accept state.
* *q*reject​ is the reject state.

**Language Recognition**: A Turing Machine can recognize languages by accepting or rejecting strings based on its transition function. If the TM enters an accept state after processing a string, it accepts the string; if it enters a reject state, , it rejects the string. A language recognized by a TM is the set of all strings that the TM accepts.

**Mathematical Representation**: Let *L* be a language recognized by a Turing Machine *M*. Then, *L* is a recursively enumerable language if and only if there exists a TM *M* such that *L*(*M*)=*L*.



Sample turing machine

**Recursively Enumerable Languages**: The class of languages recognized by Turing Machines is known as recursively enumerable languages (RE languages). A language *L* is recursively enumerable if there exists a Turing Machine *M* that accepts all strings in *L* and either rejects or loops indefinitely on strings not in *L*.

**In simple terms, turing machine is equivalent to algorithm or program or a python script.**

**Now come coming to crux of all of it.**

**Turing Decidability**:

**Definition**: A problem is said to be Turing-decidable (or simply decidable) if there exists a Turing machine that can correctly determine whether any given input satisfies the property associated with the problem, and always halts in a finite number of steps for any input.

**Formal Definition**: For a problem to be Turing-decidable, there must exist a Turing machine *M* such that, given any input *x*, *M* halts in a finite number of steps and correctly decides whether *x* satisfies the property associated with the problem.

* Turing-decidability implies that a problem can be solved algorithmically, meaning that there exists a systematic procedure for determining whether an input satisfies the property associated with the problem.
* Decidability is a fundamental concept in theoretical computer science, providing a basis for reasoning about computability and complexity.

**In our case, problem is to validate the claim, if any algorithm can be made to parse a varied forms of xls =>**

**Decidability in XLS Table Extraction**:

**Definition**: Decidability in XLS table extraction refers to the existence of an algorithmic procedure that, given any XLS file as input, can determine whether the file contains a valid table structure that can be extracted, and always halts in a finite amount of time.

**Formal Framework**:

1. **Input**:
   * Let *I* denote the set of all possible inputs, where each input *i*∈*I* corresponds to an XLS file.
2. **Property**:
   * Define the property *P* that characterizes a valid table structure or a structure encompasses every other table within an XLS file.
   * A single script parses all XLS.
   * *P* could encompass syntactic constraints (e.g., presence of headers, consistent column alignment) and semantic constraints (e.g., data consistency, calculations).
3. **Decider**:
   * A decider is a Turing machine *M* that takes an input *i* and outputs whether *i* satisfies property *P*
   * Formally, the decider *M* is a function or algorithm or a script *M*:*I*→{Yes,No} such that:
     + If *i* satisfies *P*, then *M*(*i*)=Yes.
     + If *i* does not satisfy *P*, then *M*(*i*)=No.
     + *M* halts on all inputs in a finite amount of time.

**Final Verdict :**

**The problem has both Decidable Parts** and **Undecidable Aspects.**

1. **Parsing Algorithm**:
   * Deciding how to read and interpret the structure of XLS files can be approached algorithmically. We can design a systematic method, like a recipe, for going through each cell, row, and column to identify potential tables.
   * This involves defining rules for recognizing table-like structures, such as looking for consistent patterns in cell formatting, detecting headers, and identifying rows and columns of data.
2. **Syntactic Validity**:
   * We can create algorithms to check the syntactic validity of the table structure. This involves ensuring that the layout follows expected patterns, such as having headers in the first row, data in subsequent rows, and consistent column alignments.
   * By applying predefined rules and checks, we can determine whether the XLS file conforms to the expected syntactic structure of a table.
3. **Simple Data Consistency Checks**:
   * Basic checks for data consistency, such as ensuring that numerical columns contain only numbers or that dates are formatted correctly, can be implemented algorithmically.
   * These checks involve straightforward comparisons and validations that can be performed efficiently within a parsing algorithm.

**Undecidable Parts:**

1. **Semantic Validity**:
   * Determining whether the data within a table makes sense in a broader context can be highly complex and sometimes undecidable. For example, determining where one table starts and where it finishes, exact number of distinct tables on a document etc
   * Semantic validity often involves understanding the meaning and context of the data, which can be subjective and difficult to formalize algorithmically.

**So this problem is decidable, but it has undecidable parts!!**

**Suggestion to counter this problem.**

**Use ML classier to identify parts of tables.**

**OR**

**CFG grammar trick!!**

write a context-free grammar (CFG) to define the structure of a general table in XLS files. A CFG consists of a set of production rules that specify how strings in the language are formed from terminal and non-terminal symbols. These rules can capture the hierarchical structure of a table, including headers, rows, and columns. Here's a simplified example of how you might define such a grammar:

**Example CFG for General Table Structure**:

Table -> Header Row DataRows Footer

Header -> Row

Row -> Cell (',' Cell)\* '\n'

Cell -> String | Number | Date | Formula | Empty

DataRows -> Row+

Footer -> Row?

String -> "any sequence of characters"

Number -> "numeric value"

Date -> "date value"

Formula -> "arithmetic or logical expression"

Empty -> "empty cell"

You can built bottom up or top down parser with syntactic rules And write scripts using grammar parsing!!!