
# *A Syntactic Pattern Recognition Approach to Wave Analysis*

James D Bateman

*Cardiff University*

School of Computing and Informatics, Cardiff University, 2021

---

## ABSTRACT

Safety is of constant concern to boat pilots. While there are good mobile applications that can give an approximation of current weather conditions, and some that can track other local vessels, there are no systems which address the safety of current wave conditions.

This paper presents an application of syntactic pattern recognition to the analysis of ocean waves in the context of boat safety. The solution uses a boat's dimensions to build a parser that is capable of classifying a time series constructed from 1-dimensional, surface-height ocean wave data, as would be collected by a sensor on the prow of a boat, presented in the form of a sentence that is produced by a tokenizer. Two distinct models of parsing are described and implemented in order to contrast their distinguishing features in terms of precision and performance. Attribute Grammars are an augmented form of Context Free Grammars and were selected in the first model due to their powerful ability to pass values around an abstract syntax tree. Parsing Expression Grammars were selected in the second model and differ fundamentally from Context Free Grammars in that the choice of their operation is ordered. Methods of preprocessing wave data were discussed and implemented, along with methods of extracting peaks and troughs from wave data. A comparative study of these contrasting parsing methods is presented, and their performance is evaluated. The Parsing Expression Grammar was found to be a more efficient categorising method than the Attribute Grammar, but only by a matter of 2%. However, the Attribute grammar produces false positives at a rate of three times less than the Parsing Expression Grammar, making it more applicable to the task. Possible applications of the technology in a production capacity and other future work are also discussed.

---

# ACKNOWLEDGEMENTS

I would like to thank the following individuals, without whom I would not have been able to complete this dissertation.

Dr Paul Rosin, my supervisor, who helped to direct my research and implementation and whose expertise in image recognition provided many insights into the adjacent field of linguistic recognition.

Rosamund Woolverton (BPhil), my partner, whose support and interest has been fundamental to my ability to work on this project.

Glyndwr Morgan, my friend, who advised me on how to visualise data and lay out my thoughts.

My Mother, who has supported me always through my education.

My Father, who would have loved to read this.

# CONTENTS


| | | |
|---|---|---|
| **1** | **Introduction** | Page 3 |
| **2** | **Background Research** | Page 5 |
| | **2.1 Data Capture and PreProcessing** | Page 5 |
| | **2.1 Preprocessing** | Page 4 |
| | **2.2 Tokenizing, Grammars, and Parsing** | Page 6 |
| | **2.3 Precision and Performance** | Page 9 |
| **3** | **The Specification** | Page 11 |
| | **3.1 The Core Pipeline** | Page 11 |
| | **3.2 The Architecture - Collections and The Core** | Page 13 |
| | **3.3 Tokenizing, Grammars, and Parsing** | Page 15 |
| | **3.4 Preprocessors** | Page 18 |
| | **3.4 A Testbed** | Page 18 |
| **4** | **The Implementation** | Page 19 |
| | **4.1 Meeting Language Demands** | Page 19 |
| | **4.2 Implementing the Solution** | Page 20 |
| | **4.3 Issues Encountered and Overcome** | Page 20 |
| **5** | **Evaluation and Results** | Page 23 |
| | **5.1 Precision and Performance** | Page 23 |
| | **5.2 Comparing Methods** | Page 24 |
| | **5.3 Comparing PreProcessors** | Page 26 |
| | **5.4 Appraising The Solution** | Page 26 |
| **6** | **Future Work** | Page 28 |
| | **6.1 Furthuring the Solution** | Page 28 |
| | **6.2 Other Directions** | Page 29 |
| **7** | **Conclusions** | Page 30 |
| **8** | **Reflections** | Page 32 |
| **9** | **Glossary of Terms** | Page 34 |
| **10** | **Abbreviations** | Page 34 |
| **11** | **Appendix** | Page 35 |
| **12** | **References** | Page 41 |

# 1. INTRODUCTION

Whether large or small, the safety of a vessel is the concern of every pilot and captain alike. A vessel's safety is a function of many variables such as the wind and rain conditions, the current wave conditions, and even other vessels - concerns that have remained the same for centuries. These variables must be constantly monitored by a vessel's crew in order to maintain a safe course for the vessel and prevent the crew from coming into harm's way. Advances in computing have provided modern sailors with a myriad of tools to assist them in their task.

The internet has enabled pilots to effectively track other vessels using a location broadcasting system called AIS, short for Automatic Identification System (Wikipedia, 2016). Now, any AIS enabled ship is able to locate any other AIS enabled ship anywhere in the world. As a consequence, shipping lanes are safer for cargo ships and for smaller vessels that need to cross them.

Applications exist that will allow a sailor to track atmospheric wind and rain conditions and even access AI generated predictions of how the winds might be in the future - a major advancement over the predictive models available even 10 years ago. Modern navigators are able to plan ahead by factoring bad weather and wind conditions into their decisions, making even long crossings safe and routine.

However, there is no good, affordable system for vessels to track the wave conditions in their immediate vicinity. Modelling and buoy data do allow some good statistical estimates to be made about things like average wave height and some conclusions can be made from that data but there is no available system that tracks and analyses waves as they hit a vessel. This is desirable since average wave height does nothing to tell a vessel about obvious dangers like rogue waves (waves that are higher than would normally be expected) that might occur, or less obvious dangerous like seiche waves (rolling, sloshing action that happens in bays and harbors or other enclosed or partially enclosed bodies of water) or many medium-to-high waves occurring in a row.

This paper proposes a solution for this problem by means of applying syntactic pattern recognition to the waves that hit vessels, in order to identify trends and activity over a threshold in ocean-water surface-heights that could present a danger to the vessel. Syntactic pattern recognition is a form of pattern recognition in which each object can be represented by a set of symbolic features. This allows for "representing pattern structures, taking into account more complex interrelationships between attributes than is possible in the case of flat, numerical feature vectors of fixed dimensionality that are used in statistical classification and can be used instead of statistical pattern

recognition if there is clear structure in the patterns" (Wikipedia, 2016). Since the solution looks to find clear structures in water waves, this is an appropriate choice of paradigm for this pattern recognition purpose.

In order to apply syntactic pattern recognition to waves, those waves must be presented in such a way that their syntax can be evaluated. One way to present such structure is by means of a string of symbols from a formal language. Formal languages are well-established and there exist many ways to represent and parse them, making them a flexible means of describing data. Ocean water surface heights, as they would be experienced by a sensor single point on the prow of a boat, are used to produce a string of symbols which is cast into a sentence representing a surface height time series. The sentence is presented to the parser and a binary result is produced which classifies the time series data as either safe for a vessel to traverse or not.

This paper explores the application of syntactic pattern recognition by implementing variations of the sub-processes that make up the whole. The implementations of these identified algorithms will sit alongside one another in the final system, providing the opportunity to compare and contrast the differences between them and give an idea of their precision and performance by reference to a special type of contingency table. It is hoped that the solution presented in this paper will afford vessel crew an additional tool to complement their ability to protect the safety of those on board by supplying them with a means to determine if the local wave conditions are unsafe.

**The Aim:**
The aim of this project is to develop a solution that investigates the application of syntactic pattern recognition to the analysis of water waves in the context of boat safety. This investigation will cover all the areas associated with parsing a syntactic representation of a water wave including data capture, data preprocessing and transformation, tokenization of waves into sentences, parsing of syntactic representations, and will conclude with a comparative analysis of the selected methods.

# 2. BACKGROUND

## 2.1 DATA CAPTURE & PREPROCESSING

There are many ways of capturing ocean wave surface height data. Ocean buoys are large instruments which collect weather and ocean data from the world's oceans, providing a large array of data about the sea system in which they are situated. The data that ocean buoys provide is collected from the ocean at different frequencies, with some being capable of sub-second data collection (Wikipedia, 2016). Ocean buoys are a good source of wave data and could provide a good source of data for categorisation. Another means by which data could be collected is from a sensor on board a vessel. Many sensors are able to capture wave height, including accelerometer sensors (Bender, L. C., Guinasso, N. L., and Walpert J. N. (2009)). There are also many types of ocean waves. Categorising each individually would be a very large task. Instead, the solution should seek to categorise the underlying types of activity which make up dangerous waves.

Before we begin to tokenize water waves into symbols, we may wish to process the data in some fashion first. Real data is noisy, and noise could present an issue in classification. Particularly noisy data may create false positives or false negatives when classifying some data as belonging to a member of a set and, in the context of vessel safety, a false negative might endanger lives. When trying to remove such noise, it is vital that the process which is attempting to remove noise from data does not in the process also remove real data from the set. Accidental removal of important data that an algorithm thinks is noise represents the exact same problem as noisy data.

### 2.1.1 Median Filtering

Simply described, a median filter replaces every element in some data structure with the median of its neighbouring elements. Wikipedia (2020) defines a median filter as “non-linear digital filtering technique, often used to remove noise from an image or signal” and states that such noise reduction “is a typical pre-processing step to improve the results of later processing”. What is meant by this is that the median filter should be able to de-noise input data and it is commonly used for doing this type of activity. In their paper, Median Filter, Mi̇ček, J, and Kapitulík, J. (2020) note that the “Median filter is often used in case of rare impulse errors suppression superposed on a useful signal.“. They continue to make a relevant point about the Median Filter and its applicability in situations where ”Low-pass filters are not applicable because of picture edges blurring.”. In this context, edge blurring is analogous to loss of wave shape or definition and this is a relevant concern since a low or high pass filter might destroy data about waves which would be relevant for determining vessel safety.

### 2.1.2 Periodic Sampling

Periodic sampling of data is a means of preprocessing that acts to reduce the number of data points in the output set. This preprocessing method may yield improvements in performance but at the cost of accuracy since with fewer data points to tokenize and therefore fewer tokens to parse, a solution should spend less time executing but at the cost of not actively analysing all data points from the input set. When a signal is undersampled, a type of noise called aliasing occurs. This type of noise is readily seen in the spatial domain, for example small text on a computer screen having once been hard to read before anti-aliasing was known and commonly implemented. While the Nyquist-Shannon sampling theorem is applicable to spatial data (Miček, J. and Kapitulík J. (2003)), it may not be particularly useful for categorising wave data and this is yet to be seen.

### 2.1.3 Splines

Splines are a class of functions that are used for applications that require data to be interpolated and/or smoothed. Wikipedia (2016) defined splines as "smooth functions with which to fit data, and when used for interpolation, they do not have the oscillatory behavior that is characteristic of high-degree polynomial interpolation.". The method attempts to fit smooth curves to otherwise noisy data and may be able to denoise a sample without removing relevant data, making it an interesting noise-reducing preprocessing method.

### 2.1.4 Peak Detection

Prevailing knowledge and metrics about the size of water waves that present danger to vessels relates to the size of waves and to their periodicity (Tredup, S., 2021). Detection of peaks and troughs is therefore vital in order to produce tokens which represent the features of the water's surface that are important for determining vessel safety. There are several methods available for detecting peaks and troughs and one which works well in the context must be investigated.

## 2.2 TOKENIZING, GRAMMARS & PARSING

Any kind of wave analysis performs two distinct tasks. The first task is to recognise patterns in data and identify parameters associated with those patterns. The second task is to interpret the results of the first task. This division of tasks is useful to think about when considering wave analysis, since it means that we are able to conceptually de-couple these two parts from one another.

### 2.2.1 Tokenizing

The amplitudes of the water waves are required in order to classify individual peaks and troughs, and period indications in order to group sets of peaks and troughs together. The trough, the peak, and the straight line segment are identified as primitive patterns. This choice seems to be a natural one since the prevailing knowledge surrounding vessel safety uses peak size and trough size, and

their relationship with one another to determine whether or not something is safe for a vessel to traverse (Tredup, S., 2021).. The straight line segment is included since, while a straight line is always safe for a vessel to traverse, its relationship with the peaks and troughs that surround it are vital for determining whether or not a vessel can make safe passage.

### 2.2.2 Grammars and Parsing

A parser is a function which solves the problem of deciding if the contents of a string adhere to a set of constraints. In the field of syntax recognition we normally call this set of constraints a grammar and there are many forms that a grammar can take. In the context of applying the syntactic method to wave analysis, a grammar is a description of a set of waves that a vessel can handle.

### 2.2.3 Attribute Grammars and the CYK

A context-free grammar is a type of formal notation for expressing languages recursively. Such a grammar consists of one or more variables that represent classes of strings, with rules that describe how the strings in each are to be constructed (Hopcroft, J., Motwani, R., and Ullman, J. (1979)). Attribute Grammars are, at their core, a Context Free Grammar, albeit an augmented one, and this means that some of the technology used to parse a Context Free Grammar can be used for an attribute parser. The parsing method selected to check the adherence of a sentence to a grammar is a tabular method for parsing a Context-Free Grammar called the Cocke–Younger–Kasami Algorithm, or the CYK. Running the CYK algorithm (CYKA) involves the construction of a table in memory that allows the parser to determine which non-terminal tokens derive from which substrings of a sentence. Hopcroft, J., Motwani, R., and Ullman, J. (1979) note that "using Big O notation, the worst case running time of CYK is $O\ (n^3\ .\ |\ G|\ )$, where n is the length of the parsed string and $|\ G|$ is the size of the CNF grammar G. This makes it one of the most efficient parsing algorithms in terms of worst-case asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.". Given the CYK is known to have good performance characteristics, it should prove itself to be efficient in testing.

In their paper "Syntactic Pattern Recognition of the ECG", Trahanias, P. and Skordalakis E. (1990) outline an attempt made to apply syntactic pattern recognition to wave analysis. Their paper tackles the recognition of ECG wave patterns in the context of patient health, something which strongly parallels the problem outlined in this paper, and as such the research done in this paper continues their work and expands the applicability of their research.

The distinct feature of Attribute Grammar parsing is the notion that a grammar production may pass some computed attributes from one token to another across a syntax tree. Wikipedia (2020) states that Attribute Grammar is an "extension of context-free grammars in which each symbol has

an associated set of attributes that carry semantic information, and each production has a set of semantic rules associated with attribute computation.". Normally, no values are passed down the syntax tree during the production of a token in a Context Free Grammar and in essence this collection of values against a syntactic representation is what makes Attribute Grammars unique. Interestingly, this means that Attribute Grammars cross the boundary between syntactic and statistical representation and allow the application of methods and algorithms from the field of syntactic pattern recognition and statistical pattern recognition to the same structure - a very powerful advantage.

The CYK belongs to a class of parsers that produce and use a table ( a matrix ) in memory to perform a geometric-like analysis of a sentence. This method is well-known and understood and there exist several versions and performance-upgraded implementations. Trahanias, P. and Skordalakis E. (1990) utilised an Attribute Grammar to represent the component parts of the ECG wave in their paper and, since the aim is similar, the solution presented here selects the Attribute Grammar as one of the two parsing methods that will be explored.

### 2.2.4 Parsing Expression Grammars and Parser Combinators

Attribute Grammars consider their abstract syntax tree as they make productions and the CYK uses a geometric notion to parse the sentences produced by them. In complete juxtaposition to this is the Parsing Expression Grammar and the Parser Combinator.

Parsing Expression Grammars (PEGs) provide "an alternative, recognition-based formal foundation for describing machine oriented syntax, which solves the ambiguity problem by not introducing ambiguity in the first place. Where CFGs express nondeterministic choice between alternatives, PEGs instead use prioritized choice." (Ford, D(n. d.)).

This means that, instead of parsing a potentially large abstract syntax tree, the Parsing Expression Grammar is able to consider what it wants to consider first.

The method selected to process the Parsing Expression Grammar is the Parser Combinator - a regex-based method that relies on the idea of functional composition. The Parser Combinator builds large functions with complex functionality from many small modular functions. Parser combinators enable a recursive descent parsing strategy that facilitates modular piecewise construction and testing. This parsing technique is called combinatory parsing (Wikipedia 2021). This method is not so well studied and its application to the study of syntactic wave analysis is as yet unexplored. A fully-fledged parser combinator is a task outside the scope of this paper, and there exist full implementations of this technology in the field for programmers to make use of (Parsec (n. d.)). Since a fully-fledged implementation would introduce unknown levels of complexity, it would not make for a good comparison in terms of performance. For this reason, the scope of the implementation must be concise - representing only that functionality that a Parsing Expression Grammar requires. Many papers have been presented that outline the means by which to achieve a

functioning Parser Combinator (Hutton, G. and Meijer, E. (1996)), and the methods presented within should be considered in the specification.

### 2.2.5 Functional Programming Requirements

Parser combinators are a procedure that involves reducing many functions into a single function. Often, this means combining more than two functions at once. This is known as functional composition. In order to produce the effects of a parser combinator, the solution needs to implement a few basic functional programming primitives such as composeL, foldL, last, init, head, and tail. This functionality can be found in many libraries but functional libraries are often bound up with other functional programming concepts such as immutability, which add undesirable complexity to the program.

## 2.3 PRECISION & PERFORMANCE

Precision and performance are the two meta-metrics that this paper will consider when making comparisons between the methods of parsing and the preprocessors selected for investigation. It is important to know to a high degree how precise each method and preprocessor is in order that their ability to complete the task of categorizing data can be compared in terms of how well they are able to recognise classes of waves. A confusion matrix and the derivations thereof are selected as the means by which this paper will determine the level of precision that each method and preprocess has.

The Matthews Correlation Coefficient confusion matrix derivation is selected as the precision metric used for comparison. The Matthews Correlation Coefficient is a more powerful metric than the accuracy precision metric for the following reason: Consider a sample in which there are 10 ground truths, 1 of which should be classified false and 9 of which should be classified as true. A particular classifier may classify all the data as true. The overall accuracy would be 90%, but in fact the classifier has a 100% recognition rate for the true class - but a 0% recognition rate for the false class. The Matthews Correlation Coefficient is able to take into account false positives and negatives and is still an effective measure even in datasets where the balance of the true and false classes are of very different sizes (Wikipedia (2020)). However if the Matthews Correlation Coefficient returns either -1, 0, or 1, it is not considered a reliable indicator of how similar a categoriser is to a random selector (Wikipedia (2020)). In the case that the Matthews Correlation Coefficient returns -1, 0, or 1, the solution falls back onto the Accuracy confusion matrix derivation to determine accuracy.

Performance is to be evaluated using the metrics of total heap memory used in bytes, and time taken for the method or preprocessor to complete execution.

By investigating and implementing these algorithms, methods, and evaluation it is possible to present a comparative study of the methods available for the different subprocesses that make up the main process of applying syntactic pattern recognition to the problem. The differences between parsing methods are well-known, but their differences in the context of their application to wave analysis and to vessel safety have not been studied before.

### 2.3.1 Research Question:

In order to develop software that is able to investigate the application of syntactic pattern recognition to the analysis of water waves in the context of boat safety, this investigation will; identify how to express a vessel's dimensions, develop an algorithmic approach to producing parsers, define appropriate performance metrics, implement a set of syntactic parsers and preprocessors, and demonstrate the basis by which a judgment can be made as to whether or not this application has been a success.

# 3. THE SPECIFICATION

## 3.1 THE CORE PIPELINE

In the research question, software was outlined which would be able to investigate the application of syntactic pattern recognition to the analysis of water waves in the context of boat safety. The user of software should be able to give the program two pieces of input; some wave data and some vessel data, and receive a single piece of output; a description of the safety of the input waves from the perspective of the vessel.

### 3.1.1 The Data Structures - VesselData and WaveData

There is a natural distinction between Vessels and Waves and the properties and attributes associated with them. We would like a standard way to represent both Vessel and Wave data in such a way that all sub-processes, no matter their internal functionality, can make use of them without needing to provide tailored data to different sub-processes. In order to provide a standard schema for data we will define two data types which we name *VesselData* and *WaveData*. VesselData and WaveData are both immutable and represent Vessel and Wave data in an agnostic way for use by the rest of the program. The VesselData object serves as a data storage container for Vessel parameters. An instance of the VesselData type is an object with 5 properties defined in Figure (v) of the Appendix. The WaveData object serves as a data storage container for Wave data. An instance of a WaveData type is an object with 2 properties defined in Figure (vi) of the Appendix.

### 3.1.2 The Architecture - Functions

In order to address the research question, the proposed application must facilitate the full process of applying the syntactic method to the analysis of wave data in the context of vessel safety. Previously we have spoken about the division of the process into two distinct tasks. In order to achieve greater granularity, we can break down these two tasks into four distinct sub-tasks:

- **Recognise and Identify**
    - **Recognition:** The process of formally perceiving a datapoint.
    - **Identification:** The process of describing a datapoint or group of datapoints.
- **Contextualise and Interpret**
    - **Contextualisation:** The process of setting a stage for an interpretation to make sense.
    - **Interpretation:** The process of deriving meaning from an identification.

We can now apply these distinctions to the specific context of an application of the syntactic method to wave analysis.

### 3.1.3 Recognise and Identify

The notion of Recognition is analogous to processing WaveData into a form that can be identified, a process that might include such sub-processes as noise removal. In this implementation we adopt the name *PreProcessor* to identify a function that performs the task of processing WaveData into a recognisable form. We adopt the name *PreProcessedWaveData* to identify processed WaveData that has been returned by a PreProcessor.

Identification is analogous to tokenizing PreProcessedWaveData into a meaningful representation. Whether that representation consists just of a string literal, as would be the case with a ParserCombinator, or a string literal and accompanying attribute table, as would be the case with an AttributeParser, is of no consequence at this level of abstraction since the result of both of these examples is that the resultant is passed onto the next stage. We adopt the name *Tokenizer* to identify a function that performs the task of producing a meaningful representation from a set of PreProcessedWaveData, and the name *Sentence* to identify the meaningful representation that is produced.

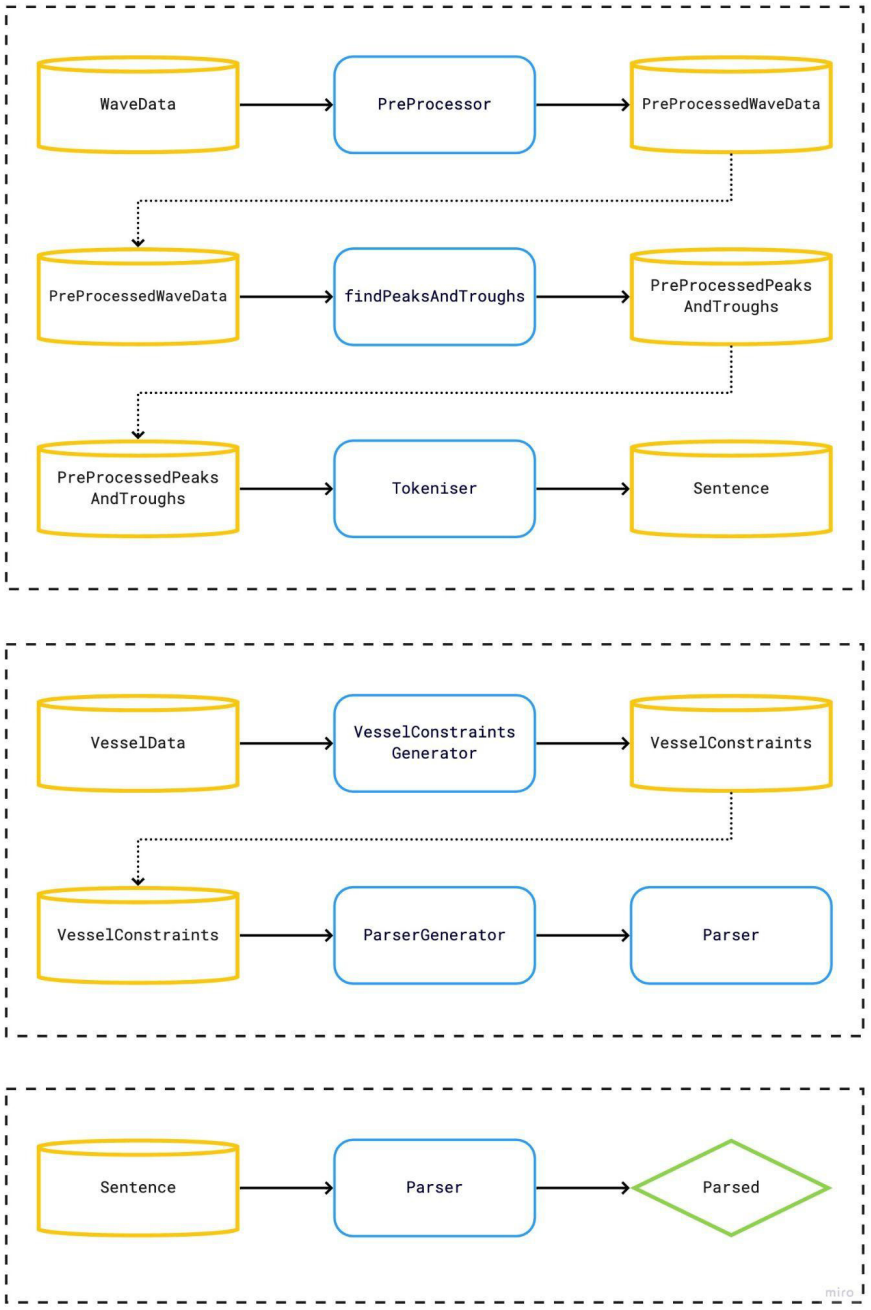
### 3.1.4 Contextualise and Interpret

Contextualisation is analogous to producing a Parser from some ParserGenerator and some context. If a ParserGenerator is to be useful to a specific Vessel it needs to receive as an argument a context representing a set of VesselData with which it can interpret its input. Since a set of VesselData will be used differently by a different Parser, it is necessary for a ParserGenerator to be able to generate its own specific configuration information during the constructuction of a Parser. We adopt the name *VesselConstraintsGenerator* to identify a function that performs the task of preparing a VesselData context for use in the generation of a Parser, and the name *VesselConstraints* to identify a VesselData context. We adopt the name *ParserGenerator* to identify a function that performs the task of producing a Parser from a set *VesselConstraints*. Interpretation is analogous to parsing a Sentence. We adopt the name *Parser* to identify a function that performs the task of parsing a Sentence to produce a result, and we adopt the name *Parsed* to identify the result of a parse.

It is entirely reasonable at this point to ask the question “Doesn’t making a distinction between Recognition and Identification, or between Contextualisation and Interpretation create a synthetic distinction that adds complexity for no benefit?”. This difference is not without benefit. A granular breakdown like this allows the four sub-tasks to be decoupled from one another. Decoupling the sub-tasks means that the internal functionality of one sub-task does not need to be known by another and they can be replaced independently. This means it must adequately accommodate the four sub-processes involved in the two tasks that make up the application. It must take in a set of sub-processes and run them, in order, producing a result. An adequate solution must provide error

checking for development, feature performance testing functionality for collecting performance results, and provide a solid framework for further work.

This whole process is visualised in the figure below.




## 3.2 The Architecture - Collections and The Core

The specification defines a ‘core’ unit that wraps each of the four sub-processes in functions that perform tests on them and runs them in order. These wrappers allow the main function of the core unit to refer agnostically to four predefined functions which makes the system resilient to problems with the algorithms during development and allows parsing errors to be passed back to the application in the same way each time during production. Ithas been previously discussed that any kind of wave analysis performs two distinct tasks - the first task being to recognise patterns and

identify parameters associated with those patterns, and the second task being to interpret the results of the first task. The specification will adopt the name *Collection* to describe a set { VesselConstraintGenerator, ParserGenerator, PreProcessor, findPeaksAndTroughs, Tokeniser }, where

```
VesselConstraintGenerator :: VesselData => VesselConstraints
ParserGenerator :: VesselConstraints => Parser
PreProcessor :: WaveData => PreProcessedWaveData
findPeaksAndTroughs :: PreProcessedWaveData => PreProcessedPeaksAndTroughs
Tokeniser :: PreProcessedWaveData => Sentence
```

And the ParserGenerator returns a value that conforms to

```
Parser :: Sentence => Parsed
```

These six functions are always performed the same regardless of the make-up of any of the individual sub-tasks that make up a Collection. Given that it is specified that the solution must investigate and test the differences between various methods of completing these sub-tasks, it must be possible to define and adopt an implementation that will allow the reuse a large central body of code whose function is to direct the operation of the underlying collection of sub-tasks in a repeatable, testable manner. First, consider the simplest possible implementation of a system that is able to direct the operation of a Collection. After identifying issues with such a system, it is possible to be able to further develop the implementation to fix these issues. The Core, in order to provide the functionality described above, need only implement a very simple pipeline. The Core could be defined

```
Core(version 1) :: (VesselData, WaveData, Collection) => Result
```

Where the Core proceeds as

```
Derive { VesselConstraintGenerator, ParserGenerator, PreProcessor,
findPeaksAndTroughs, Tokeniser } from Collection
Let VesselConstraints = VesselConstraintGenerator(VesselData)
Let Parser = ParserGenerator(VesselConstraints)
Let PreProcessedWaveData = PreProcessor(WaveData)
Let PreProcessedPeaksAndTroughs = findPeaksAndTroughs(PreProcessedWaveData)
Let Sentence = Tokenizer(PreProcessedWaveData)
Let Parsed = Parser(Sentence)
Return Parsed
```

This implementation works perfectly fine and this is the layout that was used for the prototyping. However, this implementation is not production ready for a number of reasons. Here, the solution is producing a new set of VesselConstraints, a new Parser, and a new set of PreProcessedWaveData every time we want to generate a result from a new set of WaveData. Since the Core has no notion of how long something might take, it is not desirable to spend time re-calculating sets of data and

functions that will be the same on the next run. In order to fix this minor issue, the solution defines a production version of the Core to be:

```
Core(version 2) :: (VesselData, Collection) => (WaveData) => Result
```

Where the Core proceeds as

```
Derive { VesselConstraintGenerator, ParserGenerator, PreProcessor, Tokeniser }
from Collection
Let VesselConstraints = VesselConstraintGenerator(VesselData)
Let Parser = ParserGenerator(VesselConstraints)
Define ReturnedFunction to be a function that takes (WaveData) as an argument and
proceeds as
    Let PreProcessedWaveData = PreProcessor(WaveData)
    Let Sentence = Tokenizer(PreProcessedWaveData)
    Let Parsed = Parser(Sentence)
    Return Parsed
Return ReturnedFunction
```

The second implementation of the Core fixes the first issue we identified in version 1. It is now possible define the Core to be a curried function that produces a new function that closes over the arguments we pass in. The values and functions that only need to be produced once are produced in the first step and a new function is returned which has access to those pre-computed values. The returned function no longer incurs the overheads of the first.

## 3.3 TOKENIZING, GRAMMARS & PARSERS

Due to their power in describing structural and statistical features , attribute grammars are selected and used in this paper as the model for the formulation of a pattern grammar for ECG's. Other reasons for this selection, which are common to any syntactic approach to pattern recognition, are the following: 1) an increase of parsing speed is obtained as the injection of attributes into symbols (nonterminals and terminals) reduces the grammatical complexity and 2) the technology of processing attribute grammars is fairly mature and many implementations of evaluators do exist"

### 3.3.1 An Attribute Grammar

The alphabet of symbols $W = \{ P_i , T_j , F_k \}$ is adopted for encoding the waveforms, where $P_i$ denotes a positive peak (one where the water level is above the median water level) whose name is $i$, $T$ denotes a negative peak (one where the water level is below the median water level) whose name is $j$, and $F$ denotes a flat section of water whose name is $k$. A set of attributes is assigned to each peak $P$ and each trough $T$. This set is defined as $\{ mag \}$ where:

- *(mag) is the magnitude of the peak/trough (the y distance of the highest/lowest point from the median water level)*

In this grammar, the concept of magnitude refers to the numerical value of the magnitude of a wave's amplitude.

In water wave recognition for vessel safety, groups of peaks and troughs also carry information. For example, a vessel encountering many high-medium waves within a short time is known to be dangerous. Another example being that a single wave whose amplitude is far greater than the expected normal amplitude based on current conditions is dangerous. Therefore, peaks, troughs and flat sections are grouped together into larger units. The alphabet of symbols $G = \{ \pi_l , \varphi_m , \delta_n , \alpha_o , . \text{ (period) } \}$ is adopted for encoding groups of waveforms, where $\pi$ denotes a group of many high-medium magnitude waves close to one another with little to no flat space between them whose name is $l$, $\varphi$ denotes a single wave whose amplitude is far greater than the expected normal amplitude based on current conditions whose name is $m$, $\delta$ denotes a group of consecutive waveforms whose amplitudes increases by some amount each time whose name is $n$, $\alpha$ denotes any group of waves which do not belong to the classes of groups described above whose name is $o$, and the period token denotes the end of a sequence. This token is not strictly necessary since the end of a group could be inferred by finding a token representing the start of a new group or the end of a sentence, however it is included both as a visual aid (since the eye naturally demarks sentences that end with a period) and as a computational aid since a simple (`String.split('.')`) call can now split a sentence into groups). A set of attributes is assigned to each group $\pi$, $\varphi$, $\delta$, or $\alpha$ This set is defined as $\{ max \}$ where:

- *(max) is the largest magnitude of the peak/trough in the class*

For example, there may be a sentence such as

- $\pi_0 \; p_0 \; t_0 \; p_1 \; t_1 \; . \; a_0 \; p_2 \; f_0 \; t_2 \; f_1 \; p_3 \; f_2 \; . \; \varphi_0 \; p_4 \; . \; a_1 \; p_5 \; f_3 \; t_3 \; f_4 \; . \; \delta_0 \; p_6 \; t_4 \; p_7 \; t_5 \; p_8 \; t_6 \; .$

Which, described by the above criteria, should be read as describing a series of five groups of waveforms; $\{ \pi_0 , \alpha_0 , \varphi_0 , \alpha_1 , \delta_0 \}$, which represent the groups of waveforms $\{ p_0 \; t_0 \; p_1 \; t_1 \}$ , $\{ p_2 \; f_0 \; t_2 \; f_1 \; p_3 \; f_2 \}$ , $\{ p_4 \}$ , $\{ p_5 \; f_3 \; t_3 \; f_4 \}$ , and $\{ p_6 \; t_4 \; p_7 \; t_5 \; p_8 \; t_6 \}$ respectively. Each waveform has its name in subscript, and each waveform's name is an incrementing integer - with one set of incrementing names per waveform type. In practise, the names of each waveform can be inferred by the Parser since they are consecutive integers which relate directly to the number of occurrences of that token. For example there are 9 occurrences of the P token in the above sentence. In order to access the attributes of the 9th token in the attribute table, one can simply access the 8th ( 9 - 1) P attribute on the table. Therefore we can produce a simpler sentence such as

- $\pi \; p \; t \; p \; t \; . \; \alpha \; p \; f \; t \; f \; p \; f \; . \; \varphi \; p \; . \; \alpha \; p \; f \; t \; f \; p \; f \; . \; \delta \; p \; t \; p \; t$

And infer the attribute names from their position from the start (the left) of the sentence. Parsing this sentence is now reduced to the problem of determining if any of the groups or subgroups are unacceptable for the vessel in question. The full grammar can be seen in Figure (iii) of the Appendix.

### 3.3.2 A Parsing Expression Grammar

The alphabet of symbols `A = { t5, t4, t3, t2, t1, f, p1, p2, p3, p4, p5 }` is adopted for encoding the waveforms, where each token `t`*x* denotes a negative peak of magnitude *x*, each token `p`*x* denotes a negative peak of magnitude *x*, and each token `f` denotes a period of flat water. In this grammar, magnitude is a dynamic notion that is calculated from a vessel's data. The full grammar can be seen in Figure (i) of the Appendix. Compared to the description required of the Attribute Grammar, a Parsing Expression Grammar can be described very simply. It should be noted that the simplicity of the grammar is associated with a trade-off with the complexity of the Parser Combinator required to parse this grammar.

### 3.3.3 An Augmented CYKA for an Attribute Grammar

The CYKA has been selected for parsing the Context Free Grammar that makes up the Attribute Grammar. Its implementation should follow the well-known pseudocode (reference?). Implementing a basic, un-optimised version of the CYKA will allow the fundamental characteristics of the CYKA to be investigated.

### 3.3.4 A Parser Combinator for a Parsing Expression Grammar

In order to accomplish the task of parsing a Parsing Expression Grammar, the solution must implement a Parser Combinator capable of completing five atomic Parsers which are then composed into larger, more useful functions by a Parser Generator.

**ParseChar**

ParseChar must be capable of taking as input a character and returning a new function that is capable of taking as input a representation of a string and determining if the first character of the input string matches the input character. This can be shown as: *`ParseCharλ :: Char => ( ParseChar :: String => Success )`* `assuming that the String was a match`

**RejectChar**

RejectChar must be capable of taking as input a character and returning a new function that is capable of taking as input a representation of a string and determining if the first character of the input string matches the input character, and returning false if the match is found. This can be shown as: *`RejectCharλ :: Char => ( RejectChar :: String => Failure )`* `assuming that the String was a match`

**ParseMany**

ParseMany is a meta-parser. It must be capable of taking as input a number of parsers and returning a new function capable of taking as input a representation of a string and determining if the first n characters of the input string match the input parsers in exactly the order the input parsers were handed to ParseMany. This can be shown as: *`ParseManyλ :: [Parsers] => ( ParseMany :: String => Success )`* `assuming that the String was a match`

**RejectMany**

RejectMany is also a meta-parser. It must be capable of taking as input a number of parsers and returning a new function capable of taking as input a representation of a string and determining if the first n characters of the input string match the input parsers in exactly the order the input parsers were handed to ParseMany, and returning false if the match is found.. This can be shown as: *ParseManyλ :: [Parsers] => ( ParseMany :: String => Failure )* `assuming that the String was a match`

**ParseFrom**

ParseFrom is another meta-parser. It must be capable of taking as input a number of parsers and returning a new function capable of taking as input a representation of a string and determining if the first n characters of the input string match any of the input parsers without no regard given to which is accepted. This can be shown as:*ParseFromλ :: [Parsers] => ( ParseFrom :: String => Success )* `assuming that the String was a match`

## 3.4 PREPROCESSORS

Several preprocessors have been selected in order to determine if and how their application to wave data before the tokenization of that wave data into a sentence affects the accuracy of a parser's ability to correctly classify patterns. They must be implemented in such a way that they can be easily ablated from the rest of the process so that the usefulness of their application to wave data can be studied

### 3.4.1 Finding Peaks and Troughs

Separate from the other identified preprocessing methods is the process of finding peaks and troughs in data. Identification of peaks and troughs in the data must occur for tokenization such that the magnitude of a wave can be subject to pattern recognition. There are several means of achieving this that are well described on the internet [stack overflow + other one identical], as well as a well-regarded library that is capable of doing this.

## 3.5 A TESTBED

A separate application from the solution must be implemented in which the distinctives like precision and performance can be measured for each of the parsing methods and preprocessors. Capturing this data is key to evaluating the solution and should present the user with a full set of computed data which can then be interpreted to provide insight into the differences between the methods and preprocessors and help to select the best among them. Key to the evaluation is the collection of true positives, true negatives, false positives, and false negatives. Using this data, it is possible to derive metrics which are more useful than a simple list of how many parses resulted in success.

# 4. THE IMPLEMENTATION

## 4.1 MEETING LANGUAGE DEMANDS

Not all languages meet the requirements that the specification makes on the solution. Almost all patterns can be represented in any language, and most of the functionality required by could be performed in any language. However there are a few considerations of the solution as outlined in the specification that require a language that has a certain set of features.

### 4.1.1 A Language with Closures

A Closure is a technique for lexical scoping of names in a language with first-class functions. What this means is that a language that supports closures is able to bind functionality in functions that are then passed back from a lower level of scope up into a higher level of scope. In short, closures allow a function to be programmed by some other part of a program. This is a requisite for building parser combinators, as the key functionality of a parser combinator requires that a new function be returned from a configuration function that closes over the arguments passed into the configuration function.

### 4.1.2 A Language with "Functions as First-Class Citizens"

A prerequisite for building closures is that a language recognises functions as first-class citizens. That is to say that functions must be atomic values, capable of being passed as arguments to other functions, being returned from them as values from other functions, and being assigned to variables or stored in data structures.

### 4.1.3 Choosing a Language

Given the prerequisites, Javascript provides a perfect base language to implement the feature set described in the specification. Javascript is a multi-paradigm, single-threaded language. The multi-paradigm nature of Javascript allows a programmer to implement features in whatever paradigm they choose, and even to mix paradigms. Javascript has functions as first-class citizens, and supports closures as part of its syntax. This means that an implementation can take advantage of different paradigms for different tasks. In the context of the solution, this will allow both the iterative algorithms and functional composition to be programmed in the same language.

## 4.2 IMPLEMENTING THE SOLUTION

### 4.2.1 Parser Combinators by means of Functional Composition

In the implementation of the solution outlined in the specification, regular expression matching was chosen as the means by which a ParseChar parser should recognise the character representation provided to it. This allows the ParseChar function to make use of Javascript's inbuilt regex library. A regex is supplied to ParseChar, which is closed over by a returned function that takes a statechain The RejectChar parser is built into the same function as the ParseChar parser, with this function being named parseRegexλ. The lower-case lambda (λ) is used in the implementation whenever a curried function is declared. This way, all calling scopes that use this function should be aware that the function is curried. Implementing the Parser Combinator resulted in by far the highest time investment among all of the tasks that made up completing the solution. Lots of experimentation was necessary to accommodate the error-handling and composition of the modular functions that make up the Parser Combinator. Eventually, it was determined that using different types of data for different classes of error allowed each small function to pass on or handle the input it was receiving if that input was an error.

### 4.2.2 Other Libraries

In order to fit a spline to a noise piece of data, the solution takes advantage of a spline fitting library that is freely available on Github. Of the ones tested, this was the fastest.

## 4.3 ISSUES ENCOUNTERED & OVERCOME

### 4.3.1 Chomsky Normal Form

The chomsky normal form grammar for the grammar described in the Specification (Appendix Chart (i)) can be seen in Chart (ii) of the Appendix and the chomsky normal form grammar for the grammar described in the Specification (Appendix Chart (iii))) can be seen in Chart (ii) of the Appendix. Implementing the CYK algorithm is notoriously difficult (medium.com (2021)). Doing so took much longer than expected. It was expected that the implementation would take around a day, but it required 3 or 4 day to be completed.

### 4.3.2 Data Availability

While producing the proposal for this project research was conducted into the data that is able to be collected from ocean buoys and was able to find that many buoys are capable of producing high precision surface height data (high precision being in the 1hz range). However upon attempting to retrieve data from buoys it was found that, in practise, no bouy ever really stores data at this frequency for extended periods. The sample 1Hz time series that had been seen while researching

are only examples. Instead, most buoys produce data at a frequency of 15 seconds up to 90 minutes or even longer. This data is many orders of magnitude away from being directly useful for my purposes. This data may be useful in a supplementary fashion but is likely not precise enough to use as input for a wave parser.

In order to parse waves, a solution must be able to analyse wave data. Without a suitable source of data from buoys, it was required that an investigation take place in search of other sources of data for input. There are several options available here.

Synthesis - The first option is data synthesis. There are several recognised algorithms available for generating ocean waves, perhaps the most implemented being the Tessendorf Algorithm (Tessendorf, J. (1999)). Great work has been done in the last few years using this algorithm to generate ocean waves. We see a great implementation of the algorithm available as a github repository (github.com (n. d.)). It might be possible using this algorithm to derive an ocean-surface-height-over-time graph by analysing the output data. The benefits of using this scheme of data synthesis is that the data is pseudo-random and controllable. The downsides are time taken to alter the code in the repo.

Handcrafting - Another means of data synthesis is to hand-craft waves for the parser. The benefits of this scheme are that data is exactly as the program requires it, it is fast, and specific data sets can be crafted. The downsides are that the data is completely synthetic.

Generation - Another option is data generation. This option was ultimately the option that yielded the best results and since it is real wave data, purpose-generated for this application, could be collected in a way that would be easy to present to the solution.

### 4.3.3 A Wave Data Mobile Device Capture Solution

In order to generate data for the solution, a separate piece of software was designed and implemented. This application takes the form of a NodeJS web server that allows connections from two WebSockets, one from a mobile device equipped with an accelerometer such as a mobile phone, and another from a device with a web-browser such as a laptop computer. The application, on boot, waits first for a WebSocket connection from a mobile device that is enabled with an accelerometer. When it receives that connection it then waits for a WebSocket connection from a web-browser. Once both are connected at the same time, the application, using front-end Javascript, takes live accelerometer data from the mobile device that has an accelerometer and broadcasts it across the network to the server which broadcasts that data to the web-browser for live viewing of the data, and collates that data in order to save that data to file when the application is quit.

This piece of software, while only numbering a few hundred lines of code including the server and clients, is fairly complex due to how it has to juggle WebSockets and took a few days to create and tune.

### 4.3.4 Acceleration data mimics wave height data well

Using a mobile device's accelerometers it is possible to generate data from the surface of the water by placing the mobile device on top of  a boat's sea compass (negating roll and pitch). Parsing surface data and parsing accelerometer data are different. Bender, L. C., Guinasso, N. L., and Walpert J. N. (2009) discuss how close an approximation of wave surface height can be achieved by looking at accelerometer data. There is around a 5% miss-estimation from using this technique, however; since wave data was classified after the collection of the data - the actual data that the collected data represents is irrelevant if the classification of the waves occurs after collection. That is to say, the experiment requires something that looks like wave data and is somewhat noisy as real data often is. By generating wave data and then classifying it, the experiment is provided with good ground truth.

### 4.3.5 Limitations

In practise, the stack and heap both present issues when using a Parser Combinator and a CYKA. Since a Parser Combinator is a recursive function, it is possible to reach the maximum call stack size of the V8 Javascript implementation rather quickly. One of the limitations of using a higher-level language is that in order to gain such things as garbage collection and functions as first-class-variables, it is necessary to trade off memory efficiency. Since the CYKA produces a large table in memory, it is possible to crash the heap when attempting to read sentences of length greater than around 2700 characters. This is a significant amount of data representing around an hour of wave data - far more than would be required for the average parse.

### 4.3.6 Isomorphism

Javascript is isomorphic in the context of web application design. This means that the exact same language is used for the front-end code as is used for the back-end code. Not only does this save development time in terms of the mental cost incurred by having to switch context between languages while developing networked applications, but it means that serialised data structures and even code can be shared between the front-end and back end at once during runtime. When encountering issues with data availability, it was relatively straightforward to quickly prototype a wave capture server using Javascript since I didn't have to switch context to another language in order to accomplish a side-task.

# 5. EVALUATION AND RESULTS

## 5.1 PRECISION & PERFORMANCE

When designing the solution, it was important that it be possible to sensibly demonstrate the precision and performance of the selected methods and sub-processes that make up the solution. In order to test the precision and performance of the methods chosen, the solution performs two types of analysis. First, the solution is able to determine how precise the methods are by generating a confusion matrix and derivations thereof. This allows the solution to make a determination about how efficacious the methods are. Second, the solution is able to determine how efficient the methods are by collecting their memory usage and time taken to parse a sentence. By making reference to the statistics that a method exhibits it is possible to make a distinction between methods that have a similar level of precision. For example, it may be that one method of parsing or preprocessing is just as efficacious as another method but that there exists a notable difference between the two methods in terms of their speed or the amount of memory they consume. If only a few degrees of accuracy are lost but one method proves much faster or consumes much less memory than another method, then its applicability for a use case is certainly higher. Conversely, even if a method is faster or uses less memory than another then its applicability is certainly lower.

### 5.1.1 Ground Truth and Testing By-Proxy

In order to test the effectiveness of the parsing methods investigated, the solution tests each parse by proxy. That is, in order to test a method, a solution categorises each parsing attempt by whether or not the parse successfully identifies whether or not a vessel should be able to make safe passage across the waves in the time series represented by the sentence that was parsed. Given that it is not possible to have or to create perfect ground-truth against which an evaluation of precision can be made, it was necessary to generate as good an approximation as could be made of ground truth in order to determine the efficacy of the selected methods. In order to do this a sample of 200 data sets of varying lengths were interpreted by myself and, using human perception and specialised knowledge of wave behaviour in the context of vessel safety, were classified as either representing a danger or not to three boats of varying size. This classified data was then considered to be ground truth against which more concrete analysis could be conducted. Using these 200 classified data sets, 5 preprocessors (of which 1 was a control which performs no operation), 20 sensitivity settings and 3 vessels, 60000 tests were subject to each parsing method. Tuning data was also used in the construction of the parsers. This data was hand-generated and fit-to-purpose with the intention that the parsers be able to categorise them, and as such does not form part of the 200 data sets which were used to test the parsers.

### 5.1.2 Precision - Confusion Matrices, Accuracy and the Matthews Correlation Coefficient

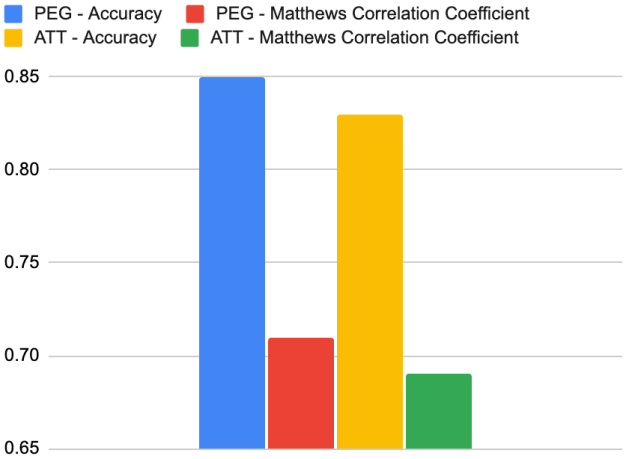
Confusion matrices were generated by the solution for each of the methods, for each of the preprocessors, and for each of the sensitivity settings separately. Collecting separate sets of confusion matrices allows the solution to present a comprehensive comparison of the results of using each method, of applying each preprocessor, and of providing the system with a different sensitivity for classifying peaks and troughs.

### 5.1.3 Performance - Heap usage and Time taken

Measuring the time taken and heap memory used for a method or sub-process to complete execution is undertaken by the solution in order to gather statistical data relating to each method and sub-process. By doing this, is it possible to ascertain the viability of the methods investigated. This is an important metric for any computing task as it can give an indication of overall usefulness - rather than just precision. If two algorithms are able to perform the same task to the same degree then it is possible to indicate which is better by reference to how constrained one or the other is in terms of how much memory the algorithm requires or how long it takes to complete that task. In this way we are able to combine precision and performance to compare the selected methods and preprocessors

## 5.2 COMPARING METHODS

Figure 1. Accuracy and Matthews Correlation Coefficient for the Parsing Expression Grammar (PEG) and Attribute Grammar (ATT)

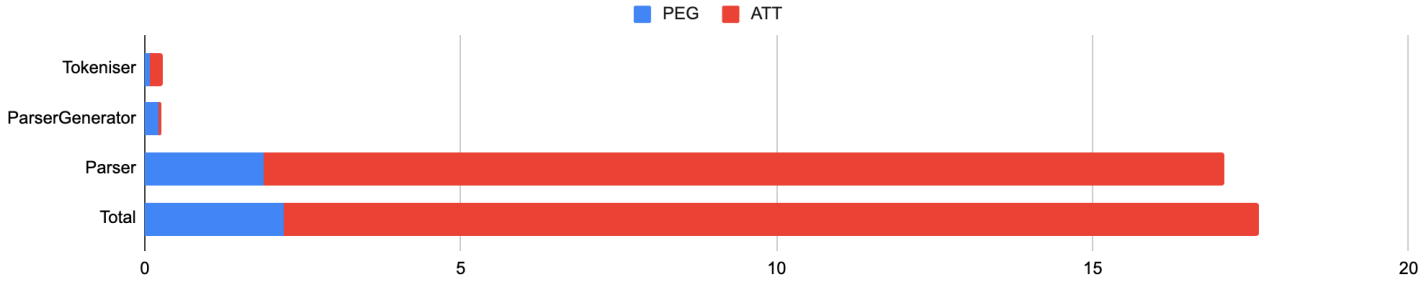



It is possible from the data returned by the solution in its evaluation phase to effectively compare the two methods, 5 preprocessors, and 20 peak-finding sensitivity settings in terms of precision and performance. With reference to Appendix iii, it is possible to note the difference between the precision of the two parsing methods as they were recorded during the control tests where no preprocessors were acting. There is a marked difference, with the Parsing Expression Grammar being 2% more precise in terms of accuracy and 2% more precise in terms of the Matthews Correlation Coefficient. This can be seen in Figure 1 (Appendix iii). However, the difference in terms of the number of false positives and negatives between the two methods is quite striking. The Attribute Grammar produces three times fewer false positives but 50% more false negatives. This can be seen in the Appendix (Chart (viii) and (ix)). In this respect, the Attribute Grammar is more cautious than the Parsing Expression Grammar. In a categorisation task where false positives are considered undesirable, the Attribute Grammar is clearly the superior parser. In the context of vessel safety, false positives are the most important

aspect of a parser's capabilities since even one false positive could put a vessel in danger if it were relying on such a system for its safety. For this reason, Attribute Grammar has proven itself the more appropriate parser for the task.

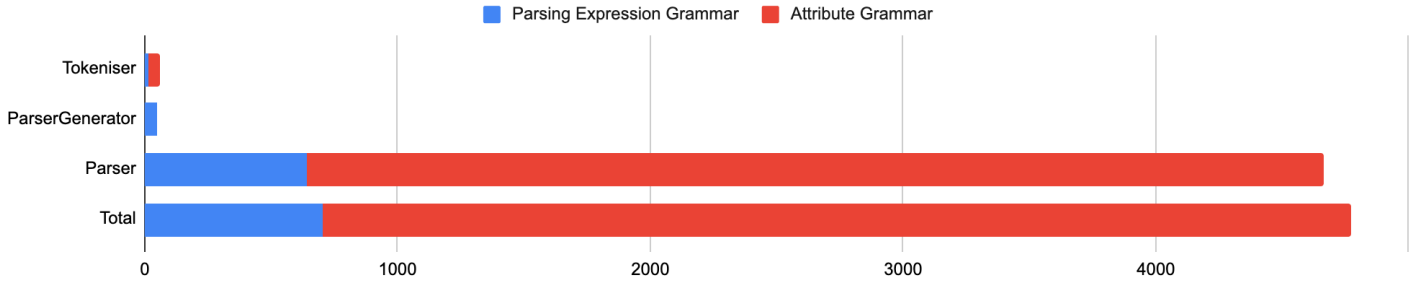
Even on small sets of (100 points) data (representing around 10 seconds) the attribute parser outperforms the parsing expression grammar in terms of false negative identification, however the performance difference is not negligible, as can be seen in Figures 2 and 3 below.

Figure 2. Average Time Taken for the Parsing Expression Grammar and the Attribute Grammar (MS)




Due to the nature of the way that the CFG that makes up the Attribute Parser is parsed, the performance difference continues to be an issue for larger sets of data and the larger the set of data, the larger the disparity between the performance difference. The increase in time taken and memory used for the PEG increases Linearly with the size of the input sentence, while the increase in time taken and memory used for the Attribute parser increases Polynomially (in the order 3).

Figure 3. Average Heap Memory Usage for the Parsing Expression Grammar and the Attribute Grammar (B)



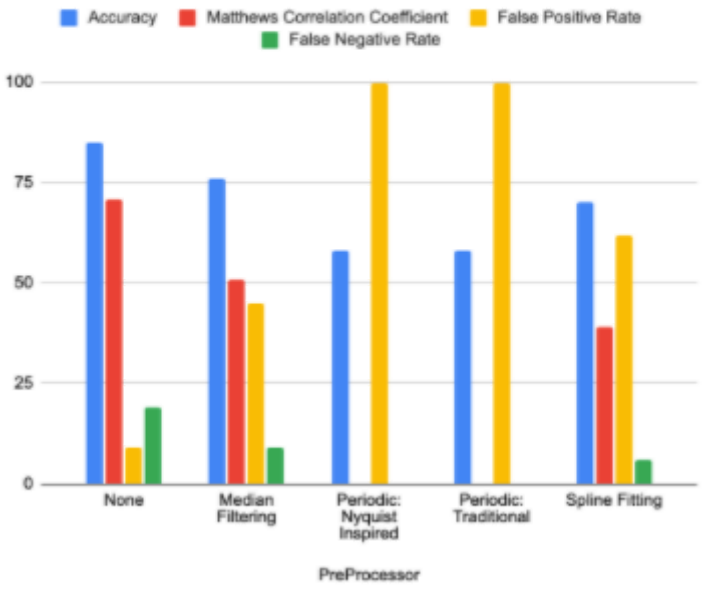

In other cases where the stringency for false positive identification is not so strong, the combination of performance and precision that the parsing expression grammar exhibits make it a more usable parser Regexes are powerful, optimised tools and the optimisation benefits that parsing expression grammars gain from using them play a part in why they are so efficient. Another reason why the parsing expression grammar is so much faster than the attribute grammar is surely its fail-fast qualities. By explicitly describing and ordering those classification that can be made immediately at the front of the ordered list of parsers that a parsing expression grammar utilizes, the method is able to quickly identify and reject certain classes of classification very quickly. In order for the attribute parser to be effective, any production system that wishes to take advantage of it must take into consideration the small data sets for which it is best suited by presenting it with

relatively small data sets. This way, the false negative identification that the method exhibits can be leveraged without incurring the associated time and memory costs that come with using it.

## 5.3 COMPARING PREPROCESSORS

The effectiveness and usefulness of the preprocessing methods selected for investigation differed wildly. In Figure 4 (right), Accuracy, Matthew Correlation Coefficient, False Positive Rate and False Negative Rate are shown for each PreProcessor for the Parsing Expression Grammar. In the control test (None) the Accuracy is 85%, the Matthew Correlation Coefficient is 71%, False Positive Rate is 9%, and the False Negative Rate is 19%. The false positive and negative rates are relatively low and as such the accuracy and Matthew Correlation Coefficient are relatively high. Full data is available in the Appendix (Chart (viii) and (ix)). Median filtering and Spline fitting have comparable accuracies, but the false positive rate for spline fitting is much higher than median filtering as such the median filtering is given a higher Matthew Correlation Coefficient than spline fitting. Both attempts at a periodic filtering were completely unsuccessful, both showing a 100% false positive rate. Clearly these two are not useful for the purpose since, in the context of vessel safety, false positives represent the most important metric. The search for a useful preprocessor was not a success in this investigation. All preprocessors investigated did not exhibit desirable characteristics in terms of precision.

Figure 4. Accuracy, Matthews Correlation Coefficient, False Positive Rate and False Negative Rate, shown for each PreProcessor, for the Parsing Expression Grammar

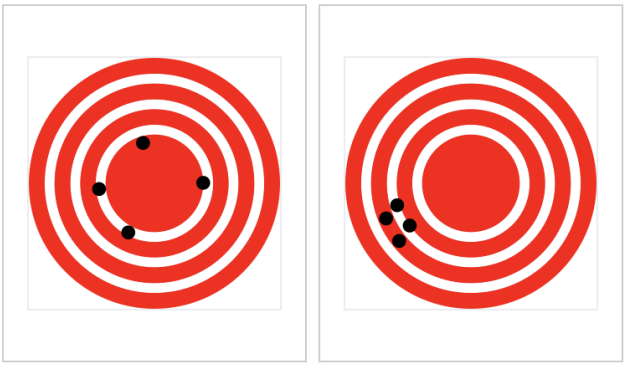



## 5.4 APPRAISING THE SOLUTION

### 5.4.1 The Core and Collections

It is difficult to determine whether the methods chosen lack greater accuracy due to a lack of precision, or a lack of trueness. In this context the words accuracy, precision, and trueness are being used in a slightly different sense that in this last chapter. The figure below attempts to describe the difference between precision and trueness with analogy to an archery target shown below (Wikipedia 2020). On the left it is possible to see an illustration of low accuracy due to a lack of precision. The black dots on the left-hand target surround the center of the target and the midpoint of these shots is roughly central to the bullseye on the target. The black dots on the right-hand

target are in a much tighter pattern in contrast to those on the left-hand target, but the pattern is offset from the center of the bullseye.



It is true to say of these targets that they have the same level of accuracy, but human perception allows a person to readily determine a difference between the two sets of scattered points on those targets; namely that the left-hand target's shots are well-centered but not accurate, while the right-hand target's shots are off-centre in the same way every time. With this knowledge, information about how to improve a shooter's performance might be generated. On the left, the shooter could be told to take their time with their shots but their aim seems fine. On the right, we might ask the shooter to re-scope the bow since it seems like the sight may be off.

The same is not known for the classifications made by each parser, that is to say that each of the thousands of parsing attempts made by each method has not been re-evaluated in the face of the information provided by the parser about its parse to determine in which way the parse went wrong since doing so would take longer than the scope of this paper would allow. Because this information is not known, it is not possible to understand if the solution lacks accuracy due to a lack of precision, or lacks accuracy due to a lack of trueness.

### 5.4.2 The Core and Collections

The design of the Core and Collections has allowed the application to effectively handle variant methods of completing the sub-tasks involved in parsing a sentence. By separating and wrapping variant methods, the core affords repeatable runs of any Collection of functions that we desire. The Core's API should allow the TestBed to carry out programmatic testing of the methods identified.

# 6. FUTURE WORK

This paper serves as the groundwork from which much future work could take its basis. There are several directions that could be explored starting from material that this paper presents and an effort is made here to sensibly divide the identified explorations that could be made into categories.

## 6.1 FURTHERING THE SOLUTION

### 6.1.1 Preprocessing

Among the ideas which I have had but not been able to express are various parser types which may have been able to provide variant and fruitful results. The Lulu Filter is noted to be able to remove impulse errors from data points. We have used the Median Filter to attempt to do this. In their PhD, Jankowitz, M. (2007) shows that "LULU smoothers have a very attractive mathematical structure and satisfy important criteria for smoothers compared to other nonlinear smoothers such as the median smoother". This is a desirable property, and would have been interesting to explore.

### 6.1.2 Production Software

Presently, the application is only capable of parsing a small to large group of waves, presented in a single array of wave data points. In order to realise this application as a production-ready system it may be desirable to analyse groups of waves one after another without having to restart the program by providing the same set of vessel data and a different set of wave data.
The currently implemented core would be capable of supporting this functionality with a few changes to the control flow and the addition of a loop that would allow a user to input new data after receiving the results of the last data.

This small change may not be enough to represent a more useful program. A more functional production implementation of this technology could instead collect wave data live from a vessels' sensors, making intelligent decisions about how often to parse the collected data. The simplest implementation of this functionality could take the form of a period tokenization and parsing of the collected data. This area of research would be ideal for those looking to use the technology presented in this paper to further the field of vessel safety.

There is a gap in the market for an application that allows sailors to track wave patterns on their mobile device using the on-board sensors and this presents an interesting opportunity for future work. The application need not be complex. Current wave conditions and historic wave conditions presented in the form of a mobile application for sailors to use to visualise the current wave

conditions is not something that currently exists and this paper serves as a basis for anyone looking to research the implementation of such a product.

### 6.1.3 More Vessel Constraints

It would have been nice to use more of the vessel's dimensions and attributes to better define the grammars produced.

## 6.2 OTHER DIRECTIONS

### 6.2.1 Experimental Parser Work

In their slide deck, L. Thomas van Binsbergen of University of London, discusses a method by which it is possible to define Parser Combinators that make use of attributes. This represents a means by which we could merge all three of the parsing methods defined in this paper. This area of research would be ideal for those looking to use the technology presented in this paper to further research into theoretical parsing methods and their applications.

# 7. CONCLUSIONS

The purpose of this paper and the solution outlined within it were to investigate the application of syntactic pattern recognition to the analysis of water waves in the context of boat safety, covering as much of the associated material involved with parsing a syntactic representation of a water wave including data capture, preprocessing and transformation, tokenization of waves into sentences, and parsing of syntactic representations, in order to create a comparative analysis of the methods selected. The solution designed was a success overall, allowing the parsing of a syntactic representation of a water wave to be investigated. Definitive conclusions were reached about which parsing method was best and the effect of preprocessors on wave data, which was the purpose of the paper.

The Attribute Grammar was clearly the best representation for water waves in the context of analysing them for vessel safety. The Attribute Parser, though it was slower and used a lot more memory, exhibited those characteristics that were most desirable - namely as low a false positive rate as possible. An Attribute Grammar's ability to store and communicate values around the abstract syntax tree and to represent those values in an attribute table allows, in this context, for a finer-grained analysis of the wave data during the tokenisation phase. The means by which the Context Free Grammar that makes up the Attribute Grammar is parsed is the CYK algorithm and this is the major problem that the Attribute Grammar has. If the Context Free Grammar were parsed by some other, faster algorithm than the CYK algorithm, the Attribute Grammar would not present the undesirable performance characteristics that it.

The Parsing Expression Grammar exhibited a greater Matthews Correlation Coefficient and also had far more desirable performance characteristics but produced three times more false positives and was therefore a terrible candidate for the better method in this context, though might be useful in other contexts since its Matthews Correlation Coefficient was higher and it had better performance characteristics.

Using both of these methods together, it should be possible to allow a parser to combine the categorisation benefits of the Attribute Grammar with the performance benefits that the Parsing Expression Grammar exhibits. In the context of vessel safety, a combined method like this would work best, allowing low rates of false positives in the categorisation , greater performance, and therefore also the ability to handle larger data sets.

None of the preprocessing methods were a success. Preprocessing wave data, though often drastically reducing the amount of time taken or the memory used for the parsing methods that accepted the preprocessed wave data, dramatically increased the rates of false positive categorisation for both parsing methods. As such not preprocessing the data is clearly the best approach, despite the performance advantages that it does provide.

Having spent many weeks in the process of experimenting with the particulars of how to best represent parser composition in Javascript, this investigator feels as though certain tasks could now be implemented again - using the knowledge gained over the last few weeks - in much less time and to a higher degree of quality. Parser composition is difficult to implement for the same reason that other composition tasks are - that is that the functions which make up a larger composed function must accept arguments and pass return values along in a sort of chain from the first to the last without knowing which function might have been composed before them or will be composed after them. One problem here is that of error handling. Some functions may be able to handle a certain class of error and not another class, and some functions may not handle or produce errors at all. Even if a function never produces or deals with errors in any way, it still must be prepared to check for them, sort them by type and pass them on, meaning that sorting input and sanity checking it becomes a huge part of the implementation for every function that may be composed into some chain of functions.

# 8. REFLECTIONS

Producing this paper and implementing the solution have been a learning experience for me. While I feel as though my programming was strong at the start of this project; I now have the experience of what it feels to bring a piece of software with this many moving parts to completion. The importance of developing a good plan and sticking to it have really been made clear to me while working my way through this project. I feel as though I planned and executed the parsing methods and the core software carefully, but that the implementation of my evaluation code was not to the same level of quality and that the overall amount of time and research that I spent planning my evaluation was less than it should have been. I intend to improve on the parser produced in the solution and produce a prototype product that incorporates all the elements discussed here into a single application that provides safety-checking for boats out on the water. Such an application could be distributed for free and, with sufficient warnings about how it is an untested prototype, might be able to provide an extra layer of security for vessel pilots. When doing so, I will take into account that any solution is only as good as the evaluation to which it can be subjected.

The solution I produced does allow great insight into the differences between the two parsers that were studied and between the preprocessors that were considered. It is possible from the results to see how different the methods are in terms of their precision and their performance. However, some assumptions made at the start of the process have been proved entirely wrong. For example, I fully expected to be able to take advantage of preprocessing to remove data errors and reduce the amount of data that would be needed to make a successful characterisation. In fact, I came to see that this was not the case at all and it is now clear to me that I made the assumption that using a preprocessor would be useful without realising I was making an assumption. The conclusion I am now making about preprocessors not being useful may also be an assumption, since I have no way of confirming that I chose the right preprocessing methods or implemented them in the best way. Much more study needs to be done into preprocessing wave data before tokenization. In the future, I intend to question the things that I assume - even if they may seem obvious.

Implementing software is important. However, in a scientific investigation, the empirical evaluation of the results gathered is of the highest importance. The scientific method involves making hypotheses and then carrying out empirical observations based on those hypotheses in order to determine the validity of those hypotheses. In the future, I will be taking this into consideration whenever I plan a project - that is to say I will ask the questions “What is my hypothesis” and “What can I do to prove or to disprove that hypothesis” and plan an approach that takes these two points into consideration rather than first thinking about how I go about constructing the software.

When writing a scientific paper, it is of the highest importance to be able to present the events, thoughts, and conclusions that an investigator goes through during the course of the investigation - so as to provide the reader with a comprehensive selection of important points. Doing this well is not possible when one does not take notes as one is going along. While I was careful to make notes about what I did in order to record them in this paper, I was not rigorous about doing so. From now on, I will be keeping a daily diary of events that occur and keeping them chronological. I feel as though doing this will aid me greatly in the next scientific paper I write as I will be able to present a more ordered paper and justify, if not only to myself, exactly where every hour has been spent.

Lastly, I have learned just how hard writing a scientific paper is. I allotted myself too little time to write a report that does justice to the work and time I have put into the project. I feel as though I haven't drawn out the conclusions to the extent that I could have given more time and I have my own planning to blame for this. I am excited to write more scientific papers in the very near future and, when I do so, planning the report will be the first thing that I focus my attention on. Remembering to keep meticulous notes will help me to organise myself, and constantly asking myself if I am doing work that will prove a hypothesis will keep me on track.

## 9. GLOSSARY OF TERMS

These terms are used in this paper.

| | |
|---|---|
| **Parser** | A parser is a function which takes in a sentence and returns the result of whether or not that sentence belongs to a set of constraints. |
| **ParserGenerator** | A ParserGenerator returns a parser. |
| **Tokeniser** | A tokenizer takes a set of data points and turns those data points into tokens. |
| **PreProcessor** | A preprocessor sits before tokenization and prepares data in some novel way. |
| **Combinator** | A combinator is a higher-order function that uses only function application and earlier defined combinators to define a result from its arguments. |
| **Functional Programming** | A programming paradigm in which programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are trees of expressions that map values to other values, rather than a sequence of imperative statements which update the running state of the program. |
| **Production** | A rewrite rule specifying a symbol substitution that can be recursively performed to generate new symbol sequences. |
| **Terminal** | Terminal symbols are the elementary symbols of the language defined by a formal grammar. |
| **Non-Terminal** (**Variable**) | Nonterminal symbols (or syntactic variables) are replaced by groups of terminal symbols according to the production rules. |
| **Accuracy** | |
| **Precision** | |
| **Trueness** | |

## 10. ABBREVIATIONS

These abbreviations are used in this paper.

| | |
|---|---|
| **CYK** | Cocke–Younger–Kasami Algorithm |
| **PEG** | Parsing Expression Grammar |
| **CFG** | Context-Free Grammar |
| **CNF** | Chomsky-Normal-Form |

# 11. APPENDIX

These figures, tables, and diagrams are references in this paper.

**Chart** (**i**) “Basic Context-Free Grammar for the Parsing Expression Grammar”

| **Human-Readable** | **Machine-Friendly** |
|---|---|
| Non-Terminals | Non-Terminals |
| Start, Grouping, Peak, Trough, Magnitude | S, G, P, T, M |
| Terminals | |
| f, p, t, 1, 2, 3, 4, 5, ε | |
| Productions | Productions |
| Start -> Grouping Start \| Grouping \| ε<br>Grouping -> Peak \| Trough \| F<br>Peak -> P Magnitude<br>Trough -> T Magnitude<br>Magnitude -> 1 \| 2 \| 3 \| 4 \| 5 | S -> G S \| G \| ε<br>G -> P \| T \| f<br>P -> p M<br>T -> t M<br>M -> 1 \| 2 \| 3 \| 4 \| 5 |

**Chart** (**ii**) “Chomsky-Normal-Form of the same Context-Free Grammar for the Parsing Expression Grammar"

| **Human-Readable** | **Machine-Friendly** |
|---|---|
| Non-Terminals | Non-Terminals |
| $Start_0$, Start, Grouping, Peak, Trough, Magnitude, a, b | Z, S, G, P, T, M |
| Terminals | |
| f, p, t, 1, 2, 3, 4, 5, ε | |
| Productions | Productions |
| $Start_0$ -> Grouping Start \| f \| T Magnitude \| P Magnitude<br>Start -> Grouping Start \| f \| T Magnitude \| P Magnitude | Z -> G S , f , T M , P M<br>S -> G S , f , T M , P M<br>G -> f , P M , T M<br>M -> 1 , 2 , 3 , 4 , 5 |

| | |
|---|---|
| Grouping -> f \| T Magnitude \| P Magnitude<br>Magnitude -> 1 \| 2 \| 3 \| 4 \| 5<br>P -> p<br>T -> t | T -> t<br>P -> p |

**Chart** (**iii**) "Basic Attributed Context-Free Grammar for the Attribute Grammar"

| Human-Readable | Machine-Friendly |
|---|---|
| Non-Terminals | Non-Terminals |
| Start, Class, Classπ, Classα, Classφ, Classδ, Group, Grouping, Peak, Trough, Magnitude | S, C, Π, A, Φ, Δ, G, Γ, L, Λ |
| Terminals | |
| π, α, φ, δ, f, p, t, ., ε | |
| Productions | Productions |
| Start -> Class Start \| Class \| ε<br>Class -> Classπ \| Classα \| Classφ \| Classδ<br>Classπ -> π Group .<br>Classα -> α Group .<br>Classφ -> φ p . \| φ t .<br>Classδ -> δ Group .<br>Group -> Group Grouping \| Grouping<br>Grouping -> p \| t \| f | S -> C S \| C \| ε<br>C -> Π \| A \| Φ \| Δ<br>Π -> π G .<br>A -> α G .<br>Φ -> φ p . \| φ t .<br>Δ -> δ G .<br>G -> Γ G \| Γ<br>Γ -> p \| t \| f |

**Chart** (**iv**) "Chomsky-Normal-Form Attributed Context-Free Grammar for the Attribute Grammar"

| Human-Readable | Machine-Friendly |
|---|---|
| Non-Terminals | Non-Terminals |
| Start, Class, ClassΠ, ClassA, ClassΦ, ClassΔ, Grouping, GroupingSelection, Peak, Trough, Magnitude | S, C, p, a, f, d, G, g, p, t, M |
| Terminals | |
| Π, A, Φ, Δ, F, P, T, ., ε | |

| Productions | Productions |
|---|---|
| Start -> Class Start \| Class \| ε<br>Class -> ClassΠ \| ClassA \| ClassΦ \| ClassΔ<br>ClassΠ -> Π Grouping .<br>ClassA -> A Grouping .<br>ClassΦ -> Φ P . \| Φ T .<br>ClassΔ -> Δ Grouping .<br>Grouping -> GroupingSelection Grouping \|<br>GroupingSelection<br>GroupingSelection -> Peak \| Trough \| F<br>Peak -> P Magnitude<br>Trough -> T Magnitude<br>Magnitude -> 1 \| 2 \| 3 \| 4 \| 5 | Z -> CS \| AX \| ΔX \| ΠΘ \| Φλ<br>S -> CS \| AX \| ΔX \| ΠΘ \| Φλ<br>C -> AX \| ΔX \| ΠΘ \| Φλ<br>A -> α<br>Δ -> δ<br>Π -> π<br>Φ -> φ<br>Ω -> .<br>X -> GΩ<br>G -> GY \| f \| p \| t<br>Y -> f \| p \| t<br>Θ -> ΓΩ<br>Γ -> ΓΣ \| p \| t<br>Σ -> p \| t<br>λ -> ΣΩ |

**Chart** (**v**) “VesselData data object definition”

| ***VesselData*** | |
|---|---|
| Length | The length of the Vessel, measured in feet from the bow to the stern. |
| Width | The width of the Vessel, measured in feet from one leeboard to another at the maximum width of the Vessel. |
| Berth | The berth of the Vessel, measured in persons. |
| Motor | Whether or not the Vessel's motor is currently engaged. |
| Sail | Whether or not the Vessel is currently under sail. |
| Name | The name of the VesselData |

**Chart** (**vii**) “WaveData data object definition”

| ***WaveData*** | |
|---|---|
| Series | The data-points of the waveform to be analysed. This is an array of objects in the form { x, y }. |
| Name | The name of the WaveData |

| | |
|---|---|
| Annotations | A Set of annotations which indicate the vessels this WaveData is safe for. |

**Chart (viii)** "True Positives, True Negatives, False Positives, False Negatives, Accuracy, Matthews Correlation Coefficient (presented as a %), False Positive Rate (presented as a %), and False Negative Rate (presented as a %) for the Parsing Expression Grammar"

| PreProcessor | TP | TN | FP | FN | ACC | MCC | FPR | FNR |
|---|---|---|---|---|---|---|---|---|
| None | 5594 | 4608 | 472 | 1326 | 85 | 71 | 9 | 19 |
| Median Filtering | 6292 | 2817 | 2263 | 628 | 76 | 51 | 45 | 9 |
| Periodic: Nyquist Inspired | 6920 | 0 | 5080 | 0 | 58 | 0 | 100 | 0 |
| Periodic: Traditional | 6920 | 0 | 5080 | 0 | 58 | 0 | 100 | 0 |
| Spline Fitting | 6484 | 1910 | 3170 | 436 | 70 | 39 | 62 | 6 |

**Chart (ix)** "True Positives, True Negatives, False Positives, False Negatives, Accuracy, Matthews Correlation Coefficient (presented as a %), False Positive Rate (presented as a %), and False Negative Rate (presented as a %) for the Attribute Grammar"

| PreProcessor | TP | TN | FP | FN | ACC | MCC | FPR | FNR |
|---|---|---|---|---|---|---|---|---|
| None | 5010 | 4936 | 144 | 1910 | 83 | 69 | 3 | 28 |
| Median Filtering | 5782 | 3257 | 1823 | 1138 | 75 | 49 | 36 | 16 |
| Periodic: Nyquist Inspired | 6920 | 0 | 5080 | 0 | 58 | 0 | 100 | 0 |
| Periodic: Traditional | 6920 | 0 | 5080 | 0 | 58 | 0 | 100 | 0 |
| Spline Fitting | 6238 | 2192 | 2888 | 682 | 70 | 39 | 57 | 10 |

**Chart** (**x**) "Average Heap Memory Used (in Bytes) for the individual sub-processes that make up the Parsing Expression Grammar"

| PreProcessor | Tokeniser (B) | ParserGenerator (B) | Parser (B) | Total (B) |
|---|---|---|---|---|
| None | 13.11 | 47.47 | 642.63 | 703.21 |
| Median Filtering | 4.64 | 47.25 | 126.64 | 178.53 |
| Periodic: Nyquist Inspired | 1.29 | 47.31 | 2.19 | 50.79 |
| Periodic: Traditional | 1.26 | 47.31 | 2.19 | 50.76 |
| Spline Fitting | 3 | 47.27 | 106.21 | 156.48 |

**Chart** (**xi**) "Average Heap Memory Used (in Bytes) for the individual sub-processes that make up the Attribute Grammar"

| PreProcessor | Tokeniser (B) | ParserGenerator (B) | Parser (B) | Total (B) |
|---|---|---|---|---|
| None | 47.81 | 0.67 | 4020.81 | 4069.29 |
| Median Filtering | 11.28 | 0.59 | 203.71 | 215.58 |
| Periodic: Nyquist Inspired | 1.86 | 0.59 | 1.05 | 3.5 |
| Periodic: Traditional | 1.8 | 0.59 | 1.05 | 3.44 |
| Spline Fitting | 7.83 | 0.59 | 139.09 | 147.51 |

**Chart** (**xii**) "Average Time Taken (in Milliseconds) for the individual sub-processes that make up the Parsing Expression Grammar"

| PreProcessor | Tokeniser (MS) | ParserGenerator (MS) | Parser (MS) | Total (MS) |
|---|---|---|---|---|
| None | 0.08 | 0.22 | 1.89 | 2.19 |
| Median Filtering | 0.06 | 0.23 | 0.57 | 0.86 |
| Periodic: Nyquist Inspired | 0.06 | 0.29 | 0.09 | 0.44 |
| Periodic: Traditional | 0.05 | 0.22 | 0.06 | 0.33 |
| Spline Fitting | 0.05 | 0.22 | 0.43 | 0.7 |

**Chart** (**xiii**) "Average Time Taken (in Milliseconds) for the individual sub-processes that make up the Attribute Grammar"

| PreProcessor (MS) | Tokeniser (MS) | ParserGenerator (MS) | Parser (MS) | Total (MS) |
|---|---|---|---|---|
| None | 0.2 | 0.04 | 15.19 | 15.43 |
| Median Filtering | 0.08 | 0.04 | 0.83 | 0.95 |
| Periodic: Nyquist Inspired | 0.05 | 0.05 | 0.05 | 0.15 |
| Periodic: Traditional | 0.05 | 0.05 | 0.05 | 0.15 |
| Spline Fitting | 0.08 | 0.04 | 0.65 | 0.77 |

# 12. REFERENCES


- Atkinson, K. (1989). An Introduction To Numerical Analysis. 2nd ed. John Wiley & Sons.

- Bender, L. C., Guinasso, N. L., and Walpert J. N. (2009). A Comparison of Methods for Determining Significant Wave Heights - Applies to a 3-m Discus Buoy during Hurricane Katrina. *Journal of Atmospheric and Oceanic Technology.* [online] Volume 27, pages 1012 - 1028. Available at: https://journals.ametsoc.org/view/journals/atot/27/6/2010jtecho724_1.xml [Accessed 10 May 2021].

- Ford, B. (n. d.). Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. [online] Bryan Ford's Homepage. Available at: https://bford.info/pub/lang/peg.pdf [Accessed 10 May 2021].

- github.com (n. d.). wave-simulation. [online] Github Repository for wave-simulation. Available at: https://github.com/ulidtko/wave-simulation [Accessed 10 May 2021].

- github.com (n. d.). OceanSurface. [online] Github Repository for OceanSurface. Available at: https://github.com/jiasli/OceanSurface [Accessed 10 May 2021].

- github.io (n. d.). Immutable collections for JavaScript. [online] Immutable JS Documentation. Available at: https://immutable-js.github.io/immutable-js [Accessed 10 May 2021].

- google.com (n. d.). Display data live on your site. [online] Google Charts Documentation. Available at: https://developers.google.com/chart [Accessed 10 May 2021].

- Grune, D. (2008). Parsing Techniques - A Practical Guide. 2nd ed. Springer US.

- Hopcroft, J., Motwani, R., and Ullman, J. (1979). Introduction To Automata Theory, Languages, and Computation. Addison-Wesley.

- Hutton, G. and Meijer, E. (1996). Monadic Parser Combinators. [online] Official Website of University of Nottingham. Available at: https://www.cs.nott.ac.uk/~pszgmh/monparsing.pdf [Accessed 10 May 2021].

- Jankowitz, M. (2007). Some Statistical Aspects of LULU Smoothers. PhD. University of Stellenbosch.

- medium.com (2021). CYK CKY. [online] Medium article. Available at: https://medium.com/swlh/cyk-cky-f63e347cf9b4 [Accessed 10 May 2021].

- Miček, J. and Kapitulík J. (2003). Median Filter. *Journal of Information, Control and Management Systems*, [online] Volume 1, Number 2, pages 51 - 56. Available at: https://www.researchgate.net/publication/267243676_Median_filter [Accessed 10 May 2021].

- ngrok.com (n. d.). Documentation. [online] ngrok Documentation. Available at: https://ngrok.com/docs [Accessed 10 May 2021].

- Por, E., van Kooten, M., and Sarkovic, V. (2019). Nyquist-Shannon sampling theory. [online] Official Website of Leiden University. Available at: https://home.strw.leidenuniv.nl/~por/AOT2019/docs/AOT_2019_Ex13_NyquistTheorem.pdf [Accessed 10 May 2021].

- pythonhosted.com (n.d.). Parsec, A Parser Combinator Library in Python. [online] Parsec Documentation. Available at: https://pythonhosted.org/parsec [Accessed 10 May 2021].

- reddit.com (2021). Looking for high resolution buoy data. [Blog] r/oceanography. Available at: https://www.reddit.com/r/oceanography/comments/lrjdfd/looking_for_high_resolution_buoy_data [Accessed 10 May 2021].

- Binsbergen, L. (n. d.). Parser Combinators and Attribute Grammars. [online] Official Website for Royal Holloway. Available at: https://pure.royalholloway.ac.uk/portal/files/26243112/poster.pdf [Accessed 10 May 2021].

- Tessendorf, J. (1999). Simulating Ocean Water. [online] Official Website of Penn State University. Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.9102&rep=rep1&type=pdf [Accessed 10 May 2021].

- Trahanias, P. and Skordalakis E. (1990). Syntactic Pattern Recognition of the ECG. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, [online] Volume 12, Number 7, pages 648 - 657. Available at: https://www.cs.rit.edu/~rlaz/PatternRecognition/slides/ECGSyntPR.pdf [Accessed 10 May 2021].

- Tredup, S (2021). Dangerous Waves and Your Boat. [online] Ocean Navigator. Available at: https://www.oceannavigator.com/dangerous-waves-and-your-boat [Accessed 10 May 2021].

- wikipedia.com (2021). Automatic Identification System. [online] Wikipedia entry for AIS. Available at: https://en.wikipedia.org/wiki/Automatic_identification_system#References [Accessed 10 May 2021].

- wikipedia.com (2020). Median Filtering. [online] Wikipedia Entry for Median Filtering. Available at: https://en.wikipedia.org/wiki/Median_filter [Accessed 10 May 2021].

- wikipedia.com (2021). Attribute Grammar. [online] Wikipedia entry for Attribute Grammar. Available at: https://en.wikipedia.org/wiki/Attribute_grammar [Accessed 10 May 2021].