Modelling of Functional Programming

Carlton Mpofu

Grahamstown, South Africa
August 6, 2025

Acknowledgements

I would like to take this opportunity to acknowledge God and my ancestors for guiding me through this challenging journey. Their presence has been a source of strength and perseverance throughout this process.

A heartfelt thank you to my mother, who has been my unwavering cheerleader and rock. Her love, support, and encouragement have kept me grounded and motivated, even in the most difficult moments.

I am deeply grateful to my supervisor, Dr. Yusuf Motara, for his exceptional guidance and patience. His unwavering support, insightful feedback, and belief in my work have been invaluable. I truly appreciate his dedication and the effort he put into guiding me through this thesis.

I also extend my gratitude to everyone who directly or indirectly contributed to this work. Special thanks to all the amazing individuals who participated in the survey and helped share it. Your contributions were essential to this research, and I sincerely appreciate your time and effort.

To all my friends, colleagues, and loved ones

Abstract

Effective system modeling is essential for understanding, communicating, and refining software architectures. While various notations exist for representing system structures, the modeling of typed functional systems remains underexplored. This research aimed to develop a structural modeling notation tailored to functional programming, integrating the Physics of Diagrams (PoD) framework to enhance clarity, usability, and cognitive effectiveness. The proposed notation was designed with a three-tiered structure to improve organization, distinguish graphical elements for perceptual discriminability, and novel symbols for representing key functional system relationships.

The effectiveness of the notation was evaluated through a survey incorporating four case studies: (1) FParsec for parsing system representation, (2) Docutils for adaptability assessment in non-functional systems, (3) Pandoc for evaluating practical applicability, and (4) a symbol-level analysis to assess user comprehension. Results demonstrated that the notation improved clarity and usability, particularly in distinguishing definitions, subtypes, and dependencies. However, challenges were identified in representing invocation relationships and alternative pathways, necessitating further refinements.

This study confirms that PoD principles provide a strong foundation for designing functional system notations but highlights the need for iterative enhancements. Future work should focus on refining specific notational elements, expanding empirical validation with a larger participant base, and exploring integrations with behavioral modeling to create a holistic notation for functional systems.

Contents

1 Introduction **11**

1.1 Problem statement 13

1.2 Research question 13

1.3 Research objectives 14

1.4 Structure 15

2 Literature review **17**

2.1 Functional Programming 17

2.1.1 Historical and Theoretical Foundations 17

2.2 Review of existing modeling languages 22

2.3 UML and its shortcomings in representing functional systems 29

2.4 The role of system design and modeling 31

2.5 HLO (Hello) 33

2.5.1 Philosophical Investigations (PI) 33

2.5.2 The Language of Mathematics (LoM) 34

2.5.3 Translating to HLO 35

2.5.4 Example 38

2.6 Conclusion 40

3 Methodology **41**

3.1 Backgound 41

3.2 Fundamental Principles 43

3.3 Design process 45

3.4 Case Studies in Action: Methodology and Selection Rationale 50

3.5 Exploring Structural Modeling Notations: Three Case Studies 52

3.6 Evaluation Process 53

3.6.1 Purpose of the Evaluation 53

3.6.2 Evaluation Design 53

3.6.3 Participants and Sampling 56

3.6.4 Criteria for Analysis 56

3.6.5 Data Collection and Ethical Considerations 59

3.6.6 Anticipated Outcomes 59

3.7 Conclusion 59

4 Implementation **61**

4.1 Notation 61

4.2 Case One 72

4.3 Case Two 77

4.4 Case Three 82

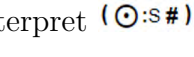
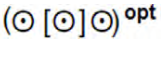
4.5 Case Four 87

4.5.1 Evaluation of the OR-Arrow Symbol 87

4.5.2 Comparative Analysis of the OR-Arrow Symbol 88

4.5.3 Evaluation of the Notation Label (::) Symbol 90
4.5.4 Evaluation of Sub-Definition Notation 91
4.5.5 Comparative Evaluation of Diagrams for Main Definitions and Sub-Definitions 92
4.5.6 Evaluation of Relationships at Level 2 93
4.5.7 Holistic Comparison of Diagrammatic Notations 94
4.6 Conclusion 96

5 Results and Discussion 98

5.1 Participant Overview and General Performance 98
5.1.1 Participant Demographics 98
5.1.2 Overall Task Performance 99
5.1.3 Performance Analysis by Participant Category 99
5.1.4 Response Categorization Criteria 102
5.2 Case One 102
5.2.1 Q1: Recognition of Main Definitions 102
5.2.2 Q2: Identifying Sub-Definitions 104
5.2.3 Q3: Identify all the definitions that rely on other definitions 106
5.2.4 Q4: What does the definition  signify in the context of parsing ? 109
5.2.5 Q5: In  , what does the presence of  versus  indicate ? 110
5.2.6 Q6: How would you interpret  ? 111
5.2.7 Q7: How would  behave in the context of parsing ? . . 112
5.3 Case Two 113

5.3.1 Q1: Recognition of Main Definitions 113

5.3.2 Q2: Identifying Sub-Definitions 115

5.3.3 Q3: What is the result of ⊕:S ..⊙:S ? 117

5.3.4 Q4: In the definition △f.△, what does the transformation f.△ imply ? 118

5.3.5 Q5: What does the combined definition (⊕:S ..⊙:S)!△ represent ? . . 119

5.3.6 Q6: If △0 (absent document tree), is passed to the writer in ⓘ▪△opt, what is the outcome ? 120

5.3.7 Q7: What does the combination (((⊕:S ..⊙:F)!△) opt imply ? 121

5.3.8 Q8: Which sequence best represents a full, successful input-to-output workflow using Level 2 definitions? 122

5.4 Case Three 123

5.4.1 Q1: Recognition of Main Definitions 123

5.4.2 Q2: Identifying Sub-Definitions 125

5.4.3 Q3: What does the ⊜.. ⊕ produce ? 127

5.4.4 Q4: What does the combined definition (((⊜:S .. ⊕:S)! ▢) represent ? 128

5.4.5 Q5: What does the notation ⊕f.▢ imply in the context of transformation ? 129

5.4.6 Q6: If the ((⊜ ++⊜) .. ⊕:S)!▢ is used, what does it indicate about the input processing ? 131

5.4.7 Q7: If (⊜:S .. ⓘ:W)!★ occurs, what is the likely outcome ? 132

5.5 Case Four 132

5.5.1 SYM1: Evaluation of the OR-Arrow Symbol 132

5.5.2 SYM2: Comparative Analysis of the OR-Arrow Symbol 136

5.5.3 SYM3: Evaluation of the Notation Label (::) Symbol 138

5.5.4 SYM4: Evaluation of Sub-Definition Notation 140

5.5.5 SYM5: Comparative Evaluation of Diagrams for Main Definitions and Sub-Definitions 142

5.5.6 SYM6: Evaluation of Relationships at Level 2 143

5.5.7 SYM7: Holistic Comparison of Diagrammatic Notations 145

5.6 Conclusion 154

6 Conclusion **158**

6.1 Summary of Objectives and Contributions 158

6.2 Key Findings from Case Studies 158

6.2.1 Case Study 1: FParsec 158

6.2.2 Case Study 2: Docutils 159

6.2.3 Case Study 3: Pandoc 159

6.3 Evaluation of Notation Elements 159

6.4 Revisiting the Research Problem and Question 159

6.5 Future Research Directions 160

A Ethics Approval **166**

B Survey questionnaire **167**

List of Figures

2.1 Example of pattern matching for branching (reproduced from (Mitkin 2012) 24

2.2 pattern matching in Drakon-Erlang (reproduced from (Mitkin 2012) 24

2.3 Example of a data flow diagram (modified from Ibrahim et al. 2010) 26

2.4 Example of a business process model with data in BPMN (reproduced from (Decker et al. 2009)) 29

2.5 Notation for definition (left) and thesaurus (right)) (Motara 2021) 36

2.6 Example of a HLO diagram (Motara 2021) 38

3.1 "Whereas Moody's "Physics of Notations" focuses exclusively on the bottom-left hand quadrant, the "Physics of Diagrams" proposed in this paper focuses on the top-left quadrant of the figure. Both artefacts leave the right-hand quadrants out of scope. (adapted from (Moody, 2009))" Pissierssens, Claes, and Poels (2019) 42

3.2 The Physics of Diagrams: Seven Core Principles for Graphic Representation Design (reproduced from (Pissierssens, Claes, and Poels 2019)) 43

4.1 Notation for level definition (left), notation for level 2 definition (middle), and thesaurus (right) 63

4.2 Definition of a parser (Motara 2021) 65

4.3 Revised definition of a parser 66

4.4 An example of a sub-definition (Motara 2021) 66

4.5 An example of a sub-definition (revised notation) 67

4.6 Foundational definitions (Motara 2021) 68

4.7 Foundational definitions (revised notation) 69

4.8 Relies relationship (Motara 2021) 70

4.9 Relies relationship (revised notation) 71

4.10 FParsec: Parser definition 72

4.11 FParsec: some type or value 72

4.12 FParsec: Level two definition (a) and (b) 73

4.13 FParsec: Level two definition (c) and (d) 73

4.14 FParsec: Level two definition (e) and (f) 74

4.15 FParsec: Level three definitions 75

4.16 FParsec diagram 76

4.17 Docutils diagram 81

4.18 Pandoc diagram 86

4.19 Level 1 definition 87

4.20 Symbol Two 89

4.21 Symbol Three 90

4.22 Symbol Four 91

4.23 Symbol Five 92

4.24 Symbol Six 93

4.25 Symbol 7 (diagram A) (Motara 2021) 95

5.1 Distribution of Overall Participant Performance Scores. 101

5.2 FParsec Q1, Q2, and Q3 options. 103

5.3 Case One (Q1) Frequency of responses by option. 104

5.4 Case One (Q2) Frequency of responses by option. 105

5.5 Case One (Q3) Frequency of responses by option. 107

5.6 Case One (Q4) Frequency of responses by option. 109

5.7 Case One (Q5) Interpretation of Symbol Presence in Level 2 Definition. . . 110

5.8 Case One (Q6) Response Distribution for Combined Level 2 Definitions. . 111

5.9 Case One (Q7) Response Distribution for Combined Level 2 Definitions. . 112

5.10 Docutils Q1, and Q2 options. 113

5.11 Case Two (Q1) Frequency of responses by option. 114

5.12 Case Two (Q2) Frequency of responses by option. 116

5.13 Case Two (Q3) Frequency of responses by option. 117

5.14 Case Two (Q4) Frequency of responses by option. 118

5.15 Case Two (Q5) Frequency of responses by option. 119

5.16 Case Two (Q6) Frequency of responses by option. 120

5.17 Case Two (Q7) Frequency of responses by option. 121

5.18 Case Two (Q8) Frequency of responses by option. 122

5.19 Pandoc Q1, and Q2 options. 124

5.20 Case Three (Q1) Frequency of responses by option. 125

5.21 Case Three (Q2) Frequency of responses by option. 126

5.22 Case Two (Q3) Frequency of responses by option. 127

5.23 Case Three (Q4) Frequency of responses by option. 129

5.24 Case Three (Q5) Frequency of responses by option. 130

5.25 Case Three (Q6) Frequency of responses by option. 131

5.26 Case Three (Q7) Frequency of responses by option. 132

List of Tables

2.1 Graphical Notation for DFD Components (Dennis, Wixom, and Roth 2008) 25

3.1 Guidelines (G) about perceptual discriminability found in literature (Pissierssens, Claes, and Poels 2019) 45

3.2 Guidelines (G) about pragmatic clarity found in literature (Pissierssens, Claes, and Poels 2019) 46

3.3 Guidelines (G) about visual expressiveness found in literature (Pissierssens, Claes, and Poels 2019) 47

3.4 Guidelines (G) about visual chunking found in literature (Pissierssens, Claes, and Poels 2019) 48

3.5 Guidelines (G) about hierarchical chunking found in literature (Pissierssens, Claes, and Poels 2019) 48

3.6 Guidelines (G) about direction of information found in literature (Pissierssens, Claes, and Poels 2019) 48

3.7 Guidelines (G) about internal and external linkage found in literature (Pissierssens, Claes, and Poels 2019) 49

5.1 Breakdown of Participant Role and Familiarity 99

5.2 Individual Participant Scores (Sorted High→Low) 100

5.3 Performance Score by Participant Role 100

5.4 Mean Performance Score by Familiarity Level 101

5.5 Mean Performance Score (%) by Role and Familiarity 101

5.6 Quantitative Results for Case 4 SYM 1 133

5.7 Quantitative Results for ase 4 SYM 2 136

5.8 Quantitative Results for Case 4 SYM 3 138

5.9 Quantitative Results for Case 4 SYM 5 140

5.10 Quantitative Results for Case 4 SYM 5 142

5.11 Quantitative Results for Case 4 SYM 6 144

5.12 Quantitative Results for Case 4 SYM 7 (Q1) 145

5.13 Quantitative Results for Case 4 SYM 7 (Q2) 147

5.14 Quantitative Results for Case 4 SYM 7 (Q3) 149

5.15 Quantitative Results for Case 4 SYM 7 (Q4) 150

5.16 Quantitative Results for Case 4 SYM 7 (Q5) 151

5.17 Summary of SYM7 Findings 153

Chapter 1

Introduction

Functional programming has existed since the 1950s, with the first functional language being LISP (Turner 2012). Functional programming is a programming style based on composing mathematical functions that, when given an input, evaluates an expression to generate an output. Functional programming emphasizes simplicity and genericity (Hughes 1989). In this paradigm, functions are treated as first-class entities, meaning they can be created, passed as parameters, returned as outputs, and stored in data structures—just like any other value. The functional paradigm avoids any kind of state mutation. A variable, once assigned a value, never changes, ensuring that it always refers to the same value. This eliminates side effects. Calling a function produces only a result with no side effects that could alter the value of an expression. As a result, expressions can be evaluated at any time, making the order of execution irrelevant and relieving the programmer of the burden of controlling execution flow.

Functional programming has grown popular over the years, with mainstream languages such as C++, Java, and C# adopting more and more of its features. C# supports lambda expressions, higher-order functions, type inference, and more. Java 8 welcomed the programming style by releasing many features that support the programming style (Saumont 2017). Features (to name a few) include support for lambda expressions and higher-order functions. The popularity of functional programming can also be observed through its presence in both conferences in academia and the industry. Many companies have adopted functional programming for software development; for example, Facebook uses Haskell for spam filtering, and Intel applies functional programming in chip design (Hu, Hughes, and Wang 2015). Functional programming is also used for financial contracts (see, for example, Jane Street Capital); WhatsApp, Twitter, Foursquare, LinkedIn, Tumblr, Klout, and

Google all somehow use functional programming concepts to achieve specific tasks (Hu, Hughes, and Wang 2015).

Despite its increasing adoption, a critical challenge persists: the absence of a standardized modeling notation that effectively captures functional systems. This challenge might pose a substantial challenge for both developers and business analysts interested in adopting functional programming. From a developer's perspective, it adds complexity to the task of designing and implementing functional systems effectively. For business analysts, it introduces difficulties in evaluating the potential risks and benefits of embracing functional programming, as well as making informed decisions about the when and how of its adoption.

Motara (2021) addressed this research gap by developing a structural modeling notation for typed functional programming using the Physics of Notations (PoN). The PoN framework is a rigorous approach to designing graphical notations, emphasizing perceptual discriminability, semiotic clarity, and cognitive fit. It aims to create notations that are both expressive and easily interpretable by their intended audience. The Physics of Diagrams (PoD) (Pissierssens, Claes, and Poels 2019) framework presents an alternative approach by focusing on diagram understanding and graphical parsimony. PoD is grounded in empirical research, consolidating 81 guidelines and 34 underlying propositions into seven foundational principles for effective graphical representation design. Unlike PoN, which emphasizes the design of notation languages, PoD is concerned with optimizing diagram representations to enhance interpretability and usability.

To address this gap, we introduce a typed modeling notation for functional programming, leveraging the Physics of Diagrams (PoD) framework. Our approach is designed to improve the expressiveness, usability, and accessibility of functional system representations. By applying PoD's principles, we aim to create a notation that enables developers to effectively design, modify, and reason about functional architectures, while also allowing business analysts to better evaluate the implications of functional programming adoption. This research bridges theoretical advancements in notation design with practical applications, contributing to a more structured and comprehensible representation of functional programming concepts.

1.1 Problem statement

While PoD provides a structured approach to diagram design, its application to domain-specific notations for functional systems remains underexplored and unvalidated in real-world contexts. The absence of a standardized and accessible notation for functional systems continues to hinder effective communication and system comprehension. This research aims to validate PoD's applicability to functional system modeling, ensuring that it enhances both technical accuracy and cognitive usability. Our goal is to bridge the gap between theoretical notation design and practical usability, providing a notation that is both rigorous and adaptable for developers, business analysts, and other stakeholders.

1.2 Research question

- Can a typed modeling notation for functional programming, designed using the Physics of Diagrams framework, improve the clarity, usability, and accessibility of functional system representations for both developers and business analysts?

Modeling and notation play a crucial role in software development, enabling developers and analysts to represent, analyze, and communicate complex system architectures. A well-structured notation provides a foundation for understanding both existing software systems and the design of new ones. Functional programming, with its emphasis on immutability, higher-order functions, and referential transparency, presents a unique challenge for traditional modeling approaches.

The effectiveness of a modeling approach is determined by its notation—the system of symbols and conventions used to represent structures and relationships within a system. Historically, widely adopted modeling notations, such as the Unified Modeling Language (UML), have been tailored primarily for object-oriented systems. UML's design revolves around objects, their interactions, and state-based representations, making it better suited for paradigms where mutable state and encapsulation are key concerns. However, functional programming diverges fundamentally from object-oriented principles, leading to a mismatch between the paradigm and the available modeling tools.

Given these limitations, there is a pressing need for a modeling notation that accurately captures the structural and behavioral characteristics of functional systems. Prior work, such as Motara's structural modeling notation, leveraged the Physics of Notations (PoN)

framework to enhance graphical notation design through principles such as perceptual discriminability and cognitive effectiveness. While PoN has advanced notation design significantly, it primarily focuses on notation languages rather than the interpretability and usability of diagrammatic representations.

This research explores an alternative approach, leveraging the Physics of Diagrams (PoD) framework to develop a typed modeling notation specifically for functional programming. Unlike PoN, PoD provides a structured, empirical basis for optimizing diagrammatic representations, emphasizing clarity, interpretability, and cognitive efficiency. The goal is to create a notation that enhances clarity by making the structures and transformations in functional systems more intuitive. Improves usability by ensuring that the notation is practical and aligns with the mental models of developers and analysts. Increases accessibility by reducing cognitive load and facilitating effective communication between technical and non-technical stakeholders. By grounding the notation design in PoD's principles, this research aims to bridge the gap between functional programming and effective system modeling, addressing the limitations of existing notations while improving the broader adoption and understanding of functional architectures.

1.3 Research objectives

The overarching goal of this research is to **design a structural modeling notation for functional programming** that effectively represents the components and relationships within such systems. To achieve this, the following objectives have been defined

- **RO1 Design a Structural Modeling Notation for Functional Systems Using the Physics of Diagrams (PoD) Framework** Develop a structural modeling notation that operationalizes the seven principles of the Physics of Diagrams (PoD) to enhance cognitive fit, semiotic clarity, and perceptual discriminability. The notation should prioritize simplicity, usability, and effectiveness in representing type definitions, transformations, and hierarchical relationships within typed functional systems, while ensuring alignment with the cognitive and perceptual needs of diverse stakeholders.
- **RO2 Applying the Notation to FParsec for Parsing System Representation:** Analyze the notation's effectiveness in representing the structure of FParsec, an F# parser combinator library. And evaluate its ability to represent modularity,

compositionality, and error-handling workflows. Measure participant comprehension of parsing logic and symbol relationships to identify strengths and areas for improvement.

- **RO3 Evaluating the Notation's Adaptability to Non-Functional Systems:** Investigate how the notation applies to Docutils, a Python-based system, and determine whether it maintains clarity and effectiveness outside functional programming environments. This includes assessing its usability in modeling system components and their interactions. Compare results with R2 to assess cross-paradigm applicability.

- **RO4 Assessing the Practicality of the Notation in Representing Pandoc's Architecture:** Evaluate how well the proposed notation captures the structural and functional aspects of Pandoc, a Haskell-based document converter. This includes analyzing whether the notation effectively represents type transformations and their relationships. Assess whether the notation effectively represents type transformations, input/output workflows, and hierarchical relationships, as measured by participant understanding and task performance in the survey.

- **RO5 Assessing User Understanding and Interpretation of the Notation's Symbols and Diagrams:** Conduct an empirical evaluation of how users interpret and understand the notation's symbols, structures, and diagrams. This includes assessing clarity, ambiguity, and effectiveness in conveying system relationships and transformations. Measure participant recall, recognition, and ease of interpretation to identify potential refinements in symbol design and overall diagram comprehension. Compare the new notation with Motara's notation to evaluate relative strengths, weaknesses, and areas for improvement.

1.4 Structure

The remainder of this thesis is structured as follows. Chapter 2, the literature review, explores the historical and theoretical foundations of functional programming, examines existing modeling languages, discusses the role of system design and modeling, and introduces HLO, a structural modeling notation for functional systems. Chapter 3, the methodology, outlines the principles of the Physics of Diagrams, details the design process for determining and combining symbols for the new notation, and explains the evaluation

design of the research. Chapter 4, the implementation, presents the notation, various system models, case studies, and the survey setup. Chapter 5 reports the survey results and interprets these findings in relation to the research objectives. Finally, Chapter 6 provides the conclusion, summarizing key insights and outlining future research directions.

Chapter 2

Literature review

This chapter explores functional programming, its historical development, and its theoretical foundations. It examines existing modeling languages and their limitations in representing functional systems. Additionally, it discusses the role of system design and modeling, focusing on the key components involved in the modeling process to achieve model abstraction. Furthermore, the chapter introduces a structural modeling notation for typed functional programming, examining its theoretical foundations and how the key components of the modeling process contribute to its design.

2.1 Functional Programming

2.1.1 Historical and Theoretical Foundations

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical expressions (Kühne 1999). In this paradigm, everything is regarded as an expression, a fundamental unit of computation that represents a value or an operation that, when evaluated, produces a value (Pollak, Layka, and Sacco 2022). Expressions can be as simple as variables or constants or as complex as functions, operators, or nested combinations of these components (Church 1985). This emphasis on expressions is a defining feature of functional programming, setting it apart from imperative programming, which often relies on statements that do not directly yield a value but instead alter the program's state.

The central role of functions in this paradigm reflects its mathematical roots. Functions in functional programming are conceptually aligned with mathematical functions, which define mappings from inputs to outputs. Unlike subroutines in many programming languages that might involve side effects or state manipulation, functions in the functional paradigm are "pure," meaning they produce the same output for the same input and do not cause side effects. For instance, a mathematical function $f(x) = x^2 + 3x + 2$ always produces the same result when x is provided, regardless of external factors. This predictable behavior contrasts with operations like reading the next line of a file, which may yield different results on successive calls depending on the file's state (Goldberg 1996).

The absence of side effects has significant implications for program design and correctness. Functions that operate without altering global variables, writing to files, or modifying external states can be reasoned about in isolation. This modularity enables developers to use the function call chain as a clear representation of data flow and as a tool for verifying the correctness of their programs (Hinsen 2009). According to Goldberg (1996), functional programming languages are designed to reflect the way humans think mathematically, focusing on abstract reasoning and problem-solving rather than the intricacies of machine-level operations.

A closely related concept in functional programming is immutability, which ensures that once a value is assigned to a variable, it cannot be modified. In contrast to imperative programming, where variables often represent mutable states, functional programming uses immutable data structures as the default. For example, appending an element to a list in a functional language does not alter the original list but instead creates a new list containing the additional element. This design choice not only prevents inadvertent changes to shared data but also enhances program reliability, especially in concurrent or parallel environments where mutable states are prone to race conditions and other synchronization issues (Mertz 2015).

The theoretical foundation of functional programming lies in lambda calculus, a formal system introduced by Alonzo Church in the early 1930s. Church developed lambda calculus as part of his efforts to establish a foundation for mathematics, presenting it in The Calculi of Lambda Conversion (Church 1985). Lambda calculus provides a minimalistic yet powerful framework for defining functions and their application, forming the conceptual basis for functional programming. In 1936, Stephen C. Kleene demonstrated the equivalence between the λ-definable numeric functions of lambda calculus and the partial recursive functions, establishing lambda calculus as a universal computational system (Rojas 2008). A year later, Alan Turing's work further solidified this universality by

proving that lambda calculus and Turing machines define the same class of computable functions, bridging the gap between theoretical computation models (Csörnyei and Dévai 2007).

One of the fundamental principles derived from lambda calculus is purity, a property where functions avoid side effects entirely. Pure functional languages are those that enforce this principle, ensuring that all operations within the language are referentially transparent (Mertz 2015). Referential transparency guarantees that a function, given the same input, will always produce the same output, irrespective of when or where it is called. This property is particularly important for reasoning about program behavior, as it allows developers to substitute function calls with their corresponding results without altering the program's meaning (Mueller 2019). For instance, in an expression x=5+3, referential transparency ensures that x can always be replaced by 8 without affecting the program. This contrasts with imperative programming, where variables may be reassigned or depend on external state, making the behavior of the program harder to predict. The absence of side effects in pure functional programming simplifies debugging and testing by ensuring that functions can be analyzed in isolation, as their behavior remains consistent and unaffected by external states or timing (Kühne 1999; Mueller 2019).

Lambda calculus itself embodies purity, with functions defined as mappings from inputs to outputs. Functions in lambda calculus can be anonymous, meaning they do not require names. For example, while a mathematical function might be written as $f(x) = x^2+3x+2$, the same function can be represented in lambda notation as $\lambda x.x^2 + 3x + 2$, where λx indicates that x is the parameter. This form, known as a lambda expression, is both concise and flexible, enabling functions to be defined and manipulated without assigning them explicit names (Baker-Finch 2013).

Since functions are anonymous in lambda calculus by default, they can be directly used as expressions without requiring a name. This flexibility allows lambda terms to be treated like any other value, such as numbers or variables. This property leads to a fundamental concept in functional programming: higher-order functions. A higher-order function is one that takes another function as input, returns a function as output, or both. Lambda calculus inherently supports higher-order functions because it treats functions as first-class citizens, meaning they can be passed around and manipulated like other expressions.

The concept of higher-order functions naturally introduces currying, another important aspect of functional programming. In lambda calculus, functions strictly take in one argument and return one value. However, by leveraging higher-order functions and currying, a

function of multiple arguments can be transformed into a chain of nested functions, each taking a single argument. For example, the lambda term $\lambda x.\lambda y.(x\ y)$ represents a curried function, which accepts two inputs sequentially—one at a time. Currying simplifies function composition and reuse, making it a powerful tool in both theoretical and practical applications of functional programming (Reynolds 1972).

Lambda calculus can be divided into two primary forms: untyped and typed. The untyped lambda calculus, the original form introduced by Church, imposes no restrictions on the types of terms, allowing any expression to serve as input to any function (Csörnyei and Dévai 2007). Its syntax consists of variables, applications, and abstractions, with terms recursively defined as follows (Barendregt 1997; Christoph 2022):

$$\text{var} = a \mid \text{var}'$$
$$\text{term} = \text{var} \mid \text{term term} \mid \lambda\ \text{var term}$$

This unrestricted flexibility enables powerful abstractions but can lead to issues such as nonsensical operations. For instance, applying a function designed for numbers to a string may produce undefined behavior. To address these challenges, the typed lambda calculus was introduced in the 1940s. By associating types with variables and expressions, the simply typed lambda calculus restricts the formation of invalid terms, ensuring greater safety and predictability in computations (Csörnyei and Dévai 2007). Types are constructed from type variables and function types, as represented by the grammar below (Barendregt 1997):

$$\text{tvar} = \alpha \mid\ \text{tvar}'$$
$$\text{type} = \text{tvar} \mid\ \text{type}\ \rightarrow \text{type}$$

The inclusion of types not only prevents errors but also facilitates reasoning about program behavior. In the simply typed lambda calculus, A→B→C is interpreted as A→(B→C), reflecting the right-associative nature of function types and supporting concepts like currying (Christoph 2022).

Functional programming also incorporates inductive types and algebraic data types for modeling complex data structures. Inductive types allow recursive definitions, enabling the creation of structures like lists and trees. These types facilitate structural recursion, where functions are defined by cases corresponding to the structure of the data (Faraldos 2023). Algebraic data types combine sum types (representing mutually exclusive possibilities) and product types (representing combinations of values), offering a flexible

framework for defining and manipulating structured data (Pepels, Plasmeijer, and Proper 2006; Pierce 2002).

Another significant concept in functional programming is evaluation strategy. Functional languages can employ eager (strict) evaluation or lazy (non-strict) evaluation (Christoph 2022; Mueller 2019). In eager evaluation, all terms in an expression are fully evaluated before the function executes, potentially wasting computational resources on terms that are ultimately unused. Conversely, lazy evaluation defers computation until the value is needed, allowing programs to handle infinite data structures and optimize resource usage (Mueller 2019). For instance, consider the expressions $F = \lambda a.aaa$ and $A = (\lambda x.C)abcdefghi$. Under eager evaluation would proceed as follows:

$$\begin{aligned} FA &= (\lambda a.aaa)((\lambda x.C)abcdefghi) \\ &\rightarrow_\beta (\lambda a.aaa)C \\ &\rightarrow_\beta CCC \end{aligned}$$

F A reduces to CCC after fully evaluating A. In contrast, lazy evaluation defers the computation of A, performing reductions only when A's value is explicitly required (Christoph 2022):

$$\begin{aligned} FA &= (\lambda a.aaa)((\lambda x.C)abcdefghi) \\ &\rightarrow_\beta ((\lambda x.C)abcdefghi)((\lambda x.C)abcdefghi)((\lambda x.C)abcdefghi) \\ &\rightarrow_\beta CCC \end{aligned}$$

In conclusion, this exploration has traced the development and core concepts of functional programming, from its theoretical foundations in lambda calculus to its practical applications in modern software development. As a foundational model of computation, lambda calculus, while historically challenging to implement directly in hardware compared to Turing machines (Christoph 2022), has profoundly influenced programming language research, informing the design of type systems in languages like Haskell and ML and even providing a framework for understanding object-oriented paradigms (Baker-Finch 2013). Though not programming languages themselves, lambda calculus and Turing machines serve as prototypes, offering abstract models for computation (Christoph 2022). While the shift to functional programming requires a significant learning investment and a departure from the imperative model prevalent in current hardware architectures (Hinsen 2009), the benefits are substantial. Functional programs are recognized for their robustness, testability (Hinsen 2009), and suitability for parallelization due to the absence of

side effects, aligning well with the trend towards multi-processor and massively parallel hardware (Kühne 1999). Although the mathematical foundations, particularly lambda calculus, present a learning curve, the concise and mathematically expressive nature of functional languages enables a stronger focus on algorithmic design, model building, and experimentation (Goldberg 1996). This ultimately results in succinct, robust, and high-performance code (Kühne 1999) that aligns more closely with human mathematical reasoning than with machine architecture (Goldberg 1996). This makes functional programming a valuable approach for both theoretical exploration and practical software engineering challenges.

2.2 Review of existing modeling languages

Flowcharts stand as one of the earliest modeling notations, dating back to the 1940s (Sandvig 1942). They serve as visual representations of the logical structure of computer programs and processes, using symbols and arrows to illustrate the flow of control and data within them. Traditionally, flowcharts found their primary applications in algorithm design and process visualization (Ensmenger 2016).

In algorithm design, a flowchart functioned as a design blueprint for a computer program. Analysts would examine problems, devise algorithmic solutions, and create a flowchart diagram that outlined the algorithm. These flowcharts were subsequently translated into machine code by programmers. As a central design document, the flowchart provided a visual depiction of the program's structure and logic.

Flowcharts also played a crucial role in process visualization, enabling people to comprehend and communicate complex systems. They served as invaluable tools to solve problems, analyze systems, plan projects, and develop solutions, allowing for the anticipation, analysis, and resolution of various challenges.

Functional programming, despite its simplicity, power, and practical utility, often faces challenges related to the visual presentation of programs (Mitkin 2012; Weck and Tichy 2016). The nature of functional programming, which heavily relies on recursion and higher-order functions, can lead to code structures that may be less intuitive for those accustomed to imperative programming paradigms (Wallingford 2002; Weck and Tichy 2016). Additonally, the human cognitive process, when dealing with textual representations of algorithms, can pose challenges, as the inherent nature of the human brain is not inherently compatible with parsing textual information (Mitkin 2012). Therefore,

traditional text-based notation can sometimes hinder the readability and intuitiveness of functional programs. This underscores the need for exploration of alternative visualization methods, such as graphical languages like DRAKON, that leverage our visual processing strengths.

Nonetheless, it's important to acknowledge that flowcharts have not been exempt from challenges. As the complexity of algorithms increased, flowcharts grew in size and complexity, often resulting in cluttered representations (Ensmenger 2016; Mitkin 2012). This clutter could render flowcharts difficult to difficult to read and understand, even for experienced developers. Despite this limitation, flowcharts have persisted in the realm of computer programming, albeit with adaptations and improvements. These adaptations include the development of structured flowcharts (Grouse 1978; Nassi and Shneiderman 1973), Data Flow Diagrams (Tao and Kung 1991), and the development of graphical programming tools that generate and execute code based on flowcharts, such as Flowgorithm, Raptor, and DRAKON (Balać and Vidaković 2017).

DRAKON is a visual language optimized for ergonomics, specifically designed to improve the readability of diagrams and algorithms. It offers simple yet effective rules that significantly enhance the clarity and understandability of the visual representation of programs. Examples of some rules are:

- Clarity Over Space: his rule emphasizes that it is better to sacrifice space than readability, ensuring that all important aspects of an algorithm are immediately visible.
- No Hidden Paths: DRAKON ensures that all important things are visible, eliminating any hidden paths in the visual representation of the algorithm. This rule prevents ambiguity and ensures that the flow of the program is transparent.
- The Further to the Right - The Worse: This rule adds an additional dimension to the algorithm, making it easier for the reader to distinguish between good and not-so-good paths. It provides a visual cue to immediately see how badly things have gone in a particular icon based on its position on the diagram.
- No Redundant Visual Details: DRAKON eliminates redundant visual details, ensuring that the visual representation of the algorithm is clear and uncluttered. This rule helps in focusing on essential elements, enhancing the overall readability.

DRAKON's focus on human visual habits and its ergonomic design ensures fast and easy understanding of algorithms, additionally, it stands out by offering a significant

```
feed(Creature) ->
    case Creature of
        {person, FirstName, SecondName, _} ->
            give_money(FirstName, SecondName);
        {animal, Name} ->
            give_food(Name)
    end.
```

Figure 2.1: Example of pattern matching for branching (reproduced from (Mitkin 2012)

improvement over traditional flowcharts, making it an excellent choice for improving the visual appearance of functional programs.

Notably, DRAKON, as cited in (Mitkin 2012), can be described as flowcharts optimized for ergonomics. Mitkin (2012) incorporates DRAKON with the functional programming language Erlang, yielding DRAKON-ERLANG, aimed at enhancing the visual clarity of functional programs.

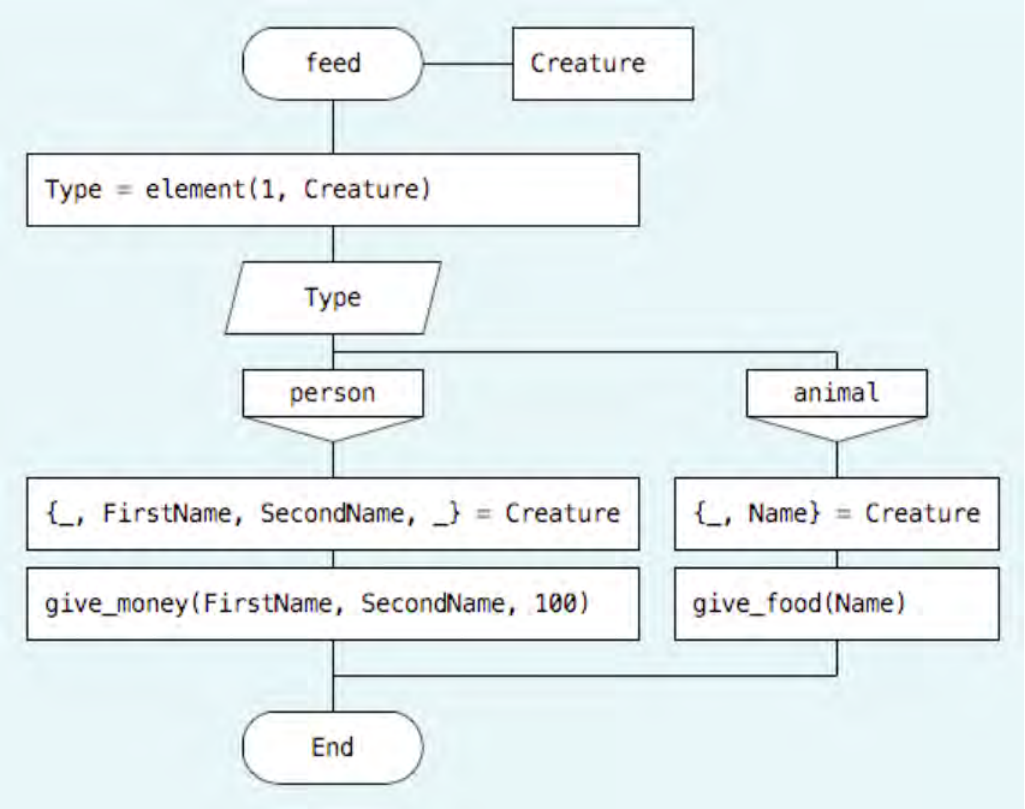


Figure 2.2: pattern matching in Drakon-Erlang (reproduced from (Mitkin 2012)

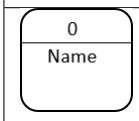
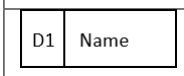
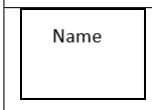
Erlang (Armstrong 2010), as the chosen functional programming language for integration with DRAKON, offers several advantages that make it a suitable candidate for this purpose. Erlang is well-known for its simplicity, reliability, and built-in support for concurrent

programming (Virding, Wikström, and Williams 1996). Its design from the beginning to facilitate concurrent programming aligns with the modern trend of increasingly prevalent multi-core processors and the growing importance of distributed computing and clustering. These features, combined with DRAKONs visual clarity, make DRAKON-Erlang a promising technology for enhancing the visual representation of functional programs. Figure 2.2 shows how the code in figure 2.1 might be represented in Drakon Erlang.

While flowcharts excel at representing processes, a significant challenge emerges when applying them to functional systems: the difficulty of depicting recursion (Roy, Kelso, and Standing 1998). Recursion, a fundamental concept in functional programming, can pose challenges when rendered in flowchart form.

One of the widely embraced business modeling techniques, emphasizing data and information systems at different abstraction levels, is the Data Flow Diagram (DFD). DFDs offer a distinct perspective by placing less emphasis on control flow and focusing on the flow of data through various functions and processes. Comprising elementary building blocks like processes, data flows, data stores, and external entities, DFDs visually represent the movement of data from one point to another. Each building block possesses a graphical notation, as detailed in Table 2.1.

Table 2.1: Graphical Notation for DFD Components (Dennis, Wixom, and Roth 2008)

Symbols	**Element Name**
	Process
	Data Flow
	Data Store
	External Entity

According to (Ibrahim et al. 2010), a process signifies an activity or function performed for a specific business purpose, a data flow represents a single piece of data or a logical collection, a data store is a repository of stored data, and an external entity is an external person, organization, or system interacting with the system. A critical insight is provided: a process lacking input data flows would be incapable of receiving data and, consequently, unable to execute any work.

An example of a DFD diagram is illustrated in figure 2.3, showcasing processes that

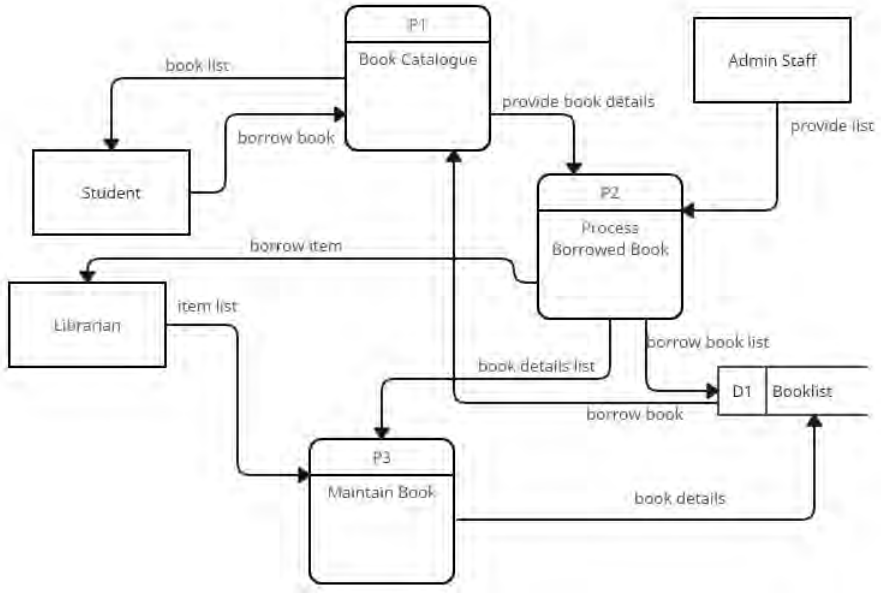


Figure 2.3: Example of a data flow diagram (modified from Ibrahim et al. 2010)

transform input data into output data, data stores housing information, and data flows connecting these elements. DFDs excel in presenting the information flow within a system and prove to be comprehensible and verifiable, often used in discussions between analysts and users. They can be further dissected into lower-level DFDs for detailed insights (Aguilar-Savén 2004). There are a set of rules that must be followed by analysts when constructing DFDs. Some examples are (Ibrahim et al. 2010):

- At least one input or output data flow for external entity: Each external entity in the DFD must have at least one input or output data flow, ensuring that the system interacts with external entities.
- At least one input data flow and/or at least one output data flow for a process: Processes in the DFD must have at least one input data flow, one output data flow, or both, representing the flow of data into and out of the processes.
- Output data flows usually have different names than input data flows for a process: This rule emphasizes the need for distinct naming conventions for input and output data flows of a process, ensuring clarity and differentiation.
- Data flows only in one direction: Data flows in the DFD should have a unidirectional flow, representing the movement of data from a source to a destination.
- Every data flow connects to at least one process: Each data flow in the DFD must

be connected to at least one process, depicting the processing of the data within the system.

- Unique name in data flow diagrams: Each element in the DFD, including processes, data flows, data stores, and external entities, must have a unique name and a description, ensuring clarity and differentiation.
- Every set of data flow diagrams must have one context diagram: A set of data flow diagrams must include a context diagram to provide an overview of the entire system
- Balancing: Every data flow, data store, and external entity on a higher-level DFD must be shown on the lower-level DFD that decomposes it

In their most basic form, DFDs provide a limited view of a system because, unlike flowcharts, they do not explicitly depict the control flow of a system. DFDs focus more on the transformation of data as it moves through various processes, rather than the specific sequences of actions or decisions that dictate this transformation. To overcome these limitations and provide a more comprehensive view of the system, DFDs are often combined with other modeling tools and techniques, such as data dictionaries, control flow diagrams, state transition diagrams, decision tables, and mini-specifications (Larsen, Plat, and Toetenel 1994). A Data dictionary is a centralized repository or document that provides detailed information about the data used in a system. It serves as a reference for data elements, their characteristics, relationships, and other relevant details (DRAFT 1989). Control flow diagrams are a visual representation of the sequence of execution of instructions in a computer program. They illustrate the flow of control from one instruction to another, typically using nodes to represent instructions and directed arcs to represent the flow of control between them. These diagrams are useful for understanding the logic of a program and are commonly used in software design and maintenance (Gustafson 1981). State transition diagrams are a graphic notation used in software engineering and system design to represent the different states of an object or system and the transitions between those states. They consist of nodes representing states of a system and arrows representing transitions between states. State transition diagrams have been widely used to model complex software systems and have played a major role in hardware design (Desharnais, Frappier, and Mili 1998). Decision tables are a tabular method used to represent complex logic and decision-making processes. They consist of segments for conditions and actions, allowing for a clear display of the relationships between different conditions and the corresponding actions. Decision tables are concise, easily interpreted,

and can readily accommodate changes, making them an effective tool for analyzing and documenting business problems and their solutions (Fisher 1966). Mini specifications are smaller documents that describe a part of a system instead of the entire system. They are used to gradually elaborate on the details of requirements in iterative development or enhancement projects. These documents are considered easy to handle, but challenges arise because a system may have many such specifications, which are interrelated (Liskin 2015).

DFDs align well with functional programming principles, which emphasize data manipulation through function composition. Notably, (Weck and Tichy 2016) explores an approach to automatically transform Haskell programs into data-flow diagrams enriched with type information. The visualization, rooted in category theory and lambda calculus, aids in understanding function interactions and data processing, supporting various use cases. However, it primarily focuses on transforming existing Haskell programs, leaving a gap in exploring DFDs for visualizing new systems before the coding phase.

An alternative widely-used method for modeling business processes, recognized as the industry standard in process modeling (Kocbek et al. 2015) and rooted in conventional flowcharting techniques (Rosing et al. 2015), is Business Process Modeling Notation (BPMN).

BPMN offers a richer set of semantics compared to DFD, and other modelling techniques, enabling the representation of a wide range of control flow patterns and sequences (Aldin and De Cesare 2009), making it a valuable tool for modelling complex processes.

A key advantage of BPMN is its ability to bridge the communication gap between business process design and implementation, serving as a common language that can be readily understood by all business stakeholders, including business analysts, technical developers, and business managers (Aldin and De Cesare 2009; Rosing et al. 2015). It supports execution semantics, mappings to other languages, and the construction of simulation models, enabling process testing and visualization prior to implementation (Aldin and De Cesare 2009; Rosing et al. 2015).

BPMN's versatility stems from its comprehensive range of notations, which cater to various modeling needs. These notations encompass different diagram types, each dedicated to representing specific facets of business processes (Rosing et al. 2015). Fig. 2.4 provides an example of a BPMN diagram. The inherent flexibility of BPMN enables the development of diverse business process models, further enhanced by its extensive support among over sixty vendors (Decker et al. 2009).

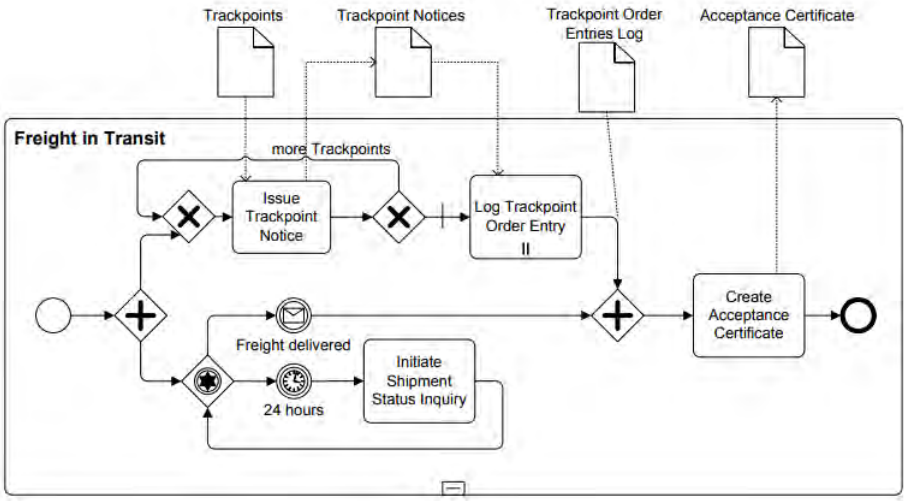


Figure 2.4: Example of a business process model with data in BPMN (reproduced from (Decker et al. 2009))

As illustrated in Fig. 2.4, data manipulation within a business process can be specified using data objects in BPMN. These data objects serve as placeholders for various types of information, including documents, data structures, and other relevant materials. Each object is identified by a name and may optionally carry a state indicative of its current status. Data objects can be linked to activities, indicating their role as inputs or outputs of those tasks. Additionally, they can be associated with flows, illustrating their transfer between activities (Decker et al. 2009). While data objects provide a means to represent the types of messages involved in a process, they lack the ability to capture the detailed content and structure of those types. This limitation makes BPMN less suitable for typed functional programming languages, that place a strong emphasis on data types and their precise definition.

2.3 UML and its shortcomings in representing functional systems

Unified Modeling Language (UML) stands as a powerful tool in the software development landscape, widely recognized for its utility in visualizing and documenting software systems. However, its prevalence in the industry as the de facto modeling language may not seamlessly extend to functional programming systems. In this section, we delve into the challenges that arise when applying UML to represent the unique characteristics of functional programming.

UML consists of a collection of diagrams that can be used to provide multiple views of the system under design. These diagrams are categorized into two main groups: structural diagrams and behavioral diagrams. Structural diagrams (Class diagrams, Object diagrams, Package diagrams, etc.) are designed to capture the static aspects of systems, such as the classes, objects, and their relationships. They provide a snapshot of the system's structure at a specific point in time. Behavioral diagrams (Use Case diagrams, Sequence diagrams, Activity diagrams, etc.) show the dynamic nature of systems, illustrating how system components interact and behave over time. They capture the system's behavior in response to various events and stimuli.

UML's design, rooted in the object-oriented paradigm, encounters challenges when attempting to model systems adhering to functional programming principles. The object-oriented emphasis on objects and relationships stands in contrast to the fundamental tenets of functional programming.

One of the fundamental challenges stems from UML's lack of explicit support for key functional programming constructs (Alam 2015). Notably absent is notational support for parametrically polymorphic functions or values, a cornerstone of many functional languages. Polymorphism is a crucial concept in functional programming, enabling functions to operate on data of different types without requiring type-specific code. UML's lack of explicit support for parametric polymorphism makes it challenging to represent such polymorphic functions. In object-oriented design, methods are encapsulated within classes, leading to the representation of functions inside classes in UML. This requirement to place functions inside classes may not align with the functional programming paradigm, where functions are typically standalone and can be passed around as values.

Functional programming's emphasis on higher-order functions and currying, techniques that enhance flexibility and composability, faces a roadblock in UML. The language's lack of native support for specifying higher-order or curried functions hinders the accurate representation of the functional programming paradigm, making it challenging to model systems where these concepts are prevalent.

An important feature of functional programming languages is that functions are treated as first-class citizens, allowing them to be assigned to variables, passed as arguments, and returned as values. Representing functional features as first-class citizens clearly in a UML class diagram can be problematic (Bijlsma et al. 2020).

UML's division between static and dynamic models introduces complexity when applied to

functional programming. The dynamic model, designed to express system behavior over time, relies on changes in system state—a concept foreign to functional programming. In a pure functional subset, where explicit state or fixed execution order is absent, UML faces difficulties in effectively modeling dynamic aspects.

The use of UML for functional programming sparks ongoing debate within the programming community. The clash between UML's foundational principles, rooted in mutable states and object relationships, and the stateless, function-centric nature of functional programs gives rise to differing opinions on its appropriateness (Alam 2015).

Despite the challenges, some have explored the adaptation of UML-style diagrams for functional programming (Bijlsma et al. 2020; Szlenk 2011; Wakeling 2001). Yet, prevailing conclusions assert that UML, with its inherent object-oriented focus, is not well-suited for functional programming development (Alam 2015).

2.4 The role of system design and modeling

There is no doubt that modelling is one of the the most important aspects of the human mind, enabling us to comprehend and interact with the world around us. As eloquently stated by Rothenberg (Rothenberg et al. 1989), "Modeling underlies our ability to think and imagine, to use signs and language, to communicate, to generalize from experience, to deal with the unexpected, and to make sense out of the raw bombardment of our sensations." Modeling is our way of comprehending the world around us. There is no doubt that modelling is one of the the most important aspects of the human mind, enabling us to comprehend and interact with the world around us. As eloquently stated by Rothenberg (Rothenberg et al. 1989), "Modeling underlies our ability to think and imagine, to use signs and language, to communicate, to generalize from experience, to deal with the unexpected, and to make sense out of the raw bombardment of our sensations." Modeling is our way of comprehending the world around us.

To distinguish models from other artifacts, Stachowiak proposes three essential features (Stachowiak 1973): the Mapping feature, where a model is based on an original system; the Reduction feature, specifying that only the relevant parts of the system should be present in the model; and the Pragmatic feature, emphasizing that a model must serve a purpose.

Regarding the Mapping feature, the original system need not be physically present. Models often come in two flavors: descriptive, portraying existing systems, and prescriptive,

acting as specifications for creations. Interestingly, a model can transition between these roles as requirements evolve. The Reduction feature ensures that the model is a simplified version of the system, containing only the pertinent parts and allowing extension with additional features to enhance it. This may seem like a loss, making the model weaker, but, in fact, it renders it more powerful, focusing on what matters (Ludewig 2003) — a power known as abstraction. Abstraction allows us to grasp complex systems by simplifying and breaking them down into their component parts, helping us avoid the complexity, danger, and irreversibility of reality (Rothenberg et al. 1989). Lastly, the Pragmatic feature ensures that models are not merely abstract constructs — it ensures that a model is purposeful; otherwise, creating a model without a purpose would be futile.

Referencing Stachowiak's work, Kühne (Kühne 2006) introduces a formal equation, providing a mathematical representation of the modeling process:

$$\alpha = \tau \circ \alpha' \circ \pi$$

In this representation, Model abstraction (α) involves a projection (π) of the original onto the model, some further abstraction (α') , and a translation (τ) to another representation.

Kühne further categorizes models based on their relationship to the original system:

- Token models: "Elements of a token model capture singular (as opposed to universal) aspects of the original's elements, i.e., they model individual properties of the elements in the system."
- Type models: "Most models used in model driven engineering are type models. In contrast to token models, type models capture the universal aspects of a system's elements by means of classification."

Token models involve no further abstraction beyond projection and translation. When creating a token model, the focus is on selecting specific properties of the original elements (projection) and representing them in a way that suits the modeling context (translation). This approach is particularly useful when the goal is to capture individual details of elements without introducing additional layers of abstraction beyond what is necessary for representation and understanding. Examples include maps in geography, blueprints in architecture, and molecular diagrams in chemistry.

Type models organize elements into types or categories, emphasizing similarities and common features. Creating a type model involves selecting specific aspects of elements,

translating them, and classifying elements into types based on shared properties. This results in a more condensed and organized representation of complex systems.

Models serve to represent both the structure (what it is) and behavior (what it does) of systems. In functional programming, the structure is often defined using immutable data types that describe entities and relationships within the system. The declarative style of functional programming, expressing what computations should be performed rather than how, aligns with modeling behavior. The emphasis on pure functions and immutability contributes to a clearer expression of behavior, making it easier to understand and reason about a system. While functional programming defines data structures and types, the primary focus is often on the functions and transformations operating on these data structures, placing more emphasis on modeling dynamic aspects.

In functional programming, types play a fundamental role in specifying the structure and characteristics of data. Strong and static type systems catch errors at compile-time and provide clarity about the expected structure of data. Types in functional programming can be expressive and capture intricate details of data structures. This expressiveness aligns with the detailed representation of elements in token models.

2.5 HLO (Hello)

This section introduces **HLO**, a notation developed by Motara in (Motara 2021) to model the structure of Typed Functional Programming (TFP). It explores the components of HLO (pronounced as Hello), its theoretical underpinnings, and the principles guiding its design. Motara (2021) emphasizes the importance of a robust philosophical and mathematical foundation for creating a modelling language that is both comprehensible and applicable across various domains. The research addresses the limitations of existing modelling approaches and proposes a structurally sound, type-focused notation applicable to diverse fields.

2.5.1 Philosophical Investigations (PI)

Wittgenstein's *Philosophical Investigations* (Wittgenstein 2009) provides a language-centric philosophical framework, emphasizing the contextual nature of language. Central to Wittgenstein's philosophy is the concept of **language-games**, which explain how meaning arises from the rules, actions, and purposes surrounding the use of language. For

instance, the word “bank” might mean a financial institution, a riverbank, or an aviation maneuver, depending on the context in which it is used.

In language-games, meaning is determined by use rather than isolated definitions. This inherent ambiguity is not a flaw but a feature of language that enhances understanding when viewed in context. Wittgenstein underscores the importance of names, which represent entities without being identical to them. For example, a name retains its meaning even if the entity it refers to no longer exists.

Motara (2021) identifies three key characteristics of ambiguity in PI. First, a word can have multiple meanings, each valid within its specific context. Second, a word may have an inexact meaning while remaining functional. Third, a word’s meaning can evolve as the language-game progresses. These principles of ambiguity align with both LoM and TFP. For example, the mathematical term “prime” might refer to prime numbers in arithmetic or prime elements in algebra, while in TFP, operations like ‘map’ or ‘fold’ might take on different meanings based on their context.

Philosophical Investigations is written as a series of numbered paragraphs offering an implicit structure, allowing for cross-referencing and logical flow.

2.5.2 The Language of Mathematics (LoM)

Mathematics is often described as a language that combines symbols and text to communicate ideas with precision and clarity. Its structured nature ensures logical consistency and facilitates the systematic organization of concepts. At its core, the Language of Mathematics (LoM) provides a framework to parse, understand, and represent arbitrary symbolic and textual expressions with full semantics, enabling a deeper comprehension of mathematical discourse (Ganesalingam and Ganesalingam 2013).

Mathematical discourse is divided into **blocks**, each with a distinct purpose. Definitions introduce terms or concepts, while lemmas, theorems, and propositions develop these ideas further, ranging from minor results to major conclusions. Corollaries follow as immediate consequences of a Lemma, Theorem, or Proposition. Cross-references link these blocks, creating a cohesive narrative that builds on foundational concepts.

Variables and definitions are central to mathematical language. Variables serve as placeholders for generalization and abstraction, while definitions provide the necessary context

for reasoning. Together, they form the backbone of mathematical communication. Mathematics operates in two modes: the **formal mode**, which comprises precise and computable statements like equations, and the **informal mode**, which includes commentary or insights providing additional context.

One of the defining features of mathematics is its **adaptivity**—the ability to refine and redefine meanings as new contexts emerge. For instance, the fraction $\frac{5}{8}$ might initially represent "five parts of eight" but later evolve to mean "five divided by eight." This adaptability parallels the evolving nature of TFP, where types and functions acquire new meanings as they are extended or composed.

2.5.3 Translating to HLO

The development of HLO draws from the principles of projection (π) and further abstraction (α') to identify and refine constructs for TFP modelling.

Projection involves identifying elements that can be directly mapped between LoM, PI, and TFP. For instance, formal and informal modes in LoM align with type annotations and comments in TFP. Similarly, definitions and variables in mathematics correspond to their functional equivalents in programming, making them foundational to HLO's design.

Further abstraction builds on these mappings, refining elements into standardized constructs. Mathematical rhetoric, for example, establishes logical relationships by linking blocks like definitions, theorems, and proofs. In TFP, types and functions serve similar rhetorical purposes, with product types grouping related data and sum types offering alternatives. However, TFP lacks explicit high-level rhetoric for connecting components into cohesive narratives.

Structured blocks and cross-referencing are another key aspect of HLO. Borrowing from LoM's block structures and PI's numbered paragraphs, block structures are created with the purpose of separating and allowing easy cross-referencing.

Variables and definitions are simplified in HLO to provide consistency and clarity. While TFP uses diverse representations for variables—such as function arguments, closed-over values, and type annotations—HLO unifies these into a simpler representation from LoM. Similarly, mathematical blocks such as lemmas, propositions, and theorems are merged into a single construct for simplicity, while corollary blocks are omitted due to their limited relevance in TFP contexts.

HLO's design follows the Physics of Notation Systematized (PoN-S) (Silva Teixeira 2017) design process to create a workable notation that adhered to the principles of good notation suggested by Moody (2009). The PoN-S process focused on improving the speed, ease, and accuracy with which a representation can be processed by the hu- man mind. It began by considering cognitive fit, determining the symbols to be used in the notation, and identifying legitimate ways in which symbols may be combined. The process also emphasized semiotic clarity (representing a one-to-one mapping relationship between visual syntax and semantics), semantic transparency (a diagram should employ a visual notation that provides unambiguous representations of its semantic constructs), perceptual discriminability (the ease with which individuals can visually distinguish and differentiate between elements or components within a visual display), and complexity management (use of a minimal number of symbol types in a diagram) of the notation .

HL0 was designed to allow a typed functional system's structure to be expressed, modified, and understood by a broad and general audience, including students, developers, and business analysts. This design process aimed to create a notation that was not only theoretically justified but also practical and accessible to a wide range of stakeholders.

In order to determine the relationships essential for the model's utility, Motara (2021) employs a representation inspired by dictionaries, thesaurus, and etymology. This involves using a set of symbols and relationships to depict elements and connections within the model.

For the dictionary notation, the focus is on addressing the question, "What words exist, and what do they mean?" A distinct shape the rub-'al-.hizb (pronounced roob-El-Hizb), is employed to encapsulate symbolic notation accompanied by expandable space for textual definitions (see Fig. 2.5). This notation ensures that definitions exclusively refer to previously established terms, creating a structured hierarchy where foundational definitions precede more advanced ones.

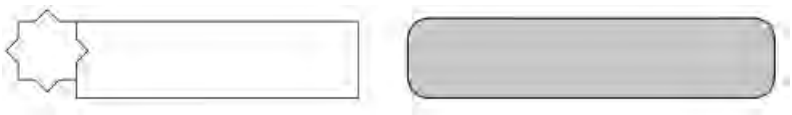


Figure 2.5: Notation for definition (left) and thesaurus (right)) (Motara 2021)

Conversely, the thesaurus notation (see Fig. 2.5) tackles the question "Which words are similar?" This notation begins with a bold title indicating the basis of similarity, followed

by a list of similar items accompanied by concise descriptions. These descriptions, prefixed with a colon, elucidate why each item belongs to the specified group. Notably, the thesaurus section follows all dictionary definitions.

The etymology notation, focused on unraveling a word's ancestry, utilizes symbols and lines to depict relationships between definitions. A diamond-terminated ◆—◇ line signifies a structural subset relationship, indicating an "alias" connection between definitions. Conversely, a circle-terminated ●—○ line denotes a "relies" relationship, highlighting a dependency where one definition builds upon another. These lines delineate etymological connections.

The following symbols are reserved in the notation:

- The symbol "►" is commonly used in textual definitions to denote a set of cases that are patterned based on the definition. These cases delineate specific scenarios or patterns within the definition, guiding how each scenario maps to its corresponding entity. This approach provides a concise and systematic way to represent conditional logic, functional mappings, or type structures. In the context of sum types, ► is typically used to separate and indicate the various cases or alternatives. A sum type, being a type that can hold one of several different values, benefits from this notation as it makes the distinctions between cases explicit and easy to understand. For product types, the "•" symbol is used to describe the components or members of a group. A product type aggregates multiple values into a single compound value, and this symbol visually represents their cohesive relationship. It succinctly indicates how different components combine to form a structured entity.

- The "→" symbol is employed to represent the mapping or transformation of one entity to another. In functional programming or similar contexts, it often signifies the transition from an input or pattern to an output or result.

- The "_" symbol is often employed as a placeholder for a fall-through case. This signifies a default scenario that captures any input or condition not explicitly handled by prior cases. It ensures robustness in definitions by accounting for unforeseen or edge cases.

- Parentheses, represented by "(" and ")", are integral for grouping and organizing expressions. They ensure clarity by explicitly delineating the scope or precedence of operations, particularly in complex expressions or nested structures.

- Subscripts are commonly utilized in sum types to distinguish between different cases or variants.
- The symbol ⚙ is used to represent 'some type or value'. This abstraction is particularly useful in the context of parametric polymorphism, where functions and data structures are designed to operate generically on any type.

2.5.4 Example

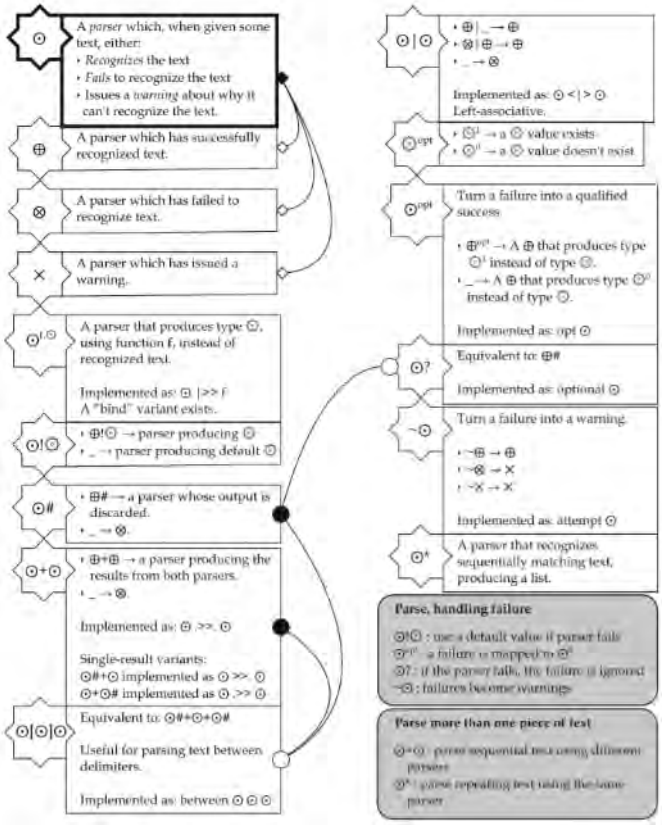


Figure 2.6: Example of a HLO diagram (Motara 2021)

Figure 2.6 presents a model of a system using the HLO notation. This section provides a brief walkthrough of the diagram to illustrate how the notation functions. The model represents FParsec, an open-source parser combinator library written in F#. The diagram is read from top to bottom, and left to right, beginning with the definition of a parser, which includes its associated symbol alias (⊙). This is followed by sub-definitions that represent more specific components of the parser, each introducing their own symbolic representations (e.g., ⊕, ⊗, ×). These sub-definitions extend the main parser and serve to modularize its functionality.

Subsequent definitions in the diagram demonstrate how these parser components are applied or composed to produce output parsers. These are illustrated by a set of symbolic definitions such as ⊙$^{f,\odot}$, ⊙!⊙, ⊙#, ⊙+⊙ ... etc, which build upon the earlier parser blocks. Definitions which rely on other definitions are indicated using a circle-terminated line; for example, the definition ⊙[⊙]⊙ relies on the definitions ⊙# and ⊙+⊙. And the definition ⊙? relies on the definition ⊙#. The diagram concludes with the two thesaurus notations shown in the bottom right, where definitions are grouped under labeled sections. Each group has a descriptive title and includes a list of relevant definitions, providing stakeholders with insight into the relationships among the definitions and helping them understand how different tasks can be achieved using the notation.

This research adopts a new set of guiding principles to develop a structural modeling notation aimed at improving the HLO framework. One key observation is that the original notation follows a staged structure: first defining fundamental building blocks, followed by sub-definitions that refine those blocks, then introducing definitions that manipulate or apply the fundamental components to generate new outputs. Finally, the diagram includes a thesaurus-like grouping of related definitions. To improve clarity, our proposed approach separates these stages explicitly into distinct visual groups:

- **Group A** – contains the fundamental definitions and their sub-definitions.
- **Group B** – contains definitions that make use of or manipulate those in Group A.
- **Group C** - contains thesaurus-style groupings that aid interpretation and task understanding.

This grouped approach has several advantages over the traditional method of combining all elements in a single, continuous diagram. By structuring the notation into clearly defined conceptual layers, it becomes easier for users to understand the flow of the system, reduce cognitive load, and avoid visual clutter—especially in complex models with many interdependencies.

Another observation relates to the use of symbolic aliases. As the model scales, the accumulation of symbols for both fundamental definitions (e.g., parser) and their respective sub-definitions can make it difficult for users to recall and differentiate between them. To address this, we propose strategies to reduce the number of distinct symbols introduced, thereby improving usability and supporting better comprehension and recall.

These examples illustrate early considerations for improving the notation. While preliminary, they suggest that even modest structural changes could significantly enhance how the notation is understood and utilized by users. Additional improvements are introduced and supported in the following chapters, each grounded in the design principles proposed in this research.

2.6 Conclusion

This literature review has systematically examined functional programming's theoretical foundations, existing modeling languages, and the principles of effective system design. Key findings reveal that while functional programming offers significant advantages—including mathematical rigor, robustness, and parallelism—its core concepts like higher-order functions, recursion, and immutability pose challenges for visual representation. Existing modeling languages (e.g., flowcharts, DFDs, BPMN, UML) exhibit limitations in capturing these functional abstractions due to their imperative or object-oriented biases.

The analysis underscores the critical role of modeling in achieving abstraction, emphasizing Stachowiak's criteria of mapping, reduction, and pragmatic purpose. In response to identified gaps, HLO emerges as a specialized structural notation for typed functional programming. Grounded in Wittgenstein's language-games and mathematical discourse principles, HLO prioritizes clarity, cross-referential integrity, and adaptability. Its design—informed by the Physics of Notation Systematized (PoN-S) framework—addresses the unique demands of functional systems through dictionary-style definitions, thesaurus-like groupings, and etymological dependencies.

While preliminary refinements to HLO (e.g., staged grouping, symbol simplification) suggest enhanced usability, further empirical validation is warranted. Ultimately, this review highlights the necessity of domain-specific modeling tools that align with functional programming's paradigm, paving the way for more intuitive, effective, and human-centric design practices.

Chapter 3

Methodology

This chapter presents the research design and methodology for developing a structural modeling notation for functional systems using the Physics of Diagrams (PoD) framework. Adopting PoD as an alternative to Physics of Notation Systematized (PoN-S), the study aims to enhance the notation's cognitive fit, semiotic clarity, and perceptual discriminability. The chapter outlines PoD's seven foundational principles, which guide the notation's design through three key steps: evaluating cognitive fit, determining symbols, and identifying legitimate symbol combinations. Validation is conducted via three case studies (Pandoc, Docutils, and FParsec) and a survey-based evaluation, gathering qualitative and quantitative feedback from students, developers, and analysts. Ethical considerations and expected outcomes are also discussed.

3.1 Backgound

While Motara (2021) introduced the Physics of Notation Systematized (PoN-S) design process as a foundational framework for creating a modeling notation, the work presented here takes a different approach by adopting The Physics of Diagrams (Pissierssens, Claes, and Poels 2019) design process. This alternative design process builds upon the principles of visual representation and cognitive processing to develop a notation that aligns with the specific requirements of our research domain. By leveraging the Physics of Diagrams, our work aims to enhance the cognitive fit, semiotic clarity, and perceptual discriminability of the notation, thereby improving the speed, ease, and accuracy with which the representation can be processed by the human mind.

In this study, we deliberately opted for a distinctive design choice, selecting The Physics of Diagrams as our framework to explore an alternative perspective. The rationale behind this choice lies in its empirical grounding and the potential it holds for elevating the quality and effectiveness of graphical representations.

The Physics of Diagrams stands out due to its empirical foundation, drawing insights from a comprehensive literature study that meticulously compiled and organized 81 guidelines and 34 underlying propositions, resulting in the derivation of 7 foundational principles for graphical representation design. This structured approach not only provides a scientific basis for diagram design principles but also offers a well-rounded framework to guide designers in making informed decisions. By bridging the gap between best practices and academic knowledge, this framework aims to mitigate the consequences of differences in understanding between novices and experts, ultimately enhancing diagram readability and usability.

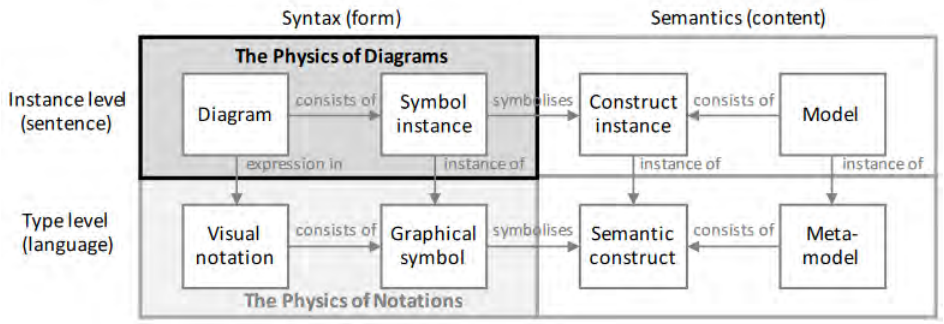


Figure 3.1: "Whereas Moody's "Physics of Notations" focuses exclusively on the bottom-left hand quadrant, the "Physics of Diagrams" proposed in this paper focuses on the top-left quadrant of the figure. Both artefacts leave the right-hand quadrants out of scope. (adapted from (Moody, 2009))" Pissierssens, Claes, and Poels (2019)

Moreover, "The Physics of Diagrams" serves as a complementary counterpart to the "Physics of Notations." While the latter concentrates on the design of graphical representation languages, the former is dedicated to the design of graphical representations themselves (see Fig. 3.1). This comprehensive approach ensures a holistic perspective on graphical representation design, addressing both the overarching language and the specific instances of graphical representations. This integration strives for a unified framework, fostering effective design practices across the entire spectrum of graphical representation.The 7 principles of The Physics of Diagrams are described in the next section.

3.2 Fundamental Principles

The Physics of Diagrams consists of seven fundamental principles for graphical representational design. These seven principles are organized into three categories: Fundamental principles, Practical principals, and Integrating principles (Fig. 3.2). The first two foun-

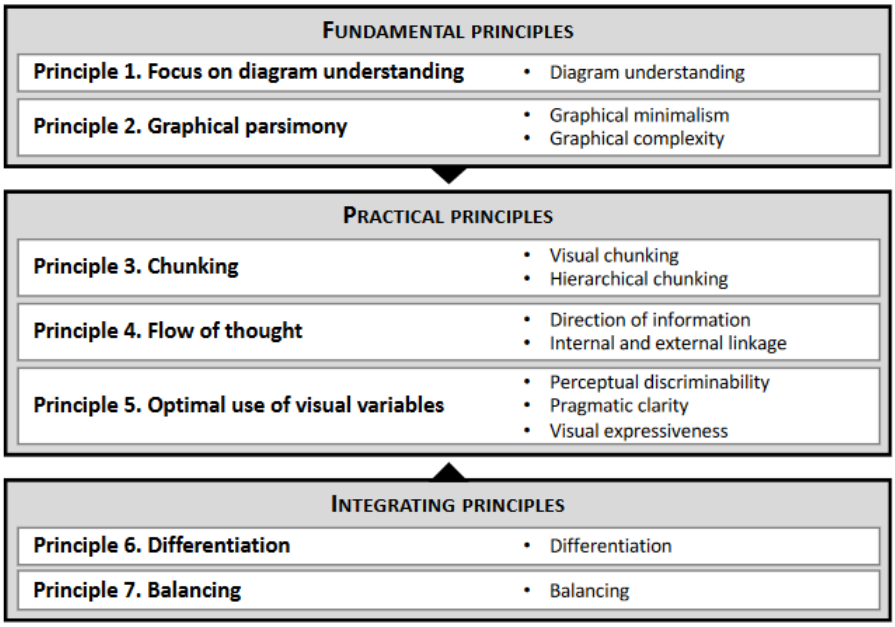


Figure 3.2: The Physics of Diagrams: Seven Core Principles for Graphic Representation Design (reproduced from (Pissierssens, Claes, and Poels 2019))

dational principles are:

- Principle 1: **Focus on diagram understanding**. This principle emphasizes the critical importance of prioritizing diagram understanding and cognitive processing. Rather than solely pursuing completeness and correctness, designers should strike a balance between cognitive ease and the completeness of diagrams, ensuring they fulfill their purpose of conveying information effectively.

- Principle 2: **Graphical parsimony**. This principle advocates for simplicity in graphical representations, urging designers to use the minimum number of symbol types and instances. Overcrowded diagrams, resulting from an excess of graphical elements, can impede comprehension. By adhering to graphical minimalism, designers enhance readability and address challenges posed by the growing complexity of data.

The following three principles provide concrete guidelines for achieving diagram effectiveness:

- Principle 3: **Chunking**. This principle recommends grouping information into meaningful parts, using hierarchical structures to represent element relationships. Chunking not only aids in understanding relationships but also supports creative thinking and simultaneous learning.
- Principle 4: **Flow of thought**. Emphasizing the importance of readability, this principle advocates for maintaining a continuous and consistent information flow direction. Internal and external linking of diagram elements enhances comprehension and prevents the loss of the logical flow of thought.
- Principle 5: **Optimal use of visual variables**. This principle highlights the role of perceptual discriminability in improving diagram readability. It stresses the need for clarity in semantics, syntax, naming, and diagram structure, achieved through the expressive use of visual variables.

The final two principles guide designers in resolving conflicts and facilitating the evaluation of design choices:

- Principle 6: **Differentiation**. This principle suggests that diagram readability can be improved by tailoring the design to user characteristics, task characteristics, and semantic requirements. Differentiating design based on these factors optimizes graphical representation effectiveness.
- Principle 7: **Balancing**. Acknowledging situations where graphical representations must accommodate multiple users and tasks, this principle advocates for using multimedia techniques and assigning weights to different design principles. Balancing ensures a thoughtful approach to meet the varied needs of users and tasks.

These principles offer a structured framework for diagram designers to make informed and optimal design decisions when creating graphical representations. They are intended to guide the design process, facilitate comparison between different designs, and formalize quality assurance of diagram design. Additionally, the principles aim to benefit various stakeholders by improving diagram readability, standardizing design processes, and minimizing the consequences of novice-expert differences in understanding graphical representations (Pissierssens, Claes, and Poels 2019).

3.3 Design process

Step 1: **Evaluate Cognitive Fit.** Our design process begins by ensuring that the notation aligns with the specific requirements and expectations of its users. This involves evaluating the cognitive fit, ensuring that the notation is well-suited for the task at hand. Similar to Motara (2021), our audience comprises students, developers, and business analysts with a broad technical background but varying levels of familiarity with software development intricacies. The specific tasks for evaluating cognitive fit include:

- Expressing the architecture of a typed functional system, encompassing its components, interactions, and relationships.
- Modifying the structure of the system to accommodate changes in requirements or design decisions.
- Facilitating understanding and analysis of the systems structure by students, developers, and business analysts.

Table 3.1: Guidelines (G) about perceptual discriminability found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G34. Consider using textual differentiation	(Cierniak et al., 2009)
G35. Use modularization	(Ambler, 2005; Gerjets et al., 2004; Regnell et al., 1996)
G36. Spread nodes uniformly over the display space	(Schuette & Rotthowe, 1998)
G37. Use barriers to separate object groups	(Ambler, 2005; Cox, 1996; Ding & Mateti, 1990)
G38. Provide sufficient distance between diagram elements	(Buagajska, 2003; Ding & Mateti, 1990)
G39. Avoid close lines, spatial overlap, and line crossings	(Ambler, 2005; Purchase, 2002; Ware et al., 2002) G40. Warrant minimum angle between crossing lines & (Purchase, 2002)
G41. Avoid diagonal or bended lines	(Ambler, 2005; Purchase, 2002)
G42. Use reasonable and consistent element sizing	(Ambler, 2005; Koshman, 2010)
G43. Use distinctive shapes and use them consistently	(Peebles & Cheng, 2003)
G44. Use colors to increase element distinctiveness	(Cox, 1996; Ding & Mateti, 1990; Moody, 2009)
G45. Use labels and diagram legends	(Ambler, 2005; Moody, 2009)

Step 2: **Determine Symbols.** Next, we embark on determining the symbols to be incorporated into the notation. This step is guided by the Principle of Optimal Use

of Visual Variables. This principle underscores the importance of leveraging visual attributes to enhance perceptual discriminability, clarity in semantics, syntax, naming, and diagram structure, as well as the expressive use of visual variables. Symbols are designed to facilitate both perceptual and cognitive processing phases. They should be easily distinguishable and visually clear to support rapid and accurate perception. Furthermore, symbols must map to concepts unambiguously, employing visual variables to express their meanings transparently. Various visual characteristics such as position, shape, size, color, hue, orientation, and texture are utilized to augment the visual expressiveness of symbols, ensuring their readability and conveying specific meanings. It is imperative to maintain visual clarity and consistency across the notation, providing annotations or legends for unfamiliar symbols as needed. Tables 3.1, 3.2, and 3.3 give the language designer important guidelines for determining the symbols to be incorporated into the notation.

Table 3.2: Guidelines (G) about pragmatic clarity found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G23. The diagram should have edge and node verticality	(Ware et al., 2002)
G24. The diagram should have top-bottom ordering	(Purchase, 2002)
G41. The diagram should have minimal bending of lines	(Ambler, 2005; Purchase, 2002)
G45. Provide a diagram legend	(Berenbach, 2004)
G46. Choose a semantically transparent notation	(De Westelinck et al., 2005; Lowe, 2003)
G46. Choose a semiotically clear notation	(Cox, 1999; De Westelinck et al., 2005; Moody, 2009)
G47. First focus on content, then on appearance	(Ambler, 2005)
G48. Ensure clarity in structure	(Cox & Brna, 1995)
G49. The diagram should have a single and clear entry point	(Berenbach, 2004)
G50. The diagram should have symmetry	(Field et al., 1993; Ware et al., 2002)
G51. The diagram should have moderate length of path	(Schuette & Rotthowe, 1998; Ware et al., 2002)
G52. Be consistent in the use of visual syntax	(Ambler, 2005; Van Merriënboer & Kester, 2005)
G53. Be consistent in diagram legend placement	(Ambler, 2005; Berenbach, 2004)
G54. Use concrete and explicit naming and labelling	(Amar & Stasko, 2004; Ambler, 2005)

Step 3: **Identify Legitimate Symbol Combinations.** In the final step, we focus on identifying legitimate ways in which symbols can be combined effectively. This process is guided by the Principles of Graphical Parsimony, Chunking, and Flow of Thought. Symbols are combined with graphical minimalism in mind, aiming to use the minimum

Table 3.3: Guidelines (G) about visual expressiveness found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G50. Maximize diagram symmetry	(Purchase, 2002; Schuette & Rotthowe, 1998)
G52. Be consistent in the use of visual syntax	(Ambler, 2005; Van Merriënboer & Kester, 2005)
G55. Balance the size of elements	(Ding Mateti, 1990)
G56. Vary in brightness	(Pilgrim, 2003)
G57. Use color to differentiate between element types	(Ormrod, 1999)
G58. Use shape to distinguish element types	(Ding & Mateti, 1990)
G59. Use texture to differentiate between element types	(Irani & Ware, 2003; Pilgrim, 2003)
G60. Be consistent in element orientation	(Field et al., 1993)
G61. Attach importance to horizontal and vertical position	(Schuette & Rotthowe, 1998; Ware et al., 2002)
G62. Use intensity to signal critical elements	(Ormrod, 1999)
G63. Use novelty/unexpectedness to increase expressiveness	(Ormrod, 1999)

possible amount of symbol types to avoid overcrowding and cognitive overload. Hierarchical structures are employed to organize symbols into meaningful groups, reducing cognitive load and enhancing comprehension. The arrangement and positioning of symbols ensure a logical and consistent flow of information within the notation, guiding users through the sequence of data or processes. Symbols are internally and externally linked to convey the flow of thought effectively, with input and output data represented logically. Throughout this process, it is essential to remain flexible, as new symbols may be identified, necessitating revisiting the symbol determination step to ensure the coherence and effectiveness of the notation. Tables 3.4, 3.5, 3.6, and 3.7 give the language designer important guidelines for identifying legitimate ways in which symbols can be combined effectively.

Step 4: **The Validation Phase.** The validation phase encompasses a multifaceted approach to ensure the robustness and effectiveness of the developed notation. It begins by employing the notation system to create a series of diagram instances, each tailored to different scenarios, use cases, or system requirements.

Furthermore, the validation process hinges on the execution of three distinct case studies, each centered on modeling a real-world system. These case studies serve as practical demonstrations of the notation's applicability, enabling an in-depth evaluation of its performance in capturing the intricacies of diverse systems. Within each case study, model instances are generated to illustrate the system's structure, providing comprehensive cov-

Table 3.4: Guidelines (G) about visual chunking found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G12. Group diagram information into 3 to 5 units	(Sweller, 1988)
G13. Chunk by spatial proximity	(Bertin, 1981; Koshman, 2010; Rockwell & Bajaj, 2005)
G14. Chunk by similarity	(Koshman, 2010; Ware, 2004) G15. Chunk through symmetry (Koshman, 2010)
G16. Chunk by common fate	(Koshman, 2010)
G17. Use visual variables to enhance category distinction	(Ware, 2005)

Table 3.5: Guidelines (G) about hierarchical chunking found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G18. Gradually increase element refinement	(Van Merriënboer & Sweller, 2005)
G19. Keep the depth of decomposition manageable	(Cox, 1999)
G20. Introduce black boxes to increase diagram complexity	(DeMarco, 2002)
G21. Ensure strong diagram cohesion	(Zugal, 2013)

Table 3.6: Guidelines (G) about direction of information found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G22. Align graph elements on continuous and smooth paths	(Field et al., 1993)
G23. Draw edges along orthogonal vertices	(Purchase, 2002; Schuette & Rotthowe, 1998)
G24. Maintain consistent flow direction	(Purchase, 2002)

erage and insight into its modeling capabilities.

To assess the quality and effectiveness of the generated diagram instances, we employ an evaluation framework rooted in the seven foundational principles of notation design. These evaluation criteria encompass factors such as visual clarity, semantic transparency, graphical parsimony, chunking, flow of thought, optimal use of visual variables, and differentiation. By systematically evaluating the diagram instances against these criteria, we can gauge their adherence to fundamental principles of notation design and identify areas for improvement.

The evaluation process begins by analyzing each diagram instance with a keen focus on visual clarity. This entails examining the extent to which the symbols and graphical ele-

Table 3.7: Guidelines (G) about internal and external linkage found in literature (Pissierssens, Claes, and Poels 2019)

Guidelines	**Sources**
G25. Model the diagram breadth first, drill down afterwards	(Berenbach, 2004)
G26. For low-interactive diagrams, make relations explicit	(Caillies et al., 2002; Rockwell & Bajaj, 2005)
G27. For high-interactive diagrams, use text segmentation	(Zugal, 2013)
G28. Activate prior knowledge	(Sweller et al., 1998; Van Merriënboer & Kester, 2005)
G29. Provide an overview	(Ormrod, 1999; Spence, 2007)
G30. Use familiar modelling languages	(Ambler, 2005; De Westelinck et al., 2005; Sweller et al., 1998)
G31. Make use of reusable patterns	(Van Merriënboer & Kester, 2005)
G32. Use extreme words, visualizations, and examples	(Boyle & Encarnacion, 1998)
G33. Use annotations	(Rockwell & Bajaj, 2005; Van Merriënboer et al., 2002; Van Merriënboer & Kester, 2005)

ments within the diagram are clearly defined and easily distinguishable. The Principle of Graphical Parsimony guides this assessment, emphasizing the importance of minimalism in graphical representation to prevent cognitive overload and enhance comprehension.

Semantic transparency is another crucial aspect evaluated during the assessment. This principle, derived from the foundational principles of notation design, emphasizes the need for symbols to transparently convey their meaning. Each symbol should intuitively represent the concept it signifies, facilitating rapid understanding and interpretation by users.

The evaluation also considers the Principle of Chunking, which advocates for organizing symbols into meaningful groups or chunks to reduce cognitive load and enhance comprehension. Diagram instances are assessed based on their ability to present information in a structured and coherent manner, leveraging hierarchical structures to represent relationships between elements.

Furthermore, the Principle of Flow of Thought is integral to the evaluation process, ensuring that the diagram instances guide the viewer through a logical sequence of information. This involves assessing the arrangement and positioning of symbols to facilitate smooth information flow and comprehension.

The evaluation also encompasses the Principle of Optimal Use of Visual Variables, which

underscores the importance of leveraging various visual characteristics such as color, shape, size, and orientation to enhance expressiveness and convey additional meaning. Diagram instances are evaluated based on their effective utilization of these visual variables to improve comprehension and interpretation.

Additionally, the Principle of Differentiation plays a pivotal role in the evaluation, focusing on the distinctiveness and clarity of symbols within the diagram instances. Each symbol should be visually distinguishable from others, minimizing the risk of confusion or misinterpretation.

Finally, the Principle of Balancing guides the evaluation by encouraging designers to make purposeful trade-offs between competing design objectives such as clarity, expressiveness, and completeness. This principle ensures that the diagram instances strike a delicate balance between these objectives, optimizing their overall quality and effectiveness.

Throughout the validation process, meticulous documentation and reporting are paramount, capturing the findings, observations, and insights gleaned from applying the notation in real-world scenarios. This documentation serves as a valuable resource for offering valuable insights into its strengths, weaknesses, and potential areas for enhancement. Ultimately, the validation phase plays a pivotal role in refining and optimizing the notation, ensuring its usability, effectiveness, and relevance in modeling complex systems.

3.4 Case Studies in Action: Methodology and Selection Rationale

Case studies are a valuable research method that involves an in-depth and detailed examination of a specific instance or phenomenon within its real-life context. This qualitative approach is particularly adept at uncovering complexities and providing nuanced insights.

In the realm of information technology and information systems research, case studies are often used to evaluate the impacts of technology on individuals and organizations (Hermes and King 2013; Jancewicz and MacKenzie 2002; Myers 1997). Case studies are an appropriate way to research areas with limited prior studies, offering valuable insights into emerging topics in the rapidly evolving field of information systems (Bourget 2014).

The practical utility of case studies is paramount when demonstrating the real-world application of proposed methodologies or interventions. Researchers, seeking to validate

the effectiveness of their findings, often turn to case studies. For instance, in (Izadikhah, Saen, and Roostaee 2018), a case study is used to showcase the applicability of a model for evaluating the sustainability of suppliers, demonstrating how proposed approaches address tangible challenges. Similarly, Odarushchenko et al. (2022) employs an industrial case study to underscore the effectiveness of a proposed approach in improving software failure rate prediction. Hajiagha et al. (2018) further exemplifies the use of a real-world case study to validate the proposed method for determining common set weights in a multi-period DEA. Numerous other examples (Edington et al. 2018; Hagspiel 2018; Jeffries and Brodsky 2017; Nobakht and Truscan 2013) reinforce the notion that case studies are pivotal in illustrating how methodologies address real challenges and generate tangible outcomes.

In the context of our research, centered on the introduction of modeling notations for functional systems, case studies take on a crucial role. The proposed notation which is designed to depict the structural aspects, needs validation in real-world scenarios. A case study offers the ideal platform to showcase their applicability and effectiveness. Our aim is to demonstrate how these notations function in practical settings.

Through the case study, we intend to provide empirical evidence, illustrating how the introduced modeling notations effectively capture the organization and relationships within functional systems. This empirical validation directly addresses the identified research gap – the absence of suitable methods for designing and modeling functional systems. By choosing a real-world system as our case study, we are not only validating the theoretical constructs but also ensuring that the proposed notations are practical and beneficial to a broader audience.

The use of case studies in our research serves a dual purpose. It not only aligns with the broader trend in software engineering research but also provides a concrete platform to demonstrate the applicability and impact of our proposed modeling notations. The case study is not merely an illustrative example; it is a crucial component in establishing the real-world relevance and effectiveness of our contributions. This approach ensures that our research is not confined to theoretical constructs but resonates with practitioners and researchers alike by showcasing the practical utility of our proposed methodologies.

3.5 Exploring Structural Modeling Notations: Three Case Studies

In this section, we delve into three comprehensive case studies aimed at elucidating the effectiveness and applicability of our proposed structural modeling notations across diverse software systems. Each case study provides valuable insights into how our notation captures the organizational intricacies of software systems, catering to various programming paradigms and domains.

Our first case study embarks on modeling the structure of Pandoc, a potent Haskell library renowned for its versatility in converting between diverse markup formats. Operating through a set of readers and writers, Pandoc parses text in various formats, generates an Abstract Syntax Tree (AST), and converts it into the desired target format. As Pandoc is deeply rooted in the functional programming paradigm, our goal is to evaluate how effectively our notation captures the organizational intricacies of functional systems. By leveraging Pandoc as our subject, we seek to demonstrate the practical utility and applicability of our notation in modeling real-world functional systems, while also garnering insights into its strengths and limitations.

In our second case study, we shift our focus to modeling Docutils, an open-source text processing system designed for generating documentation in multiple formats from plain text. Unlike Pandoc, Docutils primarily operates with reStructuredText (reST), a markup language, and processes reST input to produce various output formats. Docutils represents a non-functional system, providing valuable insights into how our proposed modeling notation functions within such environments. By examining Docutils, we can assess the adaptability and efficacy of our notation in modeling the structure of systems that operate outside the realm of functional programming. This case study serves as a crucial component in validating the versatility and applicability of our notation across a spectrum of software systems, enriching our understanding and refinement of the proposed modeling approach.

Our final case study embarks on modeling FParsec, an F# open-source parser combinator library. Utilizing parser combinators, FParsec facilitates the construction of parsers that analyze and transform textual input based on predefined rules. This case study allows for a comparative analysis between our proposed modeling notation and the original notation utilized in Motara (2021) for modeling FParsec. By examining both approaches, we can discern any advantages or drawbacks inherent in our notation, informing potential

refinements or optimizations. Similar to the Pandoc case study, FParsec offers insights into modeling a functional system; however, its use of F# presents unique challenges and considerations. Ultimately, the inclusion of FParsec as a case study enhances the comprehensiveness and robustness of our research findings, bolstering the credibility and applicability of our proposed modeling approach.

3.6 Evaluation Process

3.6.1 Purpose of the Evaluation

The purpose of this evaluation is to assess the effectiveness of the proposed unified modeling notation in representing the architecture and relationships of functional systems. The evaluation aims to determine whether the notation meets its objectives of simplicity, usability, and the ability to enhance understanding and communication among diverse audiences. By examining its application to real-world systems and gathering feedback from participants, the evaluation seeks to validate the hypothesis and identify areas for refinement.

3.6.2 Evaluation Design

The evaluation is designed to measure the notation's practicality, clarity, and effectiveness through structured case studies and participant feedback. We include case studies, representing functional systems (Pandoc and FParsec) and a non-functional system (Docutils), to ensure a comprehensive analysis of the notation's applicability across paradigms. A survey-based approach will collect qualitative and quantitative data from participants, who will interact with the notation through a series of diagrams. The evaluation is structured around a survey divided into four distinct cases to assess the modeling notation's usability, effectiveness, and clarity. Each case focuses on a different aspect of the notation:

- Case one evaluates its use in another functional system (FParsec)
- Case two explores its adaptability to a non-functional system (Docutils).
- Case three examines its application to a functional system (Pandoc).

- Case four investigates the clarity, effectiveness, and ambiguity of individual symbols and diagrams used in the notation.

The survey comprises both closed-ended and open-ended questions. For Cases 1 to 3, participants are presented with diagrams of the entire system, followed by multiple-choice questions (MCQs). These questions are designed to evaluate their understanding of the system and may have either a single correct answer or multiple correct answers, depending on the question. When a question has more than one correct answer, it is explicitly framed to ensure participants are aware of this, enhancing clarity and minimizing ambiguity. For example, a typical question based on a diagram asks participants to identify specific relationships or outcomes within the system.

Case 4 shifts focus to the evaluation of individual symbols and diagrams. Participants are presented with specific diagrams, and are asked questions aimed at gathering their opinions and perspectives on the clarity, usability, and effectiveness of the symbols or notations. These questions are designed to capture subjective feedback on the design and interpretation of the notation.

Additionally, open-ended questions are included in Case 4 to encourage participants to elaborate on their choices. For instance, after selecting an option based on a diagram, participants might encounter a follow-up prompt such as, "Please provide reasons for your choices above." In fact, all questions in Case 4 are followed by this open-ended prompt to provide richer qualitative insights into participants' reasoning and perceptions.

The open-ended questions in Case 4 are designed to capture participants' subjective experiences and perceptions of the notation. These qualitative responses complement the quantitative data from multiple-choice questions, providing richer insights into the strengths and weaknesses of the notation, particularly in terms of clarity, usability, and effectiveness.

Qualitative feedback from open-ended questions was analyzed using a structured four-step process adapted from (Ruona 2005). This process has been widely utilized across various research fields for robust qualitative data analysis, as demonstrated in studies such as (Kim et al. 2015; Noor et al. 2024; Sun et al. 2023). The four steps are as follows:

- **Data Preparation**: The survey data was collected using Google Forms, which allows for easy organization and export of responses into an Excel spreadsheet. To streamline the analysis, pivot tables were used to isolate responses for each question,

creating separate sheets for individual questions. This ensured that the analysis could focus on specific aspects of the notation without unnecessary distractions from unrelated data.

- **Familiarization**: This step involved immersing oneself in the data through repeated and thorough reading of the responses. The goal was to develop a deep understanding of participants' perspectives and identify initial patterns or recurring ideas. For example, during the familiarization phase for the OR-Arrow symbol in case four, responses revealed recurring mentions of "sequential flow" and "reliance on context," which later informed the coding process.

- **Coding**: A systematic coding approach was employed to identify themes and patterns in the data. The process began with reading through participants' responses and highlighting words, phrases, or sentences that were directly related to the questions they were answering. These segments were then assigned codes based on their content. Particular attention was given to recurring ideas or patterns that resonated across multiple participants. For example, if one participant mentioned *"The symbol shows as if the relationships are linear"* and three or more participants echoed similar sentiments, this became a focus for further analysis. This approach ensured that the analysis captured shared perspectives and common issues rather than isolated opinions.

  Once all relevant data was coded, the codes were grouped into categories based on their similarities and differences. For instance, codes related to clarity (e.g., *"Clear"* and *"Easy to understand"*) were grouped under the category "Perceived Clarity/Ease of Understanding." Finally, overarching themes were identified by examining patterns across categories. For example, the theme "Clarity and Understandability of the Notation" emerged from categories such as "Reliance on Context/Explanation for Clarity" and "Perceived Clarity/Ease of Understanding."

- **Interpretation**: The final step involved interpreting the findings and drawing conclusions based on the identified themes. This included examining how the themes addressed the evaluation objectives and considering their implications for the notation's design. For example, the theme "Misinterpretations Related to Flow, Dependency, and Alternatives" highlighted the need for clearer visual cues to reduce ambiguity in representing branching relationships. The interpretation phase also involved revisiting the data to ensure that the themes accurately reflected participants' feedback and provided meaningful insights.

This comprehensive evaluation design, combining structured case studies with both quantitative and qualitative data collection, provides a robust framework for assessing the notation's practicality, clarity, and effectiveness. By examining the notation across functional and non-functional systems (Pandoc, FParsec, and Docutils), the evaluation ensures a thorough understanding of its applicability across diverse paradigms. The inclusion of closed-ended and open-ended questions in the survey allows for a balanced analysis, with quantitative data offering measurable insights and qualitative feedback capturing participants' nuanced perspectives.

The four-step qualitative analysis process—comprising data preparation, familiarization, coding, and interpretation—ensures a systematic and rigorous approach to identifying and evaluating themes in participant feedback. By focusing on recurring patterns and shared perspectives, the analysis highlights key strengths and areas for improvement in the notation's design. For instance, themes such as "Clarity and Understandability of the Notation" and "Misinterpretations Related to Flow, Dependency, and Alternatives" provide actionable insights into how the notation can be refined to enhance its usability and effectiveness.

Ultimately, this evaluation not only assesses the current state of the notation but also lays the groundwork for iterative refinements, ensuring that it meets the needs of its intended users. The findings will inform future design decisions, contributing to the development of a notation that is both intuitive and effective for modeling complex systems.

3.6.3 Participants and Sampling

Participants will include students, developers, and business analysts with varying levels of familiarity with system design and functional programming. A purposive sampling method will be employed to ensure representation across technical and non-technical audiences. A target of 30 participants has been set to balance diverse perspectives with feasibility. Participants will be invited to complete the survey and may opt in to follow-up interviews for in-depth feedback.

3.6.4 Criteria for Analysis

The key criteria for evaluating the notation—simplicity, clarity, usability, practicality, and effectiveness—can be explicitly linked to the seven principles outlined in Section 3.2, "The

Physics of Diagrams." These connections help establish a structured approach to assessing the notation's effectiveness in representing system architectures and relationships.

1. Simplicity

Definition: The ease with which participants understand the notation.

Related Principles:

- **Principle 1: Focus on diagram understanding** – This principle emphasizes cognitive ease and the balance between completeness and comprehension. A simple notation is easier to process and understand.
- **Principle 2: Graphical parsimony** – Advocates for minimizing the number of symbols and elements in a diagram to avoid unnecessary complexity. By reducing visual clutter, diagrams become easier to understand, fulfilling the criterion of simplicity. Overcrowded diagrams hinder comprehension, whereas streamlined designs enhance participant understanding.

2. Clarity

Definition: The effectiveness of the notation in clearly representing system architectures and relationships.

Related Principles:

- **Principle 5: Optimal use of visual variables** – Clarity is enhanced through effective use of visual differentiation, ensuring that relationships and system components are easily distinguishable.
- **Principle 4: Flow of thought** – A clear notation should maintain a logical flow that guides the user through relationships and system structures, preventing confusion and misinterpretation.

3. Usability

Definition: The ability of participants to apply the notation to solve problems or interpret systems.

Related Principles:

- **Principle 3: Chunking** – Usability is improved when information is organized into meaningful groupings, making it easier to process and apply the notation effectively.
- **Principle 6: Differentiation** – Adapting the notation to users' needs and background knowledge enhances usability, ensuring that it remains accessible across various levels of expertise.

4. Practicality

Definition: The extent to which the notation can be applied to real-world systems across functional and non-functional paradigms.

Related Principles:

- **Principle 6: Differentiation** – Practicality requires a flexible notation that can accommodate different system types, which aligns with the principle of tailoring the design to task characteristics and semantic requirements.
- **Principle 7: Balancing** – The notation should strike a balance between multiple design priorities to ensure usability in varied real-world applications.

5. Effectiveness

Definition: Whether the notation facilitates improved communication and understanding across diverse audiences.

Related Principles:

- **Principle 1: Focus on diagram understanding** – The notation should be designed to optimize comprehension across different stakeholders, ensuring that it serves as an effective communication tool.

- **Principle 6: Differentiation** – Customizes diagrams for diverse stakeholders (e.g., developers vs. executives), ensuring the notation resonates across audiences.
- **Principle 7: Balancing** – Ensuring effectiveness across multiple users and tasks requires balancing various design considerations, such as accessibility and detail level, so that the notation remains functional in diverse contexts.

3.6.5 Data Collection and Ethical Considerations

Data will be collected through an online survey, capturing participant responses to questions and feedback on diagrams. Follow-up interviews, if consented to, will provide deeper insights into user experiences. All participants will be informed about the purpose of the study and their rights, including the voluntary nature of participation and the confidentiality of their responses. Ethical approval for the study has been obtained, ensuring compliance with academic and institutional guidelines (see Appendix A).

3.6.6 Anticipated Outcomes

The evaluation is expected to provide insights into the strengths and limitations of the notation. Anticipated outcomes include evidence of its effectiveness in simplifying complex system representations and fostering understanding among diverse audiences. The results will also highlight areas for improvement, ensuring the notation evolves to better meet its objectives. Success will be indicated by positive participant feedback and strong performance across the defined evaluation criteria.

3.7 Conclusion

This chapter has detailed a rigorous methodology for designing and validating a structural modeling notation for functional systems, grounded in the Physics of Diagrams (PoD) framework. Departing from prior approaches (e.g., PoN-S), PoD's empirically derived principles—focusing on cognitive processing, graphical parsimony, and perceptual discriminability—guided the notation's development through three key steps: evaluating cognitive fit, determining symbols, and defining legitimate symbol combinations. Validation was achieved via three diverse case studies (Pandoc, Docutils, FParsec), demonstrating applicability across functional and non-functional paradigms, and a mixed-methods

survey evaluating usability, clarity, and effectiveness among students, developers, and analysts. Ethical compliance ensured participant confidentiality and voluntary engagement. This systematic approach aims to deliver a notation that enhances model abstraction, communication, and comprehension in functional system design, with outcomes poised to inform both theory and practice.

Chapter 4

Implementation

This chapter details the development of the new structural modeling notation, outlining the modifications made, their rationale, and the improvements they introduce. It also presents the application of the notation across the case studies (Pandoc, Docutils, and FParsec), describing the process of model creation for each system. Additionally, the chapter discusses the survey design, explaining the questions formulated for participants to evaluate the models using the revised notation.

4.1 Notation

The target audience and purpose of our notation remain consistent with Motara's definition (Motara 2021): "The audience for the notation is *inter alia* students, developers, and business analysts; in other words, a broad and general audience which has some technical background and may be interested in software, software features, and software design, but may not necessarily be *au fait* with the details of software development. The task is to allow a typed functional system's structure to be expressed, modified, and understood by this audience." The key distinction in our work compared to Motara lies in our enhancement of the proposed notation, leveraging the principles outlined in the Physics of Diagrams. The changes are outlined below, highlighting key observations or limitations in the original notation, the corresponding design decisions and their underlying rationale, followed by example applications that demonstrate the impact of each modification.

1. Leveling the Notation:

- **(a) The original notation:** One key observation is that the original notation follows a staged structure: it begins by defining fundamental building blocks, then introduces sub-definitions that refine these components, followed by definitions that manipulate or apply them to generate new outputs. Finally, it concludes with a thesaurus-like grouping of related definitions. However, in Motara's original proposal, these stages were not visually or structurally separated. The notation presented all elements in a flat, continuous diagram, with no explicit distinction between base definitions and derived manipulations. This flatness makes it difficult for readers to discern which components represent core types and which depict transformations or outputs. This limitation becomes especially problematic in complex systems, such as Pandoc, where layered transformations and subtype hierarchies are essential for understanding system behavior.
- **(b) Design decision:** To address this, We introduce a three-tiered structure to the notation. Level 1 encompasses main definitions and their subsets. Level 2 involves definitions resulting from manipulating the main types and their subsets. Level 3 incorporates additional information or summaries for stakeholders seeking an overview of the design.
- **(c) Motivation based on design principles:** The modification introducing a three-tiered structure to the notation aligns with the principle of "Focus on diagram understanding", "Graphical parsimony" , and "Chunking." By incorporating a structured approach to the notation, the design aligns with the principle of "Focus on diagram understanding," we aim to streamline the cognitive processing of information. Level 1 provides the foundational definitions, Level 2 illustrates the manipulations of these definitions, and Level 3 offers additional context or summaries. This structured approach aligns with the principle by directing attention to specific aspects of the system's structure at each level, reducing cognitive load and enhancing overall understanding. The introduction of distinct levels corresponds to the principle of "Graphical Parsimony" by fostering a minimalist approach. Each level serves a specific purpose without unnecessary complexity. Level 1 introduces main types, Level 2 illustrates manipulations, and Level 3 provides additional information, each with a clear and focused representation. This adherence to simplicity facilitates a more efficient visual processing of the notation, ensuring that the diagrams remain clear and uncluttered. The introduction of distinct levels aligns with the "Chunking" principle by organizing information into meaningful chunks. By

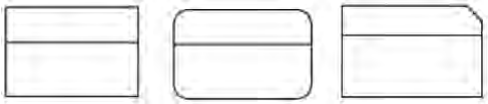


Figure 4.1: Notation for level definition (left), notation for level 2 definition (middle), and thesaurus (right)

grouping related information together within each level, we facilitate clearer visual-spatial characteristics, aiding in the comprehension and learning of the system's structure Bower 1970; Lowe 2003.

- **(d) Example application:** In the new FParsec model (see Fig. 4.16), he base definitions, such as "Parser" (and its subsets) and "some type or value" are introduced in Level 1. Their transformations (e.g., ⊙$^{f,\odot}$, ⊙!⊙, ⊙#, ⊙+⊙ ... etc.) are presented in Level 2, where specialized behaviors and derived definitions—such as transformations or combinations of Level 1 types—are modeled. A stakeholder overview diagram showing which definitions aid in (1) "parsing, handling failure" and (2) "parse more than one piece of text", is placed in Level 3. A stakeholder overview diagram—placed in Level 3—summarizes which definitions contribute to goals such as (1) *parsing while handling failure* and (2) *parsing multiple text segments.*

  Compared to the original model (see Fig. 2.6), which presented all components within a single flat diagram, the leveled version clearly separates structural definitions from transformations and high-level summaries. By segmenting the diagram into conceptual layers, users are better able to follow the logical flow of the system, reduce cognitive load, and avoid visual clutter—particularly in diagrams with many interdependencies and reused definitions.

2. Notation Alterations for Definitions and Thesaurus:

   - **(a) The original notation:** In the original notation proposed by Motara, both foundational definitions and derived ones were represented using the same visual format (see Fig. 2.5). This uniform appearance risked making it difficult for future users to distinguish between core system definitions and derived manipulations. The issue was expected to become more pronounced in models with many elements, where the visual language might impose a higher cognitive load. Additionally, the flat appearance could potentially hinder the scanning and mental grouping of elements, especially when definitions span multiple conceptual levels.

While thesaurus definitions did have a different representation, they were rendered using shading. This shading may be difficult to visualize in printed diagrams or reproduce in hand-drawn formats, as it requires filling the shape—something not easily achievable without digital tools.

- **(b) Design decision:** We revise the notation for definitions and thesaurus. Square rectangles represent Level 1 definitions, rounded rectangles signify Level 2 definitions, and shaded rectangles for Level 3 thesaurus. The updated notation is illustrated in figure 4.1. The notation resembles a class diagram, with the top part indicating the name or overall symbol for the definition and the lower part providing space for the textual definition.
- **(c) Motivation based on design principles:** The alteration in shapes and shading serves the purpose of improving perceptual discriminability and ensuring clarity in the notation's structure. The chosen visual variables contribute to the overall readability of the notation, aligning with the principle of "Optimal use of visual variables". By structuring the notation in a way reminiscent of class diagrams, we leverage a familiar visual pattern widely used in software engineering. Additionally, this aligns with the principle of "Flow of thought", as the chosen structure maintains a logical flow in representing information in the design.
- **(d) Example application:** In the revised Fparsec model (see Fig. 4.16), the definitions for "parser" (and its subsets) and "some type or value" appears in a square rectangle, indicating its status as a Level 1 definition. Derived definitions steps are presented in rounded rectangles in Level 2, reflecting their relationship as operations or refinements on core types. Finally, square rectangles with a diagonal line on the top right are used to express stakeholder summaries in Level 3.

  In the original model (see Fig. 2.6), all of these elements shared the same visual format, which made it difficult to visually distinguish between core definitions, and derived processes. This uniformity raised concerns about potential confusion, particularly in diagrams with a high number of elements. The revised design aims to reduce cognitive load by clearly separating conceptual levels and making the structure of the model easier to interpret. The effectiveness of these visual distinctions will be further examined in the Evaluation Chapter.

3. Symbol Definition Notation:

- **(a) The original notation:** In the original notation, there was no formal

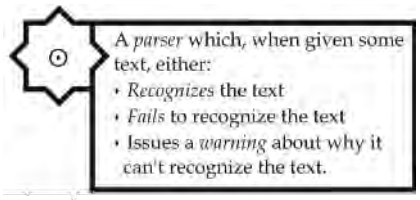


Figure 4.2: Definition of a parser (Motara 2021)

method for explicitly defining how a visual symbol related to the concept it represented. Symbols were placed on the leftmost part of the definitions, in the *rub-'al-hizb* shape, and accompanied by descriptive text. Figure 4.2 shows an example. In this format, the connection between the concept of a "parser" and the symbol (⊙) was never directly stated. Users were expected to infer this relationship based on the surrounding text. This approach relies on implicit associations, where the meaning of a symbol had to be inferred from context rather than being explicitly defined.

- **(b) Design decision:** A new notation is introduced for defining symbols for the definitions. Main types are now articulated as *'label :: symbol,'* where *'label'* denotes the name of the definition, and *'symbol'* is the representation used for the definition. The double colon signifies 'label is represented by the symbol' or 'label is denoted by the symbol.' For example, the definition *parser* :: ⊙, meaning the parser is represented by the symbol ⊙.
- **(c) Motivation based on design principles:** The introduction of a specific notation for defining symbols aligns with the principle of "Optimal use of visual variables". The new *'label :: symbol'* notation achieves two important goals: it makes the relationship between labels and symbols clear and unambiguous, ensuring that the visual elements accurately represent their intended meanings. This supports semantic transparency. Additionally, this notation enhances perceptual discriminability by making sure that the visual elements are visually distinct and clear, improving the overall interpretability and effectiveness of the graphical representation.
- **(d) Example application:** Figure 4.3 shows an example of a clearly marked section at the top of the diagram that lists symbol definitions using the *'label :: symbol'* convention. For instance, a definition such as *'parser :: ⊙'* immediately communicates to the reader that the symbol (⊙) represents the parser, without requiring them to first parse a full textual explanation. This approach enhances the user's ability to interpret the diagram by making the meaning of each

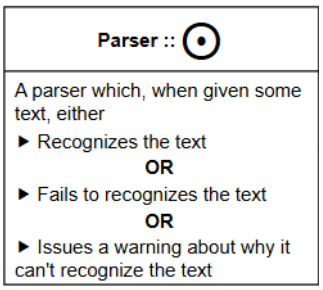


Figure 4.3: Revised definition of a parser

symbol explicitly available at a glance. As a result, readers can recognize and understand symbols more quickly, reducing the cognitive effort needed to decipher the diagram. The textual definition then serves to provide additional context or elaboration, rather than acting as the sole source of meaning. This layered structure not only supports clearer navigation and orientation within the diagram but also allows the notation to scale more effectively as complexity grows. New symbols can be introduced without compromising readability, and the diagram remains accessible even to users who return to it after some time. Ultimately, the *'label :: symbol'* format fosters a more structured and efficient interaction between the user and the visual model.

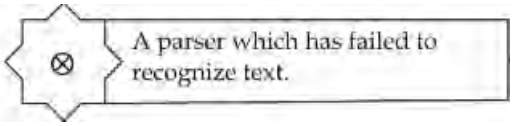


Figure 4.4: An example of a sub-definition (Motara 2021)

4. Subtype Definition Notation:

- **(a) The original notation:** In the original notation system (see Fig. 4.4), subtypes or sub-definitions were introduced in a manner similar to main definitions, using unique symbols for each. Each subtype was represented by its own unique symbol, and just like the main definition, each new symbol required the user to consult the accompanying textual explanation to understand what it represented. This meant that for every subtype introduced, a separate symbol had to be remembered, interpreted, and cross-referenced—adding unnecessary complexity. The notation not only shared the same limitations discussed earlier with symbol definition placement, but also compounded them by proliferating symbols. As more subtypes are introduced, this approach not only overloads

the user with additional symbols to memorize but also increases the cognitive load and reduces the interpretability of diagrams.

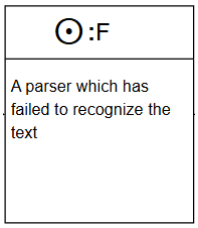


Figure 4.5: An example of a sub-definition (revised notation)

- **(b) Design decision:** Another notation is introduced for defining subtypes—definitions that structurally subset another, effectively aliasing a particular part of the larger definition. Subtypes are now expressed as *'symbol : letter,'* where *'symbol'* is the representation introduced in the main type through *'label :: symbol,'* and *'letter'* is a single character offering an intuitive understanding of the subtype's functionality. The colon signifies 'symbol of type letter,' with the 'symbol' part interchangeable with the name of the main definition introduced in *'label.'* For example, the definition $\odot$:F, represents "a parser of type F", where F is used to convey a parser which has FAILED to recognize text.
- **(c) Motivation based on design principles:** The introduction of a notation format like *"symbol : letter"* draws inspiration from common coding conventions found in functional languages (e.g., "a : int"). This familiarity serves a dual purpose. First, it aligns with established practices in typed functional programming languages, making the notation more intuitive for users familiar with these languages. Second, it adheres to the principle of "Optimal use of visual variables" by leveraging a visual format that carries semantic meaning. By mirroring a known coding pattern, the notation enhances the overall interpretability of the diagram and supports cognitive effectiveness. This notation further contributes to the semantic transparency and perceptual discriminability of graphical representations by visually representing the relationships between subtypes and their parent types in a clear and unambiguous manner. Also, By incorporating a single-letter code that intuitively conveys the subtype's purpose, the notation enhances perceptual discriminability. We went with this approach instead of introducing additional symbols due to the fact that introducing new symbols may increase the learning curve for users, especially those unfamiliar with the symbolism. Memorizing multiple symbols and their meanings could be challenging. Additionally, designing a set of symbols

that are both distinct and intuitively associated with specific subtypes can be challenging. Achieving clarity without introducing ambiguity requires careful symbol design. Additionally, introducing more symbols for each subtype could potentially clash with the principle of "Graphical Parsimony".

- **(d) Example application:** An example application of this notation can be seen in Figure 4.5. Sub-definitions in the diagram use expressions such as *'⊙ : F'* or *'⊙ : S'* to indicate subtypes of the parser, such as a failed parser ('F') or a successful one ('S'). This approach allows the user to immediately recognize the role of each subtype by referring back to the original symbol while inferring the subtype meaning from a single-letter extension. As a result, the diagram becomes easier to scan, interpret, and remember. Users are no longer required to decode or memorize a growing list of distinct symbols—instead, relationships and distinctions are conveyed through minimal and familiar visual constructs.

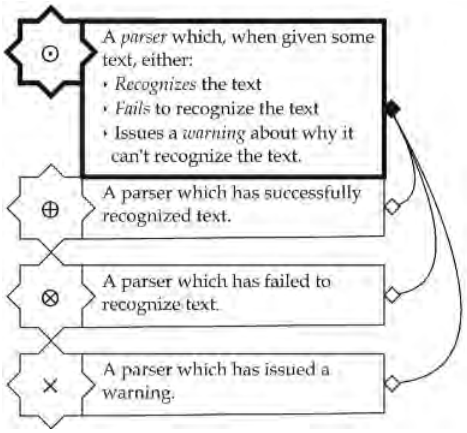


Figure 4.6: Foundational definitions (Motara 2021)

5. Introduction of OR-Arrow Symbol:

- **(a) The original notation:** In the original notation, a diamond-terminated line was used to branch from the main definition to its associated subtypes, as illustrated in Figure 4.6. This approach was sufficient in showing that the subtypes were linked to the main definition, and allows readers to interpret that the main definition encompasses several related possibilities. While this approach does convey a relationship, we wanted to explore an alternate way of expressing that the main definition could correspond to one or more distinct subtypes. Instead of a downward-branching layout, which can become vertically elongated as more subtypes are added, our revised notation introduces a horizontal layout that moves from left to right. This makes better use of space,

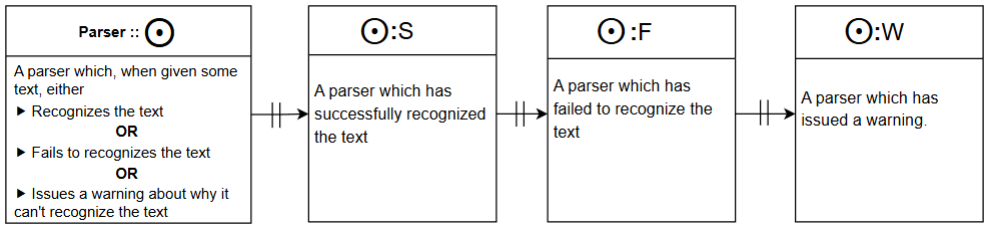


Figure 4.7: Foundational definitions (revised notation)

especially when there are many subtypes, and avoids stretching the diagram downward, which can make it harder to follow in larger examples.

- **(b) Design decision:** A new symbol, an arrow with ‖ in the middle, is introduced to signify "We can use the main definition to represent this subtype OR this subtype OR this subtype." The OR-Arrow can be chained together, with the first one originating from the main definition and subsequent ones connecting from each subtype.
- **(c) Motivation based on design principles:** The OR-Arrow symbol aligns with the principle of flow of thought, emphasizing a logical and consistent information flow direction. In the context of expressing relationships between the main definition and its subtypes, the arrow serves as a visual cue for the flow of possibilities. Additionally, the introduction of the OR-Arrow symbol optimally uses a visual variable (the arrow) to convey the concept of multiple pathways or options. This enhances the perceptual discriminability of the diagram by visually representing the idea that the main definition can lead to one or more subtypes.
- **(d) Example application:** Figure 4.7 shows an example of the OR-Arrow (⊣⊢→) in use. A single main definition is placed on the left, with multiple subtypes branching rightward using the or-arrows. This arrangement visually communicates that the main type can represent one of several alternatives. Compared to the original vertically stacked layout, this horizontal chaining makes better use of space and reduces vertical scrolling or diagram fragmentation. A comparison between this new OR-Arrow approach and the original vertical branching method will be conducted as part of the survey

6. The relies relationship:

- **(a) The original notation:** In the original notation, as shown in Figure 4.8, relationships between definitions that rely on each other were depicted using

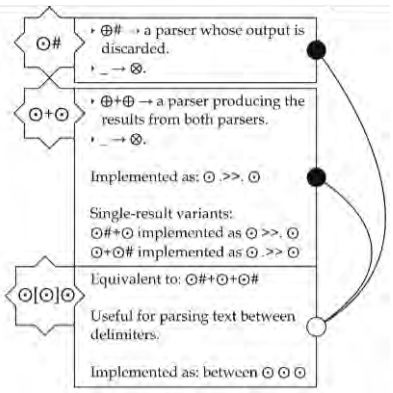


Figure 4.8: Relies relationship (Motara 2021)

circle-terminated lines pointing from the definition being relied upon to the dependent definition. This approach effectively conveyed the existence of a dependency. However, it relied on lines stretching across the diagram, which introduced visual clutter—especially when multiple definitions depended on the same source or when relationships crossed over each other. As the number of dependencies increased, the diagram could quickly become difficult to follow, with lines weaving through and around other components. While the lines succeeded in communicating the intended relationship, we sought to explore an alternative approach that would maintain clarity without increasing visual complexity or disrupting the overall flow of the diagram

- **(b) Design decision:** If a definition is built upon another definition, indicating reliance, a rectangular block is added to the bottom of the definition. This block contains the word "relies," followed by a colon and the symbol of the definition it depends on.
- **(c) Motivation based on design principles:** This addition provides explicit clarity about the relationship between definitions. It aligns with Principle 1: Focus on diagram understanding, as it prioritizes cognitive processing by explicitly representing the dependency. By visually encoding this relationship, it reduces ambiguity and enhances semantic clarity, adhering to Principle 5: Optimal use of visual variables. The graphical parsimony of this approach ensures minimal additional complexity, consistent with Principle 2: Graphical Parsimony.
- **(d) Example application:** An example of this can be seen in Figure 4.9, where a definition is extended with a relies: block placed directly beneath it. This addition immediately signals to the reader that the definition is not

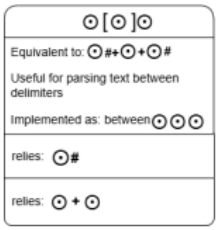


Figure 4.9: Relies relationship (revised notation)

standalone but depends on another. Rather than relying on lines that stretch from one part of the diagram to another, the relationship is captured locally within the same visual unit. This approach offers several advantages. First, it improves clarity by explicitly stating the dependency in a predictable and consistent location. Second, it is more space-efficient, particularly in diagrams with many interdependent definitions, as it avoids the need for long connecting lines. Third, by removing the visual noise caused by overlapping or intersecting lines, the diagram remains clean and readable even as it grows in complexity. Finally, embedding the relies relationship within the block itself helps reduce cognitive load, as users can quickly grasp the structure without tracing paths across the diagram. A comparison between this approach and the original line-based method will be carried out in the survey chapter.

The following set of symbols are symbols that from the original notation that have been kept due to their usefulness:

- "(" and ")" for grouping
- "⊛" for indicating "some type or value"
- subscripts for the cases of sum types
- "→" for mapping cases
- "_" for indicating a fall-through case
- "▶" and "•" for discrete cases and grouping respectively.

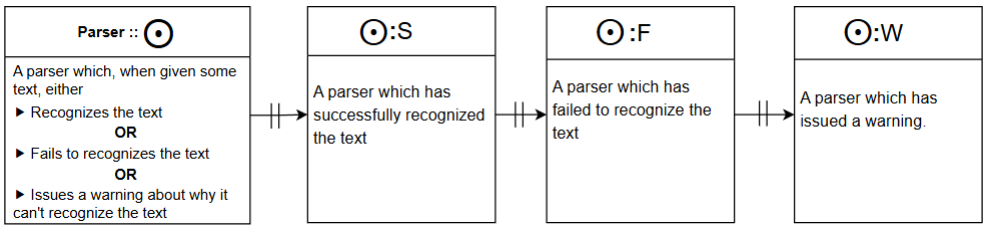


Figure 4.10: FParsec: Parser definition

4.2 Case One

FParsec [1] is a powerful parser combinator library for the F# programming language. It allows you to create efficient and readable parsers for various text-based formats by combining smaller parsers into more complex ones. When parsing fails, FParsec generates informative error messages that can help you identify and resolve parsing issues.

For our level 1 definitions, we define the main definitions and their sub-definitions. At its simplest, in FParsec, you create a parser that processes some text. This leads us to a parser that, when given some text, either recognizes the text, fails to recognize it, or issues a warning explaining why it cannot recognize the text. Figure 4.10 illustrates the main definition *parser* represented by the symbol ⊙. The OR-Arrow originating from the main definition points to the first sub-definition, ⊙*:S*, which represents a successful parser. Subsequently, another OR-Arrow extends from the first sub-definition to ⊙*:F*, indicating a parser failure. Finally, the diagram includes ⊙*:W*, denoting a parser that issues a warning. Together, these elements visually capture the relationships between the main definition and its sub-definitions.

A powerful feature of typed functional programming languages is the ability to represent generic types. Here, we define an additional sub-definition for representing some type or value (Fig. 4.11). Note the main definition does not have any sub-definitions here.

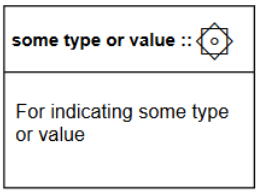


Figure 4.11: FParsec: some type or value

[1] https://www.quanttec.com/fparsec/

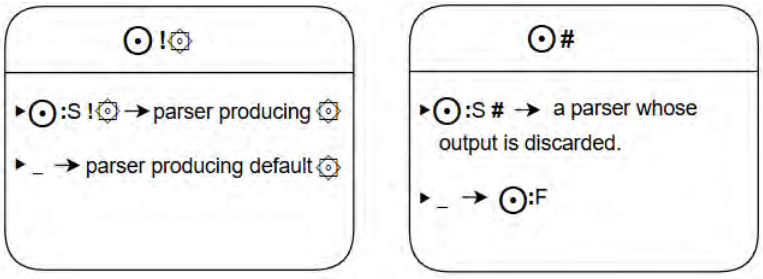


Figure 4.12: FParsec: Level two definition (a) and (b)

Definitions in Level 2 result from manipulating the definitions in Level 1. For instance, a parser that successfully parses text produces a specific value, while any other outcome—a parser that issues a warning or fails—produces a default value. Figure 4.12(a) illustrates this relationship. This is the first instance where the symbols ▶, →, and _ are introduced. The symbol ▶ marks the beginning of each case. The logic can be described as follows:

- If the parser successfully recognizes text (denoted by $\odot$*:S*), it produces the type specified after the '!' symbol.
- In any other scenario, the result is a failed parser (denoted by $\odot$*:F*).

We may want to discard certain text that is successfully parsed because it may not be needed in the final output. This is represented in figure 4.12(b).

In FParsec, we can combine parsers to form more complex ones. Therefore, we need a way to represent the combination of two parsers (4.13(c)).

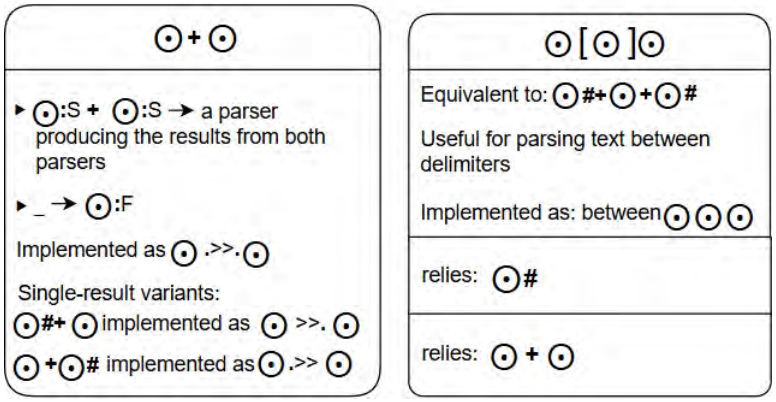


Figure 4.13: FParsec: Level two definition (c) and (d)

In certain scenarios, it may be necessary to discard delimiters at the beginning and end of a text while preserving the information between them. This can be achieved by building

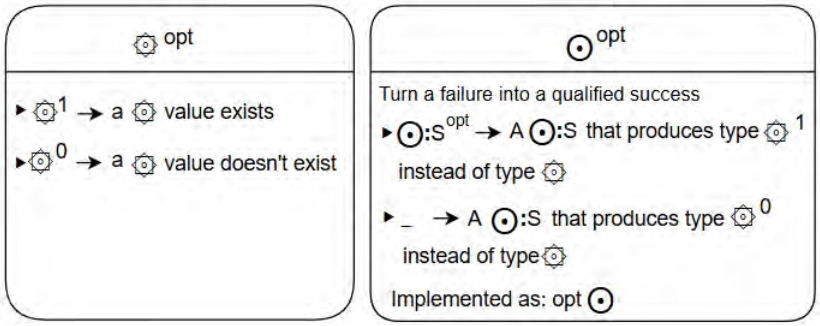


Figure 4.14: FParsec: Level two definition (e) and (f)

upon previous definitions, as demonstrated in Figure 4.13(d). In this case, rectangular blocks appended at the bottom of the definition indicate the other definitions it depends upon, leading to the text "Equivalent to:". The phrase "Equivalent to:" is used to denote semantics, describing the conceptual outcome or behavior of a definition rather than its specific implementation. This approach is particularly common in functional programming, where the paradigm inherently emphasizes the composition of functions. For example, when modeling a parser, the implementation code might not directly involve explicit function composition. However, the functionality often depends on the relationships between smaller components, such as ⊙# and ⊙+⊙. These dependencies illustrate how core operations are delegated to achieve the intended behavior, highlighting the modular and semantic nature of the system rather than focusing on implementation details.

In functional programming, it's common to represent the presence or absence of a value using types that encapsulate the possibility of a "missing" or "none" state (Fig. 4.14(e)). Such definitions enable graceful error handling by ensuring that results clearly indicate success or failure. This approach integrates error handling directly into the program's flow, reducing reliance on exceptions and promoting safer, more predictable code. We represent this by turning a failure into a qualified success (Fig. 4.14(f)). All non-alphabetic characters, such as "!", are considered to be fixed parts of symbolic definitions. The full set of Level 2 definitions, along with additional definitions, is shown in figure 4.16.

Level 3 builds upon the foundations laid in Levels 1 and 2 by providing a categorical grouping and a high-level summary of the system. This level emphasizes the modularity and composability inherent in FParsec. By grouping definitions into categories, Level 3 diagrams facilitate an overarching understanding of how individual components interact and contribute to the system's functionality. Figure 4.15 provides this categorical view, bridging the detailed manipulations of Level 2 with the holistic representation of the full diagram in Figure 4.16.

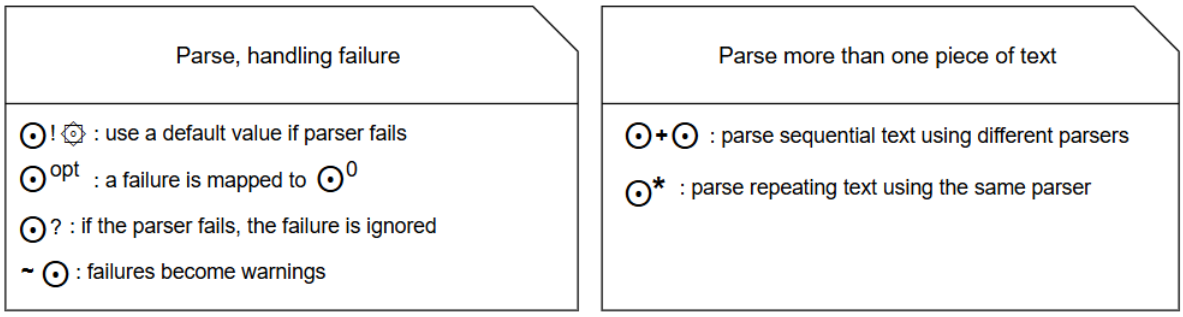


Figure 4.15: FParsec: Level three definitions

The survey for Case 1 was structured to evaluate participants' ability to understand and apply the proposed notation to represent and analyze the FParsec model. Designed in a tutorial style, similar to the structure of this section, the survey guided participants step by step through the model's notation. Participants were first asked to identify all main definitions and their associated sub-definitions in the diagram, as well as any definitions that relied on others. This line of questioning aimed to evaluate participants' ability to recognize and interpret the foundational elements of the notation while identifying how relationships between elements are expressed.

The questions then delved deeper into how definitions are manipulated. Participants were presented with specific Level 2 definitions and tasked with interpreting their meaning and functionality. They were also asked to explore specific combinations of symbolic definitions and their behavior within the context of parsing. For instance, participants were prompted with questions such as:

- What does the definition ⊙!⚙ signify in the context of parsing ?
- In ⚙opt, what does the presence of ⚙1 versus ⚙0 indicate ?
- How would you interpret (⊙:s#) + (⊙!⚙) ?
- How would (⊙ [⊙] ⊙)opt behave in the context of parsing ?

These questions encouraged participants to think critically about parsing logic and the modularity expressed within the diagrams. By focusing on the symbolic definitions and their interactions introduced in Level 2, the survey assessed participants' ability to interpret more complex relationships. Moreover, these questions emphasized the functional implications of the notation, prompting participants to apply their understanding to hypothetical parsing scenarios. This structured, tutorial-like approach aligned with the

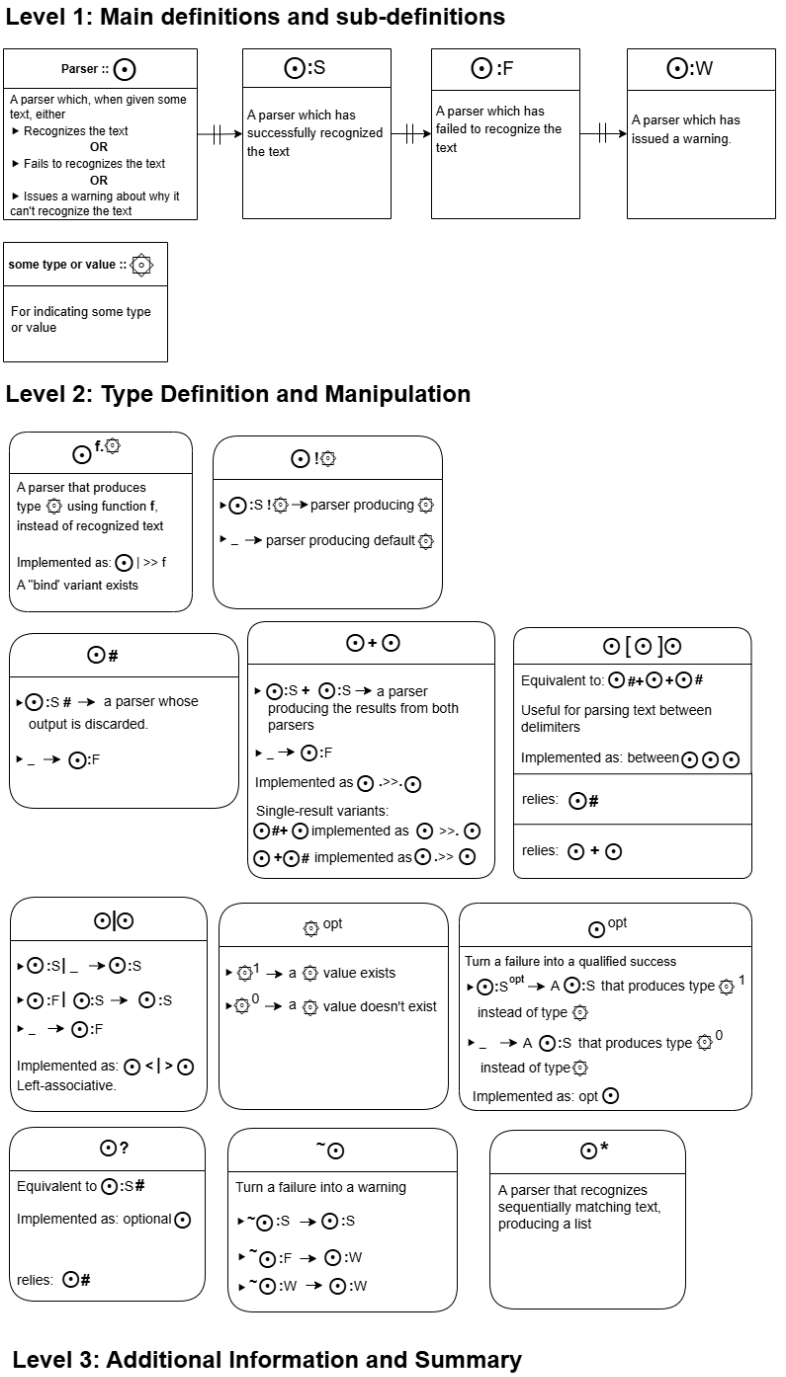


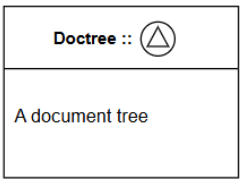
Figure 4.16: FParsec diagram

overarching goal of evaluating the notation's **clarity**, **usability**, and **practicality**. It provided a comprehensive framework to assess participants' ability to recognize, interpret, and apply the notation effectively. The participant responses and their implications will be analyzed and discussed further in the Results chapter.

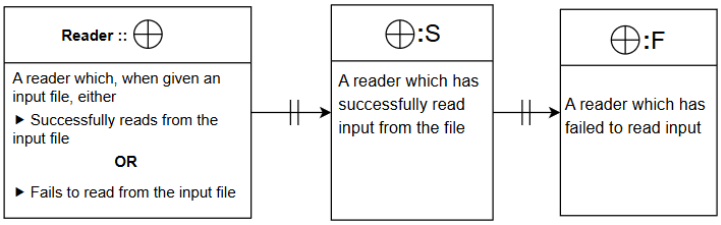
4.3 Case Two

Docutils is an open-source text processing system written in Python, designed to parse and transform reStructuredText (reST) documents. It converts reST into formats like HTML, LaTeX, and XML, making it a popular tool for technical documentation. Its architecture includes **parsers for reading input**, a **document tree** for structure representation, and **writers** for output generation. We need a definition to represent the structure of the document tree.

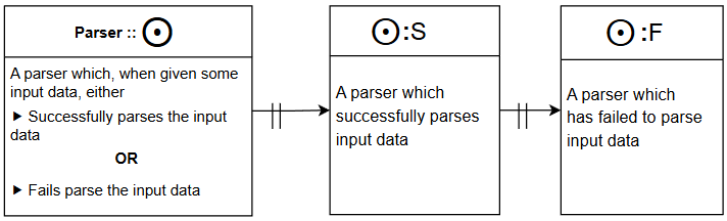
We need a definition to represent the structure of the document tree.



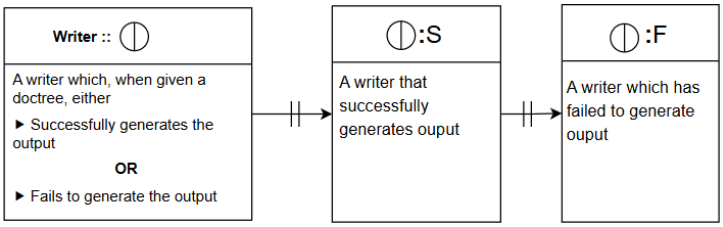
Readers play an important role in Docutils; they help read the input data and pass it to the parser. We represent the reader with the following definition:



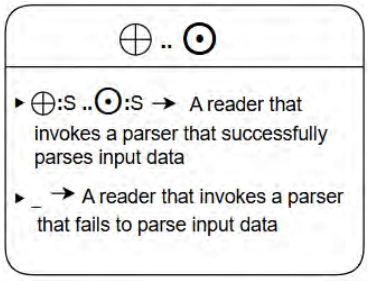
Parsers generate a document tree (doctree) from the input file. We define the parser as follows:



Finally, Docutils has writers who translate the document tree into the output format.



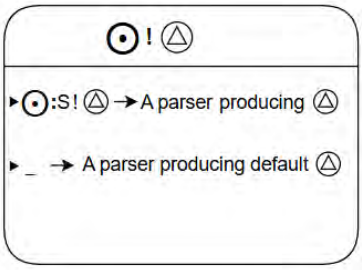
Moving on to level 2 definitions, Readers pass the input data to the parser so it can be converted into an AST:



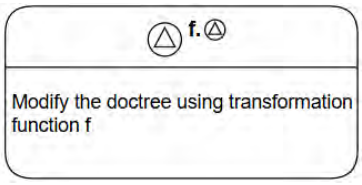
The .. notation encapsulates the act of invoking a connection between a reader (⊕) and a parser (⊙). This act encapsulates the dynamic process in which the reader, after processing or reading the input data from the source file, actively invokes the parser to further interpret and process the data. The meaning of each case can be interpreted as:

- When the reader successfully interprets the raw input, it passes the data to the parser. The parser then processes the data and generates a structured representation, typically a Document Tree (Doctree) or an equivalent abstract syntax structure. This success is denoted by the symbol *S*, indicating that both the reader and the parser have successfully completed their respective tasks, with the reader correctly reading the input and invoking the parser to generate the output structure.
- If the reader encounters an issue, such as malformed input or unsupported syntax, it generates a warning or error message. This indicates that the reader was unable to pass properly structured data to the parser, preventing the parser from generating the Document Tree and causing the process to fail.

Thus, a parser generates a doctree:

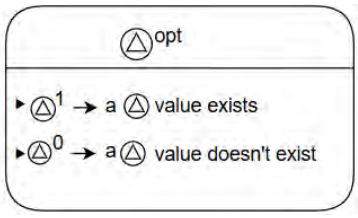


Docutils has transformers that modify the document tree:

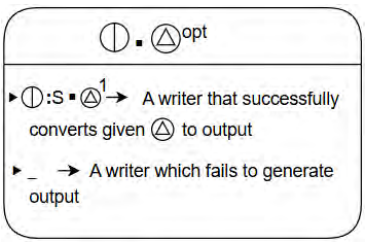


The name "f" is presented in boldface within the definition to emphasize its role as a substitutable identifier. This notation underscores that "f" serves as a placeholder or symbolic reference, allowing it to be replaced with specific instances or values as needed within the model's context. In the case of Docutils, the framework incorporates transformers designed to modify the document tree. The symbolic representation of "f" highlights its function as a flexible and substitutable transformation mechanism capable of implementing targeted modifications to the document tree to fulfill processing requirements.

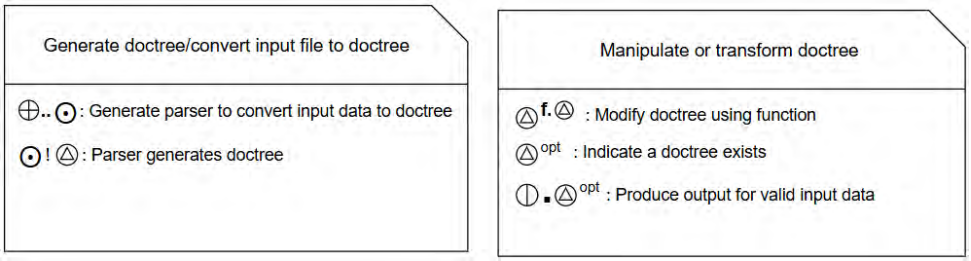
In functional programming, the presence or absence of a value is often represented using types that encapsulate the possibility of a "missing" or "none" state. We represent this in the following definition for a doctree:



Writers translate the doctree into output:



And finally, our level 3 definitions:



The full Docutils diagram is represented in figure 4.17. The survey for Case 2 was designed to assess participants' comprehension and application of the proposed notation in modeling the Docutils framework. The questions began by guiding participants to identify the main definitions and their associated sub-definitions in the Docutils diagram. This initial set of questions aimed to evaluate the participants' ability to understand the foundational elements of the model.

The survey then transitioned to a deeper exploration of the Level 2 definitions, prompting participants to interpret their functionality and implications. Specific questions explored the outcomes of applying symbolic transformations, such as identifying the results produced by a transformation function or understanding the implications of its application. Participants were also presented with specific combinations of symbolic definitions and asked to analyze their collective meaning within the context of the docutils. For example, the following questions were asked:

- What is the result of $\oplus{:}S \,..\, \odot{:}S$?
- In the definition $\triangle^{f.\triangle}$, what does the transformation $f.\triangle$ imply ?
- What does the combined definition represent $(\oplus{:}S \,..\, \odot{:}S)\,!\,\triangle$?
- If $\triangle^{0}$ (absent document tree), is passed to the writer in $\ominus \cdot \triangle^{opt}$, what is the outcome ?
- What does the combination $(((\oplus{:}S \,..\, \odot{:}F)\,!\,\triangle)^{opt}$ imply ?
- Which sequence best represents a full, successful input-to-output workflow using Level 2 definitions?

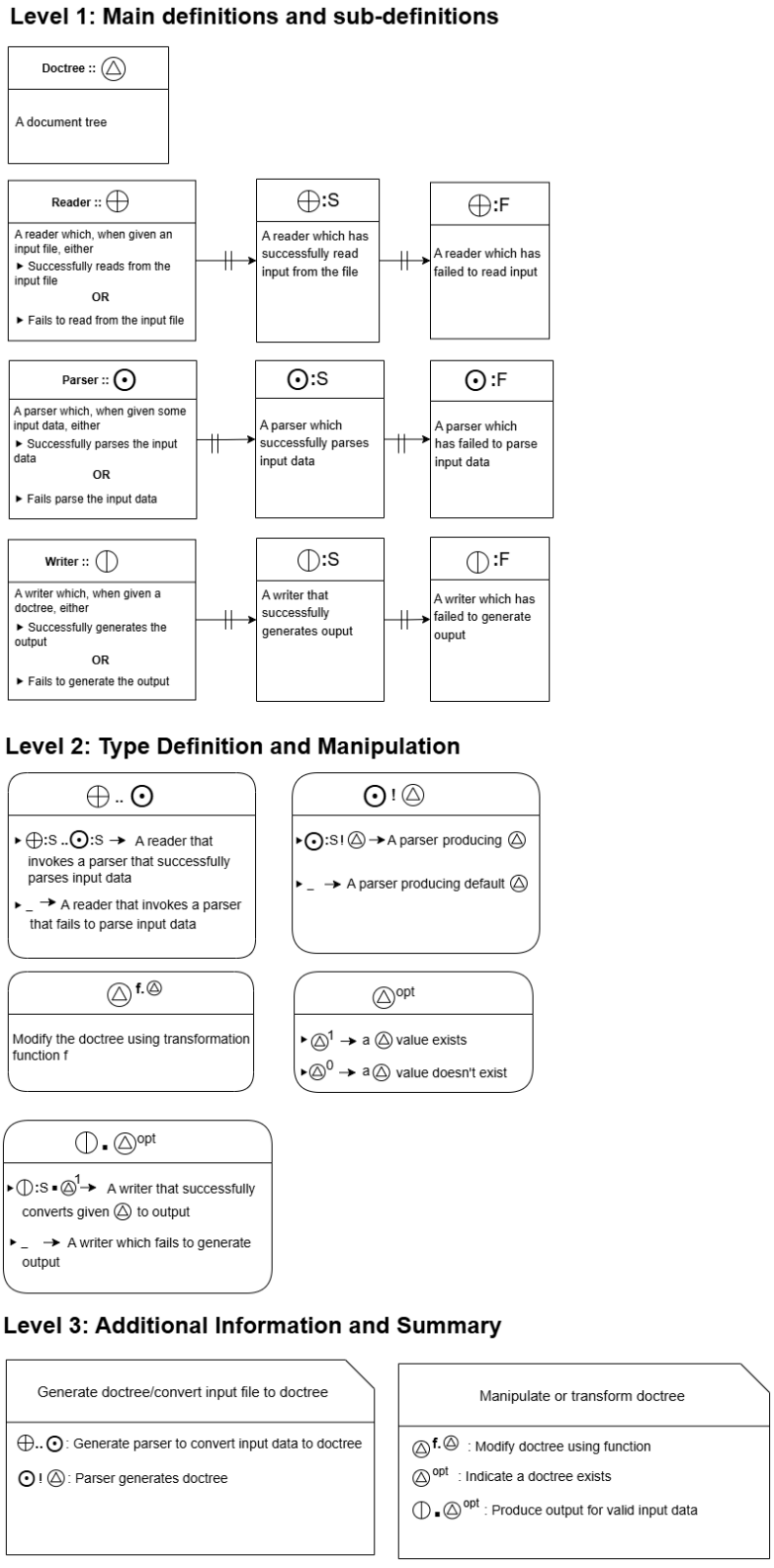


Figure 4.17: Docutils diagram

Additionally, participants were tasked with analyzing sequences within the workflow, evaluating their understanding of a full, successful input-to-output process using Level 2 definitions. These questions emphasized the functional interplay between symbolic definitions, encouraging participants to think critically about the invocation of readers, parsers, and transformations, as well as their role in generating a complete document tree.

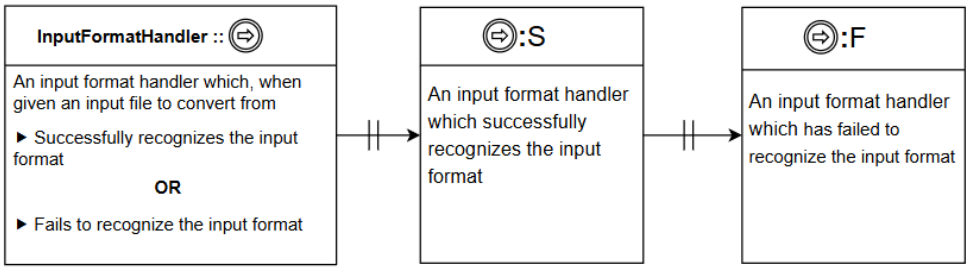
By focusing on the symbolic representations and their interactions within the Docutils framework, the survey assessed participants' ability to interpret complex relationships and workflows. Moreover, the questions underscored the functional implications of the notation, prompting participants to apply their understanding to scenarios grounded in document processing. This structured, tutorial-like approach ensured a comprehensive evaluation of the notation's **clarity**, **usability**, and **practicality**. The analysis of participant responses and their implications will be presented in the Results chapter.

4.4 Case Three

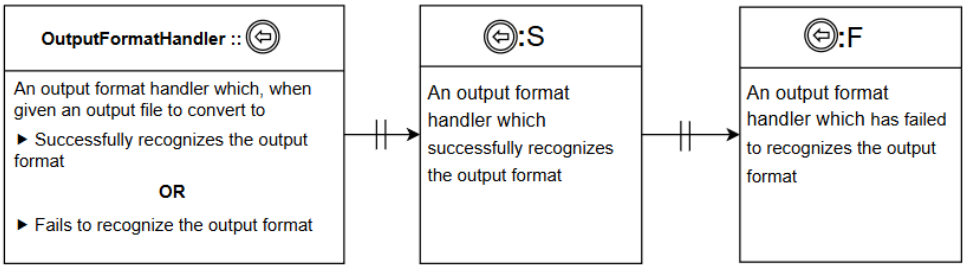
Pandoc is a powerful document converter written in Haskell that allows users to transform documents between a variety of markup formats, including Markdown, HTML, LaTeX, and PDF.

Pandoc utilizes **input parsers** to convert various markup formats, like Markdown and HTML, into an **abstract syntax tree (AST)**, which serves as a structured representation of the document's content. This AST enables efficient manipulation during the conversion process. After processing, **output writers** transform the AST into the desired output formats, such as PDF or Word documents. Additionally, **filters** allow users to customize the AST dynamically, modifying the document's content and structure before generating the final output. This architecture makes Pandoc a flexible and powerful tool for document conversion.

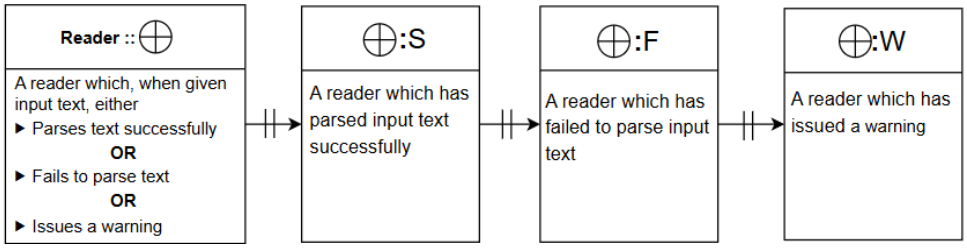
To represent these components, we define the following main concepts in level 1, we define an input handler to help manage which parsers should parse text from an input file:



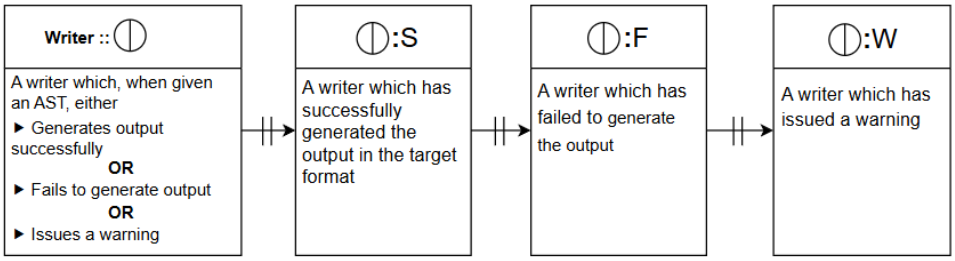
Similarly, we define an OutputFormatHandler:



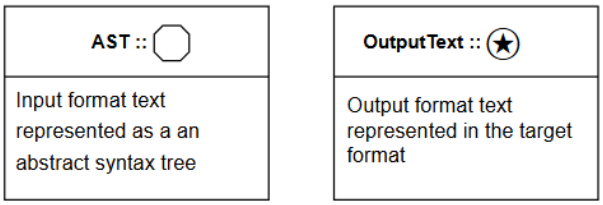
Another necessary main definition is for a Reader, which parses input text to generate an AST:



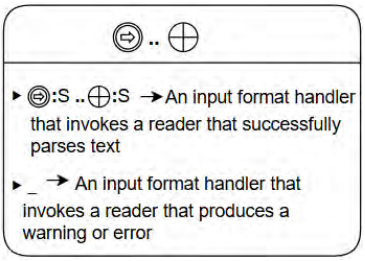
Similarly, we define a Writer:



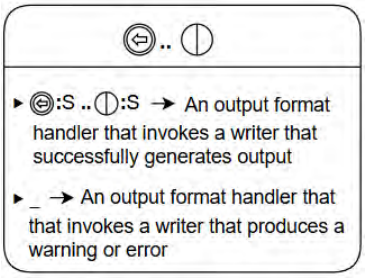
We also need definitions for an AST and output text:



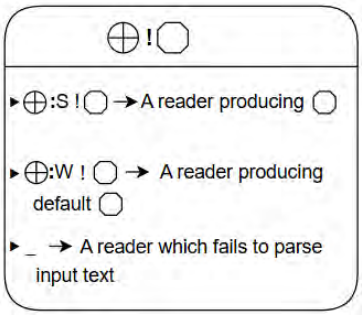
Level 2 definitions are derived from manipulating Level 1 definitions. We might have a definition for invoking a reader for a given input format to parser the text. Such a definition would look like:



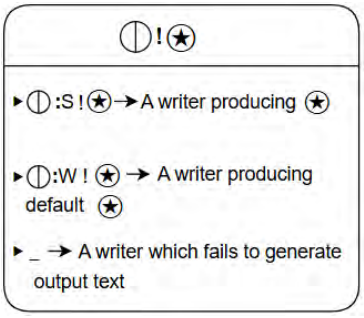
Similarly, for invoking a writer:



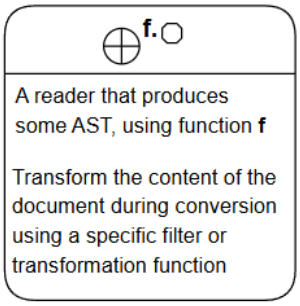
Focusing on readers, we define a process for producing an AST since readers output ASTs:



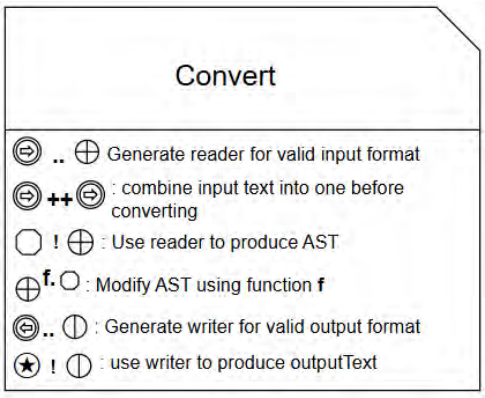
Similarly, for a writer that produces output text:



Pandoc also has filters that can transform the AST before converting it to an output format:



The complete set of Level 2 definitions, alongside supplementary definitions, is presented in Figure 4.18. And finally, our level 3 definitions:



The comprehensive Pandoc diagram, incorporating all levels of definitions, is shown in Figure 4.18. Similar to Cases 1 and 2, the survey for Case 3 included questions designed to evaluate participants' ability to comprehend the foundational elements of the Pandoc model. Participants were tasked with identifying the main definitions and their associated sub-definitions within the Pandoc diagram. This approach ensured that participants engaged with the basic structure of the notation before exploring more complex interactions.

Following the tutorial-style approach employed in previous cases, the survey aimed to assess the notation's **clarity**, **usability**, and **practicality**. Participants were presented with questions that encouraged them to analyze and interpret Level 2 definitions and their combinations. Examples of the questions include:

- What does the $\circledcirc\!\!\Rightarrow .. \oplus$ produce ?
- What does the combined definition represent $(((\circledcirc : s .. \oplus : s) ! \bigcirc)$?
- What does the notation $\oplus^{f.\bigcirc}$ imply in the context of transformation ?

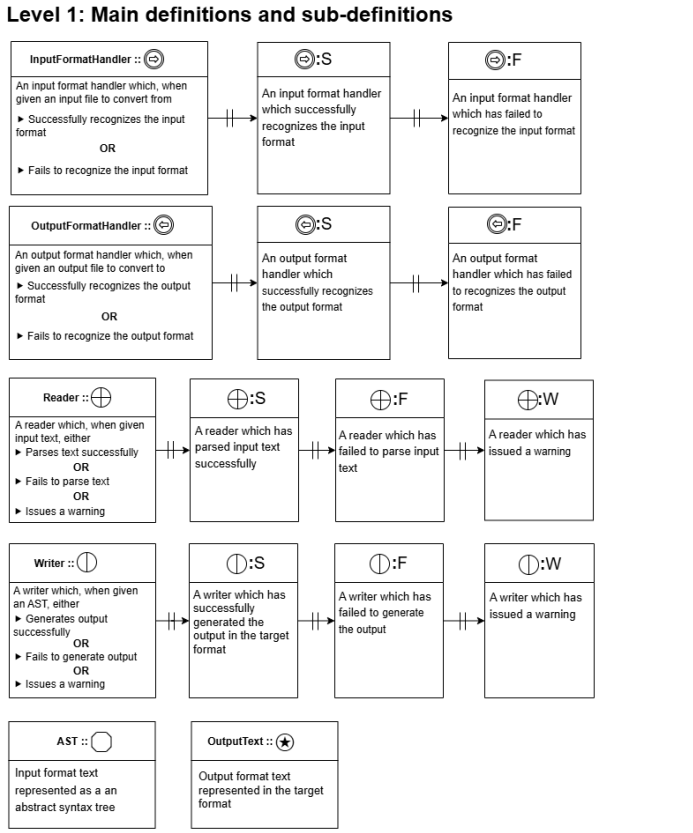
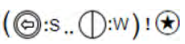


Figure 4.18: Pandoc diagram

- If the  is used, what does it indicate about the input processing ?
- If  occurs, what is the likely outcome ?

These questions were designed to test participants' ability to understand the relationships and functional implications within the model. By focusing on specific symbolic definitions and their transformations, the survey assessed participants' capacity to apply the notation effectively to hypothetical scenarios involving Pandoc's processing workflow. The participant responses and their analysis will be detailed in the Results chapter.

4.5 Case Four

In Case 4, participants were presented with targeted questions focusing on specific symbols and diagrammatic elements to evaluate their understanding and interpretation.

4.5.1 Evaluation of the OR-Arrow Symbol

For the first symbol, the focus was on the **OR-Arrow**, a key component of Level 1 definitions that conveys the possibility of a main definition leading to multiple alternative sub-definitions. The evaluation aimed to assess the symbol's **clarity**, **usability**, and **effectiveness** in representing these relationships.

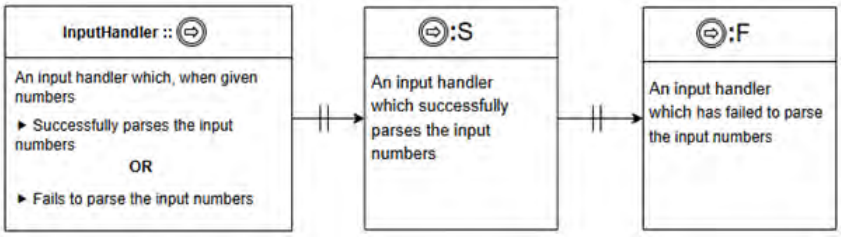


Figure 4.19: Level 1 definition

Participants were provided with a Level 1 diagram, as illustrated in Figure 4.19, which included the OR-Arrow symbol. They were then asked to engage with the diagram and respond to a combination of closed-ended and open-ended questions to provide both qualitative and quantitative feedback. The survey questions included:

1. **Yes/No Questions:**

- Does the OR-Arrow clearly communicate that the main definition can represent multiple alternatives?
- Is the use of the OR-Arrow symbol effective in conveying the either-or relationship between sub-definitions?
- Is there any ambiguity when interpreting the sub-definition relationships using the OR-Arrow symbol?

2. **Open-Ended Question:**

- Please provide reasons for your choices above.

These questions were designed to align with the evaluation criteria established in the methodology. Specifically, they assessed the **clarity** of the OR-Arrow by determining whether it effectively represented relationships within the diagram. They also explored the symbol's **simplicity**, examining whether it was intuitive and easy for participants to interpret. Lastly, the evaluation addressed **usability**, focusing on how well participants could apply the OR-Arrow to understand Level 1 definitions and their implications within the context of the model.

By employing a mixed-methods approach, the survey combined quantitative insights from the closed-ended questions with nuanced feedback from the open-ended responses. This structured evaluation ensured a comprehensive analysis of the OR-Arrow symbol's role in representing relationships. Findings from this case will contribute to refining the modeling notation, ensuring it remains both practical and accessible while maintaining clarity and simplicity for diverse systems and audiences.

4.5.2 Comparative Analysis of the OR-Arrow Symbol

For the second symbol, the evaluation continued to focus on the **OR-Arrow** symbol but introduced a comparative analysis of two different diagrams: Diagram A, representing the current notation, and Diagram B, showcasing an alternative representation of the OR-Arrow symbol. The objective was to determine which diagram more effectively conveyed the relationships between definitions.

Participants were presented with the question: "**Which diagram makes it easier to identify the relationship between definitions?**" Unlike the previous case, this question included an additional option, "Neither," allowing participants to express neutrality if they found neither diagram particularly effective. This approach ensured that participants did not feel compelled to choose a preference between the two notations.

Following this, participants answered a multiple-choice question asking: "**How many sub-definitions are visible in each diagram?**" This question had a single correct answer and was designed to verify participants' ability to interpret and understand the diagrams. Finally, participants were asked to provide feedback with an open-ended question: "**Please provide reasons for your answer.**"

This structured evaluation served multiple purposes. First, it sought to identify which diagram employed the OR-Arrow symbol in a way that was simpler and more effective for participants to comprehend. Second, by including the multiple-choice question, the survey ensured that participants were not only making subjective judgments but also demonstrating an objective understanding of the diagrams. Lastly, the open-ended feedback provided valuable qualitative insights into participants' thought processes, highlighting any challenges or preferences related to the notation. By comparing the two notations

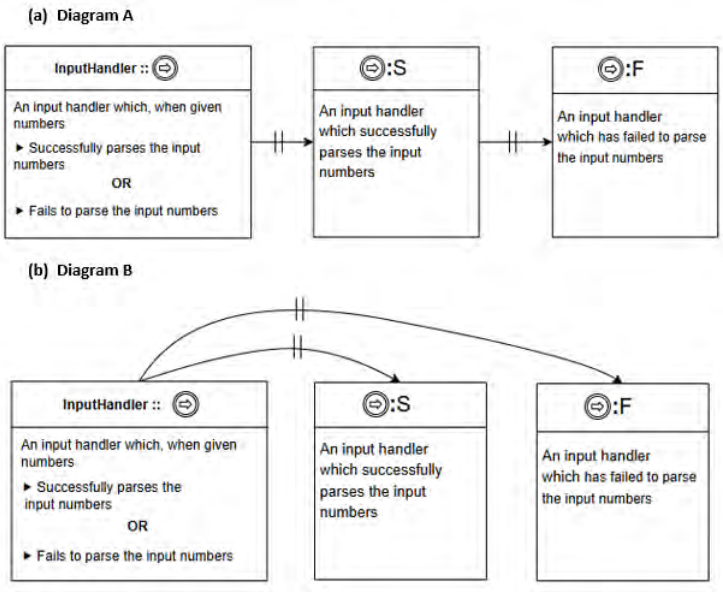


Figure 4.20: Symbol Two

and analyzing the results, this evaluation contributes to refining the design of the OR-Arrow symbol, ensuring it remains intuitive and effective in representing relationships between definitions across diverse system models.

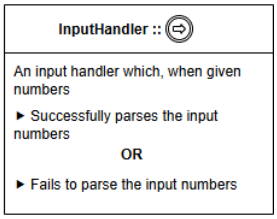


Figure 4.21: Symbol Three

4.5.3 Evaluation of the Notation Label (::) Symbol

For the third symbol, the focus shifted to the notation label (::) symbol, which is used to define the name and corresponding symbol of a main definition (Fig. 4.21). Participants were presented with a diagram featuring the (::) symbol and asked a series of targeted questions to evaluate their understanding of its function and clarity.

The first question was a yes/no inquiry: "**Is the notation InputHandler :: clear in representing the relationship between the name of a definition and its corresponding symbol?**" This question aimed to assess participants' perceptions of the symbol's clarity in establishing the relationship between a definition's name and its visual representation.

Participants were then asked a more specific question: "**What does the symbol represent?**" This multiple-choice question, with only one correct answer, served as a verification step to determine whether participants accurately understood the connection between the name and the symbol as represented in the diagram.

Finally, participants were invited to elaborate on their responses with the open-ended question: "**Please provide reasons for your choices above.**" This question allowed for qualitative feedback, offering insights into how participants interpreted the (::) symbol and any challenges they encountered in doing so.

By combining these questions, the evaluation of the (::) symbol focused on its clarity in representing relationships, its usability in helping participants interpret the diagram, and its overall effectiveness as part of the notation. The structured approach ensured a comprehensive analysis of participants' understanding while also capturing their subjective experiences, contributing valuable input for refining the notation.

4.5.4 Evaluation of Sub-Definition Notation

For the fourth symbol, the focus shifted to the sub-definition notation expressed as symbol: letter, which is used to define subtypes as subsets of a main definition. Participants were presented with a diagram (4.22) featuring this notation and asked a series of questions to evaluate their understanding of its purpose and clarity. The first two questions were yes/no

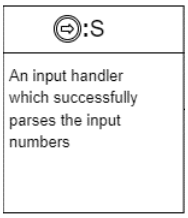


Figure 4.22: Symbol Four

inquiries designed to assess participants' comprehension of the notation's functionality:

- Does the :S notation clearly convey that combining a symbol from the main definition with a letter indicates a sub-definition of that main definition?
- Do you understand the single letter (e.g., S) to intuitively represent the sub-definition's functionality?

These questions aimed to determine whether the symbol:letter notation effectively communicates the relationship between a main definition and its sub-definitions, as well as whether the single-letter representation intuitively conveys the intended meaning.

To verify understanding, participants were then asked a specific question: "**In the :S notation, what does the S represent?**" This multiple-choice question, with only one correct answer, served as a means to ensure participants could accurately interpret the diagram and its symbolic components.

Finally, participants were invited to elaborate on their responses with the open-ended question: "**Please provide reasons for your choices above.**" This question allowed for qualitative insights, offering a deeper understanding of the participants' thought processes and any difficulties they may have encountered.

By combining closed-ended and open-ended questions, the evaluation of the symbol:letter notation aimed to assess its clarity, simplicity, and usability in representing sub-definitions. The feedback gathered from this structured approach will contribute to refining the notation, ensuring it remains intuitive and effective in communicating the hierarchical relationships between definitions.

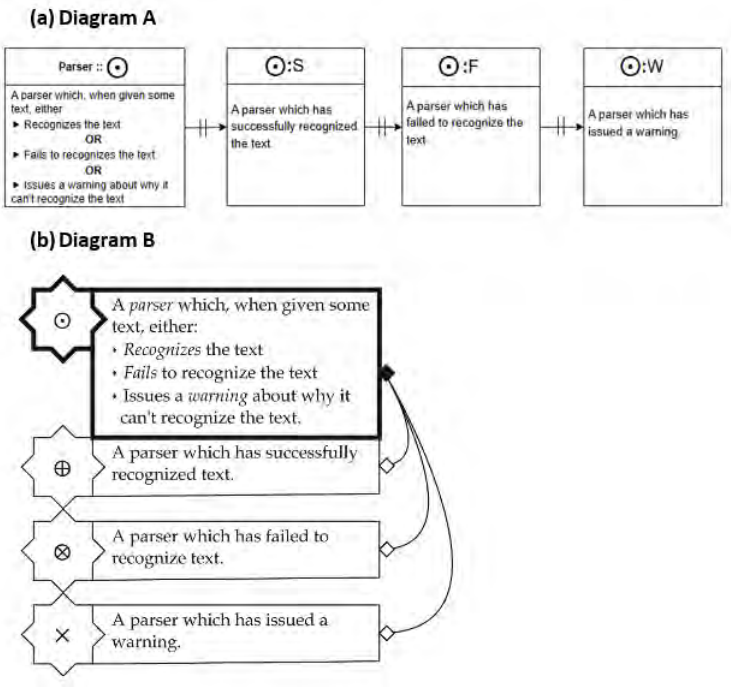


Figure 4.23: Symbol Five

4.5.5 Comparative Evaluation of Diagrams for Main Definitions and Sub-Definitions

For the fifth symbol, the evaluation aimed to compare the effectiveness of two diagrammatic representations, A and B, in conveying main definitions and their sub-definitions (Fig. 4.23). Diagram A utilized the newly proposed notation, designed to enhance clarity and usability, while Diagram B adhered to the older notation developed by Motara. Both diagrams represented the same conceptual information but differed in their visual and structural approaches. The goal was to determine whether the new notation offered improvements over the older approach in terms of understanding relationships and recall.

Participants were asked targeted questions such as "**Which diagram makes it easier to identify the relationship between definitions?**" and "**Which diagram makes it easier to recall what the symbols represent?**" These questions sought to assess the relative clarity, simplicity, and effectiveness of the new notation in visually representing hierarchical relationships. To ensure a comprehensive evaluation, participants were also encouraged to provide qualitative feedback through an open-ended question: "**Please provide the reasons for your choices above.**" This allowed the study to capture nuanced insights into participants' thought processes, preferences, and any challenges they encountered. By comparing the new and old approaches, the analysis identified specific

strengths and weaknesses of the new notation, contributing to its further refinement and improvement for practical use.

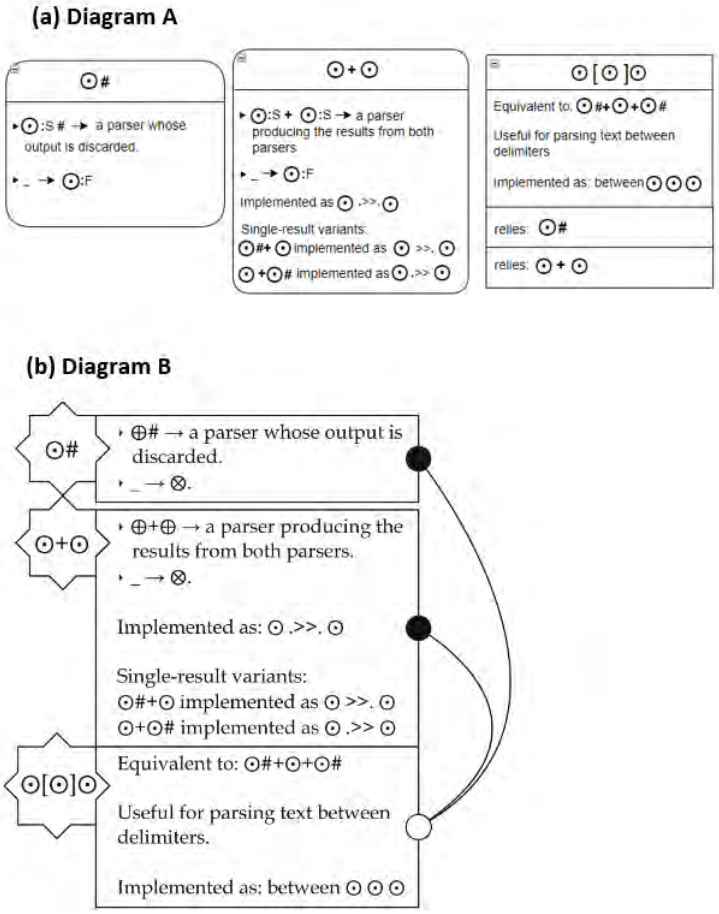


Figure 4.24: Symbol Six

4.5.6 Evaluation of Relationships at Level 2

For the sixth symbol, the focus was on evaluating how relationships at Level 2 are represented within the diagrams. Participants were provided with two diagrams, A and B, where Diagram A utilized the new notation, and Diagram B reflected the older notation proposed by Motara (Fig. 4.24). Both diagrams illustrated the concept of a "relies" relationship between definitions. Participants were asked to respond to the question, "**Which notation conveys that there is a relationship between definitions?**", with the additional option to select "neither" to avoid forcing a preference. This was followed by an open-ended question, "**Please provide the reasons for your choices above**," aimed at capturing participants' reasoning and insights. This approach allowed for a thorough

evaluation of how effectively each notation communicates relationships, offering valuable feedback for refining the clarity and usability of the new notation.

4.5.7 Holistic Comparison of Diagrammatic Notations

For the seventh symbol, participants were provided with full diagrams of the FParsec system, one rendered in the old notation (Fig. 4.25) and the other in the new notation (Fig. 4.16). The objective was to perform a comprehensive evaluation of the new notation's overall clarity, usability, and practicality in comparison to the older notation. This was achieved through a series of targeted questions designed to capture participants' preferences and insights about the two notations. Each question was carefully followed by an open-ended prompt, asking participants to explain the reasoning behind their choices. This dual approach ensured both quantitative and qualitative feedback, which was critical for identifying areas of strength and improvement in the new notation.

The first question, "**Which diagram makes it easier to keep track of the symbols?**" aimed to assess the cognitive ease with which participants could recognize and follow the symbols throughout the diagrams. By including the "neither" option, participants were not compelled to choose between the diagrams if they found both equally challenging or effective. The corresponding open-ended question allowed participants to elaborate on their reasoning, providing insights into whether the symbols in the new notation were distinct, memorable, and consistently represented.

The second question, "**Which diagram makes it easy to mentally map the symbol to its underlying concept?**" focused on the intuitiveness of the notations. This question sought to determine whether participants could readily associate each symbol with its intended meaning or functionality, a critical factor for ensuring the practical usability of the notation. The open-ended responses here were particularly valuable in highlighting whether the new notation successfully bridged the gap between abstract representation and conceptual understanding.

The third question, "**Which notation makes it easier to understand the relationships between components?**" addressed the visual and structural clarity of the diagrams. This question was essential for evaluating how well the new notation conveyed the interconnections and dependencies between various components within the FParsec system. Participants' justifications for their choices provided qualitative data on whether the new notation effectively improved the depiction of these relationships compared to the older notation.

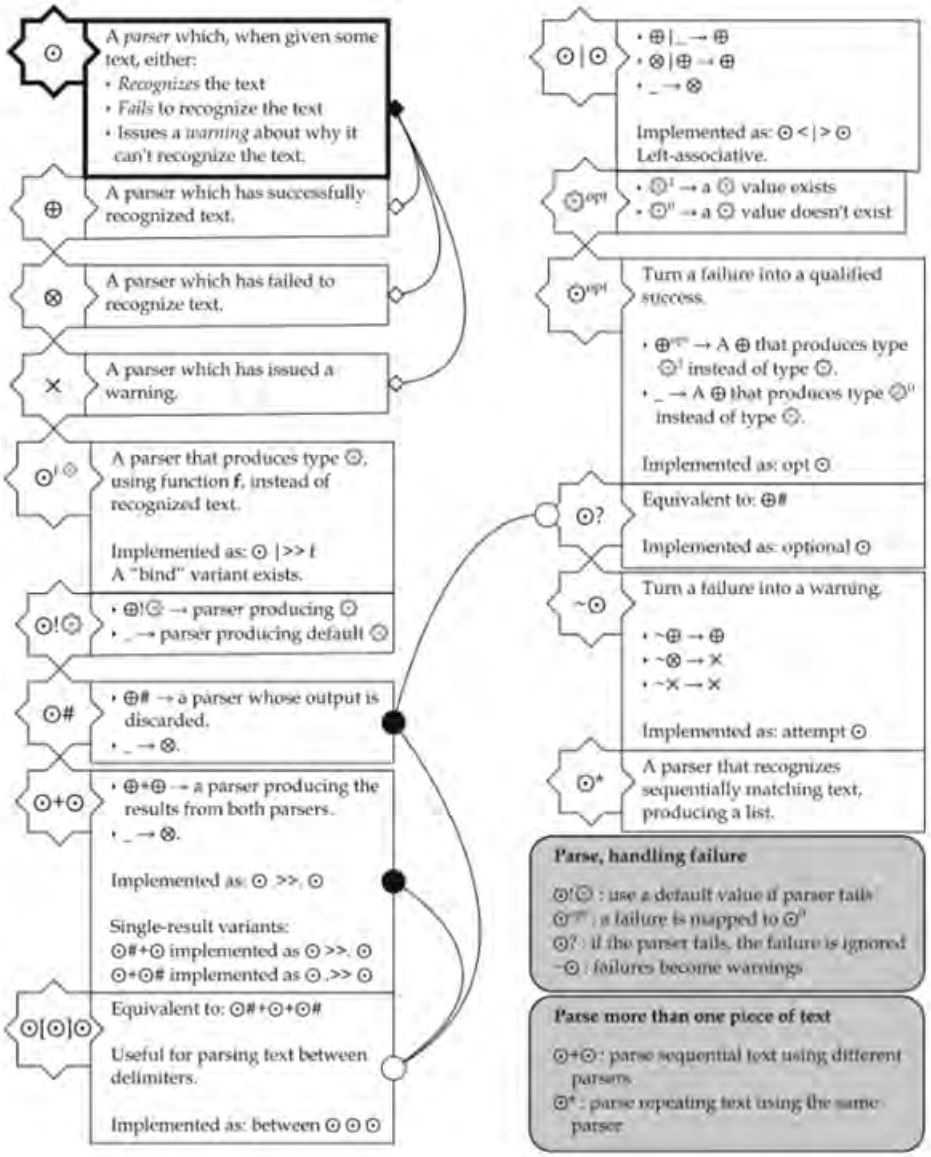


Figure 4.25: Symbol 7 (diagram A) (Motara 2021)

The fourth question, "**Which notation organizes the components and relationships more clearly in terms of layout and flow?**" evaluated the overall diagrammatic structure, focusing on the readability and logical progression of the representations. Clear organization is a vital characteristic of any notation, as it directly impacts users' ability to navigate and interpret complex systems. The open-ended follow-up allowed participants to critique specific aspects of the layout, such as spacing, alignment, and visual hierarchy, offering actionable feedback for refinement.

The final question, "**Which notation is more flexible in adapting to changes in the system's structure or requirements?**" examined the adaptability of the nota-

tions. This question aimed to assess whether the new notation was versatile enough to accommodate modifications in the system's architecture, a crucial factor for its applicability in dynamic environments. Participants' explanations provided insights into the perceived scalability and extensibility of the notation.

Overall, this evaluation of the seventh symbol incorporated a thorough comparison of the old and new notations across multiple dimensions, from clarity and organization to adaptability and conceptual mapping. By integrating closed-ended questions with open-ended prompts, the study captured a holistic view of participants' preferences and experiences, ensuring that the new notation could be fine-tuned to better meet user needs and enhance its practical applicability.

4.6 Conclusion

This chapter detailed the implementation of a refined structural modeling notation guided by the Physics of Diagrams (PoD) framework, introducing key modifications to enhance cognitive fit, clarity, and usability. A three-tiered leveling system was implemented—separating core definitions (Level 1), derived manipulations (Level 2), and stakeholder summaries (Level 3)—to reduce cognitive load and improve modularity. Visual differentiation was achieved through distinct shapes: square rectangles for foundational definitions, rounded rectangles for derived operations, and shaded rectangles for thesaurus groupings, enhancing perceptual discriminability. Symbol definition clarity was prioritized via explicit *'label :: symbol'* notation (e.g., parser :: ⊙), ensuring unambiguous mapping between concepts and visual representations. Sub-definition notation was streamlined (e.g., ⊙:F for a failed parser) to minimize symbol proliferation while leveraging intuitive single-letter codes. The new OR-Arrow symbol replaced vertical branching with horizontal chaining to represent alternative subtypes, optimizing spatial efficiency. Additionally, "relies" relationships were embedded directly within definition blocks to eliminate dependency-line clutter.

These refinements were rigorously applied across three case studies: FParsec modeled parser combinators, emphasizing failure handling and text transformations; Docutils captured reStructuredText processing workflows, demonstrating adaptability to non-functional systems; and Pandoc illustrated AST-based document conversion, validating scalability for complex functional architectures. To evaluate effectiveness, a multi-faceted survey was designed, integrating case-specific questions (testing comprehension of definitions,

manipulations, and workflows), symbol-focused evaluations (assessing OR-Arrow clarity, :: notation, and sub-definition syntax), and comparative analyses pitting the revised notation against Motara's original (see Appendix B). This implementation establishes a robust foundation for empirical validation, ensuring the notation aligns with PoD principles while addressing real-world modeling challenges across diverse systems.

Chapter 5

Results and Discussion

This chapter presents the findings from the case studies and survey evaluations, analyzing how the new structural modeling notation performed in practice. It includes qualitative and quantitative assessments of the notation's clarity, usability, and effectiveness, based on participant feedback. Appendix B presents the full questionnaire.

5.1 Participant Overview and General Performance

This section summarizes the demographics of the participants, their overall task performance, and performance trends across different participant categories. These provide a contextual baseline for interpreting results in the specific case studies that follow. This is followed by a discussion of the evaluation outcomes and the subsequent actions taken to refine the notation and improve its overall presentation and usability.

5.1.1 Participant Demographics

The study involved a total of 30 participants from various professional backgrounds. Participants were asked to self-report their primary role and their level of familiarity with the subject matter (*'Beginner'*, *'Intermediate'*, or *'Advanced'*).

A detailed breakdown of participant categories is presented in Table 5.1. The participant pool was primarily composed of *'Developers'* (n=13) and *'Students'* (n=14). The remaining participants identified as *'Data Engineer Intern'* (n=1), *'Business Analyst'*

(n=1), and *'Author'* (n=1), and are grouped into *'Other roles'* (n=3). Among the two largest groups, developers were distributed across all familiarity levels, while the majority of students (n=9) identified as beginners.

Table 5.1: Breakdown of Participant Role and Familiarity

Role	**Advanced**	**Beginner**	**intermediate**	**Total**
Developer	3	5	5	**13**
Student	0	9	5	**14**
Other	0	3	0	**3**
Total	**3**	**17**	**10**	**30**

5.1.2 Overall Task Performance

To quantify participant performance, a composite score was calculated for each individual. This score represents the sum of correctly answered items across the four experimental cases. The maximum possible score was 26 (Case One: 7, Case Two: 8, Case Three: 7, Case Four: 4). For comparative analysis, this raw score was then converted into an overall percentage. Table 5.2 presents the individual participant scores, ordered from highest to lowest.

The distribution of these overall performance scores is presented in Figure 5.1. The results indicate a strong performance from the majority of participants, with a pronounced clustering of scores above 70%. The distribution is negatively skewed, suggesting that most participants found the notation understandable. However, the presence of a tail in the lower score range indicates that a subset of participants experienced significant difficulties, which answers the question of whether confusion was widespread or isolated to a smaller group.

5.1.3 Performance Analysis by Participant Category

To investigate the influence of professional background and self-reported familiarity on performance, the mean scores for different participant sub-groups were compared.

As shown in Table 5.3, a clear difference in performance was observed when categorizing by role. *'Developer'* participants achieved the highest average score (M = 84.0%), while *'Other'* roles scored lowest (M = 48.7%).

Table 5.2: Individual Participant Scores (Sorted High→Low)

Role	Familiarity	Score
Developer	Beginner	100.00
Developer	Beginner	100.00
Student	Beginner	100.00
Developer	Advanced	96.15
Student	Beginner	96.15
Student	Beginner	92.31
Developer	Advanced	88.46
Developer	Beginner	88.46
Developer	Intermediate	88.46
Developer	Intermediate	88.46
Developer	Intermediate	88.46
Student	Intermediate	88.46
Developer	Beginner	84.62
Student	Beginner	84.62
Student	Beginner	84.62
Student	Intermediate	80.77
Data Engineer Intern	Beginner	76.92
Developer	Intermediate	76.92
Student	Beginner	76.92
Student	Intermediate	76.92
Student	Beginner	73.08
Developer	Intermediate	69.23
Student	Beginner	69.23
Developer	Advanced	65.38
Developer	Beginner	57.69
Student	Intermediate	57.69
Business Analyst	Beginner	50.00
Student	Beginner	34.62
Author	Beginner	19.23
Student	Intermediate	19.23

Table 5.3: Performance Score by Participant Role

Role	Average Score (%)
Developer	84.0
Student	73.9
Other	48.7

Further analysis based on self-reported familiarity revealed a less predictable trend (Table 5.4). Participants identifying as *'Advanced'* (M = 83.3%) and *'Beginner'* (M = 75.8%) outperformed those who identified as *'Intermediate'* (M = 73.5%).

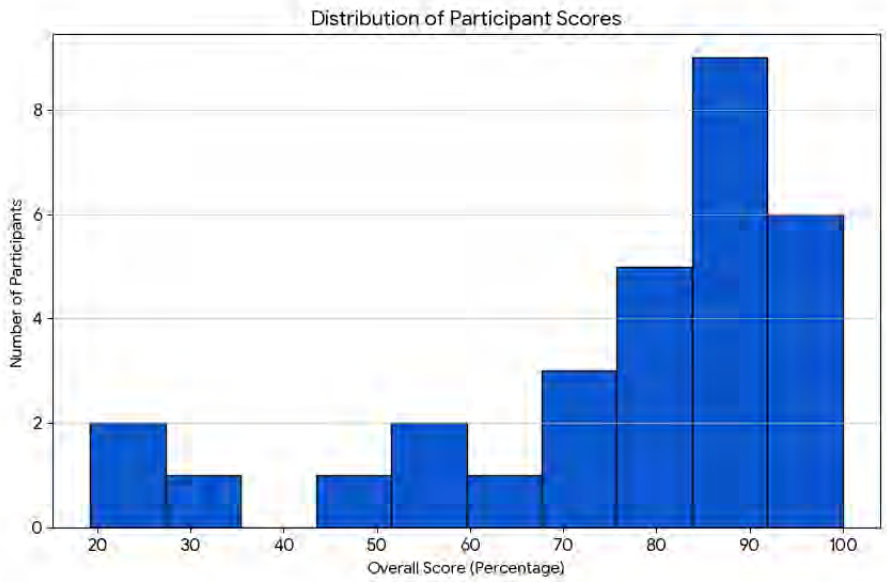


Figure 5.1: Distribution of Overall Participant Performance Scores.

Table 5.4: Mean Performance Score by Familiarity Level

Familiarity	**Average Score (%)**
Advanced	83.3
Beginner	75.8
Intermediate	73.5

A more granular breakdown of performance by both role and familiarity is provided in Table 5.5. This analysis further illuminates the trend observed in the student subgroup, where *'Beginner'* students (M = 79.1%) on average scored notably higher than their *'Intermediate'* counterparts (M = 64.6%). In contrast, developer performance was consistently high across all self-reported familiarity levels. The *'Other'* category, comprising non-developer and non-student roles, recorded the lowest average performance (M = 48.7%).

Table 5.5: Mean Performance Score (%) by Role and Familiarity

Role	**Advanced**	**Beginner)**	**Intermediate**
Developer	83.3	86.2	82.3
Student	-	79.1	64.6
Other	-	48.7	-

A more granular breakdown of performance by both role and familiarity is provided in Table 4.4. This analysis further illuminates the trend observed in the student subgroup, where *'Beginner'* students (M = 79.1%) on average scored notably higher than

their *'Intermediate'* counterparts (M = 64.6%). In contrast, developer performance was consistently high across all self-reported familiarity levels. The *'Other'* category, comprising non-developer and non-student roles, recorded the lowest average performance (M = 48.7%).

5.1.4 Response Categorization Criteria

To analyze participant answers consistently across all case studies, responses to questions with multiple correct answers are classified into the following categories:

- **Correct**: Participants identify all correct options.
- **Partially Correct**: Participants identify some but not all of the correct options, and or including other options.
- **Overinclusive Responses**: Participants identify all the correct options including other distractor options
- **Incorrect Responses**: Participants identify incorrect options only.

5.2 Case One

Case 1 focused on modeling the functional programming system **FParsec** using the developed modeling notation and evaluating participants' comprehension through seven questions. This section presents the results for each question, highlighting performance trends and descriptive statistics.

Average Score: 69.05% across all questions.

5.2.1 Q1: Recognition of Main Definitions

The objective of this question is to evaluate participants' ability to identify the main definitions in the Fparsec diagram (see Fig. 4.16). Figure 5.3 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.2.

Correct Responses:

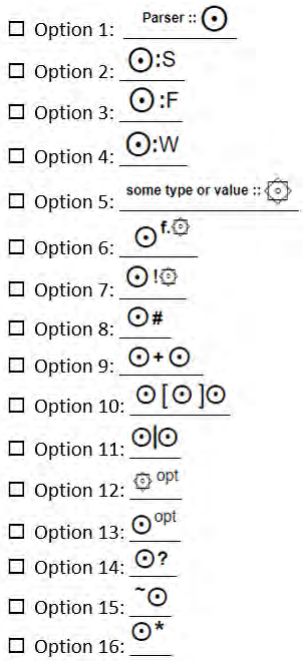


Figure 5.2: FParsec Q1, Q2, and Q3 options.

- **15 participants (50%)** correctly identified both Option 1 (parser) and Option 5 (some type or value) as the main definitions.

Partially Correct Responses:

- **9 participants (30%)** identified Option 1 but did not include Option 5 or included additional options. Examples include:
  - Option 1, Option 8 (Count: 1).
  - Option 1, Option 2, Option 3, Option 4, Option 7 (Count: 1).
  - Option 1, Option 2, Option 3, Option 4, Option 6–16 (Count: 1).
  - Option 1, Option 2, Option 3, Option 4 (Count: 1).
  - Option 1 only (Count: 5).

Overinclusive Responses:

- **5 participants (16.67%)** selected unrelated distractor options alongside correct answers. Examples include:
  - Option 1, Option 2, Option 3, Option 4, Option 5 (Count: 5).

Incorrect Responses:

- **1 participant (3.33%)** provided responses excluding both correct definitions (for example, Option 2, Option 3, Option 4, Option 8, Option 14 (Count: 1)).

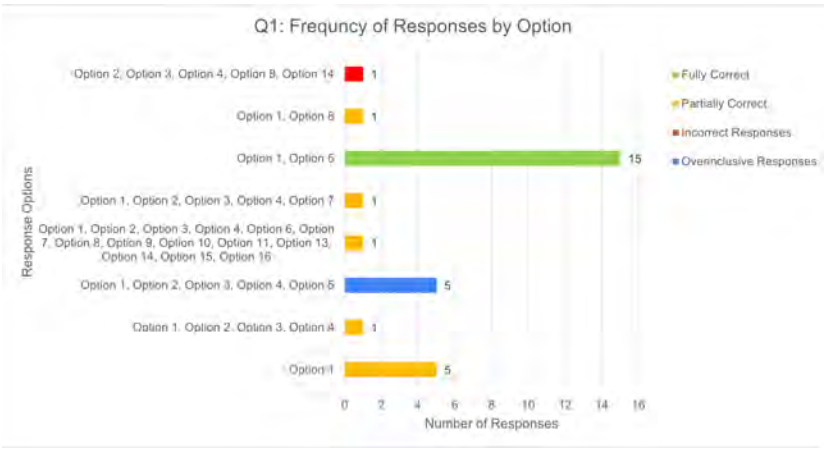


Figure 5.3: Case One (Q1) Frequency of responses by option.

Trends:

- Option 1 (reader) was the most frequently selected definition (identified by 96.67% of participants).
- Option 5 (some type or value) was selected by 66.67% of participants, indicating slightly lower recognition compared to Option 1.

5.2.2 Q2: Identifying Sub-Definitions

Figure 5.4 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.2.

Correct Responses:

- **18 participants (60%)** correctly identified all three sub-definitions: Options 2, 3, and 4.

Partially Correct Responses:

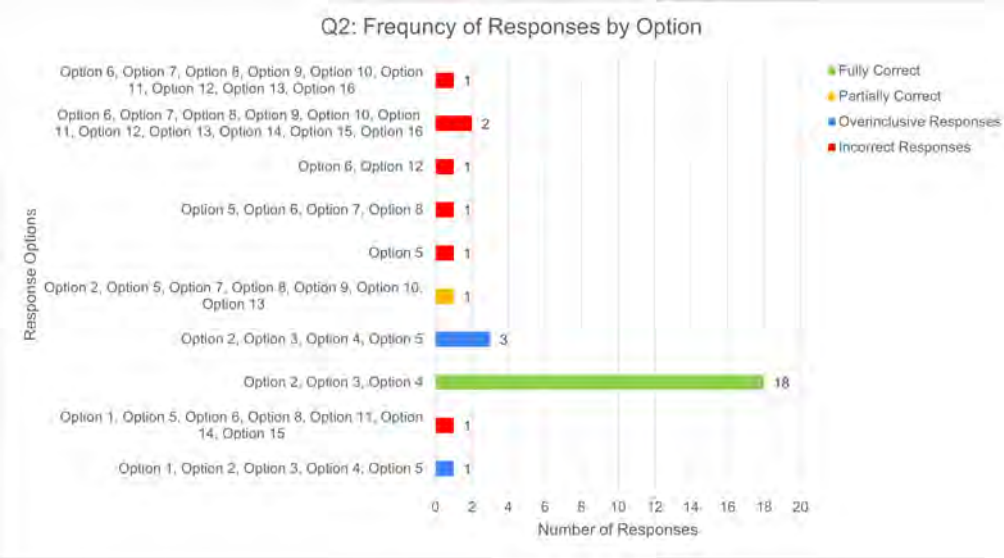


Figure 5.4: Case One (Q2) Frequency of responses by option.

- **1 participant (3.33%)** identified Option 2 but did not include Option 3 and 4 and included additional options. Examples include:
  - Option 2, Option 5, Option 7, Option 8, Option 9, Option 10, Option 13 (Count: 1).

Overinclusive Responses:

- **4 participants (13.33%)** selected all of options 2, 3, and 4 and other incorrect options. Examples include:
  - Option 2, Option 3, Option 4, Option 5 (Count: 3).
  - Option 1, Option 2, Option 3, Option 4, Option 5 (Count: 1).

Incorrect Responses:

- **7 participants (23.33%)** provided responses excluding all three correct sub-definitions
  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 16 (Count: 1).
  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14, Option 15, Option 16 (Count: 2).
  - Option 6, Option 12 (Count: 1).

- Option 5, Option 6, Option 7, Option 8 (Count: 1).
- Option 5 (Count: 1).
- Option 1, Option 5, Option 6, Option 8, Option 11, Option 14, Option 15 (Count: 1).

Trends:

- The most prominent trend is the strong performance of the participants, with 60% correctly identifying all three sub-definitions (Options 2, 3, and 4). This suggests a good overall understanding of the core concept being tested.
- Option 5 appears in the overinclusive, partially correct, and several incorrect responses. This reinforces the idea that Option 5 acted as a significant distractor, pulling respondents away from the completely correct answer.
- Overinclusive responses frequently included unrelated distractors, particularly Options 6 through 16, indicating a tendency toward over-selection in these cases.
- The presence of overinclusive responses (13.33%) suggests that these participants had a partial understanding of the concept, correctly identifying all the core components (Options 2, 3, and 4) but then adding an incorrect option, most frequently Option 5.

5.2.3 Q3: Identify all the definitions that rely on other definitions

Figure 5.5 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.2.

Correct Responses:

- **14 participants (46.67%)** correctly identified both Options 10 and 14 as the level 2 definitions that rely on other definitions.

Partially Correct Responses:

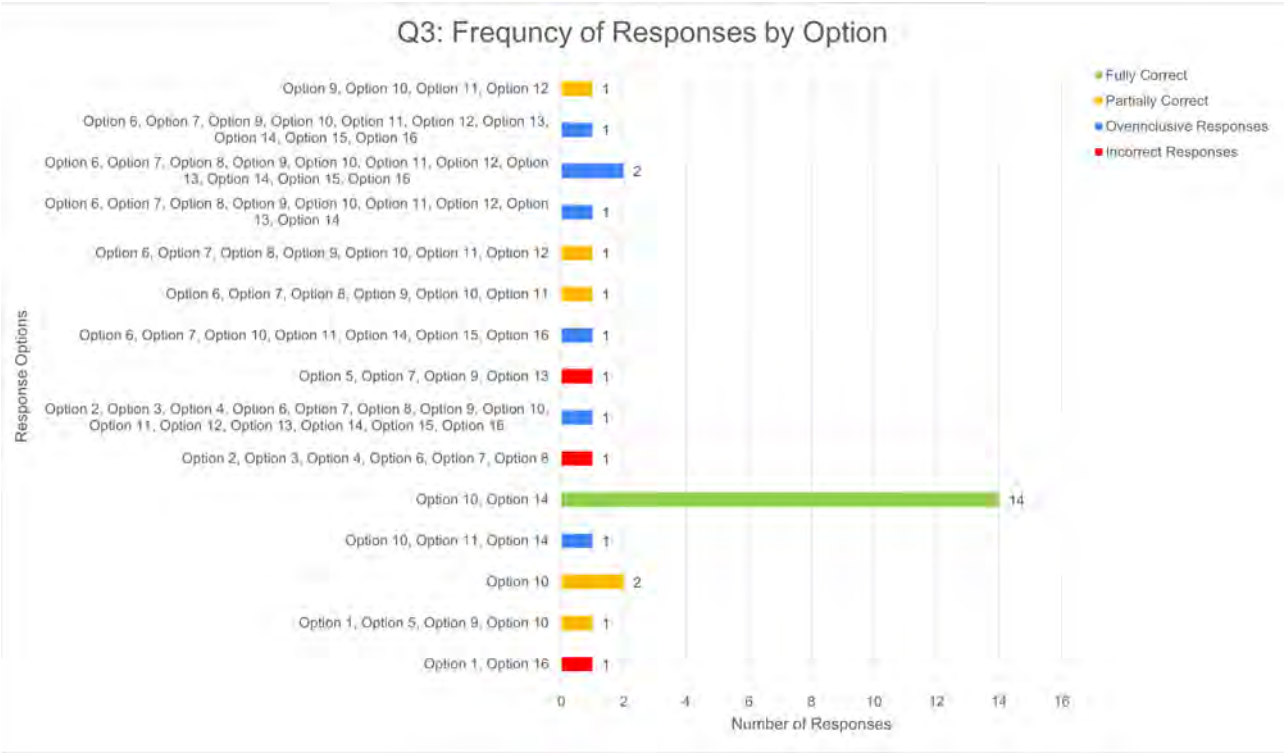


Figure 5.5: Case One (Q3) Frequency of responses by option.

- **6 participants (20%)** selected only one of the correct options (Option 10) . Examples include:
  - Option 9, Option 10, Option 11, Option 12 (Count: 1).
  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12 (Count: 1).
  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11 (Count: 1).
  - Option 10 (Count: 2).
  - Option 1, Option 5, Option 9, Option 10 (Count 1).

Overinclusive Responses:

- **7 participants (23.33%)** selected both of options 10 and 14 and other incorrect options. Examples include:
  - Option 6, Option 7, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14, Option 15, Option 16 (Count: 1).

  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14, Option 15, Option 16 (Count: 2).
  - Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14 (Count: 1).
  - Option 6, Option 7, Option 10, Option 11, Option 14, Option 15, Option 16 (Count: 1).
  - Option 2, Option 3, Option 4, Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14, Option 15, Option 16 (Count: 1).
  - Option 10, Option 11, Option 14 (Count: 1).

Incorrect Responses:

- **3 participants (10%)** provided responses excluding both correct options. Examples include:
  - Option 5, Option 7, Option 9, Option 13 (Count: 1).
  - Option 2, Option 3, Option 4, Option 6, Option 7, Option 8 (Count: 1).
  - Option 1, Option 16 (Count: 1).

Trends:

- Compared to the previous data, the overall success rate (46.7%) is significantly lower. This suggests the question was more difficult or the concepts being tested were less well understood.
- Examining the partially correct responses, Option 10 was selected more frequently than Option 14. This suggests Option 10 might have been more salient or easier to identify as a correct option. Two participants selected Option 10 alone.
- A notable trend is the high number of overinclusive responses often contained long lists of unrelated options. This suggests some participants may have been unsure and selected a large number of options to increase their chances of including the correct ones, rather than demonstrating a clear understanding.

5.2.4 Q4: What does the definition ⊙!⚙ signify in the context of parsing ?

Objective: Test participants' understanding of a specific Level 2 definition. Figure 5.5 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option.

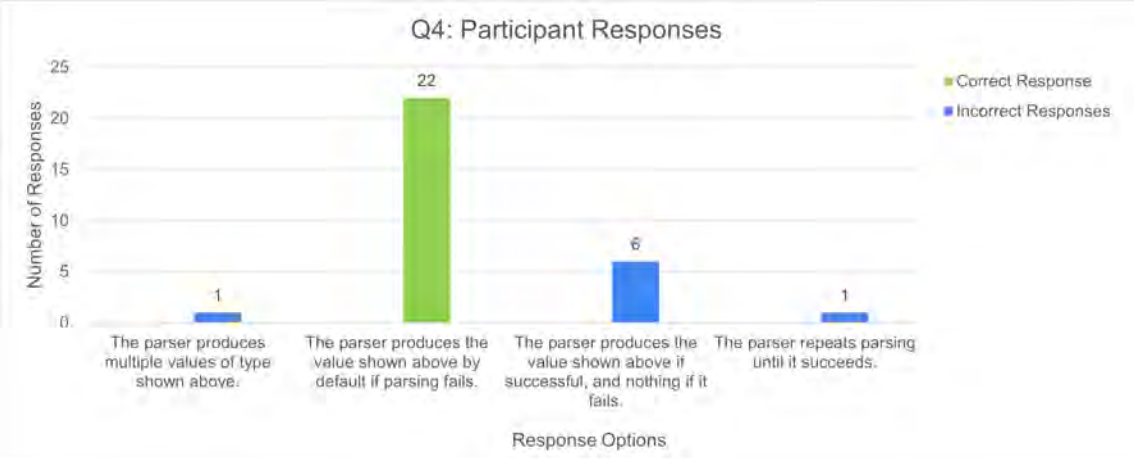


Figure 5.6: Case One (Q4) Frequency of responses by option.

Correct Responses:

- **22 participants (73.33%)** correctly interpreted the definition as *"the parser produces the value shown above by default if parsing fails."*

Incorrect Responses:

- **6 participants (20%)** selected *"the parser produces the value shown above if successful, and nothing if it fails."*
- **1 participant each (3.33%)** selected the distractor options:
  - *"The parser produces multiple values of the type shown above."*
  - *"The parser repeats parsing until it succeeds."*

Trends:

- The most significant trend is the high percentage of correct answers (73.33%). This suggests that the concept being tested was generally well understood by the participants.
- The option *"The parser produces the value shown above if successful, and nothing if it fails"* was chosen by 6 out of the 8 incorrect respondents. This indicates that this option was a particularly effective distractor. It likely appealed to those who had a partial understanding of parsing or were unsure about the specific behavior in case of failure.

5.2.5 Q5: In ⚙opt, what does the presence of ⚙1 versus ⚙0 indicate ?

Objective: Test participants' understanding of symbols indicating presence versus absence. Figure 5.7 shows a chart that representing the percentage distribution of responses between the two options, highlighting the higher selection rate for the correct answer.

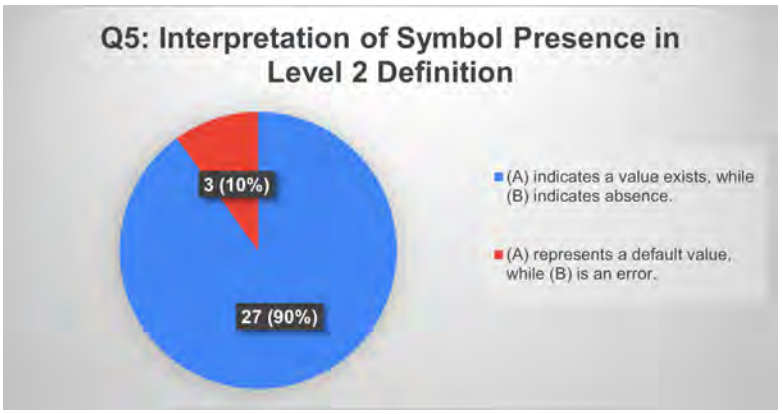


Figure 5.7: Case One (Q5) Interpretation of Symbol Presence in Level 2 Definition.

Trends:

- The (27 out of 30, or 90%) success rate indicates a very strong understanding of the concept being tested. This suggests the question was either very straightforward or the concept was exceptionally well understood.

5.2.6 Q6: How would you interpret (⊙:S#) + (⊙!⚙) ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.8 shows a chart that representing the percentage distribution of responses between the two options, highlighting the higher selection rate for the correct answer.

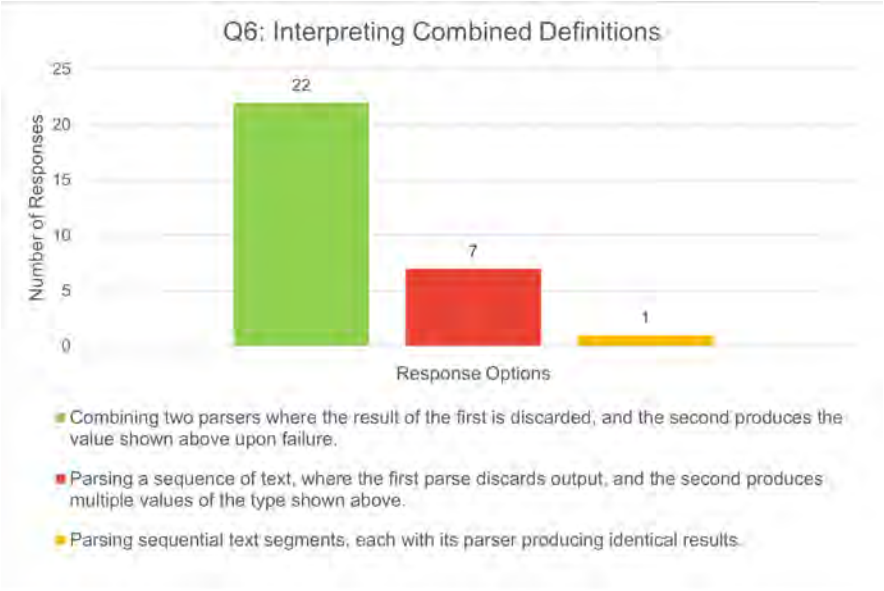


Figure 5.8: Case One (Q6) Response Distribution for Combined Level 2 Definitions.

Correct Responses:

- **22 participants (73.33%)** correctly interpreted the definition as *"combining two parsers where the result of the first is discarded, and the second produces the value shown above upon failure."*

Incorrect Responses:

- **7 participants (23.33%)** selected *"parsing a sequence of text, where the first parse discards output, and the second produces multiple values of the type shown above."*
- **1 participant (3.33%)** selected *"parsing sequential text segments, each with its parser producing identical results."*

Trends:

- High comprehension was reflected by the majority selecting the correct answer (73.33%).
- The option *"Parsing a sequence of text, where the first parse discards output, and the second produces multiple values of the type shown above"* was chosen by 7 out of the 8 incorrect respondents. This suggests this option acted as a strong distractor, likely appealing to those who understood the discarding aspect but misinterpreted the second parsers behavior (producing multiple values instead of a value on failure).
- Both Q4 and Q6 have similar success rates (73.3% and 73.3%, respectively) and both feature one particularly strong distractor. This might suggest a common underlying issue in understanding specific parsing behaviors, particularly around failure scenarios.

5.2.7 Q7: How would $(\odot\,[\odot]\,\odot)^{\text{opt}}$ behave in the context of parsing ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.9 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option.

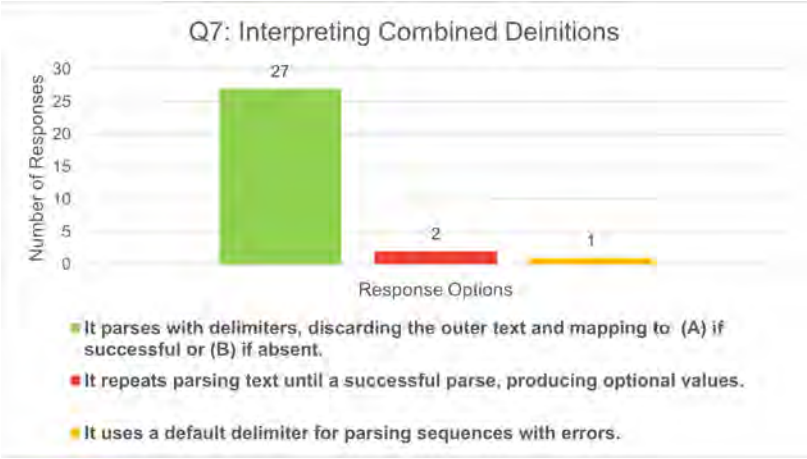


Figure 5.9: Case One (Q7) Response Distribution for Combined Level 2 Definitions.

Trends:

- The (27 out of 30, or 90%) success rate indicates a strong understanding of the concept being tested. This is consistent with the high success rate seen in Q5 (also 90%).

5.3 Case Two

Case Two focused on modeling the system **Docutils** using the developed modeling notation and evaluating participants' comprehension through eight questions. This section presents the results for each question, highlighting performance trends and descriptive statistics.

Average Score: 79.58% across all questions.

5.3.1 Q1: Recognition of Main Definitions

The objective of this question is to evaluate participants' ability to identify the main definitions in the Docutils diagram (see Fig. 4.17). Figure 5.11 displays the frequency

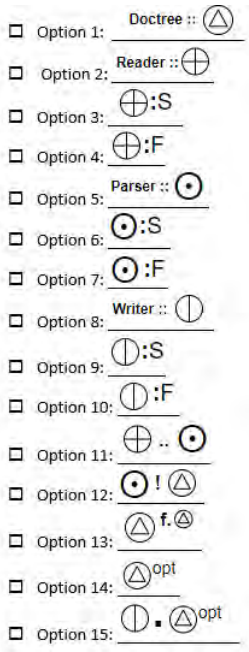


Figure 5.10: Docutils Q1, and Q2 options.

of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

Correct Responses:

- **23 participants (76.67%)** correctly identified Option 1 (doctree), Option 2 (reader), Option 5 (parser) and Option 8 (writer) as the main definitions.

Partially Correct Responses:

- **4 participants (13.33%)** identified some of the correct options including other options. Examples include:
  - Option 2, Option 5, Option 8 (Count 2).
  - Option 2, Option 3, Option 4, Option 8, Option 9, Option 10 (Count 1).
  - Option 1, Option 4, Option 5, Option 6, Option 9, Option 14 (Count 1).

Overinclusive Responses:

- **3 participants (10%)** identified all of the correct options including other distractor options. Examples include:
  - Option 1, Option 2, Option 3, Option 4, Option 5, Option 6, Option 7, Option 8, Option 9, Option 10, Option 14 (Count 1).
  - Option 1, Option 2, Option 3, Option 4, Option 5, Option 6, Option 7, Option 8, Option 9, Option 10 (Count 2).

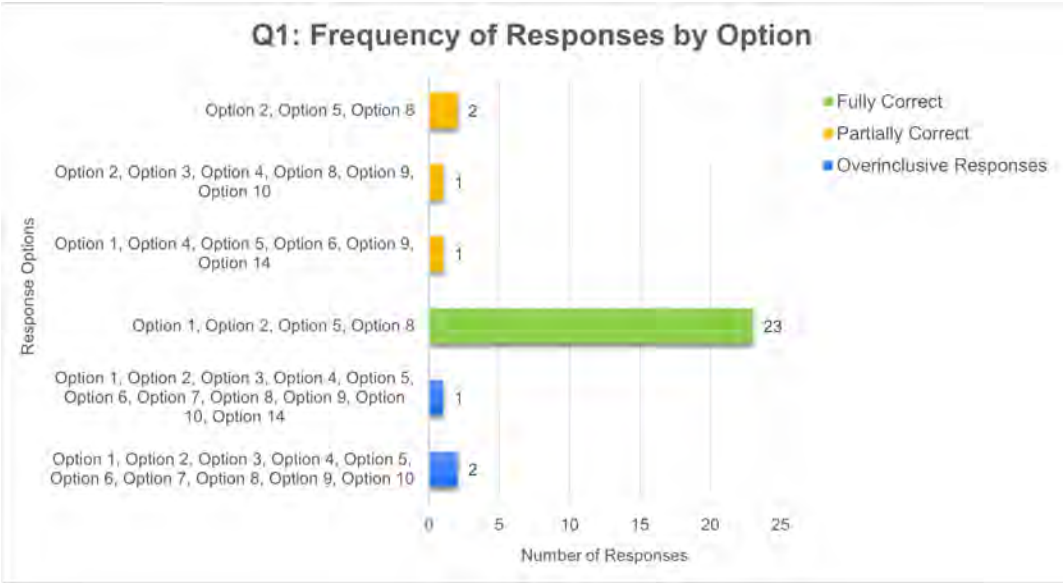


Figure 5.11: Case Two (Q1) Frequency of responses by option.

Trends:

- The most prominent trend is the strong overall performance on this question. A large majority (23 out of 30, or approximately 76.67%) of participants correctly identified all four main definitions. This suggests that the key concepts being tested were generally well understood.
- The four partially correct responses indicate that some participants grasped some of the key definitions but struggled to identify all four correctly. This suggests partial understanding of the concepts, with some confusion or difficulty in distinguishing between closely related concepts.
- The fact that no participant selected only incorrect options is a positive sign. It suggests that while there was some uncertainty leading to overinclusion or partial correctness, no one was completely off-base in their understanding of the main definitions.
- A significant observation is that most responses included options up to and including Option 10. Given that these options represent the building blocks or "level one" elements of the notation, this suggests that participants primarily focused on these fundamental components. This concentration on the foundational elements indicates a good grasp of the basic building blocks of the notation.

5.3.2 Q2: Identifying Sub-Definitions

Figure 5.12 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

Correct Responses:

- **23 participants (76.67%)** correctly identified all sub-definitions (Options 3, 4, 6, 7, 9, 10).

Partially Correct Responses:

- **5 participants (16.67%)** identified some of the correct options including other options. Examples include:
  - Option 8, Option 9, Option 10 (Count 1).

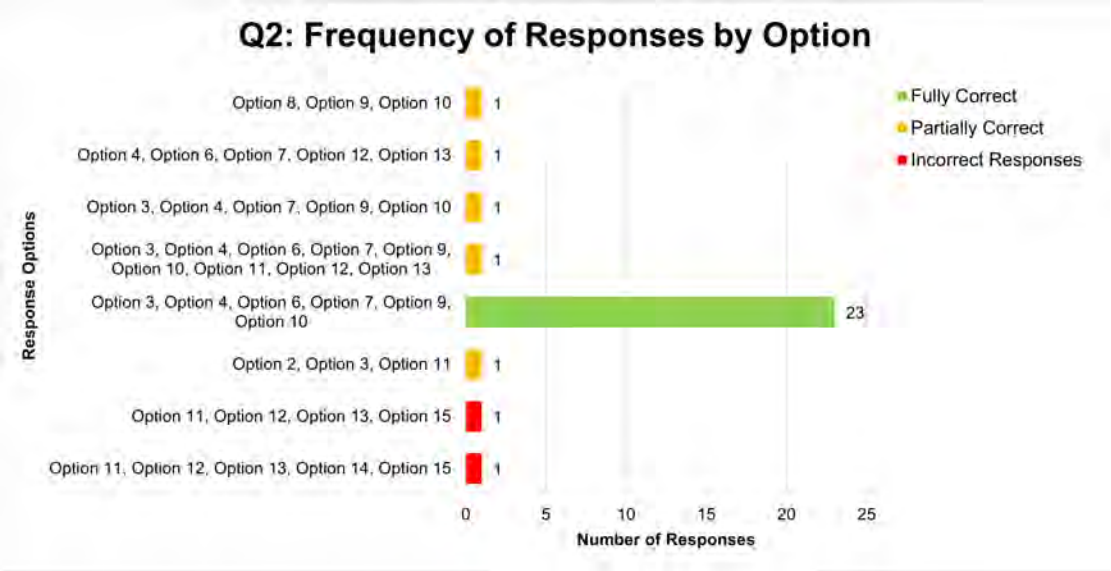


Figure 5.12: Case Two (Q2) Frequency of responses by option.

- Option 4, Option 6, Option 7, Option 12, Option 13 (Count 1).
- Option 3, Option 4, Option 7, Option 9, Option 10 (Count 1).
- Option 3, Option 4, Option 6, Option 7, Option 9, Option 10, Option 11, Option 12, Option 13 (Count 1).
- Option 2, Option 3, Option 11 (Count 1).

Incorrect Responses:

- **2 participants (6.67%)** identified none of the correct options including other options. Examples include:
  - Option 11, Option 12, Option 13, Option 15 (Count 1).
  - Option 11, Option 12, Option 13, Option 14, Option 15 (Count 1).

Trends:

- Similar to Q1, a very strong majority (23 out of 30, approximately 77%) of participants correctly identified all six sub-definitions (Options 3, 4, 6, 7, 9, and 10). This indicates a good understanding of the sub-definitions within the Docutils diagram.

5.3.3 Q3: What is the result of ⊕:S ..⊙:S ?

Objective: Test participants' understanding of a specific Level 2 definition. Figure 5.13 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

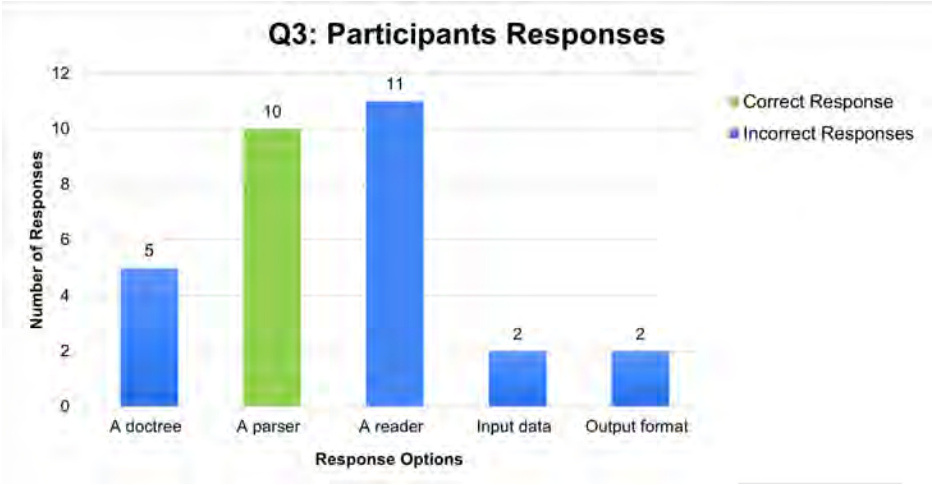


Figure 5.13: Case Two (Q3) Frequency of responses by option.

Correct Responses:

- **10 participants (33.33%)** identified *"A parser"* as the result of the notation.

Incorrect Responses:

- **11 participants (36.67%)** selected *"A reader,"* making it the most commonly chosen incorrect response.
- **5 participants (16.67%)** selected *"A doctree."*
- **2 participants (6.67%)** selected *"Input data."*
- **2 participants (6.67%)** selected *"Output format."*

Trends:

- A notable portion of participants (67%) selected incorrect answers, indicating potential misunderstanding of the question or the notation's intended output.
- The option "A reader" was chosen by more than a third of the participants (36.67%), making it a very strong distractor. This suggests a specific confusion between the roles of a parser and a reader in the construct.

5.3.4 Q4: In the definition △f.△, what does the transformation f.△ imply ?

Objective: Test participants' understanding of a specific Level 2 definition. Figure 5.14 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

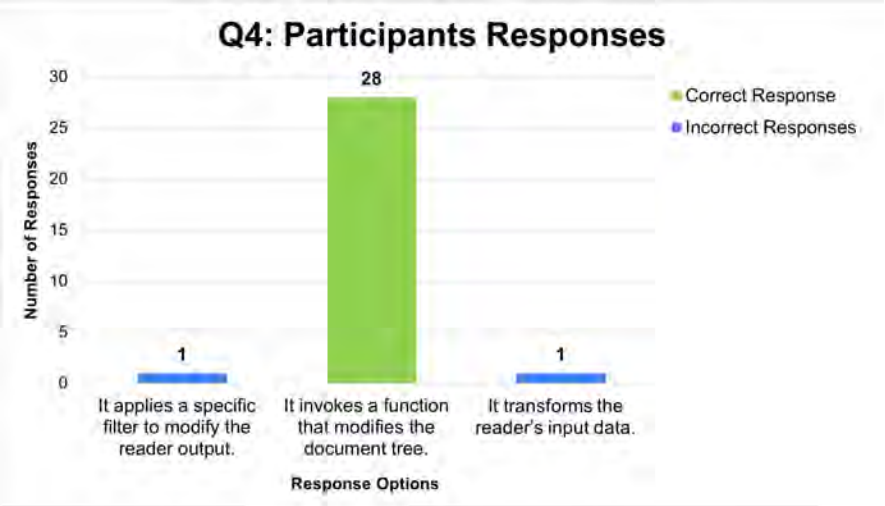


Figure 5.14: Case Two (Q4) Frequency of responses by option.

Correct Responses:

- **28 participants (93.33%)** correctly identified that "*It invokes a function that modifies the document tree.*"

Incorrect Responses:

- **1 participant (3.33%)** selected *"It applies a specific filter to modify the reader output."*
- **1 participant (3.33%)** selected *"It transforms the reader's input data."*

Trends:

- An overwhelming majority of participants (93%) correctly identified the intended transformation, indicating strong comprehension of the question and notation.

- Only 2 participants (7%) selected incorrect options, suggesting the question and options were generally well understood.
- The correct answer was significantly more popular than the distractors, reflecting clarity in the notation or familiarity with the concept.

5.3.5 Q5: What does the combined definition (⊕:s ..⊙:s)!△ represent ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.15 shows a chart that representing the percentage distribution of responses between the two options, highlighting the higher selection rate for the correct answer.

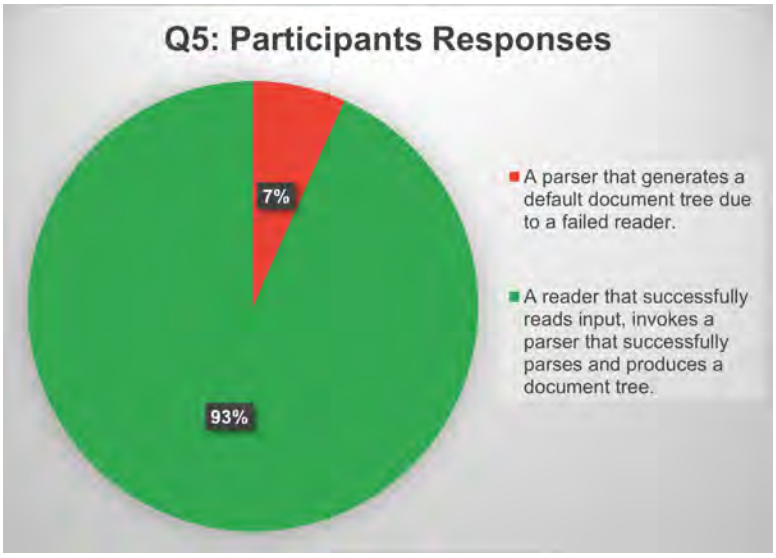


Figure 5.15: Case Two (Q5) Frequency of responses by option.

Trends:

- A large majority of 28 participants (93%) selected the correct option, indicating a clear understanding of the combined definition.
- Only 2 participants (7%) chose the incorrect response, suggesting the question was well understood and the distractor was not compelling for most participants.

5.3.6 Q6: If ◬⁰ (absent document tree), is passed to the writer in ⓘ▪◬ᵒᵖᵗ, what is the outcome ?

Objective: Test participants' understanding of a specific Level 2 definition. Figure 5.16 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

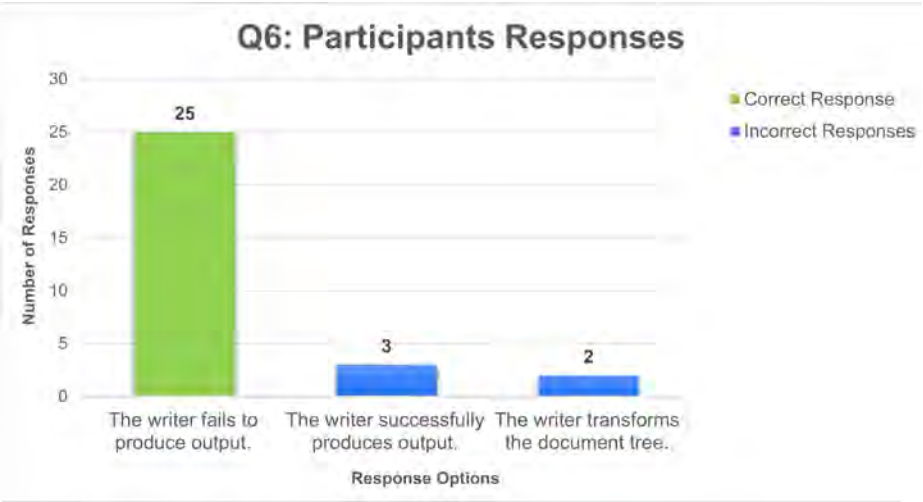


Figure 5.16: Case Two (Q6) Frequency of responses by option.

Correct Responses:

- **25 participants (83.33%)** correctly identified that "*The writer fails to produce output.*"

Incorrect Responses:

- **3 participants (10%)** selected *"The writer successfully produces output."*
- **2 participants (6.67%)** selected *"The writer transforms the document tree."*

Trends:

- A large majority of participants (83%) correctly identified the intended outcome, demonstrating strong comprehension of the intended behavior.

5.3.7 Q7: What does the combination (((⊕:S .. ⊙:F) ! △)^opt imply ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.17 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

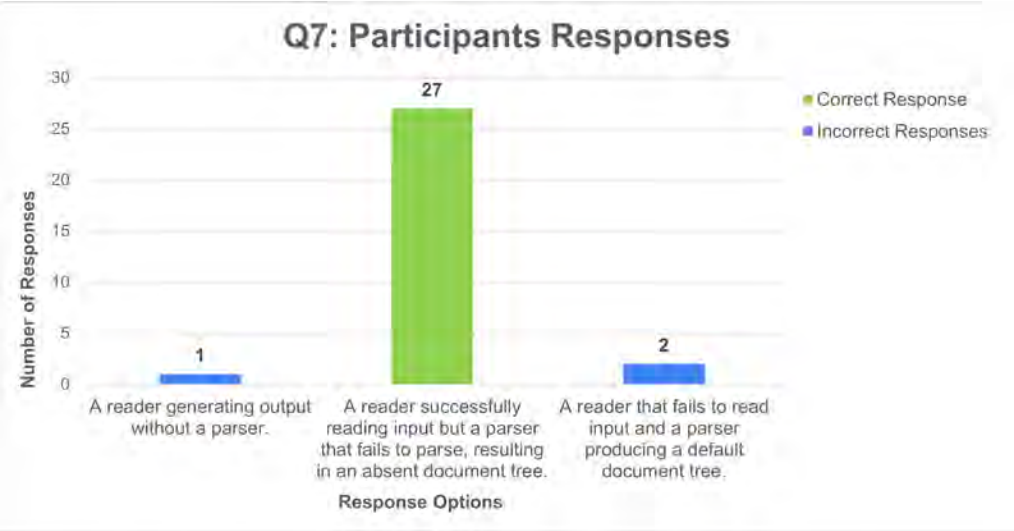


Figure 5.17: Case Two (Q7) Frequency of responses by option.

Correct Responses:

- **27 participants (90%)** correctly identified that "*A reader successfully reading input but a parser that fails to parse, resulting in an absent document tree.*"

Incorrect Responses:

- **2 participants (6.67%)** selected *"A reader that fails to read input and a parser producing a default document tree."*
- **1 participants (3.33%)** selected *"A reader generating output without a parser."*

Trends:

- he majority of participants (90%) demonstrated a strong understanding of the correct implications of the combination.

5.3.8 Q8: Which sequence best represents a full, successful input-to-output workflow using Level 2 definitions?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.18 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.10.

Figure 5.18: Case Two (Q8) Frequency of responses by option.

Correct Responses:

- **27 participants (90%)** correctly selected "A" as the sequence representing a full, successful input-to-output workflow.

Incorrect Responses:

- **2 participants (6.67%)** selected "C", indicating an incomplete or alternate interpretation of the sequence.
- **1 participants (3.33%)** selected "B", reflecting significant divergence from the correct workflow.

Trends:

- The vast majority of participants (90%) successfully identified the correct sequence, demonstrating strong comprehension of the workflow.

- A small number of participants (10%) selected incorrect sequences, with the majority opting for "C" over "B."
- The results reflect that most participants possess a clear understanding of the logical progression of Level 2 definitions.

5.4 Case Three

Case Two focused on modeling the system **Pandoc** using the developed modeling notation and evaluating participants' comprehension through seven questions. This section presents the results for each question, highlighting performance trends and descriptive statistics.

Average Score: 72.86% across all questions.

5.4.1 Q1: Recognition of Main Definitions

The objective of this question is to evaluate participants' ability to identify the main definitions in the Pandoc diagram (see Fig. 4.18). Figure 5.20 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.19.

Correct Responses:

- **18 participants (60%)** identified all six main definitions: Option 1 (InputFormatHandler), Option 4 (OutputFormatHandler), Option 7 (parser), Option 11 (writer), Option 15 (AST) and Option 16 (OutputText).

Partially Correct Responses:

- **10 participants (33.33%)** identified some of the correct options and or including other distractor options. Examples include:
  - Option 2, Option 3, Option 6, Option 9, Option 10, Option 12, Option 15, Option 18 (Count 1).

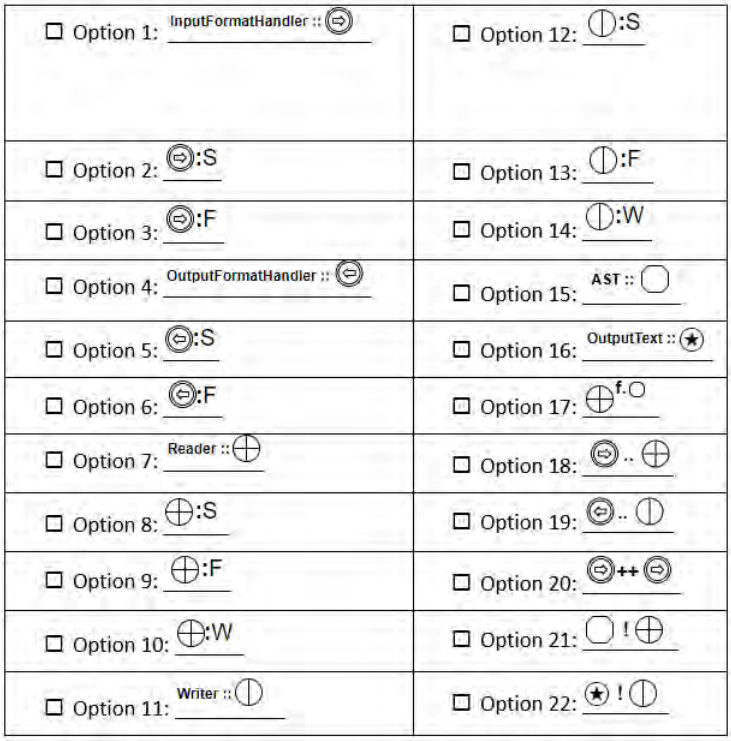


Figure 5.19: Pandoc Q1, and Q2 options.

- Option 1, Option 4, Option 7, Option 11, Option 16 (Count 3).
- Option 1, Option 4, Option 7, Option 11, Option 15 (Count 1).
- Option 1, Option 4, Option 7, Option 11 (Count 2).
- Option 1, Option 4, Option 7, Option 11, Option 16 (Count 3).
- Option 1, Option 2, Option 3, Option 4, Option 5, Option 6, Option 18, Option 19, Option 20 (Count 1).
- Option 1, Option 2, Option 3, Option 4, Option 5, Option 6 (Count 1).
- Option 1, Option 2 (Count 1).

Overinclusive Responses:

- **2 participants (6.67%)** identified all of the correct options including other distractor options. Examples include:
  - Option 1, Option 2, Option 3, Option 4, Option 5, Option 6, Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14, Option 15, Option 16 (Count 2).

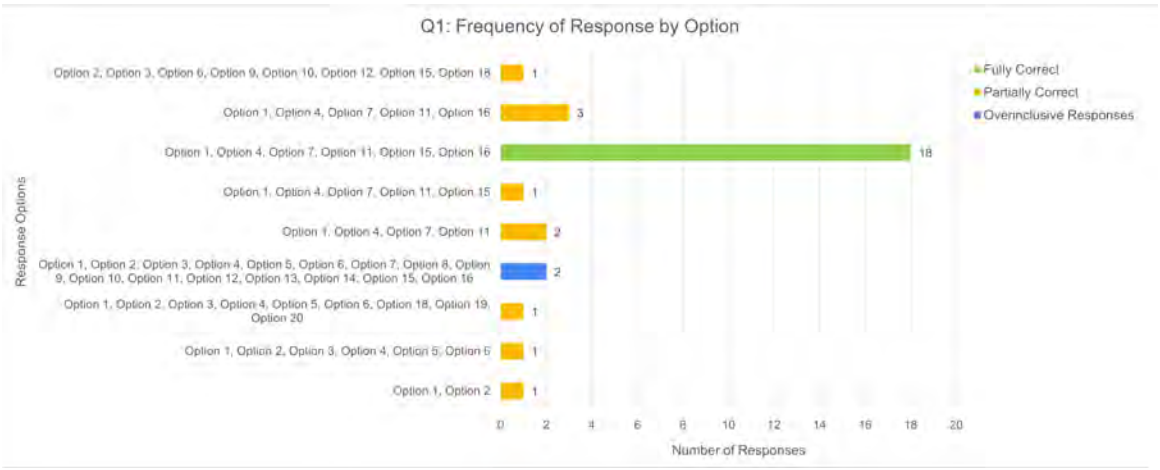


Figure 5.20: Case Three (Q1) Frequency of responses by option.

Trends:

- Option 1, Option 4, Option 7, and Option 11 were the most frequently selected correct definitions.
- Option 15 and Option 16 had slightly lower selection rates, contributing to most of the partially correct responses.
- There is a substantial increase in partially correct responses (10 participants) compared to previous questions. This indicates that participants had more difficulty distinguishing the correct main definitions from other options in this case.

5.4.2 Q2: Identifying Sub-Definitions

Figure 5.21 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option in figure 5.19.

Correct Responses:

- **21 participants (70%)** correctly identified all sub-definitions (Option 2, Option 3, Option 5, Option 6, Option 8, Option 9, Option 10, Option 12, Option 13, and Option 14).

Partially Correct Responses:

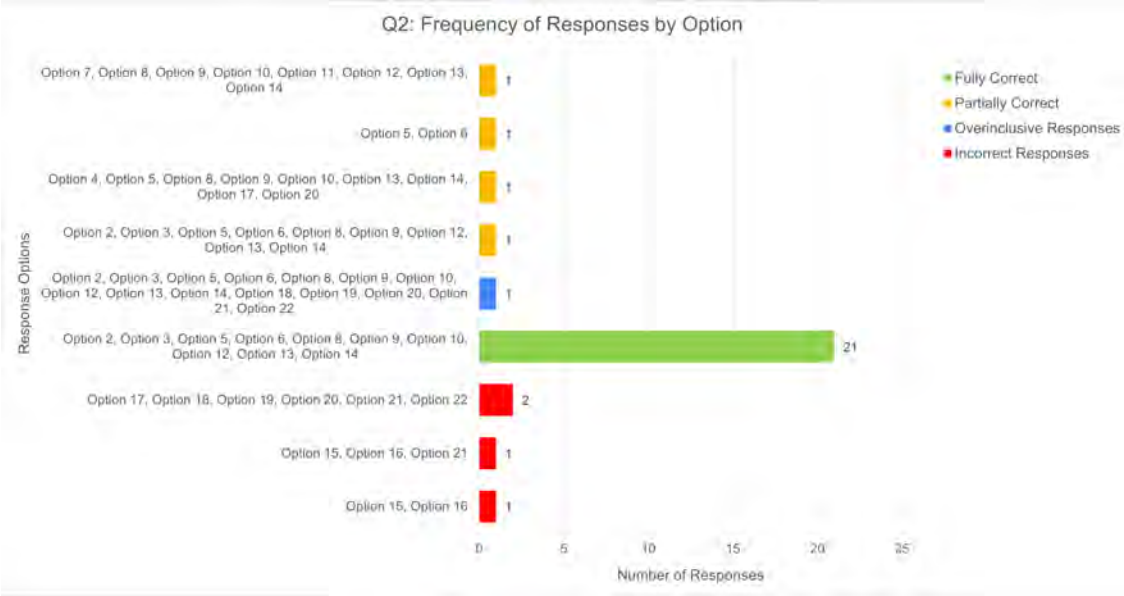


Figure 5.21: Case Three (Q2) Frequency of responses by option.

- **4 participants (13.33%)** identified some of the correct options and or including other distractor options. Examples include:
  - Option 7, Option 8, Option 9, Option 10, Option 11, Option 12, Option 13, Option 14 (Count 1).
  - Option 5, Option 6 (Count 1).
  - Option 4, Option 5, Option 8, Option 9, Option 10, Option 13, Option 14, Option 17, Option 20 (Count 1).
  - Option 2, Option 3, Option 5, Option 6, Option 8, Option 9, Option 12, Option 13, Option 14 (Count 1).

Overinclusive Responses:

- **1 participants (3.33%)** identified all of the correct options including other distractor options. Examples include:
  - Option 2, Option 3, Option 5, Option 6, Option 8, Option 9, Option 10, Option 12, Option 13, Option 14, Option 18, Option 19, Option 20, Option 21, Option 22 (Count 1).(Count 1).

Incorrect Responses:

- **4 participants (13.33%)** identified none of the correct options including other options. Examples include:
  - Option 17, Option 18, Option 19, Option 20, Option 21, Option 22 (Count 2).
  - Option 15, Option 16, Option 21 (Count 1).
  - Option 15, Option 16 (Count 1).

Trends:

- While a majority (21 out of 30, or 70%) of participants correctly identified all ten sub-definitions, the performance is slightly lower and shows more variation than in Case Two's Q1 and Q2.

5.4.3 Q3: What does the ⊚ ·· ⊕ produce ?

Objective: Test participants' understanding of a specific Level 2 definition. Figure 5.22 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option.

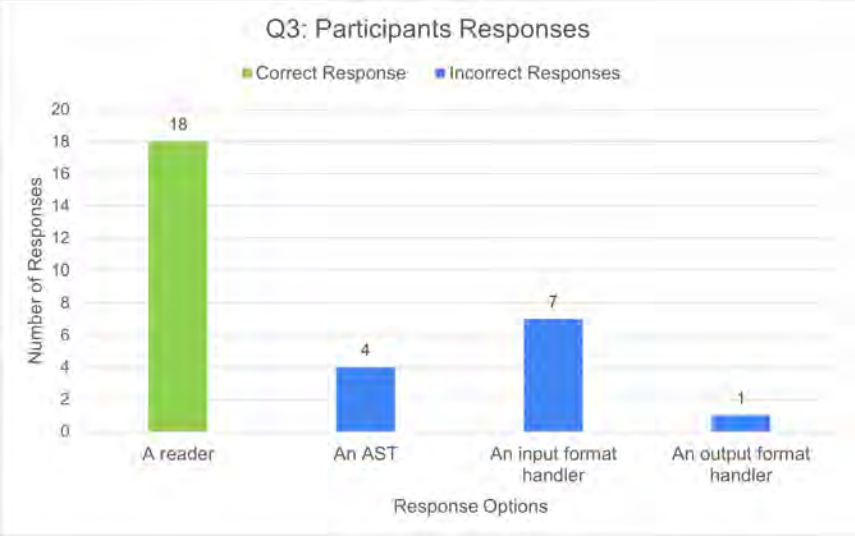


Figure 5.22: Case Two (Q3) Frequency of responses by option.

Correct Responses:

- **18 participants (60%)** correctly identified the output as *"A reader."*

Incorrect Responses:

- **12 participants (40%)** selected incorrect answers. These can be broken down as follows:
  - "An input format handler" (Count: 7)
  - "An AST" (Count: 4)
  - "An output format handler" (Count: 1)

Trends:

- A majority of participants (60%) successfully identified the correct answer, indicating a reasonable understanding of this aspect of the system.
- The option "An input format handler" was chosen by a significant portion of participants (23.3%), making it a strong distractor. This suggests a confusion between the process of handling input and the product of the construct (a reader).
- Four participants chose "An AST," suggesting confusion between the role of a reader (which handles input) and a parser (which produces an Abstract Syntax Tree - AST).
- In Case Two, Q3 had a much lower correct response rate (33.3%) and showed confusion between a parser and a reader. In this case, the confusion is more between a reader and an input format handler, as well as some confusion with an AST. Compare to

5.4.4 Q4: What does the combined definition (((◎:s .. ⊕:s)! ○) represent ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.23 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option.

Correct Responses:

- **28 participants (93.33%)** correctly identified that *"An input format handler that successfully recognizes the format and invokes a reader that successfully parses text, producing an AST."*

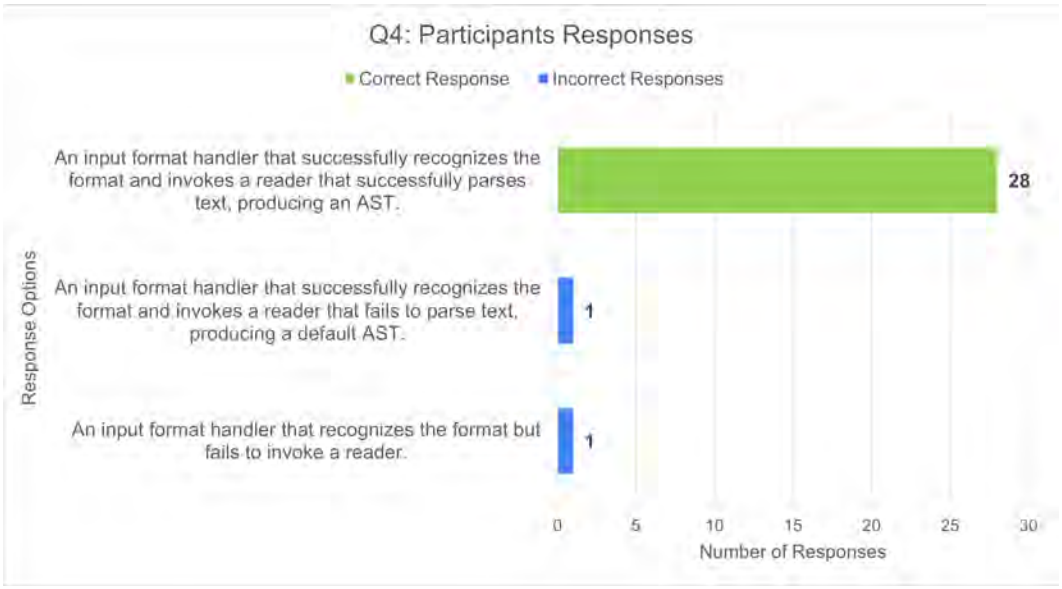


Figure 5.23: Case Three (Q4) Frequency of responses by option.

Incorrect Responses:

- **2 participants (6.67%)** selected incorrect answers. These can be broken down as follows:
  - "An input format handler that recognizes the format but fails to invoke a reader" (Count: 1).
  - "An input format handler that successfully recognizes the format and invokes a reader that fails to parse text, producing a default AST" (Count: 1).

Trends:

- The overwhelming majority (93%) of participants demonstrated a strong understanding of the combined definition's role and its expected outcome.
- Misconceptions were minimal, with only two participants selecting incorrect options, each involving failure scenarios.

5.4.5 Q5: What does the notation $\oplus^{f.\bigcirc}$ imply in the context of transformation ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.24 displays the frequency of responses for each option,

with the bars indicating the number of participants who selected each option.

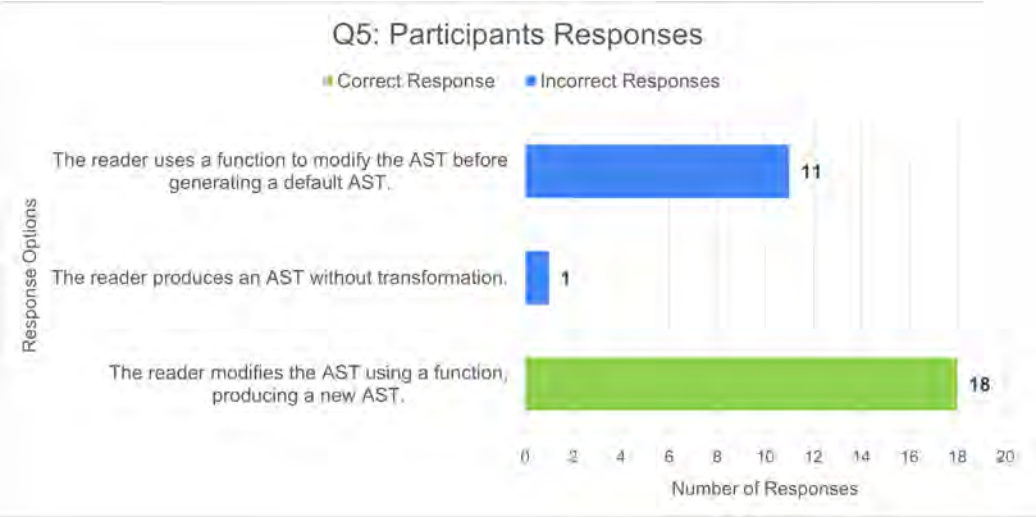


Figure 5.24: Case Three (Q5) Frequency of responses by option.

Correct Responses:

- **18 participants (60%)** correctly identified that *"The reader modifies the AST using a function, producing a new AST."*

Incorrect Responses:

- **12 participants (40%)** selected incorrect answers. These can be broken down as follows:
  - "The reader uses a function to modify the AST before generating a default AST" (Count: 11 (36.67%)).
  - "The reader produces an AST without transformation" (Count: 1 (3.33%)).

Trends:

- The 60% success rate indicates a moderate level of understanding. While a majority got it right, a significant proportion (37%) confused the transformation's outcome, possibly misinterpreting the "default AST" as an intermediate or final output.

5.4.6 Q6: If the (((⊜ ++ ⊜) .. ⊕:s) !○ is used, what does it indicate about the input processing ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.25 displays the frequency of responses for each option, with the bars indicating the number of participants who selected each option.

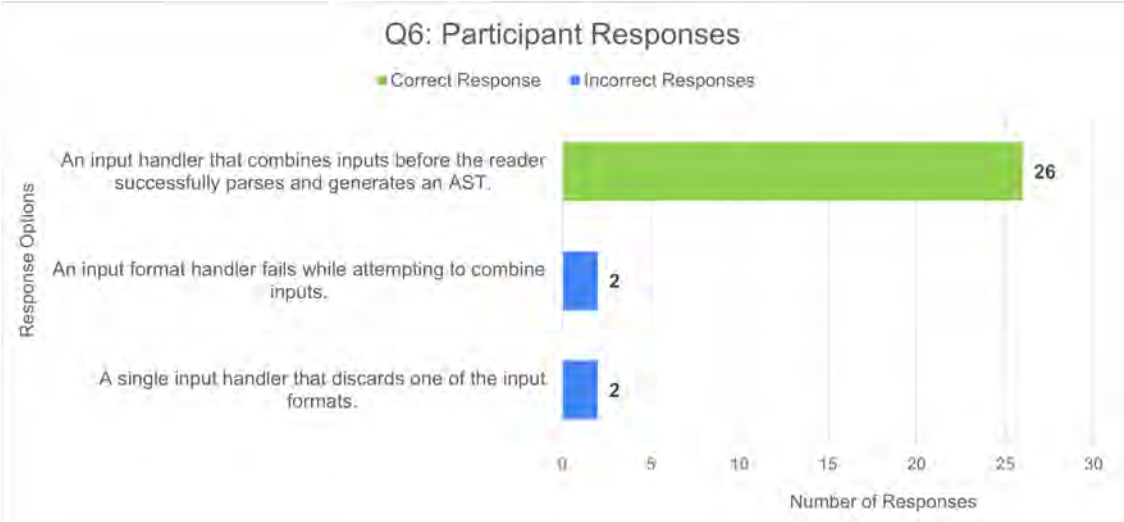


Figure 5.25: Case Three (Q6) Frequency of responses by option.

Correct Responses:

- **26 participants (86.67%)** correctly identified that *"An input handler that combines inputs before the reader successfully parses and generates an AST."*

Incorrect Responses:

- **4 participants (13.33%)** selected incorrect answers. These can be broken down as follows:
  - "A single input handler that discards one of the input formats" (Count: 2).
  - "An input format handler fails while attempting to combine inputs" (Count: 2)).

Trends:

- An overwhelming majority (26 out of 30, or approximately 87%) demonstrated strong comprehension of the concept, accurately identifying the process of combining inputs before successful parsing.

5.4.7 Q7: If (:s..:w)!★ occurs, what is the likely outcome ?

Objective: Assess participants' ability to interpret outcomes when multiple Level 2 definitions are combined. Figure 5.7 shows a chart that representing the percentage distribution of responses between the two options, highlighting the higher selection rate for the correct answer.

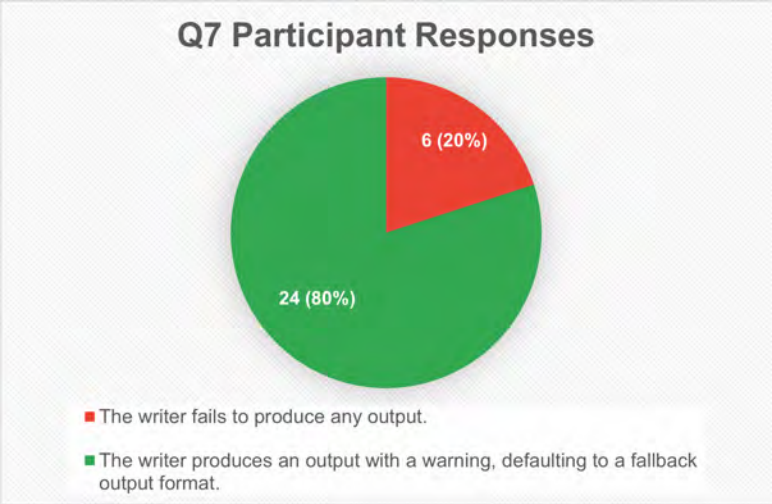


Figure 5.26: Case Three (Q7) Frequency of responses by option.

Trends:

- A significant majority (80%) displayed a clear understanding of the writer's capability to handle warnings and default to a fallback output format.
- The remaining participants (20%) may have perceived the scenario as resulting in failure rather than graceful degradation with a warning.

5.5 Case Four

5.5.1 SYM1: Evaluation of the OR-Arrow Symbol

This section presents the results of the evaluation of a specific notation (OR-Arrow, see Fig. 4.19) designed to represent multiple alternatives within a definition. Participants were asked three closed-ended questions and provided open-ended feedback explaining their choices.

Table 5.6: Quantitative Results for Case 4 SYM 1

Question	Yes	No
Clearly communicates multiple alternatives?	24 (80%)	6 (20%)
Effective in conveying either-or relationship?	18 (60%)	12 (40%)
Ambiguity in interpreting subtype relationship?	17 (57%)	13 (43%)

As Table 5.6 shows, a large majority (80%) of participants agreed that the notation clearly communicates multiple alternatives. However, there was less consensus on the notation's effectiveness in conveying the either-or relationship (60% yes, 40% no), and a slight majority (57%) perceived ambiguity in interpreting the subtype relationship.

Thematic analysis of the open-ended responses revealed two dominant themes: **"Clarity and Understandability of the Notation"** and **"Misinterpretations Related to Flow, Dependency, and Alternatives."**

Clarity and Understandability of the Notation

This theme encompasses the participants' perceptions of the symbol's inherent ambiguity and the crucial role of the provided context (introduction and case studies) in making the notation understandable. Responses were grouped into two subthemes:

Subtheme 1: Reliance on Context/Explanation for Clarity (10/16 participants): This theme encompasses all comments that address the symbol's inherent ambiguity *and* the crucial role of the provided explanation, introduction, and/or case studies in making the notation understandable. It captures the spectrum from "the symbol is meaningless on its own" to "the symbol is clear *because* of the context." Examples include:

- *"If the two parallel lines represent two outputs. However, it could be confusing because it looks like one subtype might depend on the other, which isn't clear without further explanation. The symbol already ties into the main type's definition and seems to be used in defining the subtypes too."*
- *"The arrow seems to indicate a relationship between type and success and between success and failure, without explanation."*
- *The introduction and the case studies helped me understand what the notation means. But, it also kind of looks like passing something to the next thing which can be confusing.*

- *"The symbol on its own is not inherently clear, but after it is explained and given meaning at the beginning of the survey, it is clear. The — within the symbol is helpful for those who are familiar with this variation of the OR symbol."*

Subtheme 2: Perceived Clarity/Ease of Understanding (Given Context) (6/16 participants): Other participants expressed positive evaluations of the notation's clarity or ease of understanding, assuming the provided context. Examples include:

- *"Clear."*
- *"Easy to understand."*
- *"The symbol in the FParsec diagram is used to indicate that the main definition can represent multiple alternatives. It clearly indicates that the main definition can lead to multiple alternative sub-definitions, which helps in understanding the variability and flexibility within the definitions."*
- *A symbol is intuitive if it aligns with familiar conventions, resembles its function, or is simple enough to convey meaning without additional context. And the notations were different to show that this is the main definition and the others are sub-definitions so it was clear to see and understand.*

Misinterpretations Related to Flow, Dependency, and Alternatives.

This theme encompasses the various ways participants misconstrued the relationships represented by the notation, including linear/sequential interpretations, implied dependencies, and confusion about how alternatives are represented. Responses are grouped into three sub-themes:

Sub-theme 1: Sequential/Linear Interpretation (6/12): Participants often interpreted the relationships as sequential rather than branching, despite the intended meaning. This misinterpretation was linked to the visual layout and the perceived flow of the diagram. Examples include:

- *"I think it is ambiguous because it makes me think that the process is sequential instead of branching."*
- *"The symbol shows as if the relationships are linear, as in one definition leads to another. It not clearly showing that one definition has multiple alternatives."*

- *"Maybe if the one either or block was on the other side of the main definition otherwise it looks like a chain and doesn't really show me that they are 2 different relationships"*
- *"The introduction and the case studies helped me understand what the notation means. But, it also kind of looks like passing something to the next thing which can be confusing"*

Sub-theme 2: Implied Dependency/Hierarchy (3/12): Some participants interpreted the notation as suggesting hierarchical or dependent relationships, either between the main definition and alternatives or among the alternatives themselves.

- *"if the two parallel lines represent two outputs. However, it could be confusing because it looks like one subtype might depend on the other, which isn't clear without further explanation. The symbol already ties into the main type's definition and seems to be used in defining the subtypes too."*
- *"The linked list structure suggests a false hierarchy and order, but the sign is technically sufficient and unambiguous once it has been explained."*
- *"With the way the notation was introduced it is clear that the main definition can represent multiple types and that the or arrow means it represents this or this however it could also give off the impression that the next definition is based on the previous definition"*

Sub-theme 3: Confusion Regarding Alternatives/Choice (3/12): Participants expressed confusion about how the notation represented alternatives or choice, particularly in relation to success and failure. Examples include:

- *"It gives the impression that it flows from success to failure rather than it being success or failure"*
- *"The arrow seems to indicate a relationship between type and success and between success and failure, without explanation."*
- *"The chosen symbol leaves room for uncertainty. Linking the success and failure directly to the main input handlers may remove any ambiguity. As it is now, it makes it seem like there is no direct link between the failed input and the main input handler."*

The qualitative feedback reveals a critical dependence on contextual information for understanding the notation. While some participants found the notation clear *given* the provided explanations and examples, a significant number emphasized that the symbol itself lacked intrinsic clarity, leading to potential misinterpretations related to sequential flow, implied dependencies, and the representation of alternatives. This is reflected in the quantitative data (Table 5.6), where, for example, while 80% of participants understood the concept of multiple alternatives, only 60% found the notation effective in conveying the "either-or" relationship, and 57% perceived ambiguity in interpreting subtype relationships. This suggests that while the notation can be effective with sufficient explanation, modifications to the symbol itself or more explicit visual cues could improve its inherent clarity and reduce the reliance on contextual information.

5.5.2 SYM2: Comparative Analysis of the OR-Arrow Symbol

This section presents the results of the comparison between two diagrams (A and B) representing the OR-Arrow symbol (See Fig. 4.20). Participants were asked to compare the diagrams and provide explanations for their preferences.

Table 5.7: Quantitative Results for ase 4 SYM 2

Question	**Diagram A Responses**	**Diagram B Responses**
Which diagram makes it easier to identify the relationship?	6 (20%)	24 (80%)
How many sub-definitions are visible in Diagram A? (Correct Answer: Two)	Two: 24 (80%) One: 4 (13.3%) Three: 1 (3.3%) Zero: 1 (3.3%)	
How many sub-definitions are visible in Diagram B? (Correct Answer: Two)		Two: 26 (86.7%) One: 1 (3.3%) Three: 1 (3.3%) Zero: 2 (6.7%)

As shown in Table 5.7, Diagram B was significantly preferred for identifying the relationship between definitions (80% vs. 20%). Both diagrams performed well in participants correctly identifying two sub-definitions, with Diagram B slightly better.

Thematic analysis of the open-ended responses revealed a dominant theme: Clarity and Representation of Relationships, with a strong preference for Diagram B. A significant

number of participants (20 responses) explicitly stated that Diagram B more clearly showed the direct relationship between the main definition and its sub-definitions. They often contrasted this with Diagram A, which was perceived as less clear or suggesting a different kind of relationship:

- *"Diagram B makes the relationships clearer because the direct arrows from the main definition to the subtypes are visually easier to follow...clearly shown as the sub-definitions of the main type."*
- *"Diagram B is better defined in terms of the relationships than Diagram A simply because you can clearly see the line connecting from the main definition to all the sub type definition in diagram B. With Diagram A, if not told about the symbols, it might be hard to understand how the Fail sub definition connects to the main definition."*
- *"In diagram B it is easier to see that the two definitions are a subset of the main definiiton"*
- *"It is easier to see the two sub-types originate from the main type"*

A smaller number of these participants (4/20) specifically pointed out that Diagram A suggested a linear or sequential relationship or implied a relationship between the sub-definitions, which was not intended (see also section 5.5.1).

- *"Diagram B better illustrates the relationships between the main definition and the sub definitions. While A shows a linear/directional relationship..."*
- *"The first diagram might be misinterpreted as if there is a relationship between two sub-definitions."*

A few participants (2) recognized that both diagrams represented the same information, just in different visual styles.

The results overwhelmingly favor Diagram B for its clearer and more direct representation of the relationships between definitions. The qualitative data reinforces this preference, with participants consistently highlighting the direct visual links in Diagram B and the tendency to misinterpret Diagram A as representing a linear or sequential relationship. While some participants found Diagram A to be clear, the strong preference for B suggests

that it more effectively communicates the intended "either/or" relationship between the main definition and its sub-definitions. This is consistent with the quantitative data where 80% preferred Diagram B, and both diagrams had a high percentage of correct answers when identifying the number of sub-definitions.

5.5.3 SYM3: Evaluation of the Notation Label (::) Symbol

This section presents the results of the evaluation of the notation label :: symbol, used to define the name and corresponding symbol of a main definition (Fig. 4.21). Participants answered closed-ended questions and provided open-ended feedback.

Table 5.8: Quantitative Results for Case 4 SYM 3

Question	Yes/Correct	No/Incorrect
Is the notation InputHandler :: clear in representing the relationship between the name of a definition and its corresponding symbol?	28 (93.3%)	2 (6.7%)
What does the symbol represent? (Correct Answer: An Input Handler)	26 (86.7%)	4 (13.3%)

As shown in Table 5.8, the vast majority of participants found the :: notation clear in representing the relationship between a definition's name and symbol (93.3%) and correctly identified its meaning as an Input Handler (86.7%).

Thematic analysis of the open-ended responses revealed a dominant theme: **Clarity and Effectiveness of the :: Notation**. This theme encompasses the positive feedback regarding the notation's ability to clearly link a definition's name and its corresponding symbol. Many participants (22 responses) explicitly stated that the :: notation was clear, unambiguous, and effectively conveyed the relationship between the name and the symbol. Examples include:

- *"Yes, because it clearly shows the symbol and the relationship between the name."*
- *"Clear and makes sense."*
- *"Does not appear ambiguous that the symbol represents the text that precedes it."*
- *"The chosen notion is clear and it links the name and the symbol clearly."*

Several participants connected the :: notation to the concept of input handling, further demonstrating their understanding of its function within the broader notation system. Examples include:

- *"The name is input handler and the symbol is followed after ::"*
- *"Yes, the term 'inputHandler' placed next to the symbol indicates its meaning."*
- *"Because is shows an input handler parsing the numbers."*

One participant specifically mentioned the use of :: in programming and mathematical contexts, suggesting familiarity with this convention.

- *"The :: operator is often used in programming and mathematical contexts to denote a type or a specific relationship"*

Participants did not correctly identify the symbol as an "Input Handler" connected the notation to the concept of input flow, likely influenced by the adjacent arrow symbol. For example:

- *"I think the arrow pointing right obviously represents something going in, given the context and knowledge of systems modelling."*
- *"Input handler and the arrow pointing to the right makes it easier to understand the representation of incoming input."*
- *"The 'Going in' corresponds clearly to the symbol"*

The qualitative data strongly supports the quantitative findings, indicating a very positive reception of the :: notation. Participants consistently described it as clear, unambiguous, and effective in linking a definition's name and symbol. The association with similar notations in programming and mathematics further reinforces its intuitive nature for some participants. A small subset connected the notation to input flow, likely influenced by the adjacent arrow, but this did not detract from the overall positive assessment of the :: symbol's clarity. The quantitative data shows a strong preference for the notation, with 93.3% of participants finding it clear. This is mirrored in the qualitative data, where the dominant theme is the clarity and directness of the:: notation. The few participants who did not find it clear connected the notation to input flow, likely influenced by the adjacent arrow. The high number of participants correctly identifying the symbol as representing an input handler further validates the notation's effectiveness.

5.5.4 SYM4: Evaluation of Sub-Definition Notation

his section presents the results of the evaluation of the sub-definition notation (symbol:letter), used to define subtypes as subsets of a main definition (Fig. 4.22). Participants answered closed-ended questions and provided open-ended feedback.

Table 5.9: Quantitative Results for Case 4 SYM 5

Question	Yes/Correct	No/Incorrect
Does the ⊜:S notation clearly convey that combining a symbol from the main definition with a letter indicates a sub-definition of that main definition?	26 (86.7%)	4 (13.6%)
Do you understand the single letter (e.g., S) to intuitively represent the sub-definitions functionality?	28 (93.3%)	2 (6.7%)
In the ⊜:S notation, what does the S represent? (Correct Answer: A specific behavior or variant of the type above such as a successful state.)	26 (86.7%)	4 (13.6%)

As shown in Table 5.9, the vast majority of participants found the sub-definition notation clear (86.7%) and intuitively understood the single letter's meaning (93.3%). A similar percentage correctly identified the meaning of "S" (86.7%).

Thematic analysis of the open-ended responses revealed a dominant theme: Clarity and Intuitive Nature of the Sub-Definition Notation. This theme highlights the positive perception of the symbol:letter notation in conveying the relationship between a main definition and its sub-definitions.

Sub-theme 1: Clear and Straightforward Representation (15/26 participants): Many participants (15/26) described the notation as clear, straightforward, and easy to understand. For example:

- *"easy to identify the meaning"*
- *"It was clear in the explanation."*
- *"The symbol and letter pairing is clear and it leaves no room for ambiguity as it relates the meaning/behavior associated with the letter back to the main symbol."*
- *"S simply meant successful and F meant fail."*

Sub-theme 2: Reliance on Prior Context/Explanation (3/26 participants): A few participants (3/26) mentioned that while the notation was clear, the initial explanation or context was essential for understanding.

- *"It goes back to the initial description of the whole model. If you are told what the symbols and letters mean, it makes it easier to follow in a diagram. In this case, S makes sense to represent a Success."*
- *"Again, without the textual introduction on the first pages it could happen not to be clear..."*
- *"a) Yes, understanding the meaning of the first input symbol makes it straightforward to deduce what the S represents. b) Yes, as 'success' starts with the letter S..."*

Sub-theme 3: Potential for Ambiguity/Alternative Interpretations (4/26 participants): A small number of participants (3/26) expressed concerns about potential ambiguity or alternative interpretations of the single letter, particularly in more complex scenarios.

- *"A single letter is able to carry the meaning well when helped by convention and the description in the box, but in cases more complex than pass/fail, it may not be intuitive."*
- *"The S and using the symbol from the main definition might not clearly show as a sub definition from the get go as the symbol can be interpreted as completely different from the main definition."*
- *"Yes, since the symbol for the main type is already defined. However, I think symbols are better than letters because letters can be interpreted in different ways."*
- *"it could be mistaken for 'start' which is a common use of the letter S in most diagrams..."*

The remaining comments were too brief, unclear, or did not provide specific reasons related to the notation's clarity. The qualitative data strongly supports the quantitative findings, revealing a generally positive perception of the sub-definition notation. Participants largely found the notation clear, straightforward, and intuitive, particularly in the context of the provided explanations and examples. However, a small subset raised valid concerns about potential ambiguity in more complex scenarios or without the initial context, suggesting that while effective in this specific application, the notation's scalability and generalizability should be considered. The high percentage of positive responses in the quantitative data (86.7% finding the notation clear, 93.3% understanding the letter's meaning) aligns with the dominant qualitative theme of clarity and straightforwardness.

The small number of negative responses and comments about potential ambiguity highlight the importance of the initial explanation and suggest that further refinements or more explicit visual cues could be beneficial for more complex applications.

5.5.5 SYM5: Comparative Evaluation of Diagrams for Main Definitions and Sub-Definitions

This section presents the results of the comparison between two diagrammatic representations (A and B) for conveying main definitions and their sub-definitions (Fig. 4.23). Diagram A used the newly proposed notation, while Diagram B used the older Motara notation.

Table 5.10: Quantitative Results for Case 4 SYM 5

Question	**Diagram A**	**Diagram B**	**Neither**
Which diagram makes it easier to identify the relationship?	15 (50%)	8 (26.7%)	7 (23.3%)
Which diagram makes it easier to recall what the symbols represent?	28 (93.3%)	1 (3.3%)	1 (3.3%)

As shown in Table 5.10, Diagram A was preferred for both identifying relationships (50% vs. 26.7% for Diagram B) and, overwhelmingly, for recall (93.3% vs. 3.3% for Diagram B). A notable portion of participants found neither diagram superior for identifying relationships.

Thematic analysis of the open-ended responses revealed two dominant themes: (1) Preference for Diagram A's Simplicity and Recall, and (2) Perceived Clarity of Relationships in Diagram B.

Sub-theme 1: Preference for Diagram A's Simplicity and Recall (18 responses): A strong preference for Diagram A emerged due to its perceived simplicity, conciseness, and ease of recall. Participants frequently cited the reduced number of symbols and the use of letter cues as key advantages. They found Diagram B's multiple unique symbols more difficult to remember. Examples include:

- *"A is simple and effective B is complex and scary."*
- *"Both diagrams work for me. However, A has less symbols to keep track of compared to B."*

- *"a) The letters beside the symbols provide clues about their meaning and help link each symbol to a specific letter, effectively offering additional options beyond just the symbols themselves"*
- *"Diagram A makes it easier to recall the symbols than in diagrams B because the letters are constant, their meaning might not change in a different diagram."*

Sub-theme 2: Perceived Clarity of Relationships in Diagram B (5 responses): While Diagram A was preferred for simplicity and recall, some participants found Diagram B more effective in visually representing the relationships between the main definition and its sub-definitions. Examples include:

- *"Diagram B clearly shows the distinction between the sub-definitions and the direct link to the main definition."*
- *"With b, it's easier to see the connection between the main type and the alternative results: Success, Failure, Warning."*
- *"Diagram B shows a more interconnected structure where the main definition connects directly to both sub-definitions."*
- *"Diagram B makes it easier to see and follow the relationships among the definitions because of the flow of lines from the main definition."*

The qualitative feedback reveals a trade-off between clarity of relationships and ease of recall. While some participants found Diagram B clearer in showing the connections between definitions, the majority preferred Diagram A for its simpler symbolic representation and easier recall. The use of a consistent main symbol with differentiating letters in Diagram A proved to be a key factor in its superior memorability. The quantitative data reflects this trade-off. While the preference for Diagram A in identifying relationships was not overwhelming (50%), the preference for recall was decisive (93.3%). This aligns with the qualitative findings: Diagram A's simplicity and use of mnemonic letters made it significantly easier to remember, even if some participants found the relationships slightly less explicit than in Diagram B.

5.5.6 SYM6: Evaluation of Relationships at Level 2

This section presents the results of the evaluation of how relationships at Level 2 are represented in diagrams (Fig. 4.24). Participants compared two diagrams, A (new notation)

and B (Motara notation), both illustrating a "relies" relationship between definitions.

Table 5.11: Quantitative Results for Case 4 SYM 6

Question	**Diagram A**	**Diagram B**	**Neither**
Which notation conveys that there is a relationship?	10 (33.3%)	15 (50%)	5 (16.7%)

As shown in Table 5.11, Diagram B was slightly preferred for conveying the presence of a relationship between definitions (50% vs. 33.3% for Diagram A). A notable portion (16.7%) found neither diagram effective.

Thematic analysis of the open-ended responses revealed two main themes: Explicit Visual Connections (Diagram B) and Implicit Relationships/Lack of Clarity (Diagram A).

Sub-theme 1: Explicit Visual Connections (Diagram B Preferred) (14 participants): A majority of participants preferred Diagram B due to its explicit visual connections (lines or arrows) between definitions, which clearly communicated the "relies" relationship. For example:

- *"B makes it clearer that there is a relationship between definitions because it links and connects them to a specific thing. In comparison, A also shows a relationship, but it's harder to follow with its dot + dot structure and so on."*

Several participants emphasized the ease of following the relationships in Diagram B:

- *"Diagram B's visual links makes it easier to follow and understand the relationship between the definitions."*
- *"Can see the link easier with the lines."*
- *"The lines in b convey that there is a relationship."*
- *"Forms are excessive in Diagram B, and I don't get the reasoning for hull / filled connectors, but thanks to links connecting the blocks and the outline and vertical flow, relationship is more visible in diagram B as for me, having the same symbol everywhere is not enough to catch it quickly."*

Sub-theme 2: Implicit Relationships/Lack of Clarity (Diagram A Concerns) (6 participants): Some participants found Diagram A less clear in conveying the relationships, often describing them as implicit or requiring more effort to discern.

- *"I can't see anything in Diagram A that shows a relationship, and Diagram B is not drawn in a clear way"*
- *"The first diagram requires a more thorough look to notice the relationships."*
- *"In a it tells you that it relies on the other definitions where as in b the you only see lines"*
- *"It tells you which definition relies on which other definitions. B just has lines with dots which I'm not really sure what they mean."*

The results suggest that Diagram B was generally perceived as more effective in conveying "relies" relationships between definitions due to its explicit visual links. While some participants found Diagram A clear, a significant portion found it less clear or requiring more effort to understand the connections. This preference for explicit visual connections is reflected in both the quantitative and qualitative data.

5.5.7 SYM7: Holistic Comparison of Diagrammatic Notations

This section presents the results of the holistic comparison between two full diagrams of the FParsec system: Diagram A (old notation) and Diagram B (new notation) (Fig. 4.25 and 4.16).

Q1: Which diagram makes it easier to keep track of the symbols?

Table 5.12 summarizes the responses to Q1. As shown in Table 5.12, a clear majority of

Table 5.12: Quantitative Results for Case 4 SYM 7 (Q1)

Question	Diagram A	Diagram B	Neither
Which diagram makes it easier to keep track of the symbols?	8 (26.7%)	18 (60%)	4 (13.3%)

participants (60%) found Diagram B easier to keep track of symbols, compared to 26.7% for Diagram A. 13.3% of participants found neither diagram easier.

Thematic analysis of the open-ended responses for Q1 revealed two dominant themes: (1) Preference for Diagram B's Simplicity and Reduced Symbol Count, and (2) Layout, Density, and Visual Flow (Diagram A)

Theme 1: Preference for Diagram B's Simplicity and Reduced Symbol Count (17 participants) This subtheme captures the strong preference for Diagram B due to its reduced symbol count, streamlined design, and less cluttered layout. Participants consistently emphasized the ease of understanding and reduced cognitive load associated with Diagram B. Examples include:

- *"It uses far fewer symbols compared to the other one."*
- *"Less symbols to memorize."*
- *"Lot less busy and easier to relate and understand."*
- *"uses a simplified or streamlined approach, with symbols consistently placed and visually distinct."*

The comments highlight that the consistency and placement of symbols in Diagram B contributed to its perceived simplicity:

- *"There is a standard symbol from the main definition that is used across Diagram B unlike in diagram A whereby there are more symbols with no obvious pattern."*

Theme 2: Layout, Density, and Visual Flow (Diagram A) (6 participants) This subtheme captures the concerns regarding Diagram A's density, perceived clutter, and layout. While some participants found aspects of Diagram A appealing (like neatness or the interconnectedness of elements), these were outweighed by concerns about its overall complexity and the difficulty in visually processing the information. Examples include:

- *"A is denser."*
- *"The star shape in A adds unnecessary noise to the diagrams."*
- *"In Diagram A it seems easy to follow the symbols since everything is connected by lines and the definitions and type manipulation all fit in sort of one area. Whereas with diagram B, you might forget how the other definition or manipulation work together since they get put separately from the main and sub definitions."*

Some participants expressed mixed feelings, acknowledging both positive and negative aspects of Diagram A:

- *"A because it is neat. The layout makes a difference. B had less symbols to recall"*
- *"I am going with the first A because it is neater. I like the second becuase it reducesa the numnber of symbols however the diagram appears to be all over the place"*
- *"The symbols are easier to spot visually in the whole diagram."*

The results of Q1 strongly favor Diagram B for ease of symbol tracking. The quantitative data shows a clear preference for B (60%), and the qualitative data reinforces this preference. Participants consistently highlighted Diagram B's reduced symbol count, clearer layout, and less busy appearance as key advantages. While some participants appreciated aspects of Diagram A, concerns about its density and complexity were more prevalent. This suggests that the new notation (Diagram B) offers a significant improvement in terms of symbol management and visual clarity, directly impacting usability. The qualitative data also reveals a key trade-off: while Diagram A's interconnectedness might initially seem appealing, it ultimately leads to visual clutter and difficulty in tracking individual symbols. Diagram B's more separated layout, while potentially making it slightly harder to see the overall interconnectedness at a glance, significantly improves symbol tracking by reducing visual noise. This trade-off should be considered when further refining the notation.

Q2: Which diagram makes it easy to mentally map the symbol to its underlying concept?

Table 5.13 summarizes the responses to Q2. As shown in Table 5.13, a majority of

Table 5.13: Quantitative Results for Case 4 SYM 7 (Q2)

Question	**Diagram A**	**Diagram B**	**Neither**
Which diagram makes it easy to mentally map the symbol to its concept?	10 (33.3%)	18 (60%)	2 (6.7%)

participants (60%) found Diagram B easier to mentally map the symbol to its underlying concept, compared to 33.3% for Diagram A. 6.7% found neither diagram easier.

Thematic analysis of the open-ended responses for Q2 revealed two dominant themes: (1) Preference for Diagram B due to Letter Cues and Structure, and (2) Preference for Diagram A due to Perceived Visual Clarity and Familiarity.

Theme 1: Preference for Diagram B due to Letter Cues and Structure (16 responses): This theme captures the strong preference for Diagram B due to the use of letter cues (labels) and its overall structure, which facilitated mental mapping of symbols to concepts.

- *"Diagram B uses the letters in its definitions and that makes it easier to mentally map to underlying concept than diagram A."*
- *"The letters next to the symbol help clarify what is happening, reducing the need to understand the symbol's meaning on its own."*
- *"The uniformity of the symbols and letters used in diagram B makes it easier to understand and map out mentally."*
- *"well structured and clear"*

Theme 2: Preference for Diagram A due to Perceived Visual Clarity and Familiarity (5 responses): This theme captures the reasons why some participants preferred Diagram A. These participants focused on aspects such as the visual connections between elements (lines, arrows), a perceived resemblance to code, or a general sense of better understandability despite acknowledging the challenges. Examples include:

- *"a does seem easier to mentally map because the symbols are connected by lines, making the relationships clearer. The lines in diagram a make it easier to follow and understand."*
- *"A looks more like code."*
- *"Even though both are very hard to understand, option A is easier to figure out than option B and is much more understandable"*
- *"The complete diagram looks better than the other one."*

The results of Q2 reveal a preference for Diagram B in facilitating mental mapping, primarily due to the use of letter cues and its clearer structure. The qualitative data supports this, with participants consistently highlighting these features as aiding in associating symbols with their meanings. While a smaller group preferred Diagram A, citing visual connections and familiarity, the majority favored Diagram B for its enhanced clarity and reduced cognitive load. This suggests that the new notation (Diagram B) is more effective in bridging the gap between symbolic representation and conceptual understanding.

Q3: Which notation makes it easier to understand the relationships between components?

Table 5.14 summarizes the responses to Q3. As shown in Table 5.14, the responses were

Table 5.14: Quantitative Results for Case 4 SYM 7 (Q3)

Question	Diagram A	Diagram B	Neither
Which notation makes it easier to understand the relationships between components?	13 (43.3%)	12 (40%)	5 (16.7%)

relatively evenly split between Diagram A (43.3%) and Diagram B (40%), with 16.7% finding neither diagram easier for understanding relationships.

Thematic analysis of the open-ended responses for Q1 revealed two dominant themes: (1) Preference for Diagram A due to Explicit Visual Connections and (2) Clearer Structure/Less Clutter (Diagram B Preferred)

Theme 1: Preference for Diagram A due to Explicit Visual Connections (10 responses): This theme captures the preference for Diagram A based on its use of explicit visual connections (lines) to represent relationships. Participants felt these lines made the connections more direct, easier to follow, and less ambiguous. Examples include:

- *"Symbols are easier to understand and also the relationship between the components because of the lines connecting them."*
- *"The connection lines are more obvious."*
- *"The layout of the diagram is nicer and makes it easier to notice the relationships because of the lines."*
- *"Similar to the above comment. The lines seem to make it easier to trace through the diagram to identify meaning of symbols including relationships"*

Theme 2: Clearer Structure/Less Clutter (Diagram B Preferred) (7 responses): This theme highlights the reasons for preferring Diagram B: its simpler structure, less visual clutter, and overall improved clarity.

- *"Simpler and easier to understand"*

- *"The logical and less cluttered arrangement ensures relationships between components are comprehensible without additional interpretation"*
- *"The relies makes it clear what the relationship is"*
- *"well structured and clear"*

The results of Q3 reveal a near-even split in preference between Diagram A and Diagram B for understanding relationships between components. The qualitative data highlights a key difference in how each diagram achieves this: Diagram A relies on explicit visual connections (lines), while Diagram B relies on a less cluttered layout and more implicit representation of relationships. This suggests a trade-off: Diagram A prioritizes direct visual connections, which some participants found helpful, while Diagram B prioritizes overall visual clarity and a less busy appearance, which others found more conducive to understanding.

Q4: Which notation organizes the components and relationships more clearly in terms of layout and flow?

Table 5.15 summarizes the responses to Q4. As shown in Table 5.15, a slight majority

Table 5.15: Quantitative Results for Case 4 SYM 7 (Q4)

Question	**Diagram A**	**Diagram B**	**Neither**
Which notation organizes the components and relationships more clearly in terms of layout and flow?	15 (50%)	11 (36.7%)	5 (13.4%)

of participants (50%) preferred Diagram A for layout and flow, compared to 36.7% for Diagram B. 13.3% found neither diagram superior.

Similarly to the results above the two dominant themes are (1) Preference for Diagram B due to Letter Cues and Structure, and (2) Preference for Diagram A due to Perceived Visual Clarity and Familiarity. With 10 responses favouring theme 1 and 8 responses favouring theme 2.

We also identify an additional theme, theme 3: Mixed/Neutral (5 responses). A small group expressed mixed opinions, and found both diagrams difficult:

- *"I like diagram A because it is laid out nicely, and I like diagram B because it separates the components into different sections"*
- *"In terms of layout and flow, I go with diagram B. The simplicity of the layout makes it easier to digest. In terms of organisation of components and relationships, I go with A. The visual distinction and links of components makes it easier to map out mentally."*
- *"For (a) the lines are not easy to follow and relate. In (b) the Level 2 relations are unclear."*
- *"They are the same."*
- *"Neither Very hard to keep up"*

The results for Q4 show a slight preference for Diagram A regarding layout and flow. The qualitative data reveals distinct reasons for these preferences: Diagram A was favored for its explicit visual connections and centralized layout, while Diagram B was favored for its hierarchical structure and clear directional flow. This suggests a trade-off between a centralized, interconnected approach (A) and a more structured, leveled approach (B). While A was slightly preferred, both approaches had merits in organizing components and relationships, indicating that elements from both could be considered in future design iterations.

Q5: Which notation is more flexible in adapting to changes in the system's structure or requirements?

Table 5.16 summarizes the responses to Q4. As shown in Table 5.16, a majority of

Table 5.16: Quantitative Results for Case 4 SYM 7 (Q5)

Question	**Diagram A**	**Diagram B**	**Neither**
Which notation is more flexible in adapting to changes in the system's structure or requirements?	9 (30%)	17 (56.7%)	4 (13.3%)

participants (56.7%) found Diagram B more flexible in adapting to changes, compared to 30% for Diagram A. 13.3% found neither diagram more flexible.

Thematic analysis of the open-ended responses for Q5 revealed a dominant theme: Modular Structure and Decoupling (Diagram B Preference).

Theme 1: Modular Structure and Decoupling (Diagram B Preference) (19 responses): A large majority of participants preferred Diagram B due to its modular structure, clear separation of levels, and decoupling of components, which they believed would make it easier to adapt to changes.

- *"Allows for better understanding in the event an change is needed"*
- *"B because Can be modified easly"*
- *"Diagram (b) likely uses a modular and adaptable design where components are separate yet clearly connected"*
- *"Each level focuses on a specific part of the system making it flexible for changes."*

The qualitative feedback strongly supports the quantitative finding that Diagram B is perceived as more flexible. The dominant theme is the advantage of its modular structure and decoupled components, which participants believe would facilitate easier adaptation to changes. The quantitative data shows a clear preference for Diagram B (56.7%) for flexibility. This is strongly reinforced by the qualitative data, where the majority of comments focused on the benefits of its modular design.

Summary of SYM7 Findings

This section presents the results of the holistic comparison of two full diagrams of the FParsec system, A (older notation) and B (new notation). The evaluation aimed to compare the overall clarity, usability, and practicality of the new notation against the older notation. The evaluation consisted of five questions, each followed by an open-ended prompt for participants to explain their reasoning. Table 5.17 summarizes the responses to all five questions.

As shown in Table 5.17, Diagram B was generally preferred for symbol tracking (Q1), mental mapping (Q2), and adaptability to change (Q5). Preferences were more balanced for understanding relationships (Q3) and layout/flow (Q4).

The combined quantitative and qualitative data from the holistic diagram comparison (SYM 7) provides a comprehensive picture of the relative strengths and weaknesses of the

Table 5.17: Summary of SYM7 Findings

Question	Diagram A	Diagram B	Neither
Q1: Easier to keep track of symbols?	8 (26.7%)	18 (60%)	4 (13.3%)
Q2: Easier to mentally map symbol to concept?	10 (33.3%)	18 (60%)	2 (6.7%)
Q3: Easier to understand relationships between components?	13 (43.3%)	12 (40%)	5 (16.7%)
Q4: Clearer organization of components and relationships (layout/flow)?	15 (50%)	11 (36.7%)	4 (13.3%)
Q5: More flexible in adapting to changes?	9 (30%)	17 (56.7%)	4 (13.3%)

new and older notations. While preferences were more balanced for understanding relationships (Q3) and layout/flow (Q4), Diagram B (the new notation) was clearly favored for symbol tracking (Q1), mental mapping (Q2), and adaptability to change (Q5).

The qualitative data reveals key reasons for these preferences. Diagram B's reduced symbol count, clearer structure, and use of letter cues were consistently cited as advantages for symbol tracking and mental mapping. Its modular design and distinct levels were also seen as beneficial for adaptability to change. While Diagram A was appreciated for its explicit visual connections and centralized layout by some participants, these advantages were often outweighed by concerns about its density, complexity, and less structured flow.

The data suggests a trade-off between explicit visual connections (favored in Diagram A) and visual clarity/reduced cognitive load (favored in Diagram B). While explicit connections can be helpful for understanding direct relationships, they can also lead to visual clutter and difficulty in managing complex diagrams. Diagram B's more modular and less interconnected approach, while potentially requiring slightly more effort to trace specific relationships, significantly improves overall comprehension and adaptability.

In conclusion, the results strongly suggest that the new notation (Diagram B) offers significant improvements over the older notation (Diagram A) in terms of usability, particularly for symbol tracking, mental mapping, and adaptability to change. While further refinements may be considered, the new notation appears to be a more effective and practical approach for representing the system.

Interviews were initially considered to gain deeper insights into participants' reasoning; however, the survey data provided sufficient qualitative and quantitative feedback to address the research objectives. Given the richness of the open-ended responses, additional interviews were deemed unnecessary.

5.6 Conclusion

The primary objective of this research was to develop a structural modeling notation capable of effectively representing the architecture and relationships of typed functional systems. Building upon the notation proposed by Motara (2021), which adheres to the Physics of Notation Systematized (PoN-S) (Silva Teixeira 2017) design process to ensure adherence to Moody's principles of good notation (Moody 2009). We extended this approach by incorporating the Physics of Diagrams (PoD) framework (Pissierssens, Claes, and Poels 2019) to refine the notation and improve its clarity, usability, and cognitive effectiveness. The PoD framework was chosen due to its empirical foundation, which draws on a comprehensive literature study that meticulously compiled 81 guidelines and 34 underlying propositions, culminating in seven foundational principles for graphical representation design. These principles—focus on diagram understanding, graphical parsimony, chunking, flow of thought, optimal use of visual variables, differentiation, and balancing—provide a holistic perspective on graphical representation design, addressing both the overarching language (as in Moody's Physics of Notations) and the specific instances of graphical representations (as in the Physics of Diagrams).

By integrating the PoD framework into our notation design, we successfully achieved our first research objective (RO1): extitDesigning a Structural Modeling Notation for Functional Systems Using the Physics of Diagrams (PoD) Framework. Several modifications were introduced to the notation, including:

- - A three-tiered structure to organize information hierarchically (Level 1: main definitions, Level 2: manipulations, Level 3: summaries).
- Revised notations for definitions and thesaurus, using distinct shapes (e.g., square rectangles for Level 1, rounded rectangles for Level 2) to enhance perceptual discriminability.
- A new **label :: symbol** notation to define the name and corresponding symbol of a main definition (e.g., parser :: ⊙).
- A **symbol : letter** notation to define subtypes, where a single letter intuitively represents the subtype's functionality (e.g., ⊙:F for a failed parser).
- The introduction of the OR-Arrow symbol (‖) to represent alternative pathways from a main definition to its subtypes.

- A new notation for depicting the relies relationship, explicitly indicating dependencies between definitions.

The effectiveness of the newly developed notation was systematically evaluated through a survey incorporating four case studies, each corresponding to one of our research objectives:

- **RO2:** Applying the notation to FParsec for parsing system representation.
- **RO3:** Evaluating the notation's adaptability to non-functional systems (Docutils).
- **RO4:** Assessing the practicality of the notation in representing Pandoc's architecture.
- **RO5:** Assessing user understanding and interpretation of the notation's symbols and diagrams.

Participants were first introduced to the notation through a brief tutorial, which provided foundational knowledge before they analyzed the models of each system. The models were presented in a structured tutorial format, covering key elements and justifications for their inclusion. Following this, participants answered questions designed to assess their comprehension and ability to interpret the notation accurately.

The FParsec (R2) case study focused on modeling a functional programming system and evaluating participants' comprehension through seven questions. The average score across all questions was 69.05%, indicating moderate success. Participants performed well in areas requiring interpretation of symbol combinations and transformations, achieving scores above 90% for questions involving these concepts. However, lower scores were observed for questions requiring the identification of main definitions (50%), sub-definitions (60%), and definitions that rely on others (46.67%). These results suggest that participants struggled with foundational elements of the notation, particularly when distinguishing between main definitions and their subtypes. Additionally, participants encountered difficulties interpreting symbols involving the **"!"** symbol, with scores in the 70% range. This highlights the need to refine the textual definitions accompanying these symbols to improve clarity.

The Docutils (R3) case study evaluated the notation's adaptability to non-functional systems through eight questions. The average score improved significantly to 79.58%,

likely due to participants' growing familiarity with the notation. Participants performed better in identifying main definitions (76.67%) and sub-definitions (76.67%), suggesting that repeated exposure to the notation enhanced their understanding of its foundational elements. However, a persistent challenge was observed with the notation in the form **X .. Y** notation, which represents the invocation of one component by another. A majority of participants incorrectly interpreted the result of **X .. Y** as X rather than Y, indicating a misunderstanding of the invocation concept. This underscores the need for clearer visual or textual cues to convey the directionality of such relationships.

The Pandoc (R4) case study assessed the notation's practicality in modeling a functional system through seven questions. The average score was 72.86%, reflecting a slight decline compared to the Docutils case study. Participants struggled with identifying main definitions (60%) and sub-definitions (70%), particularly when main definitions lacked alternatives or subtypes. This suggests that some participants may have mistakenly assumed that only definitions with alternatives qualify as main definitions. Additionally, participants faced challenges with the **X .. Y** notation (40% correct) and the $\oplus^{\mathbf{f}.\bigcirc}$ notation (60% correct), indicating that these elements require further refinement to improve interpretability.

The symbol-level (R5) evaluation revealed critical insights into the usability and clarity of individual notation elements. Participants generally found the **label :: symbol** and **symbol : letter** notations clear and straightforward, with high scores for comprehension and ease of use. However, the OR-Arrow symbol posed significant challenges. While 80% of participants understood the concept of multiple alternatives, only 60% found the symbol effective in conveying the "either-or" relationship, and 57% perceived ambiguity in interpreting subtype relationships. Participants noted that the OR-Arrow's design lacked intrinsic clarity, leading to potential misinterpretations related to sequential flow and implied dependencies.

To address this issue, an alternative representation (see Diagram B in Fig. 4.20) was introduced, featuring lines originating directly from the main definition to each subtype. This alternative was overwhelmingly preferred by participants, with 85% favoring it for its clearer and more direct representation of relationships. This finding underscores the importance of iterative design and user feedback in refining graphical notations.

A comparative analysis (R5) between the new notation (Diagram B in Fig. 4.16) and Motara's original notation (Diagram A in Fig. 4.25) revealed several key differences. Participants generally preferred Diagram B for symbol tracking, mental mapping, and adaptability to change, citing its use of letters for sub-definitions and its three-tiered structure

as significant improvements. However, Diagram B was less effective in conveying relationships and logical flow, particularly in Level 2 definitions. Participants noted that the relies relationship in Diagram B required scanning each definition to identify dependencies, unlike Diagram A, where dependencies were explicitly linked with lines. This suggests that while Diagram B excels in clarity and modularity, it could benefit from additional visual cues to enhance the representation of relationships.

The results of this study demonstrate the potential of the new notation to improve the modeling of functional systems. However, several areas for improvement have been identified, including the need for clearer textual definitions, more intuitive representation of invocation relationships, and enhanced visual cues for dependencies.

Chapter 6

Conclusion

6.1 Summary of Objectives and Contributions

This study aimed to develop a structural modeling notation for functional systems using the Physics of Diagrams (PoD) framework and to assess its effectiveness in improving system representation clarity, usability, and accessibility. The research was structured around five key objectives:

The first objective was successfully achieved through the integration of PoD principles into the notation design. The notation was structured into three hierarchical levels to facilitate comprehension, introduced a "label :: symbol" notation for main definitions, and adopted the "symbol : letter" notation for subtypes. Additional features, such as the OR-Arrow symbol and the relies relationship notation, were incorporated to improve expressiveness and usability.

6.2 Key Findings from Case Studies

6.2.1 Case Study 1: FParsec

The application of the notation to FParsec, a functional parsing library, revealed that participants could effectively interpret transformations and symbol combinations but struggled with foundational elements such as main definitions and relies relationships. The results highlighted areas where refinements were necessary, particularly in textual explanations and visual distinction of relationships.

6.2.2 Case Study 2: Docutils

The notation was tested on Docutils, a non-functional system, demonstrating its adaptability beyond functional programming. Scores improved compared to FParsec, suggesting that repeated exposure enhanced understanding. However, issues with interpreting the "X .. Y" notation indicated the need for clearer representations of invocation relationships.

6.2.3 Case Study 3: Pandoc

When applied to Pandoc, the notation effectively captured its structural elements, but participants continued to struggle with distinguishing main definitions and subtypes. This confirmed the need for further refinements to enhance clarity in hierarchical relationships.

6.3 Evaluation of Notation Elements

User feedback on individual notation elements revealed that while certain symbols, such as "label :: symbol" and "symbol : letter", were well received, others, particularly the OR-Arrow symbol, required redesign for better clarity. The comparative analysis of Diagram A (Motara's original notation) and Diagram B (the revised notation) demonstrated that the new notation improved symbol tracking and modularity but required enhancements in relationship representation.

6.4 Revisiting the Research Problem and Question

The problem statement highlighted the **lack of a standardized and accessible notation for functional systems** and the need to validate PoD's applicability to this domain. The study confirmed that PoD principles can significantly enhance notation design by improving clarity, usability, and accessibility. The empirical evaluation demonstrated that the revised notation effectively models functional and non-functional systems, bridging the gap between **theoretical notation design and practical usability**.

Can a typed modeling notation for functional programming, designed using the Physics of Diagrams framework, improve the clarity, usability, and accessibility of functional system representations for both developers and business

analysts? This study demonstrated that a **structured, hierarchical approach**, combined with **perceptually distinct graphical elements**, enhances the **clarity** of functional system representations. The usability of the notation was evident in its ability to model different system architectures, though refinements are needed for certain relationships and symbols. Accessibility was improved by aligning the notation with **cognitive design principles**, reducing interpretative ambiguity for both developers and business analysts.

6.5 Future Research Directions

While this research has laid a solid foundation for functional system modeling using PoD, several areas warrant further exploration:

- **Iterative Refinements Based on User Feedback:** Future studies should focus on refining ambiguous symbols, improving textual definitions, and enhancing visual cues for dependencies and invocation relationships.
- **Larger Sample Size for Empirical Validation:** Expanding the participant pool beyond 30 respondents will strengthen the statistical significance of findings and provide more diverse perspectives on notation usability.
- **Pairing the Notation with Behavioral Modeling:** To create a holistic modeling framework, future research should explore integrating the structural notation with behavioral modeling notations, ensuring comprehensive system representation.

By addressing these areas, future research can further enhance the clarity, usability, and adaptability of structural modeling for functional and non-functional systems alike.

Bibliography

Aguilar-Savén, Ruth Sara (2004). "Business process modelling: Review and framework". In: *International Journal of production economics* 90.2, pp. 129–149.

Alam, Abu S (2015). "A Programming System for End-User Functional Programming". PhD thesis. University of Gloucestershire.

Aldin, Laden and Sergio De Cesare (2009). "A comparative analysis of business process modelling techniques". In.

Armstrong, Joe (2010). "erlang". In: *Communications of the ACM* 53.9, pp. 68–75.

Baker-Finch, Clem (2013). "An introduction to the lambda calculus". In: *Received in 2007 as a handout from the RSISE seminar course "Beyond Classical Logic.*

Balać, Vladimir and Milan Vidaković (2017). *A model-Driven Approach to User-Specific Operations in the Development of Web Business Applications.*

Barendregt, Henk (1997). "The impact of the lambda calculus in logic and computer science". In: *Bulletin of Symbolic Logic* 3.2, pp. 181–215.

Bijlsma, A et al. (2020). "Restructering design patterns using functions in Java". In.

Bourget, Nicolle Marie (2014). "The role of information and communication technology within upriver halq'emeylem language initiatives: a case study." In.

Bower, Gordon H (1970). "Organizational factors in memory". In: *Cognitive psychology* 1.1, pp. 18–46.

Christoph, Fabian (2022). "Lambda Calculus and How It Shapes Programming Languages Today". In.

Church, Alonzo (1985). *The calculi of lambda-conversion.* 6. Princeton University Press.

Csörnyei, Zoltán and Gergely Dévai (2007). "An introduction to the lambda calculus". In: *Central European Functional Programming School.* Springer, pp. 87–111.

Decker, Gero et al. (2009). "The business process modeling notation". In: *Modern Business Process Automation: YAWL and its Support Environment.* Springer, pp. 347–368.

Dennis, Alan, Barbara Haley Wixom, and Roberta M Roth (2008). *Systems analysis and design.* John wiley & sons.

Desharnais, Jules, Marc Frappier, and Ali Mili (1998). "State transition diagrams". In: *Handbook on architectures of information systems*. Springer, pp. 147–166.

DRAFT, DENR (1989). "DATA DICTIONARY". In.

Edington, Collin D et al. (2018). "Interconnected microphysiological systems for quantitative biology and pharmacology studies". In: *Scientific reports* 8.1, p. 4530.

Ensmenger, Nathan (2016). "The multiple meanings of a flowchart". In: *Information & Culture* 51.3, pp. 321–351.

Faraldos, Pau (2023). "Literature survey on implementation techniques for type systems: Inductive data types and pattern matching". In.

Fisher, DL (1966). "Data, documentation, and decision tables". In: *Communications of the ACM* 9.1, pp. 26–31.

Ganesalingam, Mohan and Mohan Ganesalingam (2013). *The language of mathematics*. Springer.

Goldberg, Benjamin (1996). "Functional programming languages". In: *ACM Computing Surveys (CSUR)* 28.1, pp. 249–251.

Grouse, Phil (1978). "" Flowblocks" a technique for structured programming". In: *ACM SIGPLAN Notices* 13.2, pp. 46–56.

Gustafson, David A (1981). "Control flow, data flow & data independence". In: *ACM Sigplan Notices* 16.10, pp. 13–19.

Hagspiel, Simeon (2018). "Reliability with interdependent suppliers". In: *European Journal of Operational Research* 268.1, pp. 161–173.

Hajiagha, Seyed Hossein Razavi et al. (2018). "A novel common set of weights method for multi-period efficiency measurement using mean-variance criteria". In: *Measurement* 129, pp. 569–581.

Hermes, Mary and Kendall A King (2013). "Ojibwe language revitalization, multimedia technology, and family language learning". In.

Hinsen, Konrad (2009). "The promises of functional programming". In: *Computing in Science & Engineering* 11.4, pp. 86–90.

Hu, Zhenjiang, John Hughes, and Meng Wang (2015). "How functional programming mattered". In: *National Science Review* 2.3, pp. 349–370.

Hughes, John (1989). "Why functional programming matters". In: *The computer journal* 32.2, pp. 98–107.

Ibrahim, Rosziati et al. (2010). "Formalization of the data flow diagram rules for consistency check". In: *arXiv preprint arXiv:1011.0278*.

Izadikhah, Mohammad, Reza Farzipoor Saen, and Razieh Roostaee (2018). "How to assess sustainability of suppliers in the presence of volume discount and negative data in data envelopment analysis?" In: *Annals of Operations Research* 269, pp. 241–267.

Jancewicz, Bill and Marguerite MacKenzie (2002). "Applied computer technology in Cree and Naskapi language programs". In.

Jeffries, William and Alexander Brodsky (2017). "Composite Alternative Pareto Optimal Recommender System (CAPORS)". In: *International Conference on Enterprise Information Systems*. Vol. 2. SCITEPRESS, pp. 496–503.

Kim, ChanMin et al. (2015). "Robotics to promote elementary education pre-service teachers' STEM engagement, learning, and teaching". In: *Computers & education* 91, pp. 14–31.

Kocbek, Mateja et al. (2015). "Business process model and notation: The current state of affairs". In: *Computer Science and Information Systems* 12.2, pp. 509–539.

Kühne, Thomas (1999). *A functional pattern system for object-oriented design*. Kovac.

– (2006). "Matters of (meta-) modeling". In: *Software & Systems Modeling* 5, pp. 369–385.

Larsen, Peter Gorm, Nico Plat, and Hans Toetenel (1994). "A formal semantics of data flow diagrams". In: *Formal aspects of Computing* 6, pp. 586–606.

Liskin, Olga (2015). "How artifacts support and impede requirements communication". In: *Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23-26, 2015. Proceedings 21*. Springer, pp. 132–147.

Lowe, Richard K (2003). "Animation and learning: Selective processing of information in dynamic graphics". In: *Learning and instruction* 13.2, pp. 157–176.

Ludewig, Jochen (2003). "Models in software engineering–an introduction". In: *Software and Systems Modeling* 2, pp. 5–14.

Mertz, David (2015). *Functional Programming in Python*. O'Reilly Media.

Mitkin, Stepan (2012). "DRAKON-Erlang: Visual functional programming". In: *Slashdot Media.–http://drakon-editor. sourceforge. net/drakon-erlang/intro. html*.

Moody, Daniel (2009). "The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering". In: *IEEE Transactions on software engineering* 35.6, pp. 756–779.

Motara, Yusuf Moosa (2021). "High-Level Modelling for Typed Functional Programming". In: *Trends in Functional Programming: 22nd International Symposium, TFP 2021, Virtual Event, February 17–19, 2021, Revised Selected Papers 22*. Springer, pp. 69–94.

Mueller, John Paul (2019). *Functional programming for dummies*. John Wiley & Sons.

Myers, Michael D (1997). "Qualitative research in information systems". In: *MIS Quarterly* 21.2, pp. 241–242.

Nassi, Ike and Ben Shneiderman (1973). "Flowchart techniques for structured programming". In: *ACM Sigplan Notices* 8.8, pp. 12–26.

Nobakht, Mehdi and Dragos Truscan (2013). "An approach for validation, verification, and model-based testing of uml-based real-time systems". In: *the Eighth International Conference on Software Engineering Advances (ICSEA 2013)*, pp. 79–85.

Noor, Ayesha et al. (2024). "E-Learning Challenges Faced by University Teachers During COVID-19 Pandemic". In: *The Qualitative Report* 29.6, pp. 1742–1757.

Odarushchenko, O et al. (2022). "Improving the Accuracy of Software Reliability Modeling by Predicting the Number of Secondary Software Defects". In: *CEUR Workshop Proceedings*. Vol. 3156. CEUR Workshop Proceedings, pp. 198–207.

Pepels, Betsy, Rinus Plasmeijer, and Henderik Alex Proper (2006). "Fact-Oriented Modeling from a Programming Language Designer's Perspective". In: *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops: OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWWS, and SeBGIS 2006, Montpellier, France, October 29-November 3, 2006. Proceedings, Part II*. Springer, pp. 1170–1180.

Pierce, Benjamin C (2002). "Types and Programming Languages". In: *Proceedings of the Eighth ACM SIGPLAN*, pp. 141–152.

Pissierssens, Sarah, Jan Claes, and Geert Poels (2019). "The" Physics of Diagrams": Revealing the scientific basis of graphical representation design". In: *arXiv preprint arXiv:1903.05941*.

Pollak, David, Vishal Layka, and Andres Sacco (2022). "Functional programming". In: *Beginning Scala 3: A Functional and Object-Oriented Java Language*. Springer, pp. 79–109.

Reynolds, John C (1972). "Definitional interpreters for higher-order programming languages". In: *Proceedings of the ACM annual conference-Volume 2*, pp. 717–740.

Rojas, Cristóbal (2008). "Computability and Information in models of Randomness and Chaos". In: *Mathematical Structures in Computer Science* 18.2, pp. 291–307.

Rosing, Mark von et al. (2015). "Business Process Model and Notation—BPMN". In.

Rothenberg, Jeff et al. (1989). "The nature of modeling". In: *in Artificial Intelligence, Simulation and Modeling*.

Roy, Geoffrey G, Joel Kelso, and Craig Standing (1998). "Towards a visual programming environment for software development". In: *Proceedings. 1998 International Conference Software Engineering: Education and Practice (Cat. No. 98EX220)*. IEEE, pp. 381–388.

Ruona, Wendy EA (2005). "Analyzing Qualitative Data". In: *RESEARCH in ORGANIZATIONS*, p. 233.

Sandvig, Christian (1942). "Seeing the sort: The aesthetic and industrial defense of "the algorithm"". In: *Journal of the New Media Caucus—ISSN*.

Saumont, Pierre-Yves (2017). *Functional Programming in Java*. Manning Publications.

Silva Teixeira, Maria das Graças da (2017). "An ontology-based process for domain-specific visual language design". PhD thesis. Ghent University.

Stachowiak, Herbert (1973). *Allgemeine Modelltheorie*. Wien, New York: Springer Verlag. URL: `https://archive.org/details/Stachowiak1973AllgemeineModelltheorie`.

Sun, Junmei et al. (2023). "Promoting the AI teaching competency of K-12 computer science teachers: A TPACK-based professional development approach". In: *Education and Information Technologies* 28.2, pp. 1509–1533.

Szlenk, Marcin (2011). "Metamodel and uml profile for functional programming languages". In: *Dependable Computer Systems*. Springer, pp. 233–242.

Tao, Yonglei and Chenho Kung (1991). "Formal definition and verification of data flow diagrams". In: *Journal of Systems and Software* 16.1, pp. 29–36.

Turner, David A (2012). "Some history of functional programming languages". In: *International Symposium on Trends in Functional Programming*. Springer, pp. 1–20.

Virding, Robert, Claes Wikström, and Mike Williams (1996). *Concurrent programming in ERLANG*. Prentice Hall International (UK) Ltd.

Wakeling, David (2001). "A design methodology for functional programs". In: *International Workshop on Semantics, Applications, and Implementation of Program Generation*. Springer, pp. 146–161.

Wallingford, Eugene (2002). "Functional programming patterns and their role in instruction". In: *Proceedings of the international conference on functional programming*, pp. 151–163.

Weck, Tobias and Matthias Tichy (2016). "Visualizing data-flows in functional programs". In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE, pp. 293–303.

Wittgenstein, Ludwig (2009). *Philosophical investigations*. John Wiley & Sons.

Appendix A

Ethics Approval

RHODES UNIVERSITY
Where leaders learn

RU 120
1904-2024

Rhodes University Human Research Ethics Committee
Main Admin Building, Drostdy Road, Makhanda, 6139, South Africa
PO Box 94, Makhanda, 6140, South Africa
t: +27 (0) 46 603 7314
e: ethics-committee@ru.ac.za
https://www.ru.ac.za/researchgateway/ethics/
NHREC Registration number: RC-241114-045

15 October 2024

Mr Carlton Mpofu, ,

Email: mpofucarlton15@gmail.com

Review Reference: 2024-8015-9172

Dear Mr Carlton Mpofu

Re: Modelling of functional programming

Researcher: Mr Carlton Mpofu

Supervisor(s): Dr Yusuf Motara

This letter confirms that the above research proposal has been reviewed and **APPROVED** by the Rhodes University Human Research Ethics Committee (RU-HREC). Your Approval number is: 2024-8015-9172

Approval has been granted for 1 year. An annual progress report will be required in order to renew approval for an additional period. You will receive an email notifying you when the annual report is due.

Please apply for a protocol amendment should any substantive change(s) be made, for whatever reason, during the research process. This includes changes in investigators. Email your request to ethics-committee@ru.ac.za.

Please submit a brief report to the ethics committee on the completion of the research. The purpose of this report is to indicate whether the research was conducted successfully, if any aspects could not be completed, or if any problems arose that the ethical standards committee should be aware of.

If a thesis or dissertation arising from this research is submitted to the library's electronic theses and dissertations (ETD) repository, please notify the committee of the date of submission and/or any reference or cataloguing number allocated.

Sincerely,

Janet Hayward

Dr Janet Hayward

Chair: Rhodes University Human Research Ethics Committee (RU-HREC)

Appendix B

Survey questionnaire

Welcome

Thank you for your interest in this survey.

Please enter the following information

What is your primary role?

- o Student
- o Developer
- o Business Analyst
- o Other (please specify)

How would you rate your familiarity with functional programming?

- o Beginner
- o Intermediate
- o Advanced

Purpose

The notation is designed to represent the structure of typed functional programming systems, emphasizing the clear visualization of their components and the relationships between them. It provides a framework for modeling functional architectures, capturing the essence of typed systems, their definitions, and how different elements within the system interact with one another.

Core Structure

Level 1 consists of the main definitions, represented by **square rectangles**. These elements define core system concepts such as main types and subtypes.

Level 2 focuses on the manipulation of the main types, illustrated with **rounded rectangles**. These represent derived definitions that interact with the main types.

Level 3 introduces thesaurus entries, depicted as **rectangles with a diagonal slash corner**. These provide additional information or summaries that enhance understanding without altering the core functionality.

Relationships between these definitions, such as "alias" or "relies" connections, are expressed using specific lines and symbols that indicate how elements build upon one another.

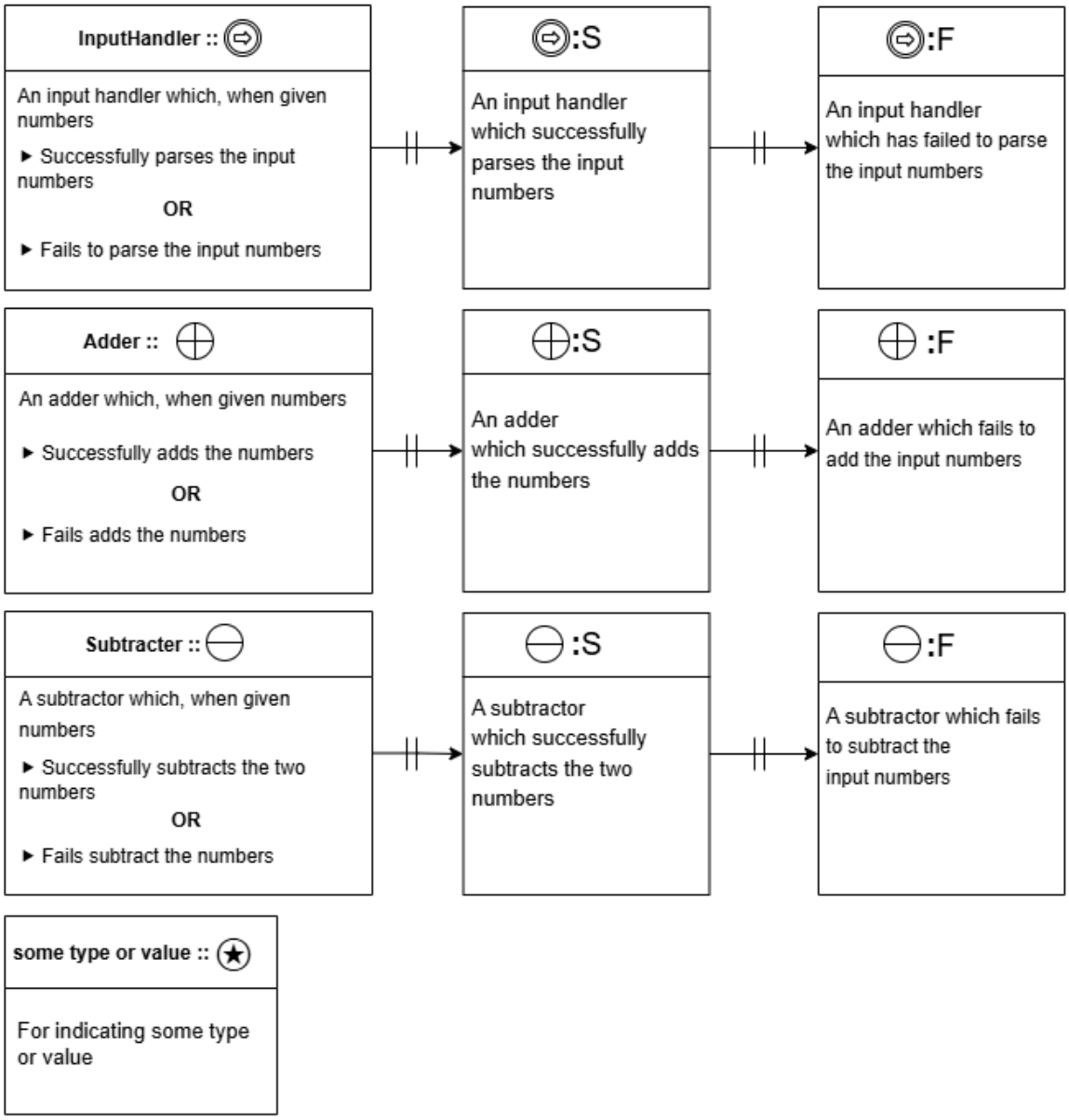
Example

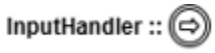
Let’s try to model a simple system that performs basic arithmetic operations: addition and subtraction. It takes two numbers as input and produces their sum or difference as output.

Level 1

In Level 1, we define the primary types or main definitions along with their subtypes or sub-definitions. For example, a diagram at this level might include an **inputHandler** that checks whether inputs are numbers before proceeding with calculations. It could also feature an **adder** and a **subtractor** for performing addition and subtraction on the numbers, respectively. Additionally, a definition might be included to represent **some type or value** that is generated as a result of these operations.

Level 1: Main definitions and sub-definitions



In the diagram above for the **InputHandler**, we have the main definition and the symbol used to represent it, shown as 

where **InputHandler** is the name of the main definition and  is the symbol used to represent it. We use the  symbol, pronounced as ***Or-Arrow***, to break down the notation into its sub-definitions. The first ***Or-Arrow*** starts from the main definition and points to the first sub-definition, with subsequent ***Or-Arrows*** points to the next sub-definition. The notation can be interpreted as: “The main definition can be represented using this sub-definition OR this sub-definition OR this sub-definition”.

Sub-definitions are represented in the form “**symbol: label**”, where the symbol corresponds to the one from the main definition, and the label indicates the nature of the sub-definition. For example, 

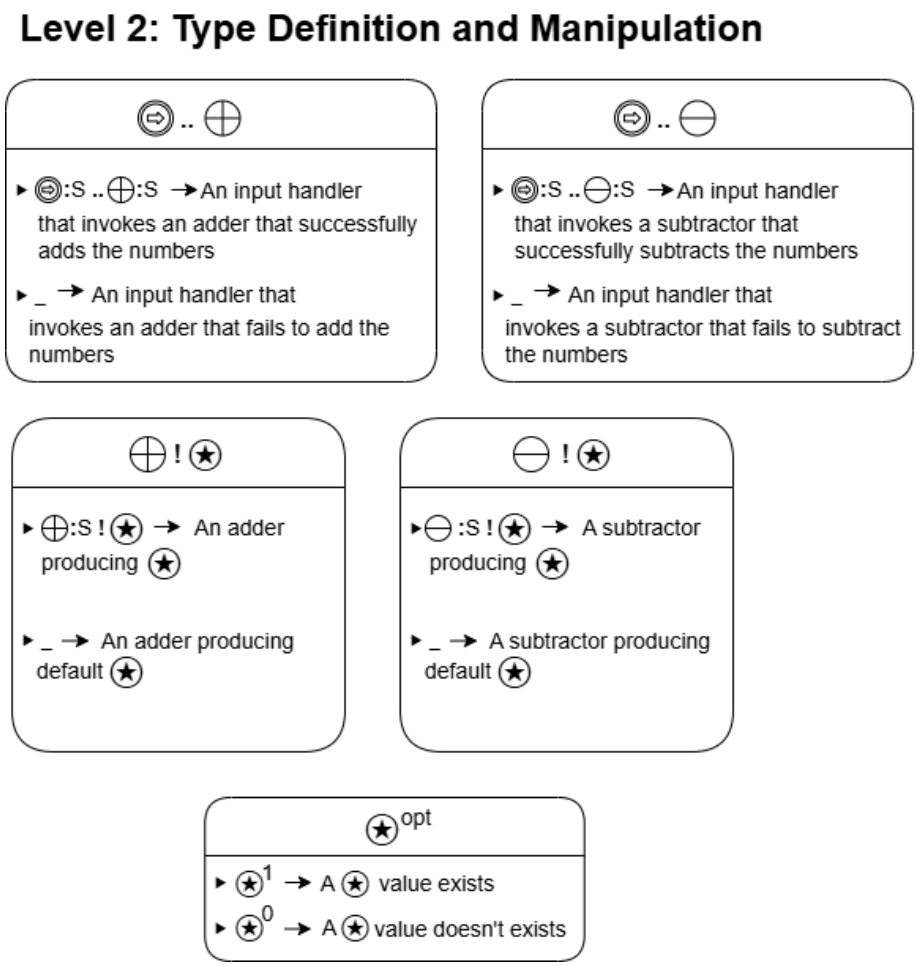
and 

represent an **adder** that **Successfully** adds the two numbers and an adder that **Fails** to add the numbers, respectively.

These definitions may represent a function, library, or module depending on the context.

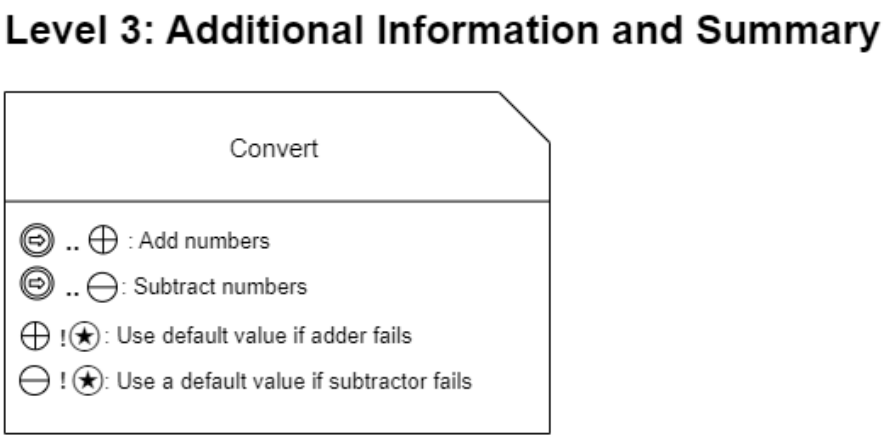
Level 2

In Level 2, we define definitions that result from manipulating the main definitions introduced in Level 1. For instance, this level might include definitions for **invoking** the adder or subtractor, representing **successful addition** or **subtraction** of numbers. Additionally, it could account for cases where the **operation fails**, capturing both successful and unsuccessful outcomes as part of the system's behavior.



Level 3

Finally, in Level 3, we group the definitions from Level 2 and provide additional information for analysts seeking a higher-level overview of the system.



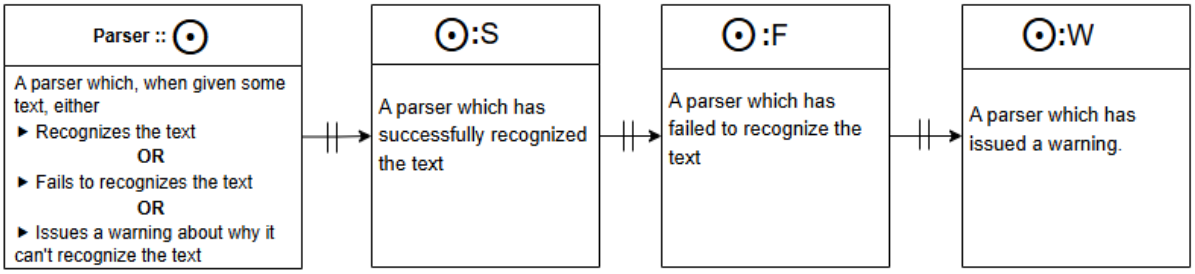
CASE 1

FParsec is a powerful parser combinator library for the F# programming language. It allows you to create efficient and readable parsers for various text-based formats by combining smaller parsers into more complex ones. When parsing fails, FParsec generates informative error messages that can help you identify and resolve parsing issues.

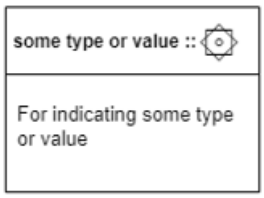
Model Creation

For our level 1 definitions, we define the main definitions and their sub-definitions.

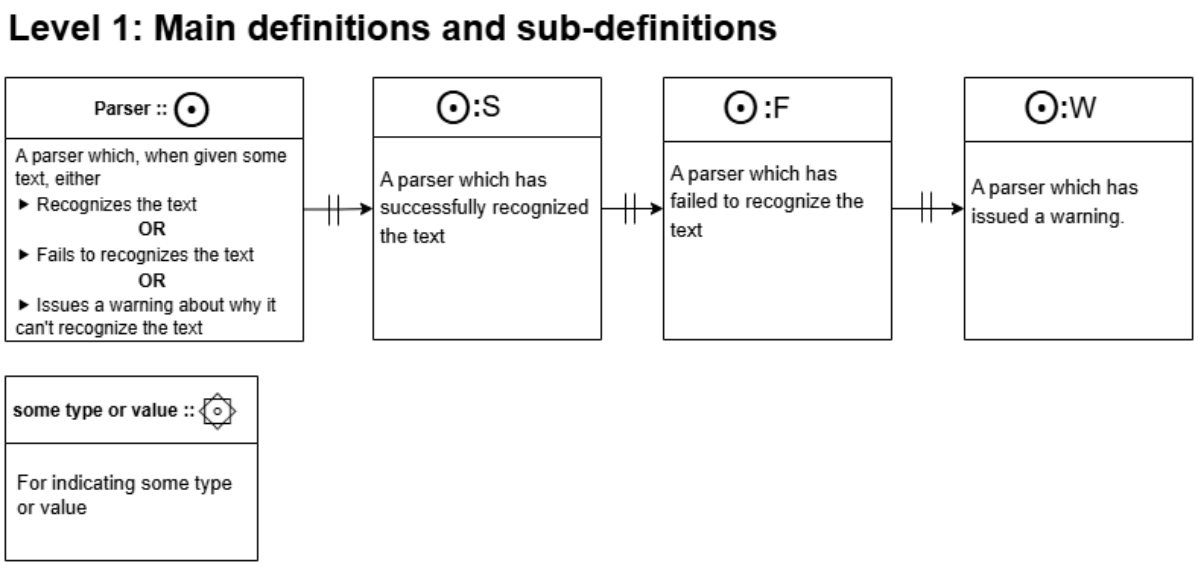
At its simplest, in FParsec, you create a parser that processes some text. This leads us to a parser that, when given some text, either recognizes the text, fails to recognize it, or issues a warning explaining why it cannot recognize the text. The main definition with its sub-definitions is represented below.



A powerful feature of typed functional programming languages is the ability to represent generic types. Here we define an additional sub-definition for representing some type or value.



The full set of level 1 definitions is:

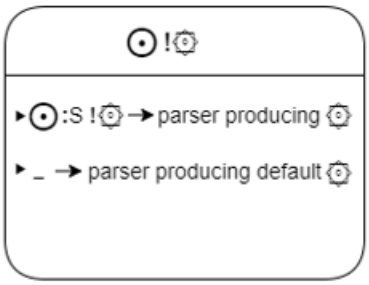


Moving on to Level 2

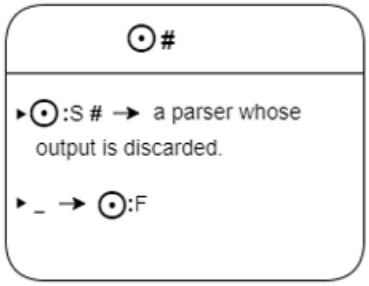
Definitions in Level 2 result from manipulating the definitions in Level 1.

It is important to note that, `▸` is used to represent discrete cases or branches in functional programming, indicating specific conditions or choices within a function or expression that result in different behaviours based on the input. The `_` represents any other case or a fall-through scenario. → is used to denote the mapping of one case or value to another.

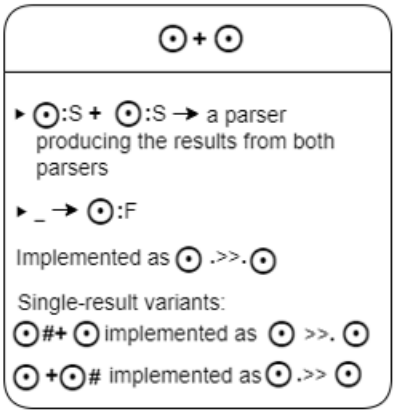
A parser that has successfully parsed text will produce some kind of value, while anything else (a parser that produces a warning or fails) will produce a default value. This is represented by the following definition:



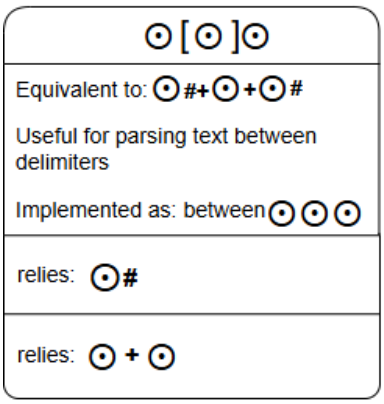
We may want to discard certain text that is successfully parsed because it may not be needed in the final output. This can be represented as follows:



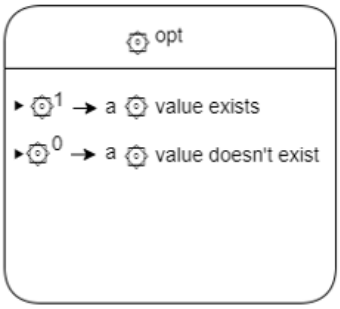
In FParsec, we can combine parsers to form more complex ones. Therefore, we need a way to represent the combination of two parsers:



You might want to discard certain delimiters at the beginning and end of some text while keeping the information in between. We can achieve this by **relying** on the previous definitions to create the following definition:

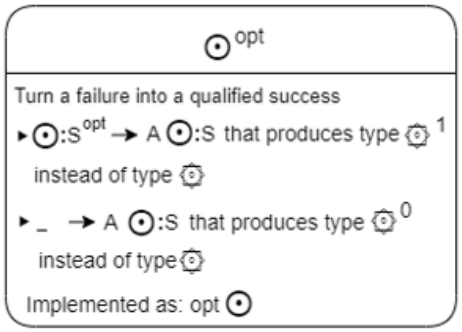


In functional programming, it’s common to represent the presence or absence of a value using types that encapsulate the possibility of a "missing" or "none" state. We represent this in the following definition:



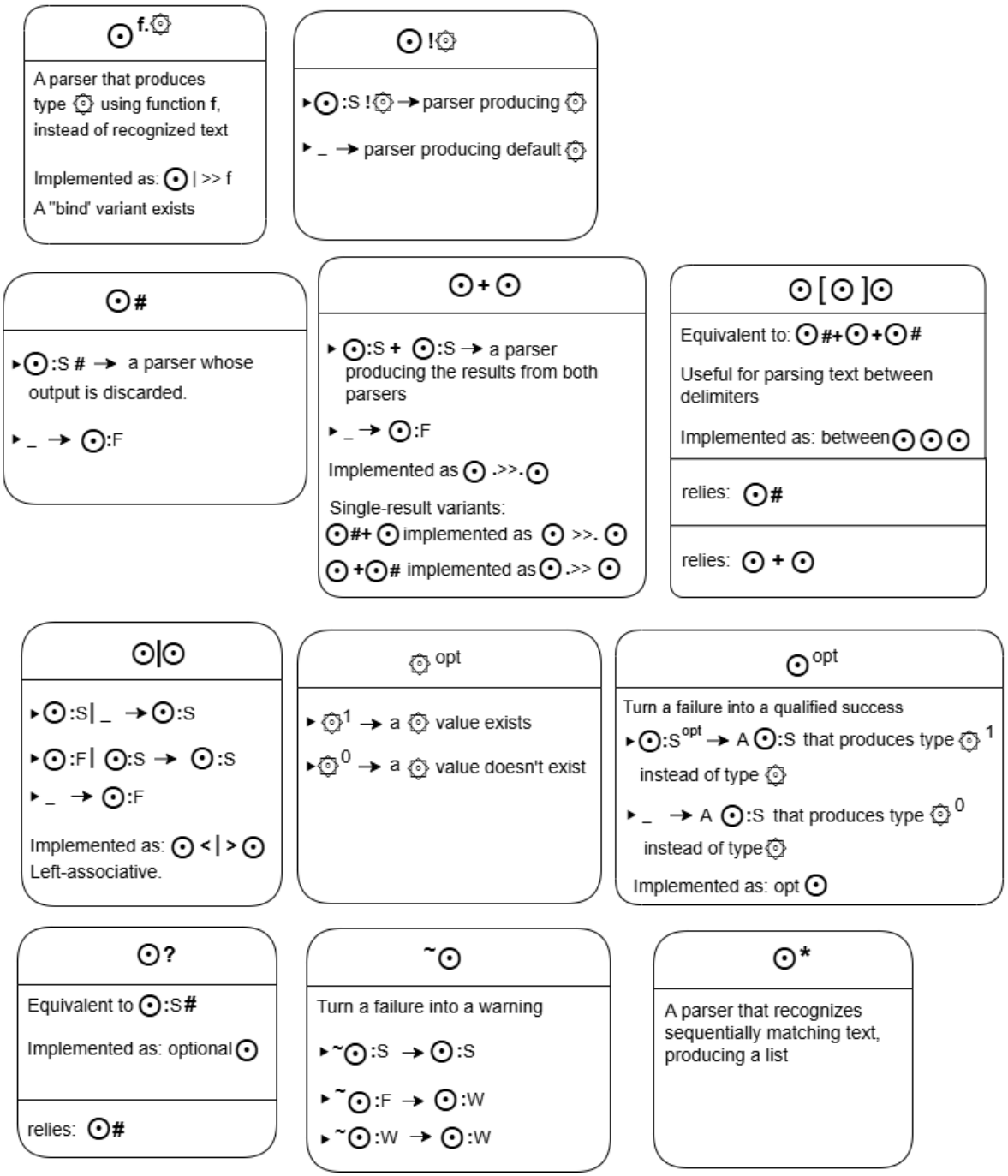
In functional programming, such definitions enable graceful error handling by ensuring that results clearly indicate success or failure. This approach integrates error handling directly into the program's flow, reducing reliance on exceptions and promoting safer, more predictable code.

We can represent this by turning a failure into a qualified success:



The full set of Level 2 definitions, along with additional definitions, is shown below.

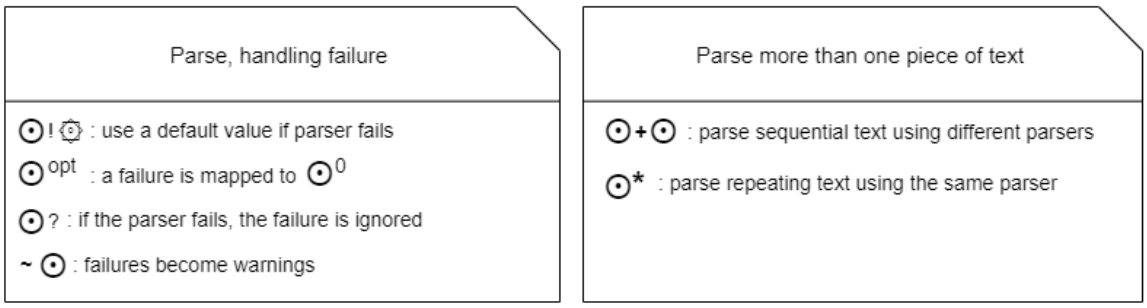
Level 2: Type Definition and Manipulation



Finally, Level 3

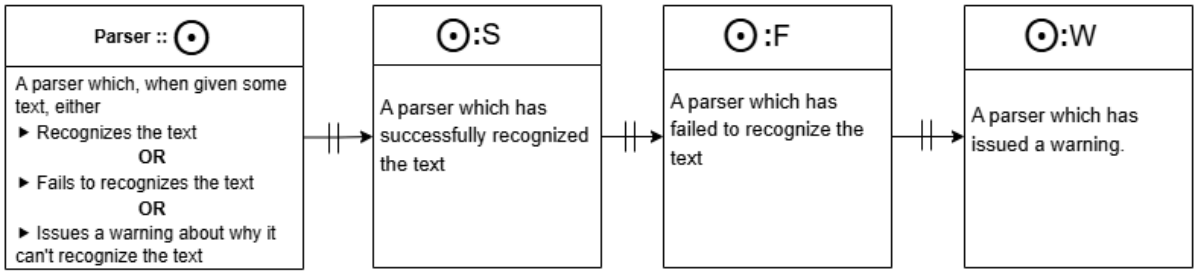
Level 3 essentially groups the Level 2 definitions into categories and provides a summary of what each definition does, giving an overview of the system.

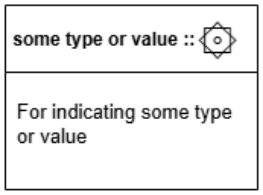
Level 3: Additional Information and Summary



The full FParsec diagram can be found below

Level 1: Main definitions and sub-definitions





Level 2: Type Definition and Manipulation

⊙ f. ⚙

A parser that produces type ⚙ using function **f**, instead of recognized text

Implemented as: ⊙ | >> f

A "bind' variant exists

⊙ ! ⚙

▸ ⊙:S ! ⚙ → parser producing ⚙

▸ _ → parser producing default ⚙

⊙

▸ ⊙:S # → a parser whose output is discarded.

▸ _ → ⊙:F

⊙ + ⊙

▸ ⊙:S + ⊙:S → a parser producing the results from both parsers

▸ _ → ⊙:F

Implemented as ⊙ .>>. ⊙

Single-result variants:

⊙ #+ ⊙ implemented as ⊙ >>. ⊙

⊙ +⊙ # implemented as ⊙ .>> ⊙

⊙ [⊙] ⊙

Equivalent to: ⊙ #+ ⊙ +⊙ #

Useful for parsing text between delimiters

Implemented as: between ⊙ ⊙ ⊙

relies: ⊙ #

relies: ⊙ + ⊙

⊙ | ⊙

▸ ⊙:S | _ → ⊙:S

▸ ⊙:F | ⊙:S → ⊙:S

▸ _ → ⊙:F

Implemented as: ⊙ < | > ⊙

Left-associative.

⚙ opt

▸ ⚙ 1 → a ⚙ value exists

▸ ⚙ 0 → a ⚙ value doesn't exist

⊙ opt

Turn a failure into a qualified success

▸ ⊙:S opt → A ⊙:S that produces type ⚙ 1 instead of type ⚙

▸ _ → A ⊙:S that produces type ⚙ 0 instead of type ⚙

Implemented as: opt ⊙

⊙ ?

Equivalent to ⊙:S#

Implemented as: optional ⊙

relies: ⊙ #

~⊙

Turn a failure into a warning

▸ ~⊙:S → ⊙:S

▸ ~⊙:F → ⊙:W

▸ ~⊙:W → ⊙:W

⊙ *

A parser that recognizes sequentially matching text, producing a list

Level 3: Additional Information and Summary

Parse, handling failure	Parse more than one piece of text
⊙! ⚙ : use a default value if parser fails ⊙opt : a failure is mapped to ⊙0 ⊙? : if the parser fails, the failure is ignored ~ ⊙ : failures become warnings	⊙+⊙ : parse sequential text using different parsers ⊙* : parse repeating text using the same parser

Questions

1. Identify all the main definitions in the FParsec diagram

- ☐ Parser :: ⊙
- ☐ ⊙:S
- ☐ ⊙:F
- ☐ ⊙:W
- ☐ some type or value :: ⚙
- ☐ ⊙f.⚙
- ☐ ⊙!⚙
- ☐ ⊙#
- ☐ ⊙+⊙
- ☐ ⊙[⊙]⊙
- ☐ ⊙|⊙
- ☐ ⚙opt
- ☐ ⊙opt
- ☐ ⊙?
- ☐ ~⊙
- ☐ ⊙*

2. Identify all the sub-definitions in the FParsec diagram

- ☐ Parser :: ⊙
- ☐ ⊙:S
- ☐ ⊙:F
- ☐ ⊙:W
- ☐ some type or value :: ⯐
- ☐ ⊙.⯐
- ☐ ⊙!⯐
- ☐ ⊙#
- ☐ ⊙+⊙
- ☐ ⊙[⊙]⊙
- ☐ ⊙|⊙
- ☐ ⯐ opt
- ☐ ⊙opt
- ☐ ⊙?
- ☐ ~⊙
- ☐ ⊙*

3. Identify all the definitions that rely on other definitions

- ☐ Parser :: ⊙
- ☐ ⊙:S
- ☐ ⊙:F
- ☐ ⊙:W
- ☐ some type or value :: ⯐

- ☐ _ ⊙f.⚙
- ☐ _ ⊙!⚙
- ☐ _ ⊙#
- ☐ _ ⊙+⊙
- ☐ _ ⊙[⊙]⊙
- ☐ _ ⊙|⊙
- ☐ _ ⚙opt
- ☐ _ ⊙opt
- ☐ _ ⊙?
- ☐ _ ~⊙
- ☐ _ ⊙*

4. What does the definition ⊙!⚙ signify in the context of parsing?

- A) The parser repeats parsing until it succeeds.
- B) The parser produces a value _⚙ if successful, and nothing if it fails.
- C) The parser produces a value ⚙ by default if parsing fails.
- D) The parser produces multiple values of type ⚙.

5. **In _⚙opt, what does the presence of ⚙1 versus _⚙0 indicate?**

- A) ⚙1 represents a default value, while ⚙0 is an error.
- B) ⚙1 indicates a value exists, while ⚙0 indicates absence.
- C) ⚙1 indicates an optional value, while ⚙0 represents success.
- D) ⚙1 indicates failure, while ⚙0 represents success.

6. **How would you interpret _ (⊙:S#) + (⊙!⚙) _?**

- A) Combining two parsers where the result of the first is discarded, and the second produces ⚙ on failure.
- B) Parsing a sequence of text, where the first parse discards output, and the second produces multiple values of ⚙.
- C) Parsing sequential text segments, each with its parser producing identical results.
- D) A successful parse produces the same result from both parsers.

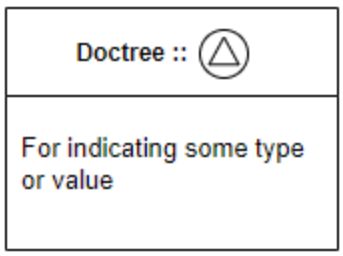
7. **How would _(⊙ [⊙] ⊙)^{opt} behave in the context of parsing?**

- A) It produces a discarded result for failures, mapping success to ⚙^{0}.
- B) It parses with delimiters, discarding the outer text and mapping to ⚙^{1} if successful or ⚙^{0} if absent.
- C) It repeats parsing text until a successful parse, producing optional values.
- D) It uses a default delimiter for parsing sequences with errors.

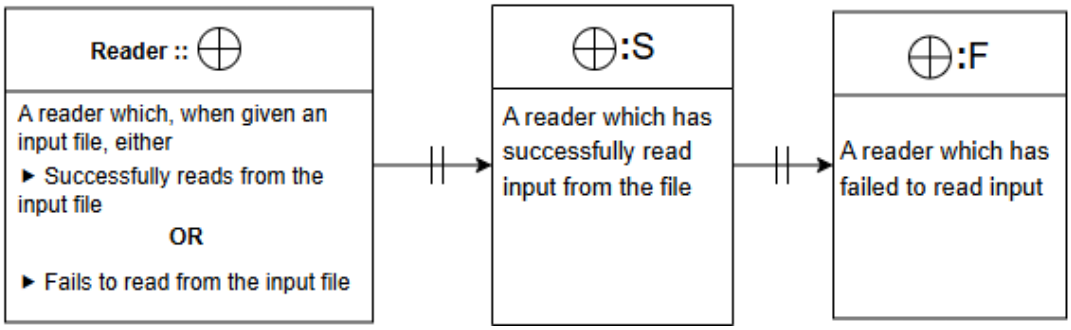
Case 2

Docutils is an open-source text processing system written in Python, designed to parse and transform reStructuredText (reST) documents. It converts reST into formats like HTML, LaTeX, and XML, making it a popular tool for technical documentation. Its architecture includes **parsers for reading input**, a **document tree** for structure representation, and **writers** for output generation.

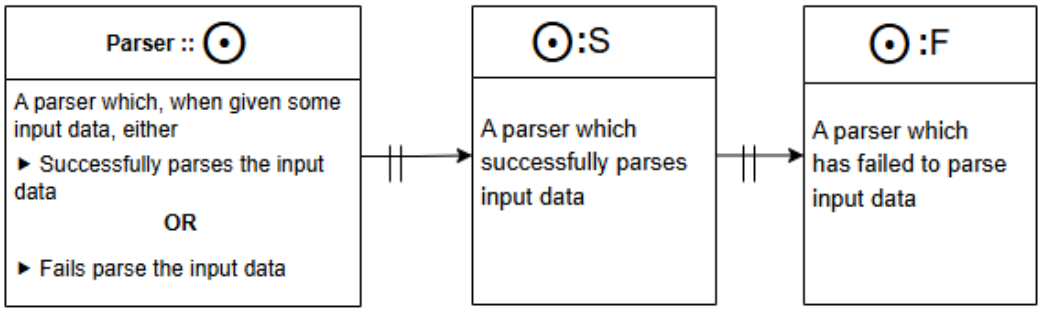
We need a definition to represent the structure of the document tree.



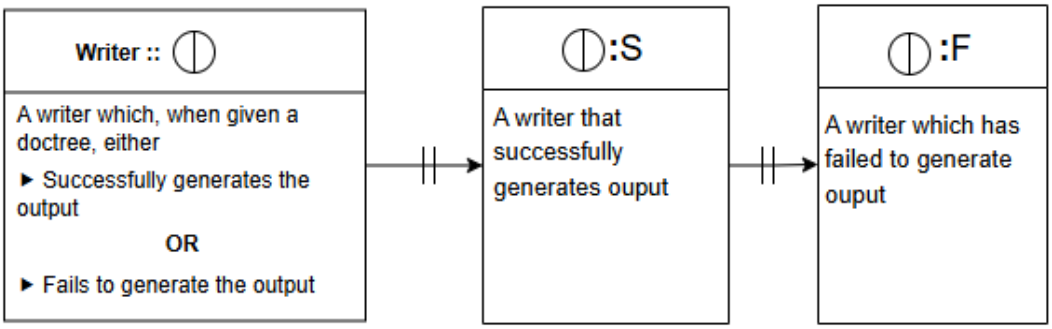
Readers play an important role in Docutils; they help read the input data and pass it to the parser.
We represent the reader with the following definition:



Parsers generate a document tree (doctree) from the input file. We define the parser as follows:

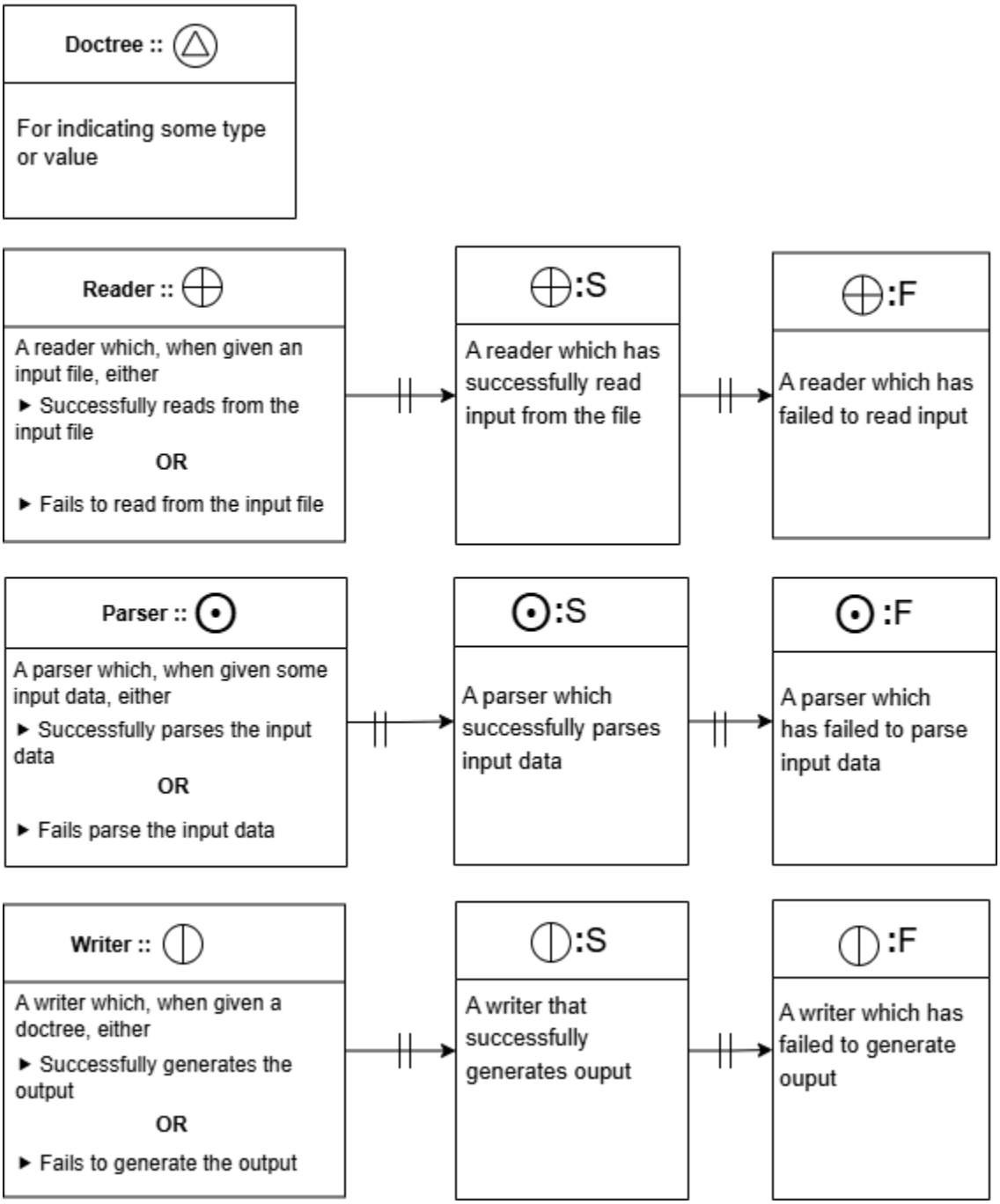


Docutils has writers that translate the document tree into the output format.



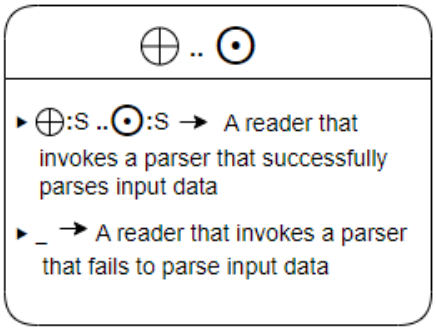
The complete level 1 diagram is found below:

Level 1: Main definitions and sub-definitions

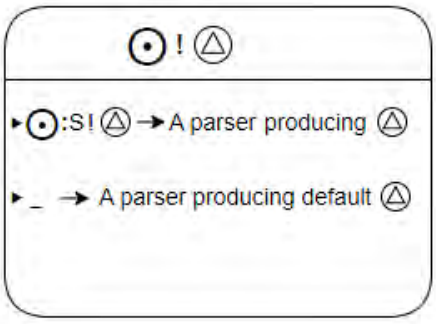


Moving on to level 2

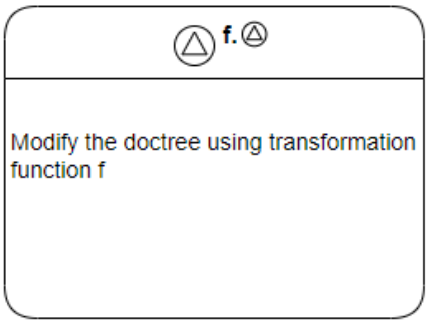
Readers pass the input data to the parser so it can be converted into an AST:



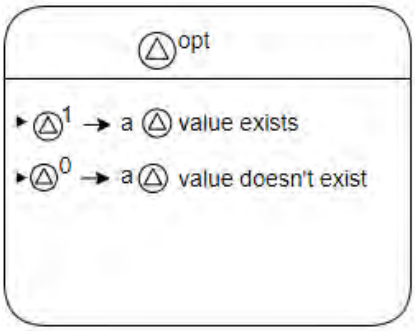
A parser generates a doctree:



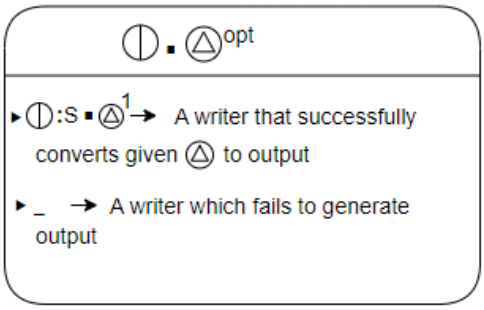
Docutils has transformers that modify the document tree:



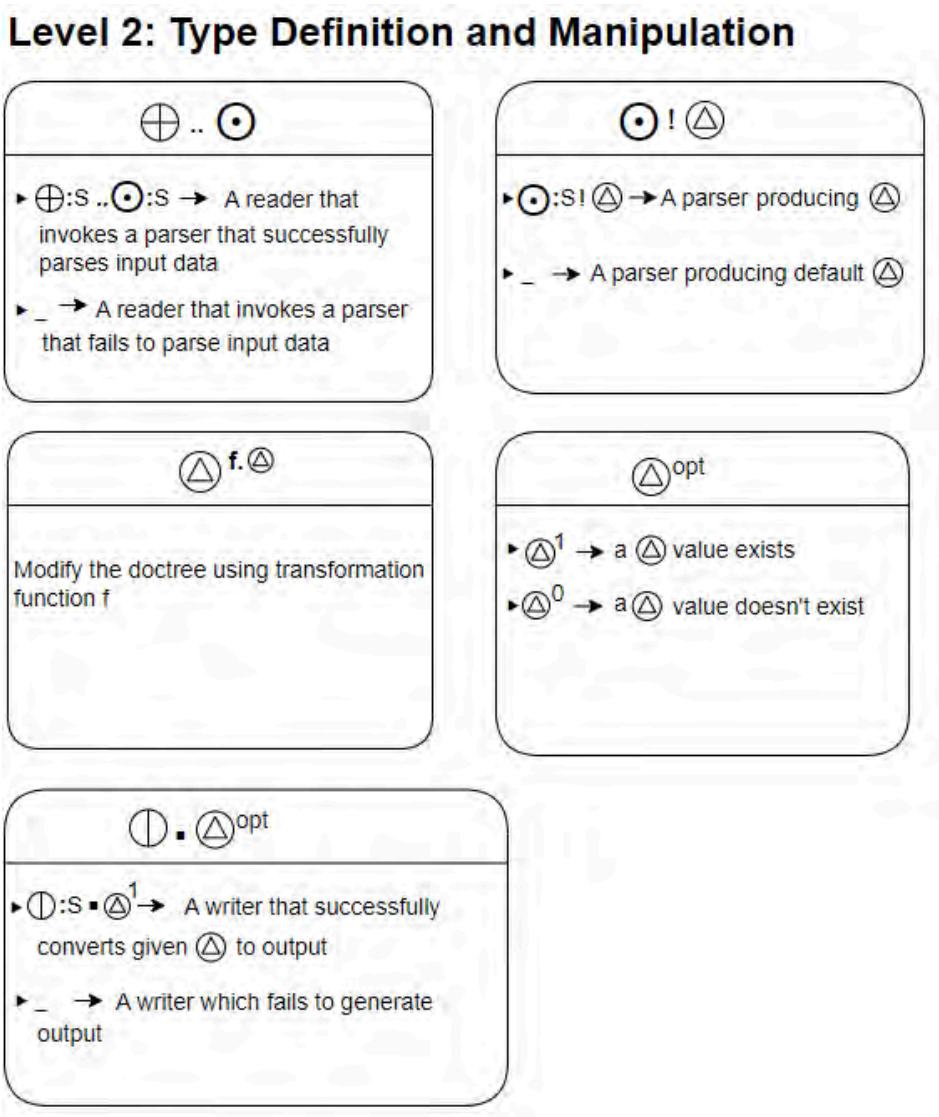
In functional programming, the presence or absence of a value is often represented using types that encapsulate the possibility of a "missing" or "none" state. We represent this in the following definition for a doctree:



Writers translate the doctree into output:

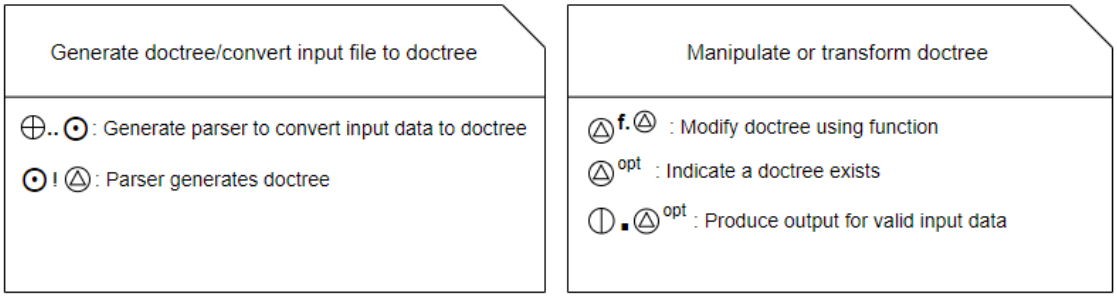


The complete level 2 diagram is found below.



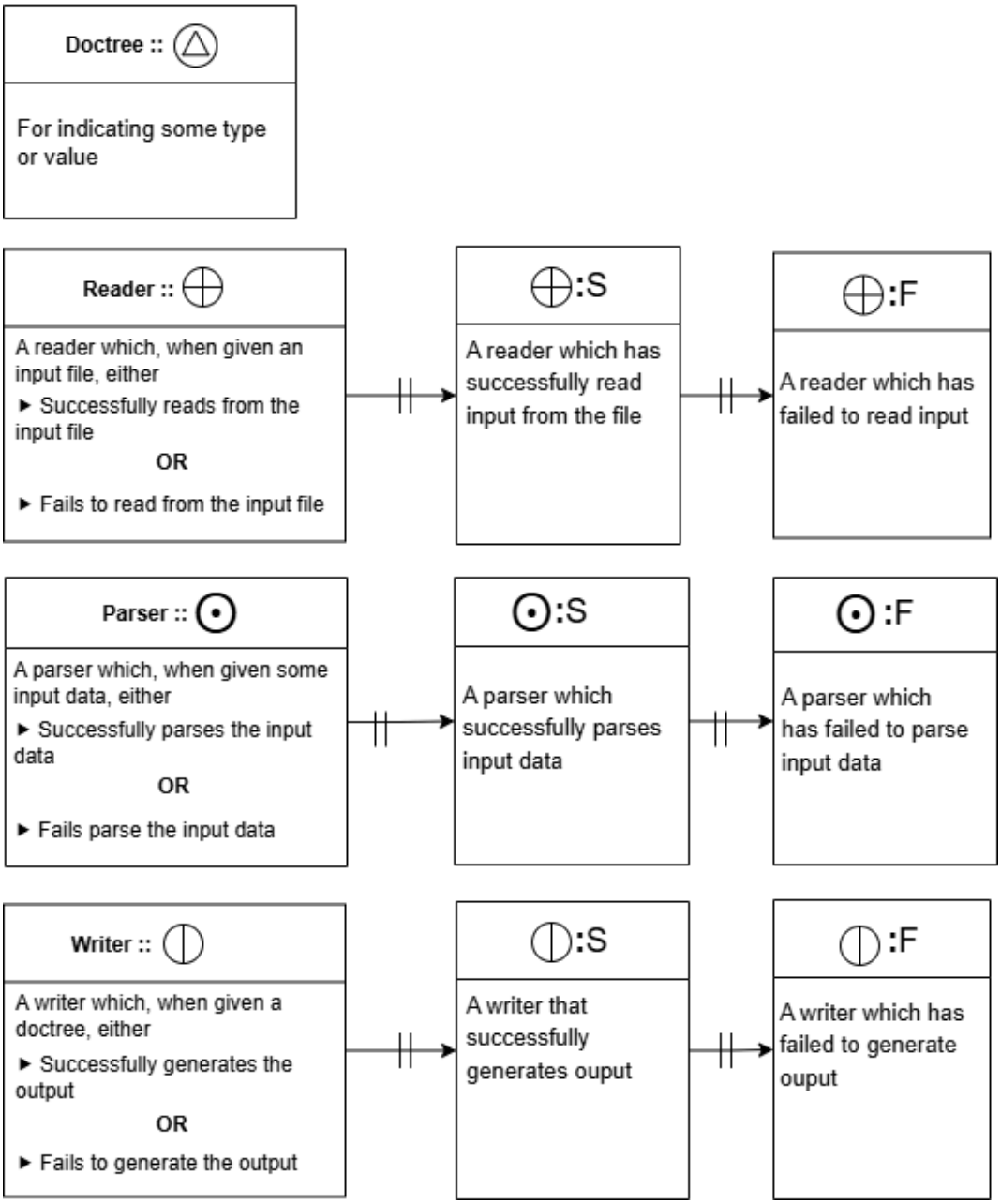
And finally, level 3

Level 3: Additional Information and Summary

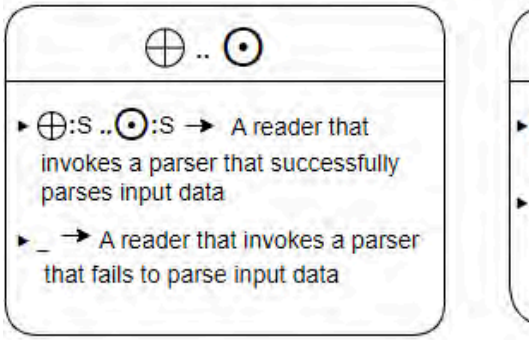


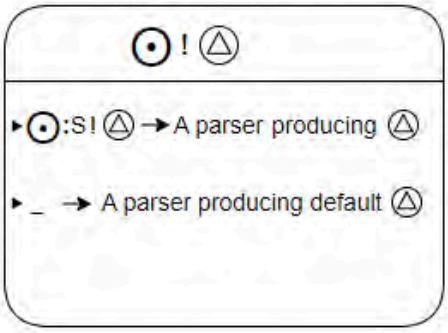
The complete Docutils diagram is found below.

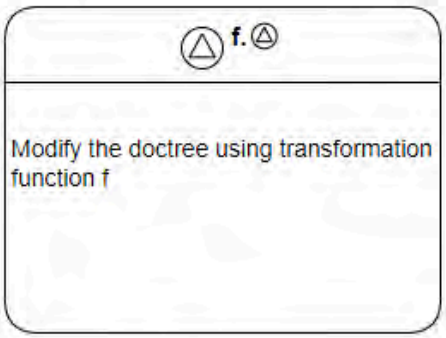
Level 1: Main definitions and sub-definitions

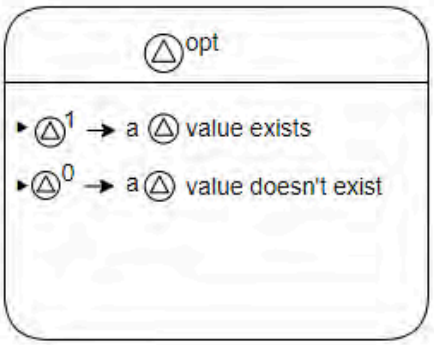


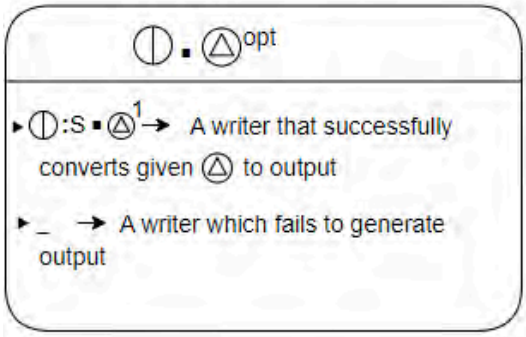
Level 2: Type Definition and Manipulation



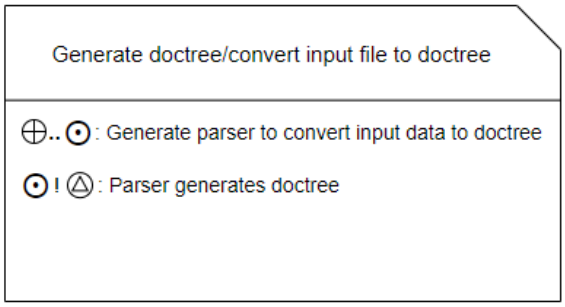


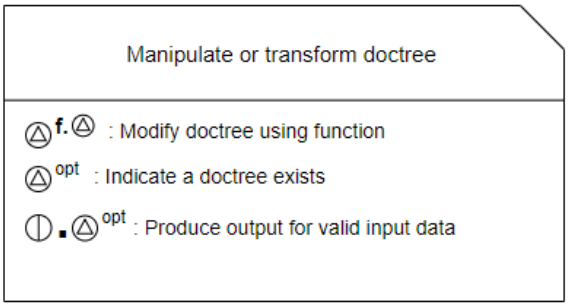






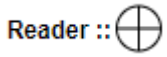
Level 3: Additional Information and Summary





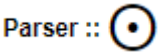
Questions

1. Identify all the main definitions in the Docutils diagram
   - □ Option 1: _ **Doctree ::** △

- ☐ Option 2: 

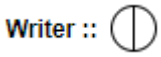
- ☐ Option 3: 

- ☐ Option 4: 

- ☐ Option 5: 

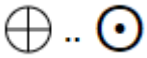
- ☐ Option 6: 

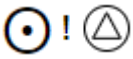
- ☐ Option 7: 

- ☐ Option 8: 

- ☐ Option 9: 

- ☐ Option 10: 

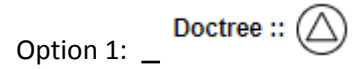
- ☐ Option 11: 

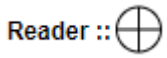
- ☐ Option 12: 

- ☐ Option 13: 

- ☐ Option 14: 

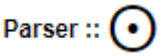
- ☐ Option 15: 

2. Identify all the sub-definitions in the Docutils diagram
   - ☐ Option 1: 

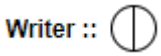
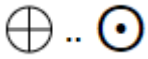
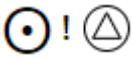
   - ☐ Option 2: 

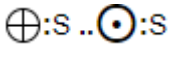
   - ☐ Option 3: 

   - ☐ Option 4: 

   - ☐ Option 5: 

   - ☐ Option 6: 

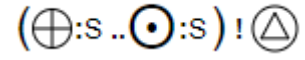
- Option 7: 
- Option 8: 
- Option 9: 
- Option 10: 
- Option 11: 
- Option 12: 
- Option 13: 
- Option 14: 
- Option 15: 

3. What is the result of _  _?

- A) A reader.
- B) Input data.
- C) A doctree.
- D) A parser.
- E) A writer.
- F) output format.

4. In the definition _  , what does the transformation _  imply?

- **A.** It applies a specific filter to modify the reader output.
- **B.** It invokes a function that modifies the document tree.
- **C.** It discards unneeded elements in the output.
- **D.** It transforms the reader's input data.

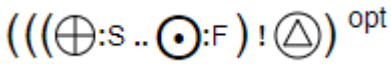
5. What does the combined definition _  __represent?

- **A)** A reader that successfully reads input, invokes a parser that fails to parse, and outputs an empty document tree.
- **B)** A reader that successfully reads input, invokes a parser that successfully parses and produces a document tree.
- **C)** A parser that generates a default document tree due to a failed reader.
- **D)** A writer generating output directly from the input file.

6. If a _  (absent document tree) is passed to the writer in _  ,

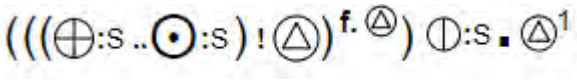
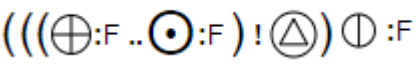
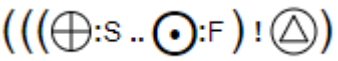
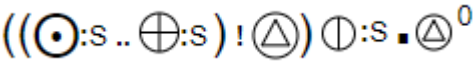
what is the outcome?

- **A)** The writer successfully produces output.
- **B)** The writer fails to produce output.
- **C)** The writer transforms the document tree.
- **D)** The parser re-generates the document tree.

7. What does the combination _  imply?

- **A)** A reader that fails to read input and a parser producing a default document tree.
- **B)** A reader successfully reading input but a parser that fails to parse, resulting in an absent document tree.
- **C)** A successful parser and reader generating an incomplete document tree.
- **D)** A reader generating output without a parser.

8. Which sequence best represents a full, successful input-to-output workflow using Level 2 definitions?

- **A)** _ 
- **B)** _ 
- **C)** _ 
- **D)** _ 

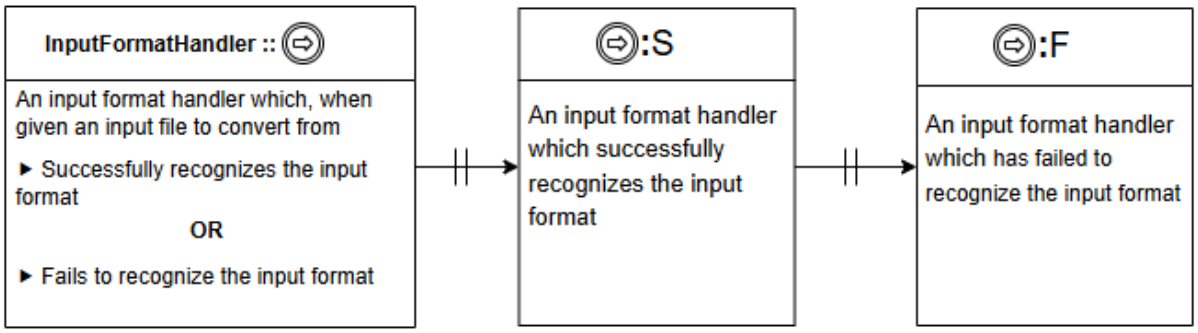
Case 3

Pandoc is a powerful document converter written in Haskell that allows users to transform documents between a variety of markup formats, including Markdown, HTML, LaTeX, and PDF.

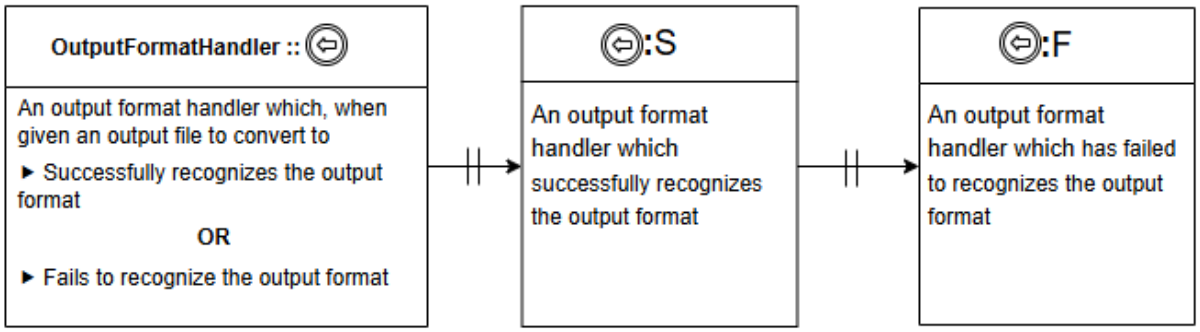
Pandoc utilizes **input parsers** to convert various markup formats, like Markdown and HTML, into an **abstract syntax tree (AST),** which serves as a structured representation of the document's content. This AST enables efficient manipulation during the conversion process. After processing, **output writers** transform the AST into the desired output formats, such as PDF or Word documents. Additionally, **filters** allow users to customize the AST dynamically, modifying the document's content and structure before generating the final output. This architecture makes Pandoc a flexible and powerful tool for document conversion.

To represent these components, we define the following main concepts in level 1:

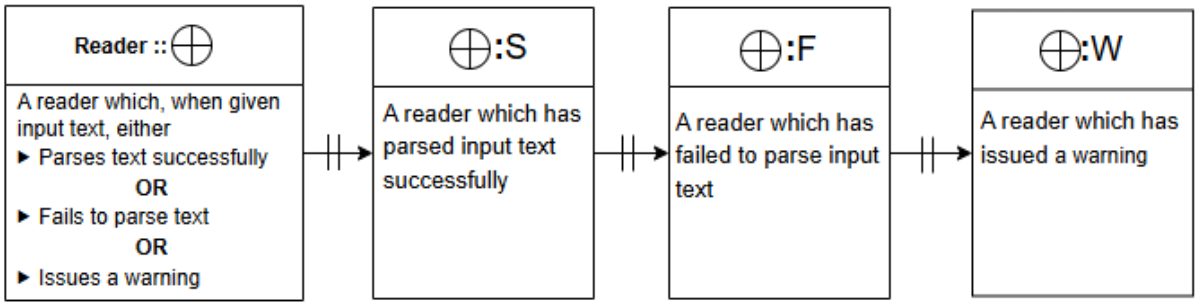
We define an input handler to help manage which parsers should parse text from an input file.



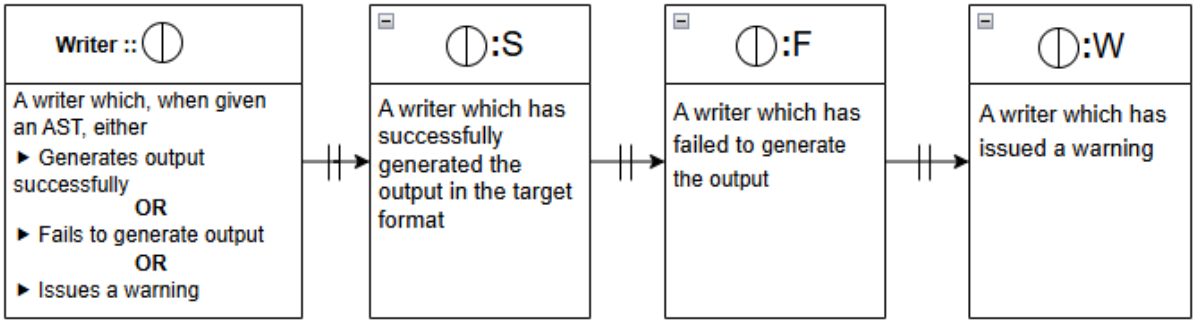
Similarly, we define an **OutputFormatHandler**



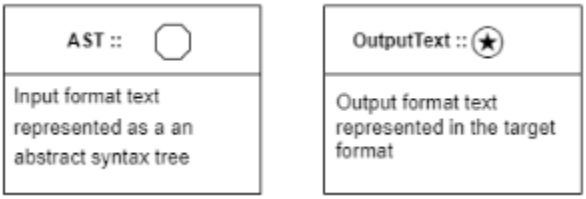
Another necessary main definition is for a **Reader**, which parses input text to generate an AST.



Similarly, we define a **Writer**

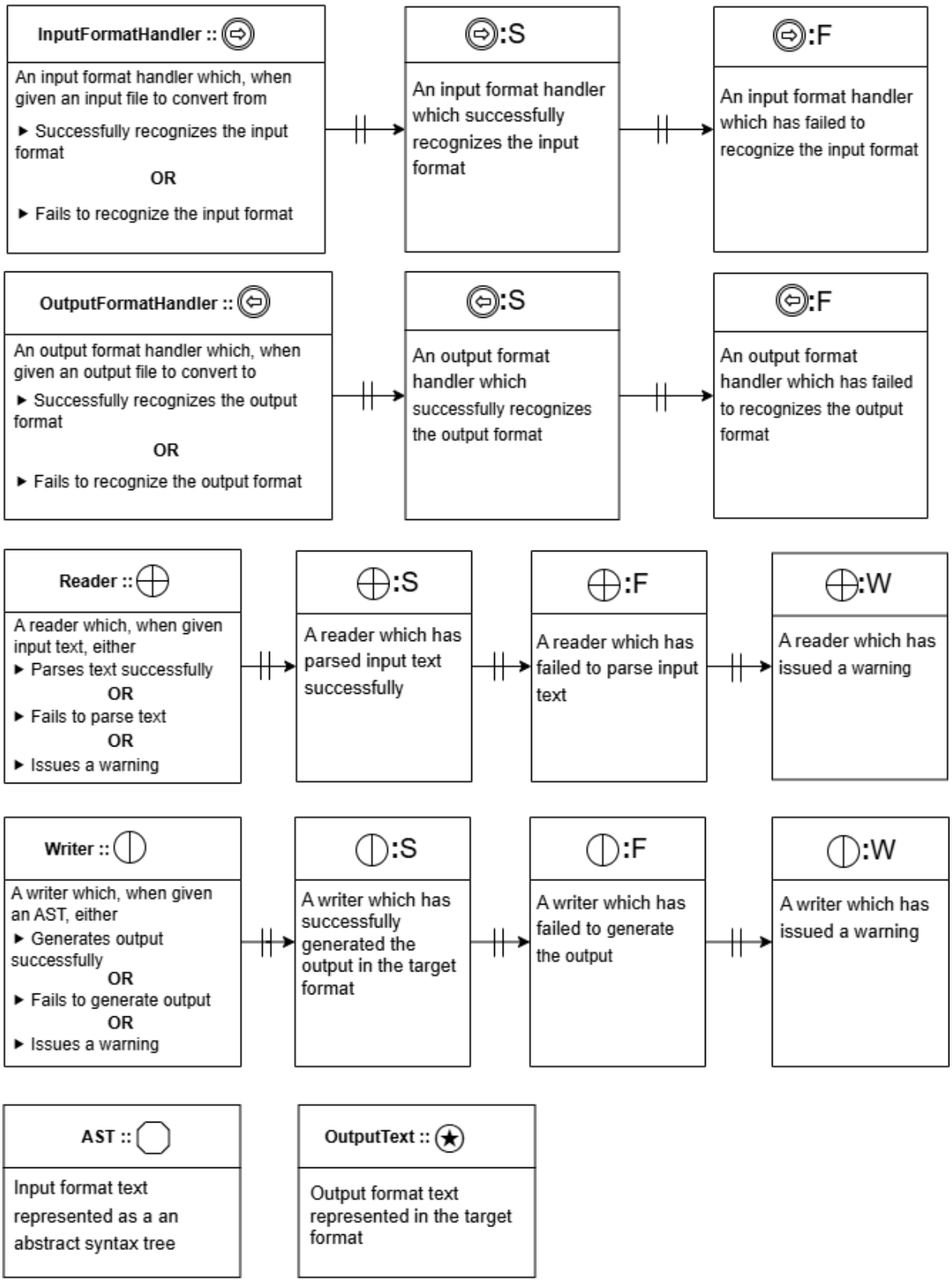


We also need definitions for an **AST** and **output text**:



The full set of level 1 definitions is:

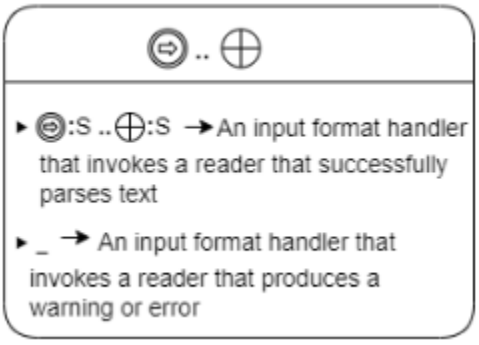
Level 1: Main definitions and sub-definitions



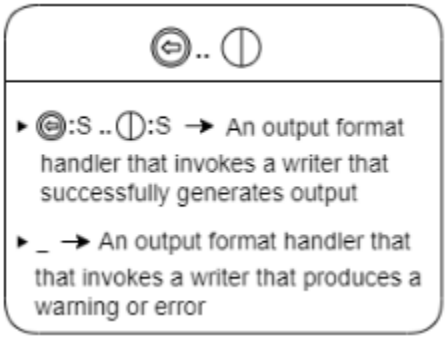
Moving on to level 2

Level 2 definitions are derived from manipulating Level 1 definitions.

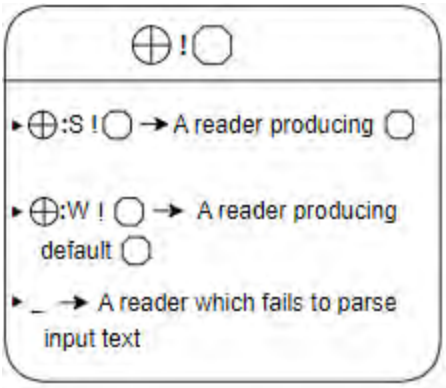
we might have a definition for invoking a reader for a given input format to parser the text.
Such a definition would look like



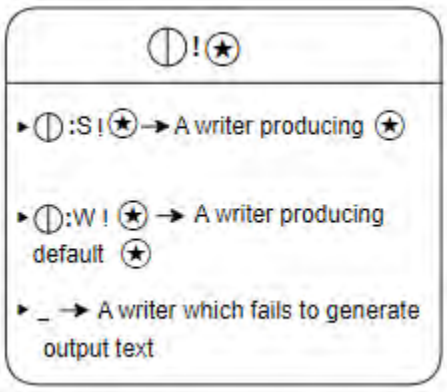
Similarly for invoking a writer



Focusing on readers, we define a process for producing an AST since readers output ASTs:



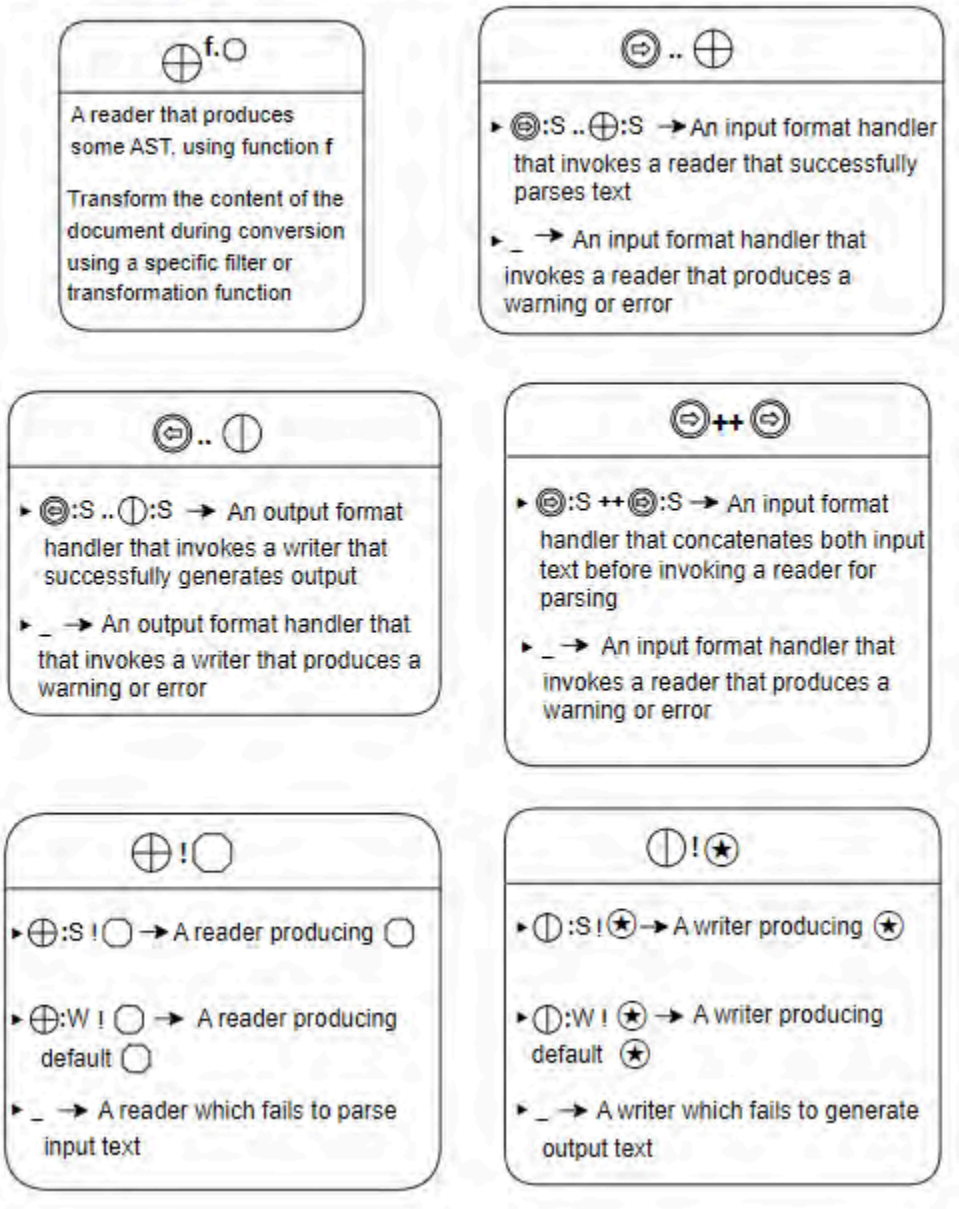
Similarly for a writer that produces output text



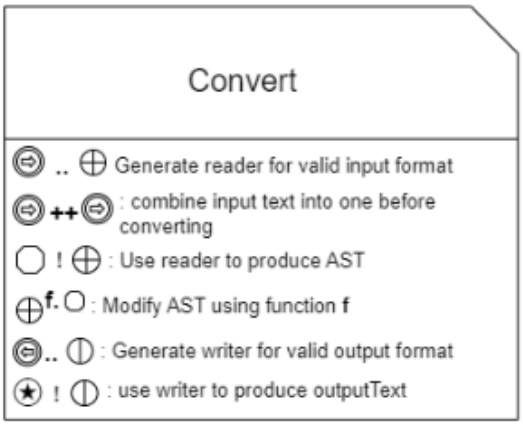
Pandoc also has filters that can transform the AST before converting it to an output format

The complete list of Level 2 definitions can be found below.

Level 2: Type Definition and Manipulation

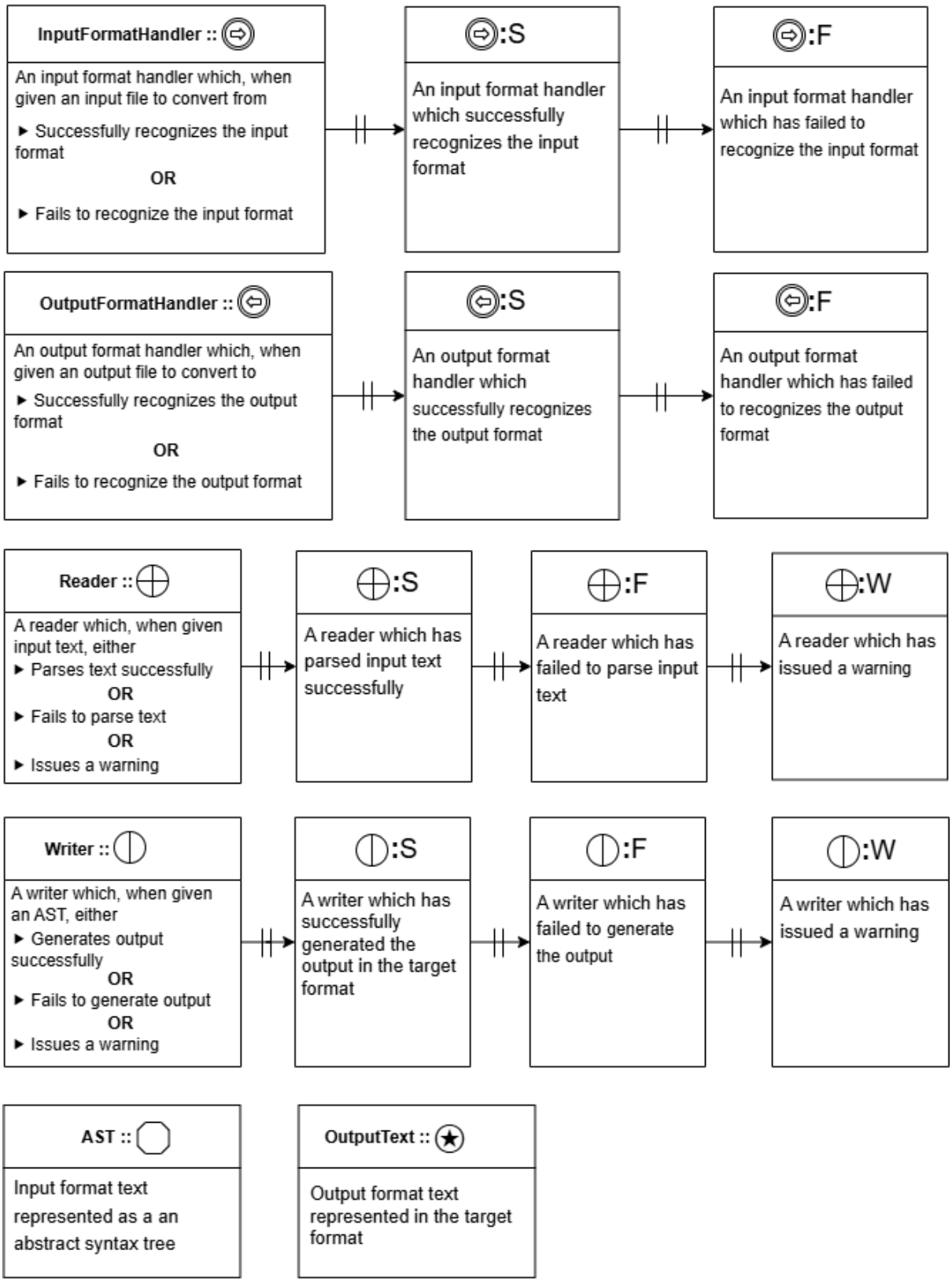


And finally, the level 3 definitions

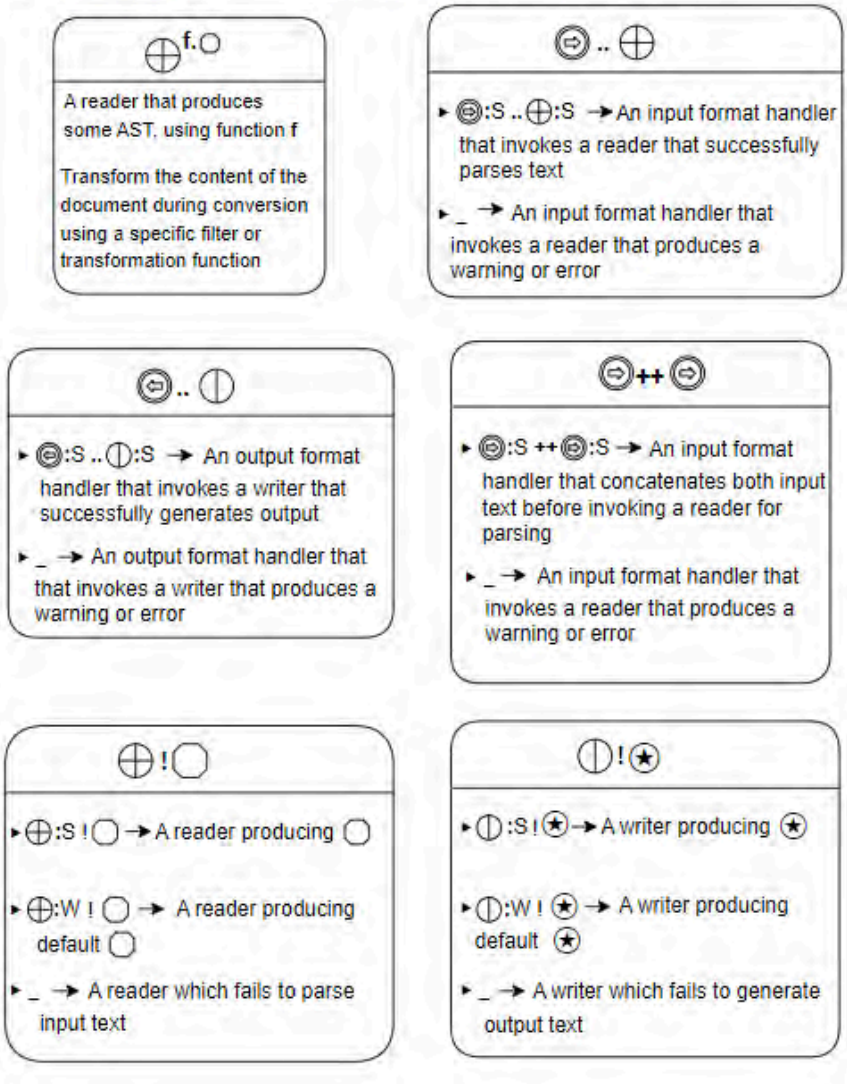


The complete diagram of Pandoc can be found below.

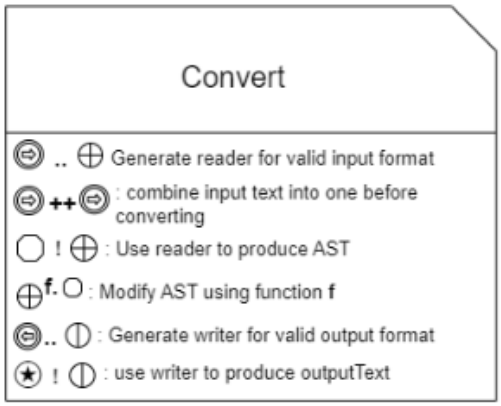
Level 1: Main definitions and sub-definitions



Level 2: Type Definition and Manipulation

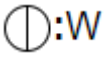
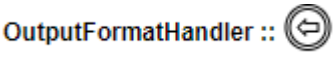
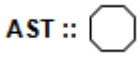
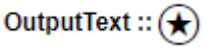
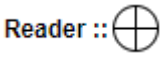
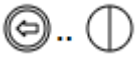
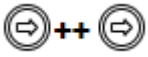
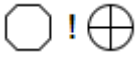
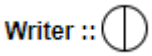
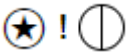


Level 3: Additional Information and Summary

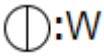
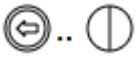
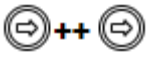
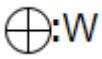
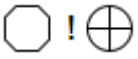


Questions

1. Identify all the main definitions in the Pandoc diagr

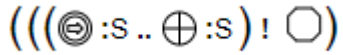
□ Option 1: 	□ Option 12: 
□ Option 2: 	□ Option 13: 
□ Option 3: 	□ Option 14: 
□ Option 4: 	□ Option 15: 
□ Option 5: 	□ Option 16: 
□ Option 6: 	□ Option 17: 
□ Option 7: 	□ Option 18: 
□ Option 8: 	□ Option 19: 
□ Option 9: 	□ Option 20: 
□ Option 10: 	□ Option 21: 
□ Option 11: 	□ Option 22: 

2. Identify all the sub-definitions in the Pandoc diagram

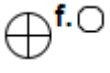
□ Option 1: InputFormatHandler :: 	□ Option 12: 
□ Option 2: 	□ Option 13: 
□ Option 3: 	□ Option 14: 
□ Option 4: OutputFormatHandler :: 	□ Option 15: AST :: 
□ Option 5: 	□ Option 16: OutputText :: 
□ Option 6: 	□ Option 17: 
□ Option 7: Reader :: 	□ Option 18: 
□ Option 8: 	□ Option 19: 
□ Option 9: 	□ Option 20: 
□ Option 10: 	□ Option 21: 
□ Option 11: Writer :: 	□ Option 22: 

3. **What does the**  **produce?**

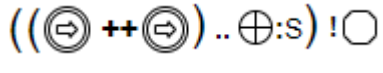
- **A)** An input format handler
- **B)** A reader
- **C)** An AST
- **D)** An output format handler
- **E)** A writer

4. What does the combined definition _  **_represent?**

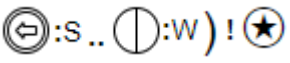
- **A)** An input format handler that successfully recognizes the format and invokes a reader that fails to parse text, producing a default AST.
- **B)** An input format handler that successfully recognizes the format and invokes a reader that successfully parses text, producing an AST.
- **C)** An input format handler that recognizes the format but fails to invoke a reader.
- **D)** A reader that successfully parses text without an input handler.

5. What does the notation _  **imply in the context of transformation?**

- **A)** The reader produces an AST without transformation.
- **B)** The reader uses a function to modify the AST before generating a default AST.
- **C)** The reader modifies the AST using a function, producing a new AST.
- **D)** The writer transforms the AST before output.

6. If _  **_is used, what does it indicate about the input processing?**

- **A)** A single input handler that discards one of the input formats.
- **B)** An input handler that combines inputs before the reader successfully parses and generates an AST.
- **C)** An input format handler fails while attempting to combine inputs.
- **D)** An input handler invokes a writer instead of a reader.

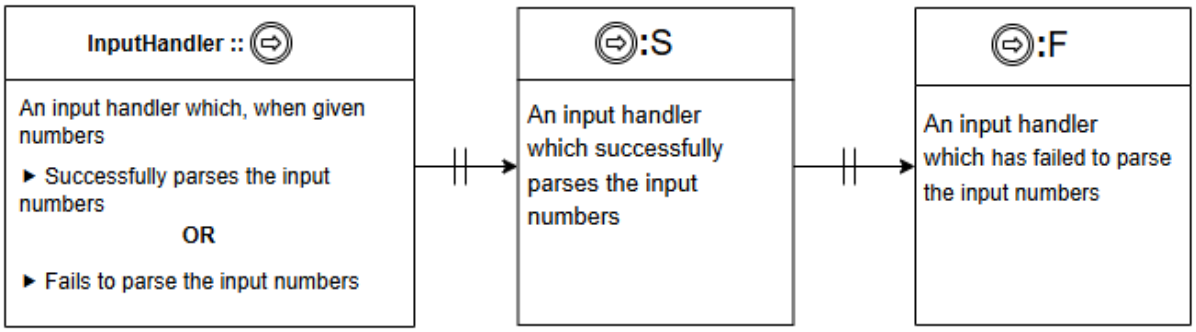
7. If _  **occurs, what is the likely outcome?**

- **A)** The writer produces an output with a warning, defaulting to a fallback output format.
- **B)** The writer fails to produce any output.
- **C)** The reader produces a default output text.
- **D)** The writer skips the transformation.

CASE 4

Symbol 1 (Evaluation of the OR-Arrow Symbol)

Consider the following diagram:



Does the ⫴→ clearly communicate that the main definition can represent multiple alternatives?

- o Yes
- o No

Is the use of the ⫴→ symbol effective in conveying the either-or relationship between subtypes?

- o Yes
- o No

Is there an ambiguity when interpreting the subtype relationships using the ⫴→ symbol?

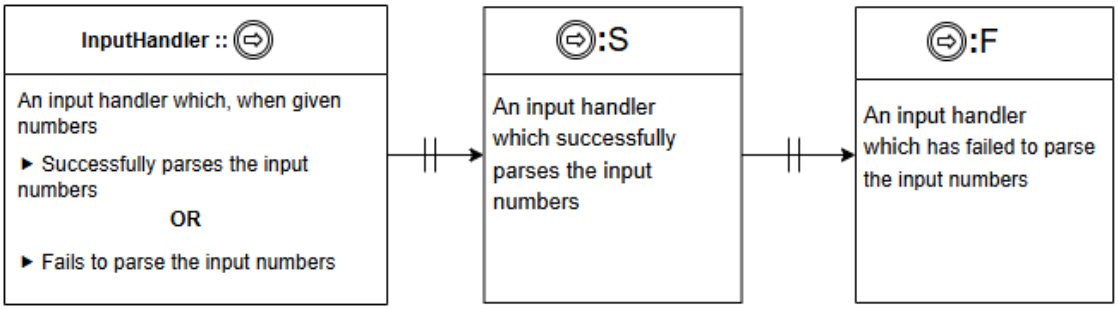
- o Yes
- o No

Please provide reasons for your choices above:

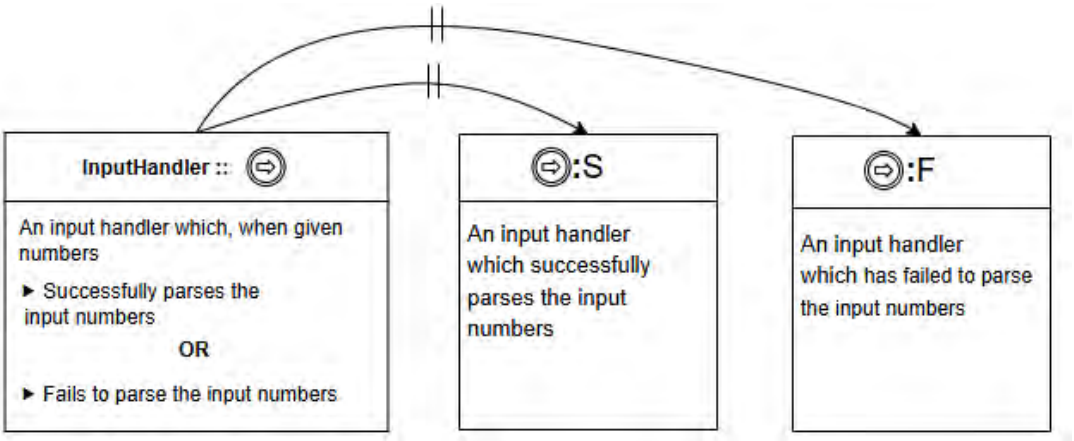
Symbol 2 (Comparative Analysis of the OR-Arrow Symbol)

Consider the following two diagrams which have the same meaning:

(a) Diagram A



(b) Diagram B



Which diagram makes it easier to identify the relationship between definitions?

- o (a)
- o (b)
- o Neither

How many sub-definitions are visible in the diagram A?

- o Zero
- o One
- o Two
- o Three

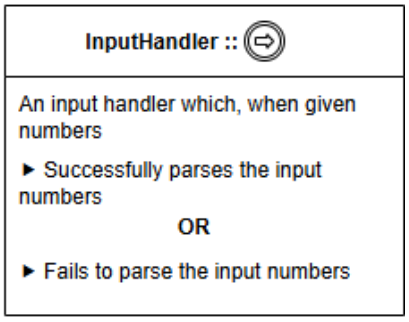
How many sub-definitions are visible in the diagram B?

- o Zero
- o One
- o Two
- o Three

Please provide reasons for your choices above:

Symbol 3 (Evaluation of the Notation Label (::) Symbol)

Consider the following notation:



Is the notation InputHandler :: clear in representing the relationship between the name of a definition and its symbol?

- o Yes
- o No

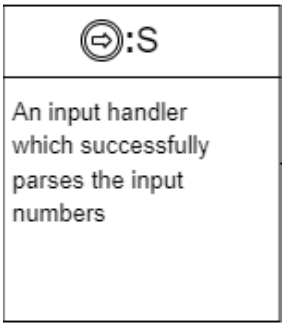
What does the symbol represent?

- o Going in
- o An Input Handler
- o A circle with an arrow
- o Moving right

Please provide the reasons for your choice:

Symbol 4 (Evaluation of Sub-Definition Notation)

Consider the following notation:



Does the :S notation clearly convey combining a symbol from the main definition with a letter indicates a sub-definition of that main definition?

- o Yes
- o No

Do you understand the single letter (e.g., S) to intuitively represent the sub-definitions functionality?

- o Yes
- o No

In the :S notation what does the S represent?

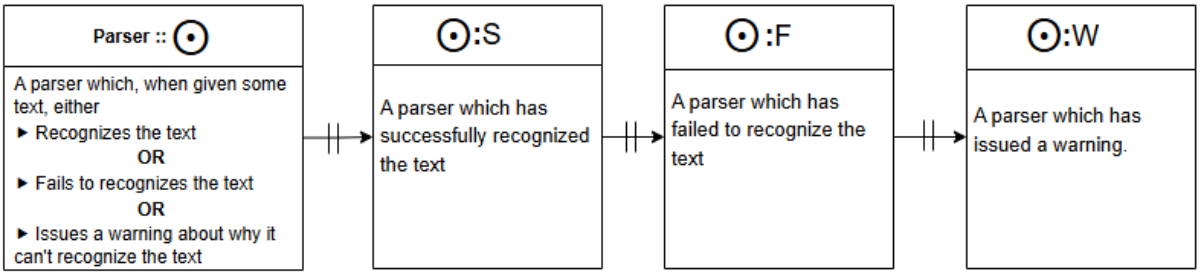
- o A new, unrelated type that has no connection to
- o A specific behavior or variant of the type such as a successful state.
- o The name of the symbol
- o A numerical value associated with

Please provide the reasons for your choices above.

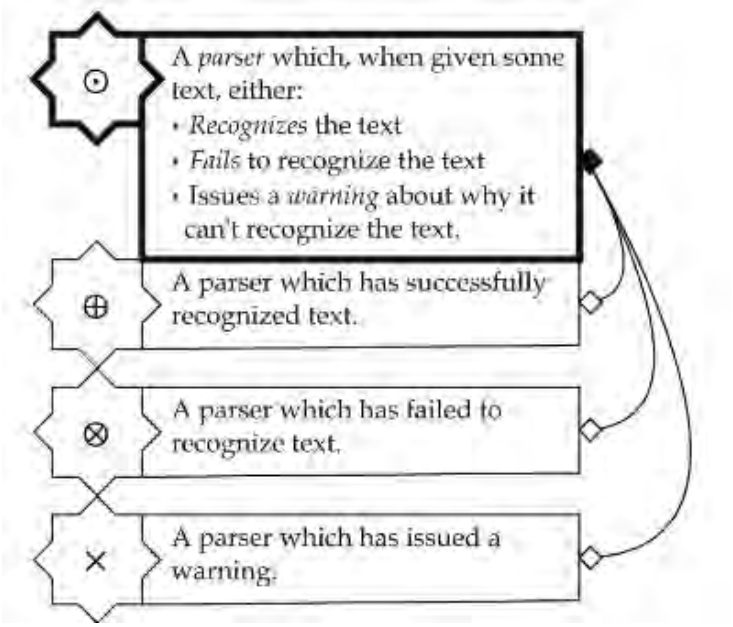
Symbol 5 (Comparative Evaluation of Diagrams for Main Definitions and Sub-Definitions)

Consider the following two diagrams which have the same meaning:

(a) Diagram A



(b) Diagram B



Which diagram makes it easier to identify the relationship between definitions?

- o (a)
- o (b)
- o Neither

Which diagram makes it easier to recall what the symbols represent?

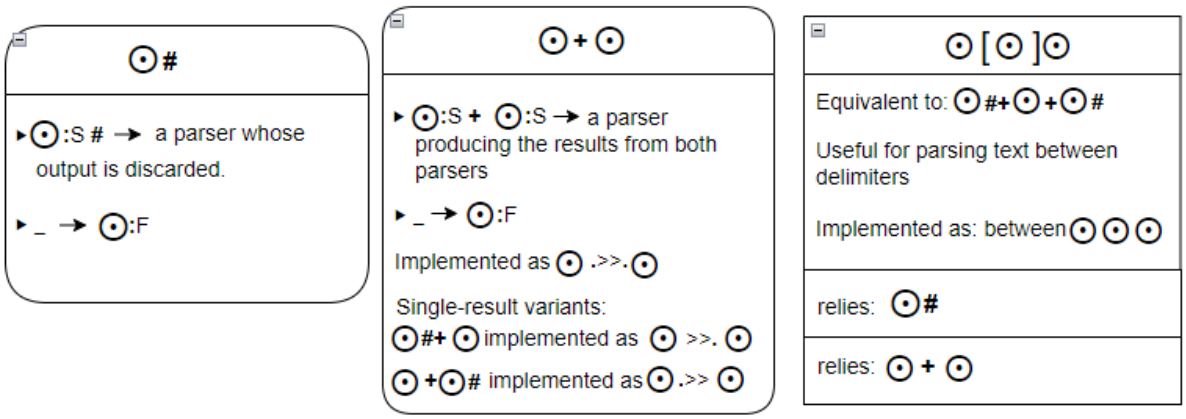
- o (a)
- o (b)
- o Neither

Please provide the reasons for your choices:

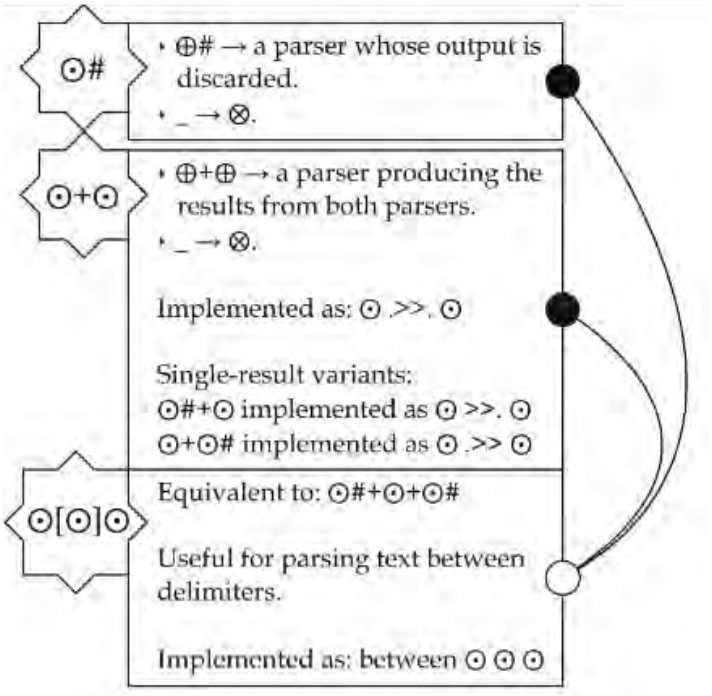
Symbol 6 (Evaluation of Relationships at Level 2)

Consider the following two diagrams:

(a) Diagram A



(b) Diagram B



Which notation conveys that there is a relationship between definitions?

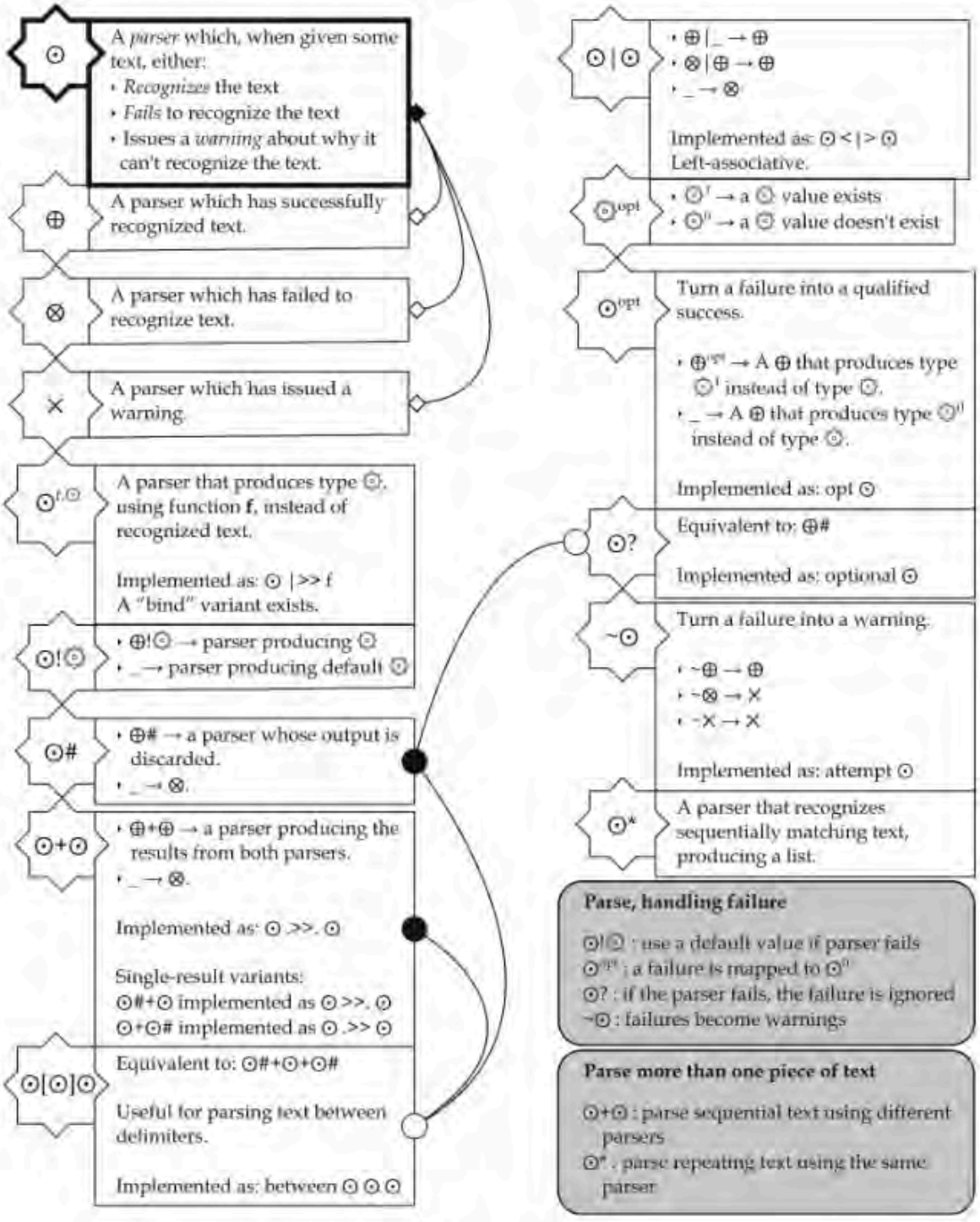
- (a)
- (b)
- Neither

Please provide the reason for your choice:

Symbol 7 (Holistic Comparison of Diagrammatic Notations)

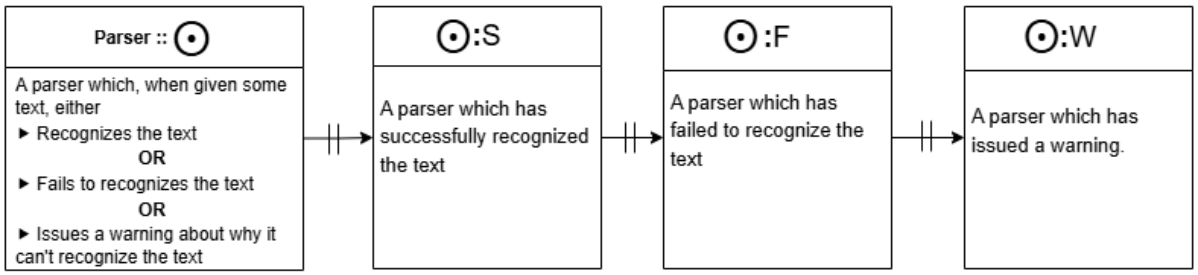
Consider these two diagrams that model the same system:

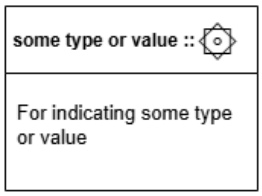
(a) Diagram A


A *parser* which, when given some text, either:
• *Recognizes* the text
• *Fails* to recognize the text
• Issues a *warning* about why it can't recognize the text.
A parser which has successfully recognized text.
A parser which has failed to recognize text.
A parser which has issued a warning.
A parser that produces type ⊙, using function **f**, instead of recognized text.
Implemented as: ⊙ |>> f
A "bind" variant exists.
• ⊕!⊙ → parser producing ⊙
• _ → parser producing default ⊙
• ⊕# → a parser whose output is discarded.
• _ → ⊗.
• ⊕+⊕ → a parser producing the results from both parsers.
• _ → ⊗.
Implemented as: ⊙ .>>. ⊙
Single-result variants:
⊙#+⊙ implemented as ⊙ >>. ⊙
⊙+⊙# implemented as ⊙ .>> ⊙
Equivalent to: ⊙#+⊙+⊙#
Useful for parsing text between delimiters.
Implemented as: between ⊙ ⊙ ⊙
• ⊕|_ → ⊕
• ⊗|⊕ → ⊕
• _ → ⊗
Implemented as: ⊙ <|> ⊙
Left-associative.
• ⊙ᵀ → a ⊙ value exists
• ⊙ᶠ → a ⊙ value doesn't exist
Turn a failure into a qualified success.
• ⊕ᵂᵀ → A ⊕ that produces type ⊙ᵀ instead of type ⊙.
• _ → A ⊕ that produces type ⊙ᶠ instead of type ⊙.
Implemented as: opt ⊙
Equivalent to: ⊕#
Implemented as: optional ⊙
Turn a failure into a warning.
• ~⊕ → ⊕
• ~⊗ → ✕
• ~✕ → ✕
Implemented as: attempt ⊙
A parser that recognizes sequentially matching text, producing a list.
Parse, handling failure
⊙!⊙ : use a default value if parser fails
⊙ᵂᵀ : a failure is mapped to ⊙ᶠ
⊙? : if the parser fails, the failure is ignored
~⊙ : failures become warnings
Parse more than one piece of text
⊙+⊙ : parse sequential text using different parsers
⊙* : parse repeating text using the same parser

(b) Diagram B

Level 1: Main definitions and sub-definitions





Level 2: Type Definition and Manipulation

⊙ f.⚙

A parser that produces type ⚙ using function **f**, instead of recognized text

Implemented as: ⊙ | >> f

A "bind' variant exists

⊙ !⚙

▸⊙:S !⚙ → parser producing ⚙

▸ _ → parser producing default ⚙

⊙#

▸⊙:S # → a parser whose output is discarded.

▸ _ → ⊙:F

⊙ + ⊙

▸ ⊙:S + ⊙:S → a parser producing the results from both parsers

▸ _ → ⊙:F

Implemented as ⊙ .>>. ⊙

Single-result variants:

⊙#+ ⊙ implemented as ⊙ >>. ⊙

⊙ +⊙# implemented as ⊙ .>> ⊙

⊙ [⊙]⊙

Equivalent to: ⊙#+⊙+⊙#

Useful for parsing text between delimiters

Implemented as: between ⊙ ⊙ ⊙

relies: ⊙#

relies: ⊙ + ⊙

⊙|⊙

▸⊙:S| _ →⊙:S

▸⊙:F| ⊙:S → ⊙:S

▸ _ → ⊙:F

Implemented as: ⊙ <| > ⊙

Left-associative.

⚙ opt

▸⚙^{1} → a ⚙ value exists

▸⚙^{0} → a ⚙ value doesn't exist

⊙ opt

Turn a failure into a qualified success

▸⊙:S^{opt} → A ⊙:S that produces type ⚙^{1} instead of type ⚙

▸ _ → A ⊙:S that produces type ⚙^{0} instead of type ⚙

Implemented as: opt ⊙

⊙?

Equivalent to ⊙:S#

Implemented as: optional ⊙

relies: ⊙#

~⊙

Turn a failure into a warning

▸~⊙:S → ⊙:S

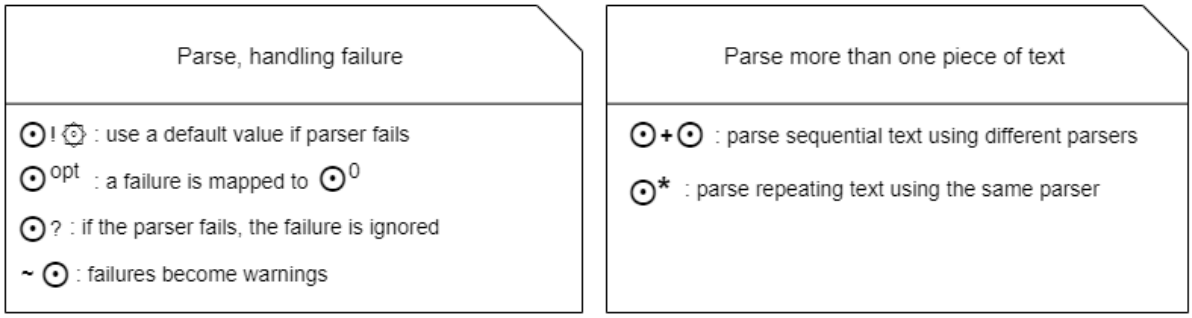
▸~⊙:F → ⊙:W

▸~⊙:W → ⊙:W

⊙*

A parser that recognizes sequentially matching text, producing a list

Level 3: Additional Information and Summary



Which diagram makes it easier to keep track of the symbols?

- (a)
- (b)
- Neither

Please provide a reason for your choice:

Which diagram makes it easy to mentally map the symbol to its underlying concept?

- (a)
- (b)
- Neither

Please provide a reason for your choice:

Which notation makes it easier to understand the relationships between components?

- (a)
- (b)
- Neither

Please provide a reason for your choice:

Which notation organizes the components and relationships more clearly in terms of layout and flow?

- (a)
- (b)
- Neither

Please provide a reason for your choice:

Which notation is more flexible in adapting to changes in the system's structure or requirements?

- (a)
- (b)
- Neither

Please provide a reason for your choice: