A RULE-BASED APPROACH FOR RECOGNITION OF CHEMICAL STRUCTURE DIAGRAMS

by

NOUREDDIN SADAWI

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
May 2013

Abstract

In chemical literature much information is given in the form of diagrams depicting chemical structures. In order to access this information electronically, diagrams have to be recognised and translated into a processable format. Although a number of approaches have been proposed for the recognition of molecule diagrams in the literature, they traditionally employ procedural methods with limited flexibility and extensibility.

This thesis presents a novel approach that models the principal recognition steps for molecule diagrams in a strictly rule based system. We develop a framework that enables the definition of a set of rules for the recognition of different bond types and arrangements as well as for resolving possible ambiguities. This allows us to view the diagram recognition problem as a process of rewriting an initial set of geometric artifacts into a graph representation of a chemical diagram without the need to adhere to a rigid procedure. We demonstrate the flexibility of the approach by extending it to capture new bond types and compositions. In experimental evaluation we can show that an implementation of our approach outperforms the currently available leading open source system. Finally, we discuss how our framework could be applied to other automatic diagram recognition tasks.

ACKNOWLEDGEMENTS

I would like to express my heartfelt thanks to my supervisor; Volker Sorge, for his support, encouragement and guidance. Volker has stood by me during the difficult times after my sponsorship was suspended, and has shown faith in me when I doubted my own ability. His direction has helped me learn to evaluate and express my work clearly - I shall cherish my work with him for many years to come.

I would also like to thank the members of my *research monitoring group*; Iain Styles and Peter Hancox, for their helpful suggestions and academic support. In addition, after having the opportunity to collaborate with Alan Sexton, I would like to thank him for his thoughtful advice.

The unlimited support from my parents; Mahdi and Muna, makes me feel deeply indebted to them - for without their assistance, I would not have been able to reach this stage. I would like to say that I will always be grateful to them.

Finally, there are no words to show my beloved wife, Reem, how much I appreciate her help, patience and support.

CONTENTS

1 Introduction **1**

1.1 Hypotheses 3

1.2 Overview of the Approach 4

1.2.1 Vectorisation 4

1.2.2 Rule System 5

1.3 Contributions 5

1.4 Publications 6

1.5 Thesis Overview 7

1.6 Summary 8

I Background **9**

2 Overview of Chemical Structure Diagrams **10**

2.1 Molecule Representation/Drawing 12

2.1.1 Molecular formulae 12

2.1.2 Structural formulae 13

2.1.2.1 Displayed formulae 13

2.1.2.2 Skeletal formulae 14

2.1.2.3 Representing Aromatic Rings 15

2.1.3 Two Dimensional Structural Formulae for Three Dimensional Representations 16

2.1.4 Markush Structures 19

2.2 More on Diagram Notation . . . 21
2.2.1 Double and Triple Bonds . . . 21
2.2.2 Implicit Atoms . . . 21
2.2.3 Bridge Bonds . . . 22
2.2.4 Bond Directions . . . 23
2.2.5 Superatoms . . . 23
2.3 Summary . . . 25

3 Electronic Representation of Chemical Structure Diagrams 26
3.1 Electronic Representation of Molecular Structures . . . 28
3.1.1 Chemical Graph . . . 28
3.1.2 Adjacency Matrix Representation . . . 29
3.1.3 Linear Representations . . . 31
3.1.3.1 The Molecular Formula . . . 31
3.1.3.2 The SMILES String . . . 32
3.1.3.3 IUPAC Nomenclature . . . 34
3.1.4 Chemical Markup Language . . . 34
3.1.5 Connection Tables (MOL files) . . . 35
3.1.5.1 Detailed Structure of MOL file . . . 36
3.1.5.2 Stereochemistry in MOL files . . . 37
3.1.5.3 Superatoms in MOL files . . . 38
3.2 Summary . . . 39

4 Existing Approaches to Chemical Diagram Recognition 40
4.1 The tool developed by IBM's Almaden Research Centre . . . 41
4.2 CLiDE . . . 43
4.3 ChemOCR . . . 45
4.4 OSRA . . . 47
4.5 ChemReader . . . 49

4.6 ChemInfty . . . 51
4.7 Imago . . . 51
4.8 AsteriX Server . . . 52
4.9 Chemink . . . 54
4.10 Summary . . . 56

II Rule-based Recognition System 57

5 Specifications 58

5.1 The Overall Philosophy . . . 59
5.2 System Architecture . . . 60
5.3 Implementation Methodology . . . 63
5.4 Why Rule-based . . . 65
5.5 Testing and Evaluation Methodology . . . 66
5.6 Summary . . . 66

6 Vectorisation 68

6.1 Image Thinning . . . 69
6.1.1 Types of Thinning Algorithms . . . 72
6.1.1.1 Sequential Thinning . . . 72
6.1.1.2 Parallel Thinning . . . 72
6.1.2 The Thinning Algorithm we have used . . . 73
6.2 Extraction of Precise Line Endpoints . . . 74
6.2.1 How Douglas-Peucker Algorithm works . . . 75
6.2.2 Barb (Spike) Removal . . . 77
6.3 Detecting Solid Triangles and Bold Lines . . . 79
6.4 Detecting Arrows . . . 83
6.5 Summary . . . 84

7 A Description of The Rules **85**

7.1 More on our Implementation of Rules . . . 86

7.2 Preliminaries . . . 88

7.2.1 Geometrical Primitives . . . 88

7.2.2 Concepts . . . 90

7.2.3 Parameters . . . 92

7.3 Bond Cutting . . . 94

7.4 Summary of Rules . . . 95

7.5 Sample Rules . . . 97

7.5.1 R1: Planar Single Bond . . . 97

7.5.2 R3: Planar Triple Bond . . . 98

7.5.3 R7: Dashed Bond . . . 101

7.5.4 R12: Wavy Bond . . . 103

7.6 A Fully Worked Example . . . 105

7.7 Summary . . . 109

8 Graph Formation and Output Generation **110**

8.1 Formation of Graph . . . 111

8.2 Heuristics for Graph Formation . . . 113

8.3 Expanding Superatoms . . . 114

8.4 Guessing the Stereocentre . . . 116

8.5 Outputting MOL File . . . 117

8.6 Steps of Procedural Implementation . . . 118

8.7 Summary . . . 122

9 Extensibility and Flexibility of the Approach **123**

9.1 Discussion of Rules . . . 124

9.1.1 Rule Conflicts . . . 124

9.1.2 System Termination . . . 125

9.1.3 Circularity . . . 126
9.2 More Bond Types . . . 126
9.2.1 R19: Aromatic Bond . . . 126
9.2.2 R20: Tautomeric Bond . . . 128
9.2.3 R21: Double Bond with Type 1 Stereochemistry . . . 131
9.2.4 R22: Double Bond with Type 2 Stereochemistry . . . 132
9.3 Summary . . . 134

III Evaluation **135**

10 Evaluation **136**
10.1 Repeatability on Selection of Images . . . 137
10.2 Comparison with Existing Systems . . . 141
10.2.1 Experiments using the OSRA Dataset . . . 141
10.2.2 Experiments using our Dataset . . . 142
10.2.3 Experiments using TREC11's Datasets . . . 143
10.2.4 Experiments using CLEF12's Dataset . . . 144
10.3 Analysis of Results . . . 146
10.3.1 Recognition Errors due to Rule Mismatch . . . 146
10.3.2 Recognition Errors due to Vectorisation . . . 149
10.4 Summary . . . 151

11 Conclusions & Future Work **152**
11.1 Conclusions . . . 153
11.2 Future Work . . . 154
11.2.1 Improve OCR . . . 154
11.2.2 Better Bond Segmentation . . . 154
11.2.3 Handle Markush Structures . . . 154
11.2.4 Superatoms and Stereocentre . . . 155

11.2.5 Examine Flexibility of our Approach . . . 155
11.2.5.1 Electronic Circuit Recognition . . . 155
11.2.5.2 Flowchart Recognition . . . 156

List of References **158**

Appendices **169**

A Optical Character Recognition (OCR) **170**
A.1 Optical Character Recognition . . . 171
A.2 Connected Component Labelling . . . 171
A.2.1 Calculating Connected Components . . . 173
A.2.2 Bounding Boxes . . . 174
A.3 Performing OCR and Removal of Characters . . . 175
A.3.1 Feature Extraction . . . 176
A.3.2 Classification . . . 178
A.3.3 Removal of Symbols . . . 180
A.4 Formation of Atoms (Chargroups) . . . 180
A.4.1 Alignment of Characters . . . 181
A.4.2 Consecutivity of Characters . . . 182
A.4.3 Inter-character Distance . . . 183
A.4.4 Forming the Atoms . . . 183
A.5 Identifying Rings/Circles . . . 183
A.6 Summary . . . 184

B The Rules System **185**
B.1 R2: Double Planar Bond . . . 186
B.2 R4: Dashed Bold Bond vs Triple Bond . . . 187
B.3 R5: Dashed Wedge vs Triple Bond . . . 188
B.4 R6: Dashed Wedge vs Double Bond . . . 188

B.5 R8: Dashed Bold Bond 189
B.6 R9: Dashed Wedge Bond 191
B.7 R10: Hollow Wedge Bond - case 1 192
B.8 R11: Hollow Wedge Bond - case 2 193
B.9 R13: Arrow Bond 194
B.10 R14: Solid Wedge Bond 195
B.11 R15: Bold Bond 195
B.12 R16: Aromatic Ring 196
B.13 R17: Open Bridge Bond 197
B.14 R18: Closed Bridge Bond 199
B.15 Summary 200

C The Douglas-Peucker Algorithm **201**

LIST OF FIGURES

2.1 Two different molecules with the same formula 13

2.2 Methane and its displayed formula (image from [22]) 14

2.3 Ethanoic Acid (image from [22]) 14

2.4 Cyclohexane - C_6H_{12} 15

2.5 Benzene C_6H_6 - The three structures are equivalent 16

2.6 Significance of Stereochemistry 17

2.7 Common Types of Bond Representation on Paper 18

2.8 Markush Structure 20

2.9 Planar Double and Triple Bonds 21

2.10 Embedded Node on Shadowed Side(s) of Planar Double and Triple Bonds . . . 22

2.11 Open and Closed Bridge Bonds (shadowed) 22

2.12 Example Superatoms (Highlighted) 24

3.1 Example Chemical Structure Diagram 29

3.2 The Corresponding Chemical Graph of the Structure in Figure 3.1 30

3.3 A Chemical Structure and its Adjacency Matrix Representation 31

3.4 SMILES Example 33

3.5 An Example Structure with its CML file 35

3.6 A Basic Connection Table for a Simple Molecular Structure 36

3.7 The Corresponding MOL file of the structure in Figure 3.5(a) 37

3.8 A Superatom with its equivalent Substructure 38

5.1 Components of a Rule-based System 60

5.2 Components of Our System 61

6.1 Vectorisation Steps 69
6.2 Image Thinning 70
6.3 Example Thinning Kernels 71
6.4 Example 8-Neighbourhood 74
6.5 Thinning using Guo and Hall's Thinning Algorithm 74
6.6 Foreground Pixel Tracing 75
6.7 Example connected pixels and their neighbours 76
6.8 How Douglas Peucker Algorithm works (image from [109]) 77
6.9 Spike resulting after thinning of rough line (shadowed) 78
6.10 A Simple Blurring Kernel 79
6.11 Detection of Solid Triangles and Thick Lines 80
6.12 Sequence Behaviour 82
6.13 Example Structure with Dative Bonds 83

7.1 Line Segment with its measurable attributes 88
7.2 An Arrow Primitive 89
7.3 A Triangle Primitive 89
7.4 A Circle Primitive 90
7.5 A Chargroup Primitive 90
7.6 Radius of collinearity, Dash length and Dash separation 92
7.7 Bold dash length and Bold dash width 93
7.8 Bold bond width and Wedge base 93
7.9 Bond separation 93
7.10 Parallel line overlap 94
7.11 Single, Double and Triple Planar Bonds 94
7.12 Implicit Nodes (shadowed) 94
7.13 Illustration of Cut Lines 95

7.14 A Single Planar bond . . . 97
7.15 A Triple Planar bond . . . 99
7.16 Cutting produces two double bonds rather than a triple bond . . . 100
7.17 A Dashed Bond . . . 101
7.18 A Wavy Bond . . . 103
7.19 Dashes in a Wavy Bond . . . 104
7.20 A Molecule Diagram with its Corresponding Set of Primitives . . . 106
7.21 A Fully Worked Example of how our Recognition System Works . . . 108

8.1 Graph Representation of a Chemical Structure . . . 112
8.2 Closed and Open Node Examples . . . 113
8.3 Line Close to Atom but not pointing towards it . . . 114
8.4 Example Superatoms (Highlighted) . . . 114
8.5 Superatom Dictionary . . . 115
8.6 Bonds with Unknown Stereochemistry . . . 116
8.7 Which atom is the stereocentre? . . . 116
8.8 Determining the Stereocentre . . . 117
8.9 An Example Structure with its MOL file . . . 118
8.10 Recognition steps for the molecule C_9H_9NO. . . . 119

9.1 Aromatic Bond . . . 126
9.2 Tautomeric Bond . . . 128
9.3 Double Bond with Type 1 Stereochemistry . . . 131
9.4 Double Bond with Type 2 Stereochemistry . . . 133

10.1 An Example image from OSRA Dataset with its Rule Execution Sequence . . . 138
10.2 Another Example image from OSRA Dataset with its Rule Execution Sequence 139
10.3 An Example image from Maybridge Dataset with its Rule Execution Sequence 139
10.4 Another Example image from Maybridge Dataset with its Rule Execution Sequence . . . 139

10.5 An Example image from TREC11 Dataset with its Rule Execution Sequence . 140
10.6 Another Example image from TREC11 Dataset with its Rule Execution Sequence 140
10.7 An Example image from CLEF12 Dataset with its Rule Execution Sequence . . 141
10.8 Another Example image from CLEF12 Dataset with its Rule Execution Sequence 141
10.9 Number of Structures Identified in each run at TREC11 144
10.10 An ambiguous Open-bridge Bond (shadowed) 147
10.11 An ambiguous Closed-bridge Bond 147
10.12 R12 Error 148
10.13 R20 Error 148
10.14 Unhandled Bond Type 149
10.15 Various Touching Components 149
10.16 Example Broken Characters 150
10.17 Ambiguous Connectivity of Atoms and Bonds 150
10.18 Superatoms with 2 or more connections to the main structure 151

11.1 Example Electronic Circuit 156
11.2 Example Flowchart 156

A.1 Components of an OCR Engine 172
A.2 Two Types of Pixel Neighbourhoods 173
A.3 CCL using Run Length Encoding 174
A.4 Coordinates of a Character's Bounding Box 175
A.5 Zoning of a character image. 177
A.6 The Removal of Symbols from a Molecular Structure Image 180
A.7 Coordinates and Extreme Points of a Character 181
A.8 Circle used to represent aromatic ring in aspirin 184

B.1 A Double Planar bond 186
B.2 Triple Bond or Dashed Bold Bond? 187
B.3 A Dashed Wedge Formed of only Two Dashes (shadowed) 189

B.4 A Dashed Bold Bond 190
B.5 Dashes in a Dashed Bold Bond 190
B.6 A Dashed Wedge Bond 191
B.7 Dashes in a Dashed Wedge Bond 191
B.8 A Hollow Wedge Bond - Case 1 192
B.9 A Hollow Wedge Bond - Case 2 194
B.10 A Dative Bond 194
B.11 A Solid Wedge Bond 195
B.12 A Bold Bond 196
B.13 Aromatic Ring (these structures are equivalent) 196
B.14 Open Bridge Bond (Bridge is shadowed) 198
B.15 Closed Bridge Bond (Bridge is shadowed) 199

LIST OF TABLES

2.1 Example Elements from the Periodic Table 16

6.1 Example Radius Values 81

7.1 Textual Summary of our 18 Rules 96

7.2 Possible Rule Execution Sequences with their Occurrence Frequency 107

10.1 Results of Running our Tool several times on Selection of Images 138

10.2 Procedural MolRec vs. OSRA - OSRA dataset 142

10.3 Random MolRec vs. OSRA - OSRA dataset 142

10.4 Procedural MolRec vs. OSRA - our dataset 143

10.5 Random MolRec vs. OSRA - our dataset 143

10.6 List of Participants at TREC11's I2S Task 144

10.7 Four Runs on the Manual Evaluation Set (95 images) 146

10.8 Four Runs on the Automatic Evaluation Set (865 images) 146

10.9 Chemical diagram recognition results at CLEF12 146

A.1 An Example Dataset 179

CHAPTER 1

INTRODUCTION

Over the years, the use of diagrams and sketches of chemical structures to capture and communicate ideas has become an integral part of chemists' activities. This is because these sketches can be more informative than, for instance, using a textual representation since they not only describe the internal structure of a certain compound, they also convey other information such as potential properties and intricate architectures. Hence, *a picture is worth a thousand words*.

The opposite process, which is retrieving precise information from existing chemical structure diagrams and storing it in a computer searchable format, remains a challenging task. The difficulty stems from the various ways chemists depict different formations and properties, and from image analysis processes such as the correct identification of shapes.

The current state of the art in interpreting chemical structure diagrams relies heavily on procedural approaches. In other words, the recognition process is based on a rigid list of steps that the approach goes through in order to recognise chemical structure diagrams and identify their bonds. This affects the extensibility, flexibility, and running time of the method as it becomes difficult to maintain, extend and results in repeating the same steps every time it is run.

In this thesis, we present and evaluate a technique for the recognition of chemical structure diagrams from bitmapped images. The technique is based on ideas derived from human expertise and is developed and evaluated using common chemical structure recognition evaluation methods on a variety of benchmark sets.

The technique has several advantages over existing approaches. In particular it deals with many bond drawing styles, it is easy to extend, easy to maintain and fast to execute. All these features stem from the rules at the core of the system. Since all patterns can be captured in an unambiguous list of rules, with each rule composed of a list of conditions or predicates, the whole system becomes easier to modify to handle patterns from other domains and becomes easier to extend in case new patterns are to be identified. It also provides ease of maintenance because one can easily identify which patterns are being mis-recognised.

Our rules strictly define various bond formations and architectures in a mutually exclusive

manner. By mutual exclusivity, we mean that the rules are well defined so that the order of their execution is minimised. This is realised by capturing different bonds using a sequence of strict conditions to avoid ambiguity and improves the performance of chemical diagram recognition. As we demonstrate throughout this thesis, the technique in general outperforms state of the art methods and produces results of higher accuracy when comparing results.

1.1 Hypotheses

In this thesis we address the following four hypotheses:

- A strictly rule-based approach where diverse bond drawing styles are captured using a list of strict rules can be used to accurately and efficiently recognise chemical structure diagrams.
- The rule framework can be flexible to allow for easy extension to incorporate additional diagram elements.
- The approach can be efficiently implemented in a system that can match the performance of other contemporary approaches in chemical diagram recognition.
- The extraction of geometrical primitives from chemical diagrams can be improved using novel approaches to the detection of line endpoints and geometrical objects in the shape of solid triangles which are used to depict one of the bond types.

We create a rule-based recognition system to evaluate the above hypotheses. The system will be composed of 18 rules defining different bond patterns that exist in chemical structure diagrams. These 18 rules were derived after analysing some existing benchmark datasets and catalogues of chemical structure diagrams. The system will take in as input a set of geometric objects of five possible types called primitives and will generate a suitable output format. We identify these primitives by using an existing line simplification algorithm in a novel way. A primitive can be a line segment, a triangle, an arrow, a circle or a character group. We also demonstrate the extensibility of the system as we modify these rules when

necessary to accommodate some new bond types making the number of rules 22. We also provide implementation and evaluation methodologies.

1.2 Overview of the Approach

Our rule-based approach for the recognition of chemical structure diagrams consists of two modules, a vectoriser and a rule-engine. In the vectoriser, we preprocess an input image, analyse the chemical structure diagram it represents and generate a set of geometric primitives. Each of these primitives can be one of five types (i.e. a character group, a circle, a line segment, a solid triangle or an arrow). These primitives are then picked up by the rule engine which rewrites them into a graph structure representation of the recognised molecule. In a post-processing step, we translate this graph structure into a variety of output formats such as MOL files [98] or SMILES [61, 107].

1.2.1 Vectorisation

The vectorisation works in three steps:

Image binarisation: For the first step of image binarisation we use Otsu's method [73]. This is followed by labelling of connected components.

Optical character recognition (OCR): In the second step we perform optical character recognition by extracting a set of structural features from connected components and applying a nearest neighbour classification based on a Euclidean metric. We remove all connected components recognised as characters from the image. We also detect circles at this step since their appearance is similar to that of an oxygen atom label, i.e. "O". We remove all detected characters and circles and the result of this step is a set of primitives each of which can be either a character group or a circle.

Separation of bond elements: At this point we have a new copy of the (character free) diagram. We apply a thinning algorithm to connected components to thin them to a single pixel width. We then build a polyline representation of the thinned lines. Here we employ a novel adaptation of the Douglas-Peucker line simplification algorithm to determine line end

points. At the same time we determine the average line width and average line length. We use a novel expanding disc technique to identify solid triangles and template matching to identify arrow lines. This results in having a set of primitives each of which can be a line segment, a solid triangle or an arrow. The latter two are then annotated with their respective direction.

1.2.2 Rule System

Our rule system essentially works with the geometric primitives resulting from the vectorisation as input. In particular it uses character groups and circles from the OCR step, as well as line segments, solid triangles and arrows from the bond separation. The goal of our rule system is to rewrite the input set of primitives into a graph structure that represents the molecule in terms of the atoms (or superatoms) and different types of bonds between them.

We define rules in terms of preconditions and consequences. A rule is applicable if there exist geometric objects that satisfy its preconditions. The consequence results in the removal of existing geometric objects and the addition of elements to the graph as well as possibly the addition of new geometric objects. In general, preconditions of different rules are mutually exclusive, and thus the order of rule application is minimised. Rules work with a number of parameters, both fuzzy and strict, that set certain thresholds, for instance the minimal bond length, under which decisions will be made. These parameters allow for the customisation and adaptation to particular requirements of datasets.

1.3 Contributions

The following points provide a summary of the contributions of this thesis.

1. A novel application of the Douglas-Peucker line simplification algorithm to precisely detect line endpoints in a structure containing interconnected straight line segments.

2. A novel approach using an expanding disc to accurately identify objects in the shape of solid triangles and bold lines.

3. A novel strict rule-based system to precisely identify various bond conventions and

formations. The system consists of 18 rules and takes a set of primitives as input and produces a suitable computer processable format.

4. We demonstrate the robustness of our approach by providing experimentation two implementations. A procedural implementation and a random implementation.

1.4 Publications

The following conference and workshop publications were published to summarise the work carried out in this thesis.

- Noureddin Sadawi, "Recognising Chemical Formulas from Molecule Depictions", In Pre-Proceedings of the 8th IAPR International Workshop on Graphics Recognition (GREC 2009) [86]

- Noureddin M. Sadawi, Alan P. Sexton and Volker Sorge "Performance of MolRec at TREC 2011 — Overview and Analysis of Results", The Twentieth Text REtrieval Conference Proceedings (TREC 2011) [87]

- Noureddin M. Sadawi, Alan P. Sexton and Volker Sorge "Chemical Diagram Recognition: A Rule-based Approach", 19th Document Recognition and Retrieval Conference (DRR 2012) [88]

- Noureddin M. Sadawi, Alan P. Sexton and Volker Sorge "MolRec at CLEF 2012 — Overview and Analysis of Results", Conference and Labs of the Evaluation Forums (CLEF 2012) [89]

And two posters:

- Noureddin Sadawi, Alan P. Sexton and Volker Sorge. Performance of MolRec at TREC 2011: Overview and Analysis of Results, November 2011.

- Noureddin Sadawi, Alan P. Sexton and Volker Sorge. Performance of MolRec at CLEF 2012: Overview and Analysis of Results, September 2012.

1.5 Thesis Overview

This thesis consists of three parts. Part I is an overview of background and related work relevant to the topic of this thesis. Chapter 2 briefly introduces methods used to represent chemical structures and provides an explanation of several ways used to depict bonds and bond arrangements. Chapter 3 looks at some of the common methods used to represent chemical structures electronically. It focuses on MOL files as it is the format used for evaluating the tool developed as part of the work carried out in this thesis. Finally, Chapter 4 presents existing approaches for chemical structure recognition and analyses them in a "how-things-work" manner to provide the user with an overview of how procedural the existing methods are. Some advantages and disadvantages of these approaches are also discussed.

Part II presents our approach to recognition of chemical structure diagrams. Our method to carry out optical character recognition (OCR) and identify characters and circles has been explained in Appendix A. Chapter 5 provides specifications of our overall recognition system. It explains the philosophy of our system as well as its architecture and embeddings. It also presents an overview of our implementation methodology and contains justification of why our rule-based recognition system is favourable when compared to the traditional procedural approach. This chapter then ends by explaining our testing and validation methodology. Chapter 6 details our methods of using the Douglas-Peucker algorithm to precisely detect line segment endpoints, our method of using an expanding disc to identify solid triangles and bold lines and our method for spotting arrow lines. In Chapter 7, we provide more information about our rules and explain various concepts and parameters necessary to build them. After that, we present a summary table of our 18 rules and then discusses in detail a selection of four of these rules (with the remaining rules provided in Appendix B). The discussion includes an explanation of each rule's predicates and how they capture and rule out various patterns. Not only this, but it also provides sample OCAML code to show how we have implemented these rules. A simple fully work example to illustrate how our entire system works is provided at the end of this chapter. Chapter 8 explains what happens after our system of rules terminates. We

detail how we form a graph data structure to represent the diagram in question. We also show how we post-process the graph by expanding superatoms and guessing stereocentres. After that, we explain how we generate a suitable output. Additionally, we show the steps for a procedural approach that we have developed. The last chapter in this part is Chapter 9. This chapter begins by addressing rule conflicts, rule execution order, system termination and the issue of circularity. After that, we describe how our system can be easily extended to accommodate more bond types where we introduce four new bond types to the system. The chapter also shows which of our original 18 rules introduced in Chapter 7 and Appendix B are affected and how they can be modified to avoid conflict and ambiguity.

The final part of this thesis is Part III which provides details of the validation, robustness and performance of our rule-based recognition system. It also presents the results of our experimental evaluation after comparing our implementation with other state of the art systems in Chapter 10. This part ends in Chapter 11 with conclusions and suggestions for what can be done in the future to extend and improve this work.

1.6 Summary

This chapter provided material to motivate the reader, introduce the main research idea and provide a concise hypothesis statement. The overall structure of the rule-based approach for recognition of chemical structure diagrams was also given. Additionally, an overview of the structure of this thesis was also given.

Part I

Background

CHAPTER 2

OVERVIEW OF CHEMICAL STRUCTURE DIAGRAMS

This thesis is about extracting precise information from chemical diagrams and saving this information in a format processable by a computer. By computer processable format, we mean a format that can be used in various operations such as comparison, frequent editing, copying, pasting and others. It is obvious that for such operations, a textual format, for example, is easier and more efficient to use than an image format. Notice that the terms *chemical structure diagram* and *molecule diagram* are going to be used interchangeably throughout the remainder of this thesis.

In this chapter, we will provide a brief overview of some necessary knowledge from the Chemistry domain. We will focus on how chemical structures are often represented and on the contents of chemical diagrams. This thesis is on recognition of chemical structure diagrams so it applies several concepts from Chemistry. However, a detailed account of chemistry is out of the scope of this work. Further information can be found in [18, 96, 85, 83].

Chemistry used to be defined as the study of matter's nature, properties, decomposition and the changes that it undergoes. According to [85], this definition is not precise and it is almost impossible to accurately define it. This is because there is huge overlap between Chemistry and other scientific disciplines such as Physics, Geology, Astronomy and others. However, one can devise a general definition such as: It is the field of science that is concerned with simplifying how elements and compounds work to facilitate understanding and creation of new useful compounds. It deals with collections of atoms such as molecules, their properties, their transformations and interactions to form materials encountered in everyday life. In other words, it is the science of matter at the atomic to molecular level. Not only this, but Chemistry is often denoted the **central science** [18] since it embraces all other sciences. It is the foundation upon which other sciences such as material science, biology, pharmacy and others are based and connected through.

Numerous topics fall under the umbrella of Chemistry. For instance, the atomic structure of matter is an essential aspect. An atom is the basic chemical building block of matter, it is the smallest particle of a chemical element which retains its chemical properties. Also, the study of chemical bonding and how atoms join together is another very important field because

elements are formed by atoms which have the same chemical properties [96]. Known elements are conveniently presented in what is known as the **periodic table** where symbol abbreviations for each known chemical element and other information are listed.

Two or more atoms can join together to form what is referred to as a **molecule**. If the atoms are from different elements, the molecule is known as a **compound**. Hence, all compounds can be considered molecules, though on the other hand, not all molecules can be considered compounds. An example of this is H_2, (or molecular hydrogen), which is a molecule and not a compound because it is made from one element. Whereas Water (H_2O) is a compound and not a molecule because it is composed of two elements.

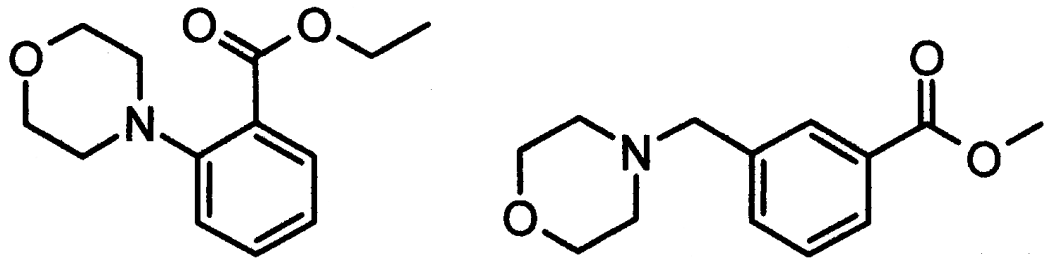
2.1 Molecule Representation/Drawing

The attraction between atoms to form molecules results in creation of chemical bonds. As proposed by the American chemist Gilbert Newton Lewis in 1916, the formation of these bonds is a result of interaction between electrons from atoms. There are two major types of bonds, ionic and covalent. However, after the interaction of atoms and creation of molecules, a means of communicating molecules including their structures and properties is needed. In the following sections, We are going to explain some of the various ways that molecules can be represented on paper or on screen.

2.1.1 Molecular formulae

In molecular formulae the number of occurrences of each atom present in the molecule is counted. They do not provide any information about the way they are joined together. For example, the molecular formula of butane is C_4H_{10}; this means that butane consists of four carbon atoms and ten hydrogen atoms. Another example is ethanol which is composed of two carbon atoms, six hydrogen atoms and one oxygen atom; hence, the molecular formula of ethanol is C_2H_6O. However, different molecules can have the same molecular formula. As can be seen in Figure 2.1, the two provided molecules have the same formula of $C_{13}H_{17}NO_3$ but the way their atoms are bonded is different. These disadvantages limit the use of

molecular formulae since information about the bonding formation cannot be obtained from such representations [22].



(a) (b)

Figure 2.1: Two different molecules with the same formula

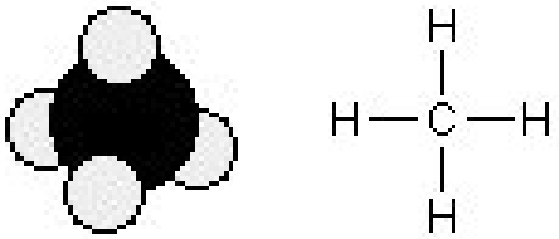
2.1.2 Structural formulae

The way various atoms are bonded is sometimes shown in structural formulae. These are depictions illustrating which atoms of the molecule are bonded to which atoms with the type of bond is also shown. We are going to introduce some of the existing ways to display molecular formulae in the coming few sections. We will also provide more information about bond types later in this chapter and throughout the remaining chapters of this thesis.

2.1.2.1 Displayed formulae

A good way to communicate molecules is through a pictorial description because much information can be conveyed via a single diagram. A diagrammatic or displayed formula shows all the bonds in the molecule as individual lines. Each line represents a pair of shared electrons. For example, Figure 2.2(a) provides a model of methane, whose molecular formula is CH_4, showing its 3-dimensional nature. The black ball represents a carbon atom and the four white balls represent hydrogen atoms (observe that this is not common in chemistry related publications). On the other hand, Figure 2.2(b) shows methane's displayed formula. Notice that the way methane is drawn does not resemble the real shape of the molecule - methane is not flat with 90°bond angles [22].

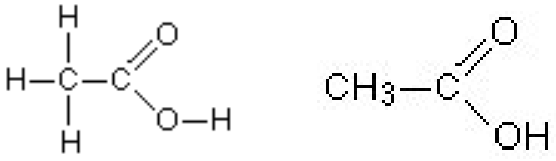
However, for large and complicated molecules, drawing a fully displayed formula can be tedious and occupy much space - especially all of the carbon-hydrogen bonds. However, the



(a) Structural formula (b) Skeletal formula

Figure 2.2: Methane and its displayed formula (image from [22])

formula can be simplified by writing, for example, CH_3 or CH_2 instead of showing all these bonds. So for instance, ethanoic acid would be shown in a fully displayed form and a simplified form as Figure 2.3 illustrates. In fact, it can be condensed further to CH_3COOH but the way atoms are bonded will not be seen [22].



(a) Fully displayed form (b) Simplified form

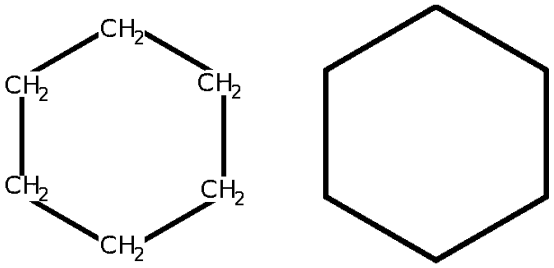
Figure 2.3: Ethanoic Acid (image from [22])

Observe that the carbon and hydrogen atoms forming methane are connected via single bonds only, whereas for ethanoic acid, the carbon, hydrogen and oxygen atoms are connected via single and double bonds. We are going to explain various bond types later in this chapter.

2.1.2.2 Skeletal formulae

One of the common ways to describe molecules is to use skeletal formulae. As the term suggests, it is an abbreviated pictorial description of molecules. In a skeletal formula, all the hydrogen atoms attached directly to carbon atoms are omitted, which results in a carbon skeleton with functional groups attached to it [22]. So, when interpreting such representations, a hydrogen atom is assumed where necessary. In fact, it is a very common practice to omit carbon and hydrogen atoms from molecular depictions, especially in research papers, advanced

textbooks and other areas.



(a) structural formula (b) skeletal formula

Figure 2.4: Cyclohexane - C_6H_{12}

In a skeletal diagram of this type.

- There is a carbon atom at each junction between bonds in a chain and at the end of each bond (unless there is something else there already).
- There are enough hydrogen atoms attached to each carbon to make the total number of bonds on that carbon up to 4 (observe that it is common to make this assumption for other atoms).
- Skeletal diagrams take less space and can be easily and quickly drawn - electronically or manually.
- They are easy to interpret, and it is easier to illustrate important parts of the molecules.

Figure 2.4 shows the structural formula and the skeletal formula for cyclohexane, C_6H_{12}. As it can be seen, cyclohexane is a ring of carbon atoms each with two hydrogens attached.

Notice that the carbon and hydrogens are not the only atoms involved in chemical molecules. Several elements can be used to form such structures depending on their properties and on the desired target molecule. A list of some of the most common elements is provided in Table 2.1.

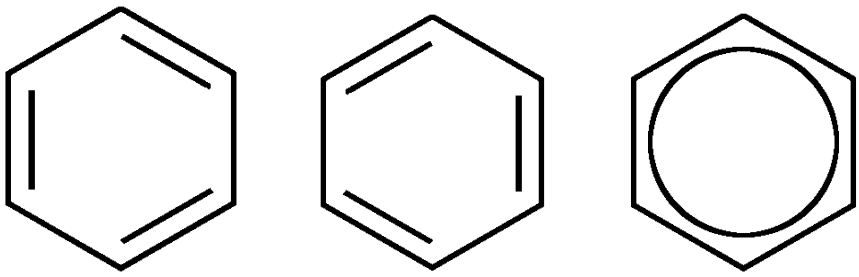
2.1.2.3 Representing Aromatic Rings

Aromaticity is a chemical property of some significance to chemists. It exists in certain cyclic, or ring, structures such as the benzene ring shown in Figure 2.5(c). Benzene, (C_6H_6), is

Table 2.1: Example Elements from the Periodic Table

Element name	Abbreviation
Carbon	C
Hydrogen	H
Nitrogen	N
Oxygen	O
Phosphorus	P
Sulphur	S
Chlorine	Cl
Fluorine	Fl
Bromine	Br

composed of six carbon atoms and six hydrogen atoms connected via single and double bonds. It is often depicted as a ring of six carbon atoms, with alternating single bonds and double bonds. However, because of reasons related to the aromaticity of the ring, the structures shown in Figure 2.5 are equivalent. Therefore, it is a common practice to indicate the aromaticity of a ring by drawing a circle inside it (Figure 2.5(c)).



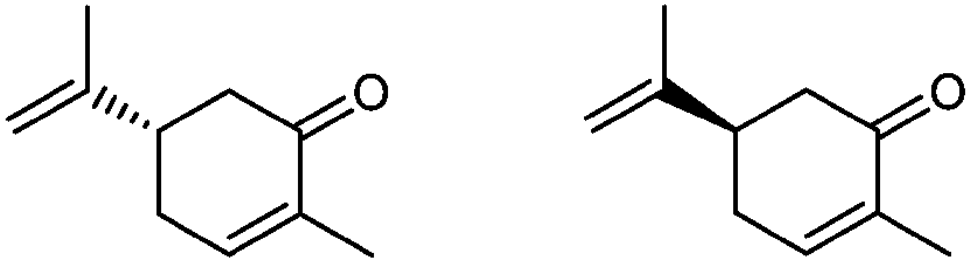
(a) (b) (c)

Figure 2.5: Benzene C_6H_6 - The three structures are equivalent

2.1.3 Two Dimensional Structural Formulae for Three Dimensional Representations

Molecules are inherently 3-dimensional objects albeit they are usually displayed on 2-dimensional media such as paper or computer displays. This was first suggested by the German chemist August Kekulé (1829-1896). Hence, sketching any 3-dimensional entity on 2-dimensional media involves change in shape, restriction in terms of expressiveness and other problems. Therefore, a field of Chemistry related to the spatial formation of atoms in molecular

objects had to be introduced. This field is called **Stereochemistry**. So, the relationship between atoms in 3-dimensional space determines their stereochemical configuration. The field is also concerned with the consequences of having different spatial arrangements of atoms in molecules. According to [16], two enantiomers of carvone provide a widely known example of how significant stereochemistry is in everyday life. As shown in Figure 2.6, the two compounds have different stereochemistry and therefore they have different smells.



(a) S-carvone (smells like caraway) (b) R-carvone (smells like spearmint)

Figure 2.6: Significance of Stereochemistry

The need to convey 3-dimensional information in molecule entities on 2-dimensional surfaces led to the use of perspective. Fortunately, several drawing conventions were suggested to at least illustrate the 3-dimensional configuration of molecules in 2-dimensional media. The most common way to achieve this is the use of special shaped bonds. As Figure 2.7 shows, different shapes of bonds are used to imply the existence, or non-existence, of 3-dimensional arrangement between two atoms. According to recommendations of International Union of Pure and Applied Chemistry (IUPAC) for graphical representation of stereochemical configuration [16], in case of 3-dimensional arrangements, the architecture specifies that one of the two concerned atoms is protruding or receding from the plane of depiction and the other atom is the **stereogenic** centre.

Planar bonds, Figure 2.7(a) are usually drawn as normal lines to indicate that the bond is in the plane of the drawing surface (even if the two atoms at each end are not co-planar). By normal we mean they are not thick, wavy, dashed or any shape other than an ordinary straight line.

If a structure has atom(s) above the drawing plane, bond(s) to such atoms are usually

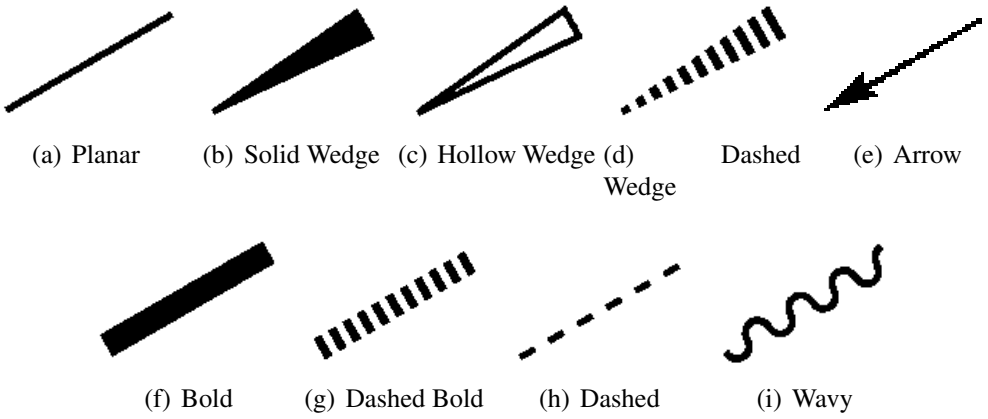


Figure 2.7: Common Types of Bond Representation on Paper

illustrated with solid wedge (Figure 2.7(b)), with the wide end of the wedge connected to these atoms (the narrow end of the wedge is connected to the stereogenic centre). However, using a hollow wedge such as the one in Figure 2.7(c) can also be encountered (several examples can be found in [104]). Additionally, a bold bond, which is usually shown as a thick line as in Figure 2.7(f), is sometimes used although it is often used to show that both atoms are above the plane of drawing, or to indicate that one of the atoms is above plane and the other is not, which means that the stereogenic centre is unknown [16].

As for bonds that project below the plane of the drawing, different formations of short normal lines have been used [16]. These include, short parallel lines of increasing/decreasing length (a hashed wedge as in Figure 2.7(d)), short parallel lines of the same length (a hashed bold bond as in Figure 2.7(g)) and short collinear lines of the same length (Figure 2.7(h)). These bond formations are usually used to show that both atoms are below the plane of drawing or to indicate that one of the atoms is below plane and the other is not, which means that the stereogenic centre is unknown [16].

However, representation of bonds going into the plane of drawing is still problematic despite earlier recommendations to use hashed bold bonds [16]. For example, there is no universal agreement on how to interpret a hashed wedge bond. That is, atoms at the wide end are sometimes considered to be in the plane of drawing. This interpretation is sometimes given to atoms at the narrow end. In addition to that, a dashed bond such as the one shown in

Figure 2.7(h) is sometimes used to represent a hydrogen bond, a partial bond or delocalisation.

A wavy line such as the one shown in Figure 2.7(i), of round or angular waves, is used to explicitly indicate that the configuration is not known. This means that the bond can be either protruding or receding (above or below plane). Hence, wavy bonds involve two ambiguities, the first is whether the bond above or below plane, and the second is which side the stereogenic centre is. The waves are usually of similar length, although wavy bonds with waves of different amplitude can be encountered (several examples can be found in [10]). Furthermore, the arrow bond shown in Figure 2.7(e) are also used to depict a dative bond. According to [104], it indicates the existence of a negatively charged atom at the arrow end of the bond.

Notice that it can be understood from the provided explanation that an author's intended meaning can be misunderstood from just a structure drawing. This is because there is no consensus amongst chemists on how to draw each architecture [16]. Having said that, this thesis is concerned with the accurate recognition of different kinds of bond shapes and arrangements. It is not intended to judge how valid, or preferable, bond shapes are, or how they should be depicted in terms of orientation or any other property.

We are going to provide more information on how different types of bonds are illustrated and introduce several common bond formations and arrangements in the coming sections of this chapter. In the next section we will introduce what is known as *Markush Structures*. Notice that automatically interpreting such structures will not form part of this thesis.

2.1.4 Markush Structures

Markush structures are used in compound classes which are described by generic notations [5]. This is done to indicate that the presented structure has some degree of variability [41]. They are mostly used in databases of patents and the notations are used to represent a series of chemical compounds. This means that a diagram can describe a family, or families, of compounds. A typical Markush structure consists of a core structure represented by a diagram and substituents which are usually listed separately [52]. Each substituent can contain variable group(s), normally referred to as R1, R2 and so on, where the introduction of

variability can be done in any of several ways. The following list provides some of these ways:

- **R-Groups.** Also known as substituent variation, this is where the structure contains a list of alternatives. An R-atom is used to denote the variable part of the structure and the enumeration of the list of alternatives is given separately.
- **Atom Lists.** A List of atom types can be present at a certain position to allow the structure to have some degree of variability.
- **Position variation.** A substituent group in the core structure may have a variable attachment position.
- **Repeating Units.** This is when the occurrence of a group can be repeated several times (frequency variation).

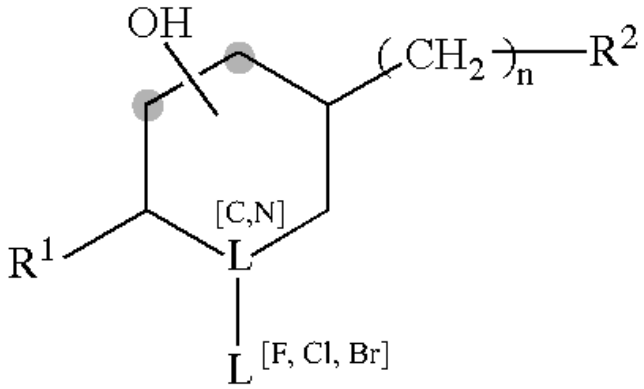


Figure 2.8: Markush Structure

An example Markush structure illustrating the mentioned types of variability is given in Figure 2.8. $R1$ and $R2$ represent substituent variation as $R1$ can be either F or Cl, whereas $R2$ can be a generic expression such as *alkyl*. The n represents frequency variation, and the OH has a position variation because it can be connected to either of the shadowed nodes. Finally, the L indicates the existence of atoms lists (provided in square brackets). The reader should note that these are only examples of ways to represent variability in Markush structures, a larger list can be found in [5].

2.2 More on Diagram Notation

As we have explained in the previous sections, molecule diagrams depict the relationship between atoms using various bond shapes. Apart from carbon and hydrogen, a list of the most common atoms has been provided in Table 2.1. Additionally, an overview of the most common bond types has been provided in Section 2.1.3. However, a molecule diagram can contain subtle combinations and arrangements of such atoms and bonds. For instance, the use of a circle inside a multi-sided polygon to indicate the existence of an aromatic ring is a common practice (Section 2.1.2.3). Another example is the contents of Markush structures which have been explained in Section 2.1.4.

2.2.1 Double and Triple Bonds

In addition to that, molecule diagrams can contain other architectures. One example is the use of two close parallel line segments and three close parallel line segments to illustrate planar double bonds and planar triple bonds respectively (Figure 2.9). Notice that the line segments have similar lengths. Observe that each line segment in each bond represents a pair of shared electrons between the the two bonded atoms.

(a) Planar Double Bond (b) Planar Triple Bond

Figure 2.9: Planar Double and Triple Bonds

2.2.2 Implicit Atoms

Another example is the use of combinations of multiple bonds. As can be seen in Figure 2.10, there is a combination of single bonds with double and triple bonds respectively. Subsequently, carbon atoms are understood to exist at the connection between the single bonds and the double and triple bonds. These will be referred to as implicit atoms, implicit nodes

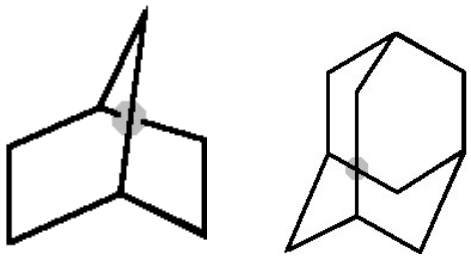
or embedded nodes in the remaining chapters of this thesis. To clarify the idea, a grey circle has been placed where an *implicit* carbon atom exists. It can be noticed that there is only one implicit carbon atom in Figures 2.10(a) and 2.10(c), whereas there are two implicit carbon atoms in Figures 2.10(b) and 2.10(d).

(a) (b) (c) (d)

Figure 2.10: Embedded Node on Shadowed Side(s) of Planar Double and Triple Bonds

2.2.3 Bridge Bonds

Chemists use another arrangement of line segments to illustrate what is denoted as *bridge bonds* (Figure 2.11). Bridge bonds are a formation of line segments to indicate the existence of a 3-dimensional structure. These are typically presented in a $2^1/_2$-dimensional perspective drawing form where there are multiple different connection paths between different parts of the chemical structure diagram. As Figure 2.11(a) illustrates, the bond line segments that cross in the drawing but do not intersect in 3-dimensions are sometimes drawn in an open form, where the back-ground bond is drawn broken leaving the viewer to deduce that the broken lines are actually one line. On the other hand, Figure 2.11(b) displays an architecture where the different connection paths between different parts of the chemical structure diagram are drawn in a closed form, leaving the reader to deduce that the crossing point does not correspond to a junction.



(a) Open Bridge (b) Closed Bridge

Figure 2.11: Open and Closed Bridge Bonds (shadowed)

Observe that the structure in Figure 2.11(a) consists of a hexagon and two pentagon shapes, whereas the structure in Figure 2.11(b) consists of three hexagon shapes. These shapes are sharing some of their sides and they can be deduced by visualising them in 3D.

2.2.4 Bond Directions

As we have explained in section 2.1.3, several bonds represent 3-dimensional architecture. These include dashed bonds, dashed bold bonds, dashed wedge bonds, wedge bonds, bold bonds, hollow wedge bonds and wavy bonds. Some of them represent below plane bonds (these are the dashed, dashed bold and dashed wedge bonds) and some others represent above plane bonds (these are the wedge, bold and hollow wedge bonds) with the wavy bond representing either below or above plane architecture.

These bonds are usually connected to what is denoted as the "*stereogenic*" centre. Using this centre, such bonds can be assigned "*directions*" to clearly indicate which end of the bond is on plane and which end is below, or above, the plane. This direction is vital to correctly interpret these bonds as it is chemically very significant. However, the stereogenic centre is unknown for all the above mentioned bonds except the wedge, hollow wedge and dashed wedge bonds. This means, a mechanism needs to be found to correctly identify stereogenic centres when they are unknown.

As far as the correct interpretation of chemical structure diagrams is concerned, usually the structures do not provide sufficient information to know the stereocentre in case of bold, dashed bold and dashed bonds. Thus, this represents one of the challenges in recognition of chemical structure diagrams. We are going to introduce our way for identifying bonds that represent 3-dimensional architecture and how we determine the stereocentre when it is unknown in Section 8.4.

2.2.5 Superatoms

The special arrangements in chemical structure diagrams are not restricted to bonds, rather, atoms are also sometimes presented in a special way. To save display space, authors often use

shortcut names to known substructures instead of drawing the entire substructure in a chemical structure diagram. These shortcut names are known as superatoms. Figure 2.12 shows several example superatoms.

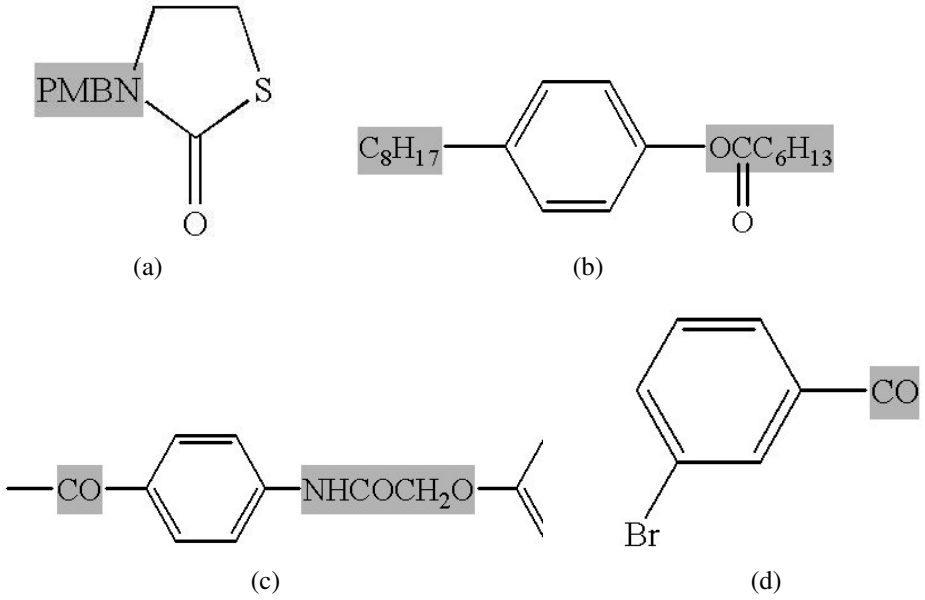


Figure 2.12: Example Superatoms (Highlighted)

Superatoms have a potentially complex substructure that can contain many atoms and bonds. Furthermore, while a chemical formula is at least potentially parseable, some of these names give no indication to their content other than via looking up a dictionary. For example, the superatom name $PMBN$ (shown in Figure 2.12(a)) identifies a substructure with the chemical formula "C_8H_9ON".

Also, while it is clear which atoms in the main structure are connected to the superatom, it is not clear which atom(s) in the superatom is/are connected to the main structure. This is evident in the shown figures. For example, the superatom OCC_6H_{13} (shown in Figure 2.12(b)) is connected to the main structure via two bonds - a single bond (to an omitted carbon atom), and a double bond (to the Oxygen atom). However, it is not obvious which atom in the superatom is connected to the single bond and which atom is connected to the double bond. This is also clear for the superatom $NHCOCH_2O$ (shown in Figure 8.4(d)). In fact, this problem is more complicated than that since there are superatoms which have the exact same name but they have a different formula, and hence, a different number of connections. This is clear in the case of

the CO superatom. In Figure 8.4(c), the main structure is connected to it via one connection only, whereas in Figure 8.4(d), the main structure is connected to it via two connections. Notice that the two superatoms have exactly the same name but a different formula because of the conventional omission of hydrogen atoms from atom names.

On the one hand, these shortcut names represent a useful way for chemists to keep the molecule drawings compact, but on the other hand, they create a problem for people trying to automatically interpret such drawings from images. That is, after optically recognising characters and symbols and grouping them, the user is left with the following problems:

1. It is not straightforward to know what the exact superatom's substructure is (although for a chemist, this is a trivial task).

2. Also, and more importantly, it is not straightforward to know how the superatom is exactly connected to the surrounding structure.

It is relatively easy to deduce the molecule structure of some chemical formulae, therefore, we will focus on the second problem. In Section 8.3, we will introduce our method for solving this problem. Additionally, we are going to provide more information on superatoms in Section 3.1.5.3.

2.3 Summary

In this chapter we have discussed the representation of chemical structures with their 3-dimensional information on a 2-dimensional surface. The chapter started by providing some domain knowledge from Chemistry and continued by introducing methods of molecule representation. As mentioned before, this thesis is not intended to be part of a work submitted to attain a degree in Chemistry. Nevertheless, the provided information should give the reader an overview of what the problem of chemical structure diagram recognition entails. In the next chapter, the focus will be on several formats of how such diagrams are represented in a computer.

CHAPTER 3

ELECTRONIC REPRESENTATION OF CHEMICAL STRUCTURE DIAGRAMS

We have introduced some of the various ways of representing chemical structure diagrams on 2-dimensional media in the previous chapter. In this chapter, we present how these diagrams can be represented electronically (i.e. in computers). The field that deals with representing information in a format that facilitates knowledge extraction is called *Chemoinformatics*. The field of Chemoinformatics involves the use of computational methods for the processing of chemical data and tackling problems in the field of Chemistry. The focus of these methods is on manipulating chemical structure information but it also encompasses several applications on chemical information such as analysis, design, management, retrieval and visualisation which are carried out in the context of drug discovery and design.

There does not seem to be a consensus on the correct spelling as there are different spellings reported in the literature. These include Cheminformatics, Chemical Informatics and Chemiinformatics. This is possibly because the field is new compared to other fields in Chemistry [60]. In fact, the field is so new that there is not even a universal agreement on its definition, albeit numerous attempts have been made to define it. Examples of these are:

> *"...the mixing of information resources to transform data into information and information into knowledge, for the intended purpose of making better decisions faster in the arena of drug lead identification and optimisation"* [17]

and

> *"Chem(o)informatics is a generic term that encompasses the design, creation, organisation, storage, management, retrieval, analysis, dissemination, visualisation and use of chemical information, not only in its own right, but as a surrogate or index for other data, information and knowledge"* [77]

Chemoinformatics is an important discipline as it involves addressing several challenges. For example, a difficult task is the efficient storage and maintenance of the huge already existing amount of data. Also, the use of the already known data to learn about cases for which the required data is not available is another ongoing task. In addition to that, it plays a vital role in understanding the biological activity of a compound and its structure.

Chemoinformatics presents an opportunity to integrate the understanding of information technology with knowledge of Chemistry, this is due to the ever expanding overlap between

the two fields. In fact, advances in this field allow chemists to accurately model chemical compounds. Chemists are now able to access various molecular properties such as shape, chemical reactivity and others.

The scope of chemoinformatics encompasses diverse applications such as:

- Structure representation
- Structure/Substructure searching and database searching
- Similarity measurement and clustering

3.1 Electronic Representation of Molecular Structures

A common way to store information about chemical compounds is by means of databases entries containing different properties and possibly representations of each compound. Several databases are now publicly available through many organisations. The number of entries in a typical chemical compound database ranges from hundreds of thousands to millions. For example, at the time of writing this thesis, the online chemical database ChemSpider [24] reports that it contains more than 28 million structures.

In the following subsections, we are going to present an overview of some of the methods used to represent a molecular structure in a computer readable format. We will use the example in Figure 3.1 as a basis and show its corresponding representation in each method.

3.1.1 Chemical Graph

A mathematical graph can be used to represent structures not only in Chemistry but in several other fields. We are going to provide a general definition of a graph in mathematics, and then show how it can be used to represent a chemical structure.

Definition 1 *A graph G is a pair (V, E) of sets, called the vertex set and the edge set. V is a set of vertices, in the form $v_1, v_2, \ldots, v_n$ and E is a set of edges, where an edge is a pair of vertices.*

Members of the set E denote the existence of an edge between every pair of vertices. Another important property related to a mathematical graph is a path. A path between two

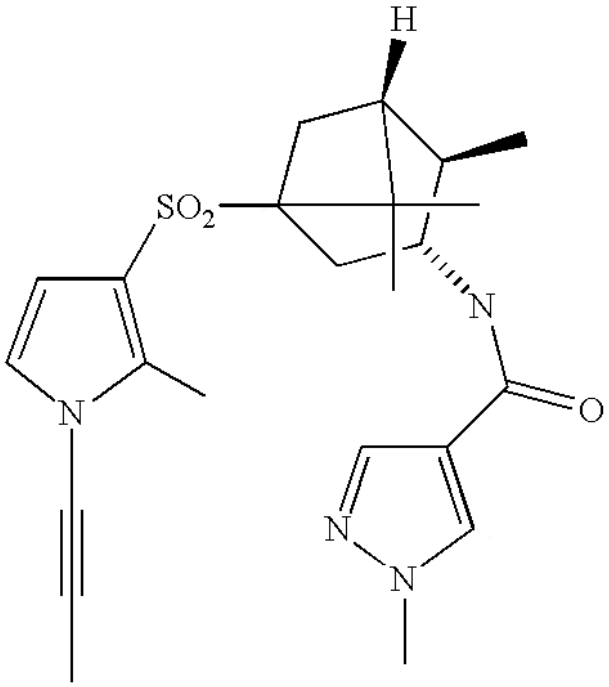


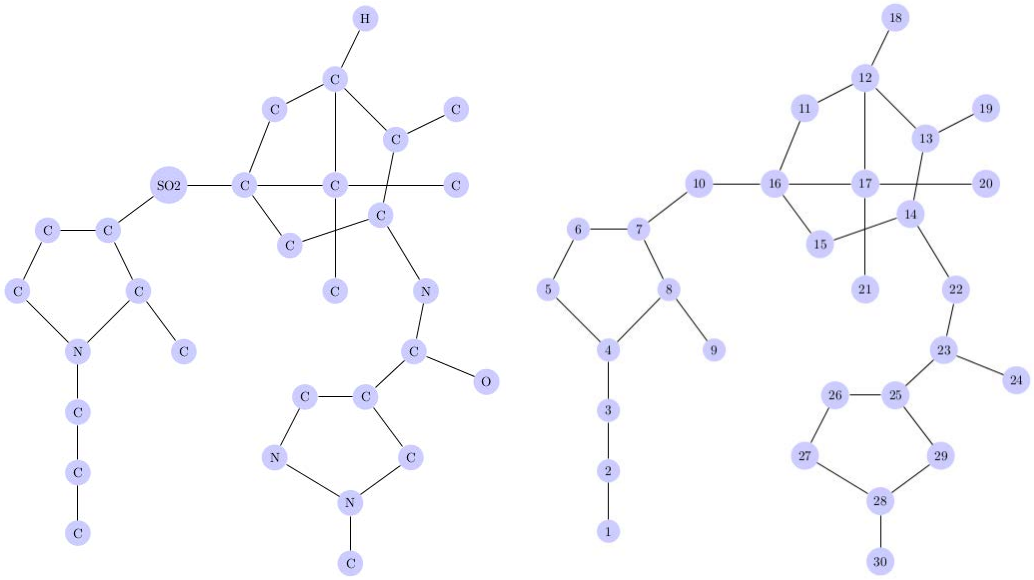
Figure 3.1: Example Chemical Structure Diagram

vertices u and v in a graph can be defined as the sequence of consecutive edges required to travel from u to v, and the length of the path is the number of edges in it.

The graph definition which we have provided is abstract and defining what the vertices and edges represent is application dependent. Therefore, a molecular structure can be interpreted as a mathematical graph where each atom is a node, and each bond is an edge [29]. It is worth mentioning that the same element can exist multiple times in the same structure. This can lead to ambiguity in distinguishing atoms (nodes) when building a corresponding chemical graph. Therefore, we are going to allocate unique identifiers to atoms. Figure 3.2 shows a graph representation of the chemical structure shown in Figure 3.1. Representing a chemical structure as a graph provides the ability to use graph theory for the mathematical processing of molecular structures.

3.1.2 Adjacency Matrix Representation

A matrix of dimensions $n \times n$ can be used to represent a molecular structure with n atoms (nodes). Remember that H atoms are often omitted and therefore we are not going



(a) Molecular Graph (b) Molecular Graph with Node IDs

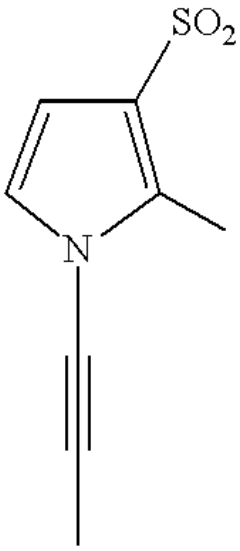
Figure 3.2: The Corresponding Chemical Graph of the Structure in Figure 3.1

to consider them as nodes. An adjacency matrix illustrates the connectivity of vertices through edges. Figure 3.3 shows the corresponding adjacency matrix of part of the structure shown in Figure 3.1. To preserve space, we are going to assume that we only have the structure represented by atoms 1 to 10 (Observe that the atom IDs and labels of the structure in Figure 3.3(a) are provided in Figure 3.2). For a molecular graph, the index of the matrix corresponds to the atoms' unique identifiers and the entries indicate whether a bond exists between any two atoms. Formally, we can define an adjacency matrix as:

Definition 2 *The $n \times n$ matrix A, used to represent a graph G with n vertices $v_1, v_2, \ldots, v_n$ such as the one defined in the previous section, is called adjacency matrix whose entries are of the value:*

- *a_{ij} = 1 if a path from v_i to v_j exists*
- *a_{ij} = 0 otherwise*

Observe that other matrix representations can also be used. For example, a *distance matrix* encoding the distances between atoms can be used to represent molecular structures. To find the distance D between two atoms a and b, the shortest path between the two atoms is



(a) Molecular Structure

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	0	1	0	1	0	0	1	0	0
5	0	0	0	1	0	1	0	0	0	0
6	0	0	0	0	1	0	1	0	0	0
7	0	0	0	0	0	1	0	1	0	1
8	0	0	0	1	0	0	1	0	1	0
9	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	1	0	0	0

(b) The Corresponding Adjacency Matrix

Figure 3.3: A Chemical Structure and its Adjacency Matrix Representation

determined, and then D is calculated as the number of bonds (edges) in this path. Another matrix representation of chemical structures is the *connectivity matrix*. Also known as the bond matrix, the connectivity matrix used to represent a molecular graph illustrates atom (node) connectivity and the connectivity order. In other words, it indicates the corresponding bond orders for any bonded atoms. Notice that the matrix representation is easy to use and fast to access, for example, when examining whether two atoms are connected. However, its disadvantage is that is has a fixed size as space will be allocated for entries of unconnected atoms.

3.1.3 Linear Representations

Representing chemical structures in a linear format is very common amongst chemists. The idea is to have a concise and descriptive representation that conveys information about the structure and saves display and storage space.

3.1.3.1 The Molecular Formula

We have briefly introduced this representation in Section 2.1.1. In a molecular formula, the number of atoms of each element that form a particular chemical structure is precisely

shown [57].

A symbol based representation is used to shorten this string representation. Each atom is represented by its atomic symbols from the periodic system. In addition, the Hill system [47] describes the order of the atoms within a molecular formula, i.e. the number of carbon atoms in a molecule is indicated first, then the number of hydrogen atoms are listed followed by the number of all other chemical elements subsequently, in alphabetical order. If we take Figure 3.4(a) as an example, its molecular formula should be $C_5H_7N_3O$.

This first systematic naming approach was a step towards more standardisation, which was commendable but not sufficient. As discussed in Section 2.1.1, scientists realised soon that different molecules can have the same chemical formula (i.e. isomerism problem). Thus, the original form of the nomenclature was not adequate.

3.1.3.2 The SMILES String

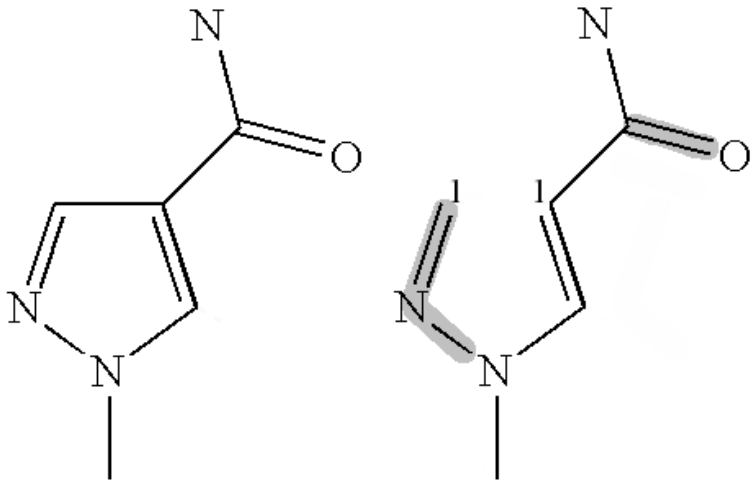
Invented by Arthur Weininger and David Weininger in the late 1980's, SMILES notation [107] is one of the common approaches to communicate molecular structures. SMILES is an abbreviation for Simplified Molecular Input Line Entry Specification. As its name suggests, SMILES is relatively easy to understand. Also, to create and interpret most of the SMILES strings, only a few rules are required.

According to [61], the SMILES notation consists of a series of characters containing no spaces (atoms are represented by their atomic symbols). Hydrogen atoms may be omitted (hydrogen-suppressed graphs) or included (hydrogen-complete graphs). Aromatic structures may be specified directly or in Kekulé form.

There are five generic SMILES encoding rules, corresponding to specification of atoms, bonds, branches, ring closures, and disconnections. Single bonds do not have a symbol representation. Double bonds are typified by '=' and triple bonds by '#'. To determine a SMILES string, the molecule graph is traversed in such a manner, that each vertex is visited only once. Cycles are described by allocating digits to the connecting ring atoms. For that purpose these connecting atoms are broken up and marked with the same digit. If a branch

point in the molecule is reached, parentheses are introduced. A left-hand parenthesis is used to symbolise a new branch and a right-hand bracket is used to indicate that all atoms in that branch were visited. Branches can be nested to any level necessary [57].

To see how simple extracting SMILES is, we are going to take part of the structure displayed in Figure 3.1 and use it as an example. As shown in Figure 3.4(a), we have taken the bottom right part (i.e. the part represented by nodes 22 to 30). To extract a SMILES string, we are going to start at the top N and walk the structure downwards. Figure 3.4(b) shows how a cycle has been broken and the digit 1 has been placed on both ends of the removed bond. The figure also shows the branches highlighted. Now applying the rules discussed in the previous paragraph, we end up with $NC(=O)C1 = CN(N = C1)C$. Notice that we can extract multiple SMILES strings for the same structure depending on our starting atom and how we traverse branches. An algorithm to extract a canonical SMILES string is introduced in [108].



(a) Molecular Structure (b) Extracting SMILES string

Figure 3.4: SMILES Example

This simplicity of SMILES led to a widespread acceptance as a universal line notation nomenclature for the representation of chemicals. In the meantime the notation has been extended several times, most notably by Daylight Chemical Information Systems Inc [61].

3.1.3.3 IUPAC Nomenclature

The concept of two dimensional molecular diagrams was a result of new experimental methods which allowed insights into molecule structures. This was the way out of the confusion and made molecules more conceivable. However, the structural bodies of different molecules with the same molecular formula differ.

A modified Geneva Nomenclature was developed by the Commission on Nomenclature of Organic Chemistry [41], formed by the IUPAC (International Union of Pure and Applied Chemistry) [70], an international non-governmental organisation established in 1919. This IUPAC nomenclature naming system is still in use today and new updates are presented regularly.

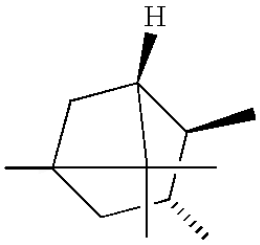
In IUPAC, names are based on the number of carbon atoms present in the longest possible chain. The inference of a molecule's name starts with determining the longest hydrocarbon skeletal structure of the molecule and identifying all functional groups in the molecule that distinguish it from the parent hydrocarbon.

The carbons in the main chain are numbered in a way that small enumerated carbons include functional groups. Generating the name is done through the connection points of the main chain-functional group. It is then followed by the functional group names and finally ended with the name of the hydrocarbon skeletal structure. Despite being systematic, including stereochemistry and allowing the reconstruction of the structural formula, this line notation has several drawbacks; it is error prone because it applies a complex naming system and the generated names are complicated.

3.1.4 Chemical Markup Language

Chemical Markup Language (CML) [1] was developed by Peter Murray-Rust and Henry Rzepa since the mid-1990's. As Figure 3.5 illustrates, CML is based on XML. The figure shows a corresponding CML representation of part of the structure shown in Figure 3.1. We have chosen the top-right part of the structure (i.e. the structure represented by nodes 11 to 20) for display space restrictions. CML provides an approach for precise management of information

in several areas and concepts such as molecular information and reactions. A recent article about the semantics of CML can be found in [66].



(a) A Chemical Structure

```
<?xml version="1.0"?>
<molecule id="MOLfile" xmlns="http://www.xml-cml.org/schema">
 <atomArray>
  <atom id="a1" elementType="C" x2="0.207100" y2="-1.069500">
   <atomParity atomRefs4="a1 a4 a8 a6">1</atomParity>
  </atom>
  <atom id="a2" elementType="C" x2="-0.112900" y2="0.030500">
   <atomParity atomRefs4="a13 a3 a4 a7">1</atomParity>
  </atom>
  <atom id="a3" elementType="C" x2="-0.752900" y2="-0.119500"/>
  <atom id="a4" elementType="C" x2="0.367100" y2="-0.419500">
   <atomParity atomRefs4="a4 a1 a9 a2">1</atomParity>
  </atom>
  <atom id="a5" elementType="C" x2="-0.912900" y2="-0.759500"/>
  <atom id="a6" elementType="C" x2="-0.482900" y2="-1.219500"/>
  <atom id="a7" elementType="C" x2="-0.022900" y2="-0.759500"/>
  <atom id="a8" elementType="C" x2="0.527100" y2="-1.419500"/>
  <atom id="a9" elementType="C" x2="0.917100" y2="-0.249500"/>
  <atom id="a10" elementType="C" x2="-1.372900" y2="-0.769500"/>
  <atom id="a11" elementType="C" x2="0.637100" y2="-0.759500"/>
  <atom id="a12" elementType="C" x2="-0.022900" y2="-1.419500"/>
  <atom id="a13" elementType="H" x2="-0.002900" y2="0.450500"/>
 </atomArray>
 <bondArray>
  <bond atomRefs2="a1 a4" order="1"/>
  <bond atomRefs2="a1 a6" order="1"/>
  <bond atomRefs2="a1 a8" order="1"/>
  <bond atomRefs2="a2 a3" order="1"/>
  <bond atomRefs2="a2 a4" order="1"/>
  <bond atomRefs2="a2 a7" order="1"/>
  <bond atomRefs2="a2 a13" order="1"/>
  <bond atomRefs2="a3 a5" order="1"/>
  <bond atomRefs2="a4 a9" order="1"/>
  <bond atomRefs2="a5 a6" order="1"/>
  <bond atomRefs2="a5 a7" order="1"/>
  <bond atomRefs2="a5 a10" order="1"/>
  <bond atomRefs2="a7 a11" order="1"/>
  <bond atomRefs2="a7 a12" order="1"/>
 </bondArray>
</molecule>
```

(b) The Corresponding CML file

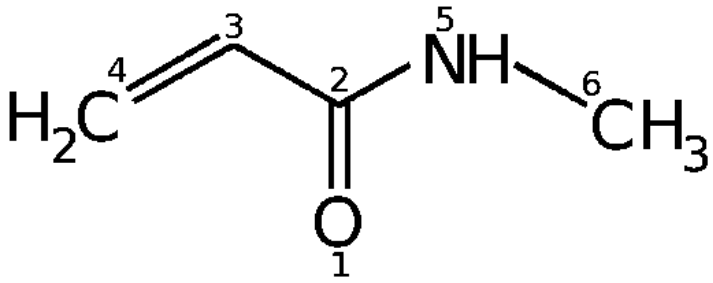
Figure 3.5: An Example Structure with its CML file

3.1.5 Connection Tables (MOL files)

As the definition of the adjacency matrix suggests, if the number of atoms in the molecule is n, the number of entries increases with respect to n^2. For that reason, connection table formats [41, 60] became more popular. By listing only the atoms that are present in a molecule and the bonds between them (Figure 3.6), the number of entries only increases as a linear function of the number of atoms.

In connection tables, atoms are assigned indices (sometimes called atomic numbers) which operate as unique identifiers. At least two sections exist in the simplest form of this kind of

format (Figure 3.6). The first section contains all indices of the atoms and their corresponding labels. The second part holds all the bonds specified as pairs of atomic numbers.



Atom List	
1	O
2	C
3	C
4	C
5	N
6	C

Bond List		
Atom 1	Atom 2	Bond Order
1	2	2
2	3	1
2	5	1
3	4	2
6	5	1

Figure 3.6: A Basic Connection Table for a Simple Molecular Structure

Connection tables can include more information. Despite the usual practice of omitting hydrogen atoms (hydrogen suppressed connection tables), details such as spatial coordinates of atoms (xy or xyz), atom charge and others are often included in connection table formats.

There are several connection table file formats reported in the literature; ranging from files to describe one molecular structure (MOL files), to SDfiles, which contain information about one or more molecular structures and others. The reader is reminded that this thesis will only focus on MOL files, and therefore it is recommended to read Symyx's CTfile standard [98] to gain a deeper insight into the exact format of MOL files.

3.1.5.1 Detailed Structure of MOL file

As shown in Figure 3.7, the MOL file consists of several blocks describing the structural properties and relationships of a group of atoms. It starts with a *header block* where general information about the structure, or the MOL file itself can be saved. The first important part is the *counts line* which contains the number of atoms, the number of bonds, the MOL file version and other information. It should be noted that the MOL file version used throughout this thesis is V2000. The next essential block is the *atom block* which specifies the atom's

spatial coordinates, symbol, charge, stereochemistry and other information. After this comes the *bond block* which, as its name suggests, holds information about atoms' connectivity. Any two atoms connected by a bond are specified (using their atomic number) with the bond type and any bond stereochemistry. The MOL file may also contain other blocks such as an *atom list block* and a *properties block*. The first shows the atomic number of the list and the atoms in the list, and the second contains additional properties such as charged atoms, isotopes and others. It ends with a *M END* entry.

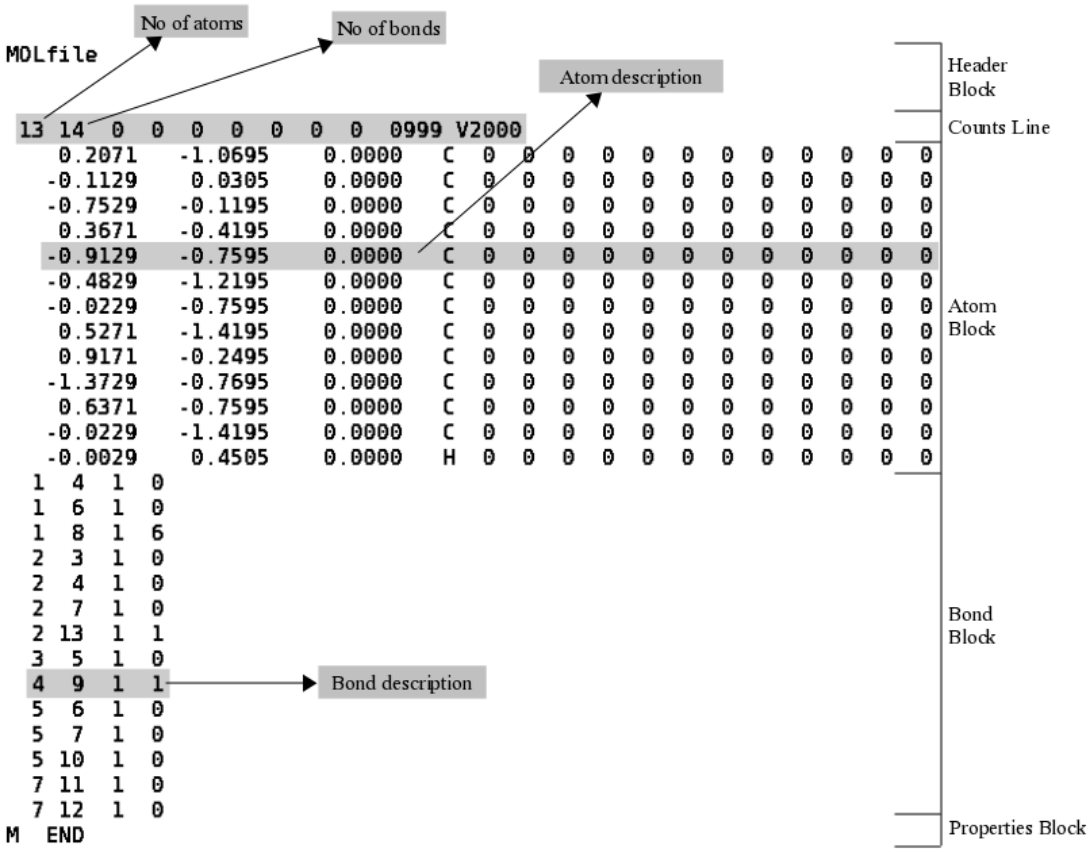


Figure 3.7: The Corresponding MOL file of the structure in Figure 3.5(a)

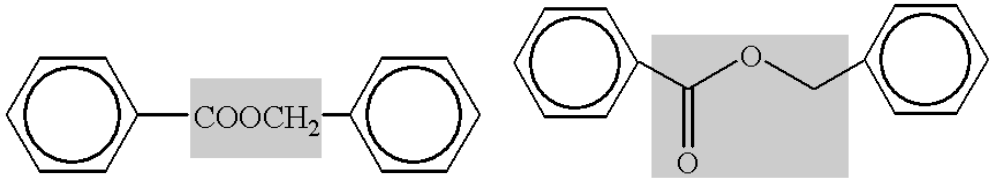
3.1.5.2 Stereochemistry in MOL files

An example was shown in section 2.1.3 to illustrate the significance of stereochemistry in chemical compounds. Two molecular structures can look the same when displayed, but their corresponding MOL files can have different stereochemistry. Although it can be argued that, if the purpose of an approach is to extract accurate information from a given molecular

depiction so that the same structure results when rendering the extracted data, then having different stereochemistry becomes insignificant. However, if the results are to be processed further, then the issue becomes essential. Therefore, it is important to have a precise description of compounds in their corresponding MOL files.

3.1.5.3 Superatoms in MOL files

We have introduced superatoms in Section 2.2. They are shortcut names used to save display space. In a MOL file, superatoms are usually placed in the properties block. A definition of a superatom starts with the line *M STY...* every superatom is given a unique integer index value. An ideal superatom block contains entry/entries listing the atoms they contain, and an entry indicating the bond(s) that connect the superatom to the main structure. It also contains an entry to show the superatom's label which is needed for display.



(a) An example Superatom (b) The Corresponding Substructure

Figure 3.8: A Superatom with its equivalent Substructure

Figure 3.8 shows an example superatom and its corresponding substructure (both are highlighted). In 3.8(a), only the shortcut name is shown, whereas the full substructure is displayed in 3.8(b). In fact, figuring out the correct interconnections between superatoms and their parent, or host, structure represent a challenge. This is due to the difficulty of knowing which atoms of the superatoms are connected to the main structure. This gives rise to an interpretation problem since the intended meaning is not immediately obvious. Not only this, but sometimes superatoms having the exact same shortcut name but with different number of connections to the parent structure are also encountered in the literature. More information about superatoms will be provided in Section 8.3.

3.2 Summary

This chapter provided a brief introduction to the field of Chemoinformatics and its applications. We have introduced several ways of how chemical structure diagrams are represented in computers. These computer processable formats allow users to edit the chemical structure diagram's connectivity information which helps in areas such as scientific research. In the next chapter, we are going to review the existing approaches to chemical structure diagram recognition. In Part II, the focus will be on the computer science side of the task. Hence, we are going to introduce our rule-based approach with all the relevant details.

CHAPTER 4

EXISTING APPROACHES TO CHEMICAL DIAGRAM RECOGNITION

There are several existing approaches for chemical diagram recognition. Some of them date back to the 1980s. In this chapter, an overview of these existing approaches will be given. The overview will be of a ”how it works“ format, meaning it will be shown how each of these approaches works.

4.1 The tool developed by IBM's Almaden Research Centre

This is one of the earliest attempts to tackle the problem [20, 15]. It was carried out at IBM's Almaden Research Centre.

The input images to this system are binary images at 300 dpi resolution. All connected components of the input image are determined. Here a multi-sided convex polygon bounding each subimage is generated instead of the conventional square shaped bounding box as it helps improving the accuracy of calculating distances between adjacent components.

A molecule clipping process is run where detecting the molecular diagram in the image is achieved by applying a size filter to the connected components. A size that is larger than the maximum expected character size is chosen. This may result in subimages which are presumed to be parts of the molecule image. These parts are combined using a predefined threshold distance to make the complete image with all its connected components.

The Graphical Image Formatting and Translating System, GIFTS, is used to vectorise the image. GIFTS, which was developed by IBM Tokyo Research Laboratory, outputs a set of coordinates representing endpoints. Short vectors are assumed to be parts of characters and a classification process is carried out to separate atoms and lines (bonds). The classification process starts by grouping connected vectors and then identifying which of them are characters, bonds or circles.

A predefined value of the ratio of the size of the group to the size of the whole structure is used to filter out symbols. This is followed by analysis of the neighbourhood of small groups, such as “l” which is considered as a character if located close to another character. Circles are identified as groups having at least eight vectors. And the rest of groups are labelled as bond structures.

An OCR engine is used to identify characters. The connected component lists of groups classified as characters are used to retrieve their corresponding pixel groups from the structure image which in turn are used as input to the OCR engine.

An attempt is also made to identify and segment glued characters. The process starts by normalising the extracted regions and sending them to the OCR engine. To identify characters, the approach referenced in [49] is employed.

This is followed by parsing the strings of characters identified by the OCR to map any strings corresponding to substructures into their original basic structure format. Unconnected strings are discarded.

The recognition process continues by preprocessing and then analysing the rest of the vectors to constitute the basic structure of the diagram. Preprocessing is done because straight lines may be split into two or more shorter lines.

This procedure involves a clean-up process to correct any lines broken near a junction (where more than two lines intersect) and to correct any single lines split into shorter segments. To correct the first case, any relatively short lines are removed and the lines connected to them are linked. The second case is handled by using a preset value for the angle at the point where two lines connect. If any lines are found to make an angle less than this threshold, the point of meeting is removed and the two lines are merged to make one longer line segment.

Circles denoting the existing of aromatic rings are identified as mentioned previously and their closest six atoms are determined and sorted (clockwise or anti-clockwise). Double bonds are placed on every second bond around each circle.

A MOL file containing connectivity information is generated at the end of this stage. The produced MOL file is then checked for valence validity and the structure is displayed on a GUI for manual correction (if needed).

A close look at the publications describing this work can reveal a few drawbacks. For example, the authors did not mention how they identify embedded nodes and various types of 3-dimensional bonds. Also, they assume that the number of atoms surrounding a circle in an aromatic ring is always six.

4.2 CLiDE

Chemical Literature Data Extraction [48, 105], or CLiDE starts with binarising the input image (full page) using the technique introduced in [9]. A multi-coloured image is binarised using a certain threshold to identify foreground and background components (foreground is turned into black whereas background is turned into white). After binarisation, a connected component labelling process is applied using a run-length technique. The result of this step is a list of all connected components in the image with their contours represented as series of directions.

Page segmentation follows to separate text from graphics using the approach introduced in [91]. In this technique, the page structure is represented as a tree starting by generating lists of words, lines, text blocks and graphic blocks and then deriving the least-cost spanning tree.

CLiDE Pro continues by attempting to divide connected components into basic categories such as atoms (text) and lines. This is carried out by calculating the value of a size threshold (using the method found in [106]) and using some other features, such as density and aspect ratio, with this threshold value to classify the components.

Representation of bonds as vectors is carried out by vectorising the connected components classified as lines and graphic components. This is achieved by analysing the contours of the connected components and generating a polygon for every contour. These polygons are approximations and they are generated by detecting straight and curved segments using a method similar to the one introduced in [92].

A preset threshold value is used in creating these polygons such that a side of the polygon is not further than this distance from every point on the original contour. Long sides of polygons indicating the existence of straight lines and short sides indicating curves result when an appropriate value for this threshold is selected. Information about the segment in the other side of the line is used to correct errors resulting in having more than one fraction for the same line, or uncutting borders where they should be cut.

At the end of this step, information about components is gathered. Since all vectors are

described by their corresponding two borders, their shapes (straight or wedge), coordinates, thickness are determined. Wavy lines are identified as relatively short vectors (shorter than characters) which connect to each other and a straight line on which they lie can be drawn.

Components classified as dashes (short line segments) are then investigated. If situated along a straight line, short parallel dashes are identified by comparing the size of dashes found at both ends of lines (dashed wedge) or as parallel dashes of similar length. Straight dash bonds are detected as collinear dashes. Any remaining dashes are investigated further according to their dimensions and location.

Characters are classified using an OCR engine based on a filter technique and analysis of their geometrical and topological features. Symbols of relatively larger size can be interpreted regardless of their size and font and, to some extent, rotation. After identifying characters, they are grouped (horizontally and vertically) according to their coordinates to build words which are in turn looked up in a database of frequently used functional groups such as R-groups and others. An attempt to parse words or atom labels not found in the database is made.

To determine connectivity information, all bond lines are associated with free-flags at their two endpoints. If two bond lines touch each other, their corresponding free-flags are set to false, otherwise they are set to true.

After this, line-atom connectivity is determined using a predefined threshold distance, making sure that the line's free-flag is set to true, the atom label is in the direction of the line and making sure that this atom is the closest to this line. A default Carbon atom is associated to all bond lines which are not connected to any atom. To join any two bond lines (detection of double/triple bonds), they must be within close proximity of each other and must be parallel.

A connection table illustrating the connectivity details of all bonds and atoms is generated at this stage and bonds connecting the same atoms are combined to make double/triple bonds. Also, atoms connected to which end of wedge and dashed wedge bonds are also distinguished in the connection table.

This is followed by the interpretation of generic structures which are usually represented as structures associated with variable groups (R-groups).

Some ambiguous patterns are dealt with during various phases of the recognition procedure. Bridge bonds are handled using a set of rules which include the analysis of the environment, length, how collinear the probed bonds are and cycle membership of candidate crossing bonds (rules found in [53]).

Also, the proximity of vertical and horizontal lines is analysed since they can cause ambiguity in various situations. For example, if the letter "C"is on the right side of a vertical line, then the vertical line is interpreted as part of a "Cl" atom except if the line is part of a dashed bond.

Also, if a relatively large circle is found inside a cycle with centres of both the circle and the cycle are close enough, then the circle is interpreted as a aromatic.

The generated connection table is analysed further for some bond formations such as when a triple bond and a single bond are joined together and when a dashed line can be incorrectly interpreted. These are identified by analysing neighbouring bonds and the average bond length.

Despite being a commercial tool, CLiDE still suffers from some weaknesses. For instance, it attempts to identify various bonds and bond arrangements in a brute-force manner. Not only this, but vectorisation seems to produce incorrect results if a line has several spikes. Also, the use of Hough transform may lead to missing double and triple bonds especially when lines are too close. Additionally, it is not clear how the stereocentre is determined in case of dashed, bold and bold dashed bonds.

4.3 ChemOCR

This is the ChemOCR project [8, 7, 113]. It starts by removing the results of applying any anti-aliasing filter and continues to binarise the image by using a dynamic binarisation algorithm. After the image has been binarised, all connected components are identified.

The resulting connected components are then fed into a OCR engine which was designed specifically to process characters existing in chemical diagrams. Wavelet functions are used to extract moments and these are used as features. The classification is carried out using a support vector machine algorithm which can automatically add incorrectly identified components.

Any components which are not identified by the OCR are assumed to be parts of the molecule's body. A graph of vectors is generated by vectorising these components. Only one vector represents each line in the image and the resulting graph should contain the same number of edges and vertices as lines and joints in the input image.

Wedge bonds are identified by superimposing all vectors in the graph with the pixels that generated them in the original image. Some statistic is used to determine how each vector and its corresponding pixels register. Bonds of the form of short parallel lines are identified as unconnected vectors which are parallel and of a naturally fitting size.

Vectors that are not identified as dotted bonds (short-parallel lines) and are unconnected are examined by building a region of interest around them. This region is defined by dilating the bounding box of the vector by a factor of two. Any other vector that is located within the region of interest is examined. Double and triple bonds are identified as two parallel and three parallel vectors respectively. It is assumed that more than three parallel vectors constitute a dotted bond (short-parallel lines). Cross bonds (closed bridge bonds) are identified as vectors which intersect at the middle part of the vector.

The results of the OCR are analysed and the atoms and superatoms's names are formed. Grouping of atoms is carried out by dilating the bounding box of all characters by a factor of two and then linking characters if their dilated bounding boxes overlap or touch. The direction of grouping (horizontal or vertical) is determined by testing the centre points of linked bounding boxes. Also, a dictionary of valid atom and superatom names is used to solve any ambiguities resulting from mistakenly grouping too close atoms or superatoms.

After this, a graph is built by assigning the atoms and superatoms to vertices in the vector graph. This is done by calculating the centre of mass for all atoms and superatoms, finding their nearest vertex and binding them. A Carbon atom is assumed at any vertex that is not linked to any atom/superatom. This graph is used to build an SDF file.

After describing how ChemOCR works, one can list several points where it can be improved. An example of these is that the authors did not mention that they handle dashed bonds, wavy bonds, any of the two types of bridge bonds, embedded nodes and bold bonds.

4.4 OSRA

OSRA [38, 37], or Optical Structure Recognition Application, is an open source tool which is freely available. OSRA's approach relies heavily on reusing existing open source tools.

OSRA starts with grayscaling and binarising the input image. The input image is grayscaled by choosing the minimum colour band in every pixel as the grayscale value rather than calculating the average band. For binarisation, OSRA does not employ the use of adaptive thresholding and rather uses a global threshold.

Image segmentation is applied after this step. This is done to automatically clip the areas where molecule images are. Detecting the molecule image is achieved by calculating the ratio of foreground pixels to the total area of the rectangle containing the molecule image, calculating the aspect ratio of this rectangle, this rectangle does not overlap with any other rectangles containing structures, the dimensions of the rectangle are within a predefined range depending on the resolution of the image.

The procedure continues by estimating how noisy the image is. This is accomplished by counting the number of pixel segments which are linear and of length 2 pixels and 3 pixels, and then calculating their ratio. This ratio, called the noise factor, is then compared against a predefined threshold and the noisiness is estimated. If the image is found to be noisy, the GREYCstoration anisotropic smoothing library [2] is used to perform noise removal and smoothing. This step is then followed by thinning the image to unit width and this is done using the algorithm found in [28]. Currently, OSRA applies the smoothing and thinning procedures only to images of resolution 300 dpi.

To detect bonds and nodes, OSRA employs the Potrace library [90] to convert bitmap images to vector graphics which in turn are used to compute the positions of bonds and atoms. In more detail, the output of the potrace library, which is a set of control points of a Bezier curve, is used to locate atoms first and then the bonds are detected as the vectors linking the detected atoms.

To perform OCR, OSRA uses the open source tools GOCR [4] and OCRAD [3]. A size filter is used and all connected groups of Bezier curves resulting from the previous step are

input to the OCR tools. Verification of some characters and symbols according to their position is performed after saving all recognised characters and removing their corresponding Bezier curves from the list created in the previous step.

To detect aromatic rings, OSRA looks for relatively large circles inside of rings. The ring atoms must be close enough to the circle and the angles between the ring bonds and the vectors to the centre of the circle must be less than 90. However, if the ring bonds and the circle touch, the process of detection fails.

Also, to measure the average bond length, OSRA repeatedly builds a sorted list of all bond lengths and then chooses the value at 0.75 by rank. The average bond width and the distance within double bonds are measured in a similar way. Double and triple bonds are detected by making sure they are parallel, they are within the measured distance and that they are within each other's "shadow".

Wedge bond detection is performed by using a least-square estimate of thickness against position within the bond and then testing for any increase/decrease along the bond. Additionally, dashed bonds are detected as relatively small shapes where a straight line can be drawn through their geometric centres and that they are positioned within the average bond length. OSRA conditions that there must be at least three such shapes in the same dashed bond.

To detect closed-bridge bonds, OSRA finds an atom which is connected to four straight lines. When looked at, these four lines form two longer straight lines. It checks whether the atom satisfies a number of conditions and performs disambiguation accordingly.

An input image to OSRA can have three different outputs because OSRA tries to handle the input image at three different resolutions. A certain confidence measure is calculated to choose the most likely solution.

Like other existing approaches, OSRA has its drawbacks. For instance, the authors do not mention that it handles wavy bonds, bold bonds and dashed bonds. Additionally, OSRA follows the usual procedural brute-force approach in identifying various architectures.

4.5 ChemReader

Chemreader [78, 79] starts by attempting to resize and denoise the input image. Resizing is performed using GREYCstoration [2], and the freely available algorithm found at [103]. The purpose of resizing is to adjust the character sizes and bond lengths in order to suit the recognition algorithms employed by ChemReader.

It continues by finding the connected components of the image using the 8-connectivity algorithm. These components are then classified into either graphics components or characters. Characters are detected as components with similar heights and areas as explained in [39]. Also, characters touching graphic components (lines) are segmented using the method found in [101]. This is followed by checking the location and horizontal/vertical run profile of components which cause disambiguity such as the letter ”l“ which is often wrongly classified as a vertical line.

To detect straight-lines, ChemReader uses the modified HT found in [111]. In this approach, pairs of pixels can selected randomly and assigned a weight which is based on the probability that the two pixels belong to the same line-segment. Unrecognised characters can be identified as short line segments and, therefore, these short line segments are re-checked.

Double and triple bonds are detected as two and three close enough lines respectively. In low resolution images, double and triple bonds wrongly identified as single bonds are processed by checking their thickness and by checking the black runs orthogonal to the bond.

To detect Wedge bonds, ChemReader lists corners using [95] and then checks all possible combinations of three corner points. It detects the wedge bond by checking whether the area of the triangle formed by the 3 points equals the number of foreground pixels in the triangle, checking whether the triangle has two equal sides and by splitting the triangle into three smaller areas and comparing their foreground pixel counts.

To detect dashed bonds, unprocessed components are checked by applying their centres to a conventional Hough Transform. A line which is perpendicular to a dashed line segment can be detected.

The approach ChemReader uses to detect circles in aromatic rings is to find a connected component whose centre is almost the same distance from all its pixels. A threshold is used to check the standard deviation of pixel distances from the centre of a candidate component. The Generalised Hough Transformation [11] is used to detect pentagon and hexagon shapes which can be wrongly identified as circles.

To perform OCR, ChemReader uses the open source library GOCR [4]. GOCR uses template matching of feature holes at various positions, density of sub-regions and back-ground to fore-ground transitions. It is mentioned that some components including GOCR are going to be removed and replaced with other tools. In addition to GOCR, a total of ten candidate characters and their confidence scores are associated to every input character. This is achieved by using a distance based OCR algorithm.

This is followed by a dictionary based chemical spell checking after grouping the identified characters according to their neighbourhood. Given a chemical word and its confidence score and the dictionary, the checker attempts to find the likeliest chemical word by using a statistical measure.

A graph based on the extracted information is built to represent the chemical structure. Nodes within a preset distance from each other are combined into one node. All nodes are assigned labels from the recognised symbols using Carbon as default for nodes with no symbol. This graph is used to generate a table illustrating connectivity information which can be used to output a conventional format.

Despite reporting encouraging results, ChemReader seems to employ the usual procedural approach as in other existing tools. The main drawback of the procedural approach is that it can be difficult to extend and it has to perform certain checks even when they are not needed. Not only this, but in terms of identifying bonds, the authors do not mention that they identify any of the two types of bridge bonds or how they spot embedded nodes.

4.6 ChemInfty

ChemInfty [67, 55, 40] is developed by a research group based in Japan. The group is known for their InftyReader [97] project. Although some publications about ChemInfty have appeared in the literature, the authors do not specify how exactly they identify and distinguish different bond formations. In [40] they made an attempt to tackle touching components in chemical diagrams using dynamic programming [42], and in [40] an attempt to build a graph representing a given chemical diagram and take advantage of a human user to make necessary corrections is made.

Nevertheless, ChemInfty's recognition procedure can be summarised as follows: the input image is binarised and smoothed first. This is followed by identifying and removing characters and caption areas (recognising characters is carried out using their InftyReader OCR engine). After that, the remaining part of the image is thinned to unit width, and crossing and bending points are detected. An attempt to recognise and remove bonds is made here before attempting to group the remaining image segments and searching for each segment group's most suitable combination. The search process is responsible for spotting and identifying any characters that may have been part of touching components.

4.7 Imago

Imago [93] starts by using a preset threshold value to binarise the input image and then computes the image's connected components. An attempt to identify dashed wedge and dashed bold bonds is then made to avoid having the short line segments affecting the results of next steps (character classification and others).

The stereocentres of any dashed bold bonds are also identified, although the procedure to identify them was not revealed in [93]. This is followed by classifying all the remaining connected components into two categories, atom labels and bonds. This is carried out by using an approximate height of capital letters that may exist in the input molecule image. A certain procedure is used to calculate this value from the input image and any connected component's

height is compared against this height value and determined to be either a symbol or a graphic (bond). Single lines which may have a similar height to that of a capital letter are handled as exceptions to the criterion.

All the resulting atom labels (symbols) are then grouped to form atom labels and superatoms.

The remaining components, which are the bonds, are used to create a skeleton molecule graph. This is done by thinning the graphics part of the image such that their width is one and then removing any foreground pixel that has more than two foreground neighbours. Douglas-Peucker algorithm [32] is used to simplify the resulting polylines. A graph is created using the simplified lines and double and triple bonds are identified as close and parallel edges. The graph is updated by merging nodes that are very close to each other or that are very close to the same atom label.

An attempt to identify solid wedge bonds is made after that step although the authors did not reveal how exactly this is carried out. According to the paper, their approach depends on the result of thinning the solid wedge, an example image containing a solid wedge was given with its thinned version but the result of the thinned wedge looks exactly like a normal line.

Superatoms are then replaced by their corresponding predefined sub-graphs and the resulting molecule graph is used to output a MOL file.

Imago is one of the recent chemical diagram recognition tools, it is not mentioned in [93] how it recognises any of the two types of bridge bonds, dashed bonds or wavy bonds. Additionally, the authors did not say whether they attempt to spot embedded nodes.

4.8 AsteriX Server

AsteriX [62] is a Web server that was designed to process PDF files containing chemical diagrams. It uses Xpdf suite of programs to decompose the input pdf file and extract text and pictures. Pictures are classified as chemical diagrams, or non-chemical diagrams. Subsequently, the detected chemical diagrams are analysed further to identify and extract 2-dimensional coordinates of components.

The process of deciding whether a given picture contains a chemical diagram (molecular structure) or something else is heuristic. The suspect image is preprocessed by connecting all components that are within a certain distance of each other (a window is centred at a foreground pixel and any components within the window are linked to the foreground pixel's parent component), the resulting image is denoted a "patch". Also, likely bonds and atoms are obtained as the connected components of the original image, atoms and bonds are denoted "motifs".

Now, a list of seven conditions is used to make decisions (to decide whether the image at hand contains a chemical diagram or not). The list includes a size filter for patches, a certain minimum and maximum number of motifs in a patch, a size filter and a certain length of vertical or horizontal lines (length compared to the patch size), one of the motifs must be one of the most common atoms used in chemical diagrams (C,N,O,...), distribution of lengths of straight line segments and distribution of values of angles between connected straight line segments.

After a decision is made, an attempt is made to identify bonds and atoms from pictures deemed as molecular structures. The process includes identifying bonds, bond connection points and atoms (an attempt to identify atoms glued to endpoints of bonds is also made here). The identification process is based on the assumption that pixel density is usually high at connection points (locations where bonds are connected to each other, or where bonds are connected to atoms). Whenever such a location is found, a template matching technique with a size filter is used to identify possible characters.

At this stage, the connectivity information of bonds and atoms is calculated to determine which bonds are connected to each other and which atoms are associated with which bonds.

It must be noticed that the authors do not mention how they identify out-of-plane and into-plane bonds (bonds of type: solid-wedge, dashed-wedge, bold, dashed-bold and dashed). Additionally, they do not mention how they identify embedded nodes as they reported only that double and triple bonds are detected when atoms are connected by two or three lines respectively. The approach continues by trying to identify aromaticity via matching centres of any detected five, or six, sided cycles with those of any identified circles or ellipses.

The final stages include handling other details such as disambiguating vertical lines and capital “I”letters, dealing with superatoms and atom charges represented by plus or minus signs. It should be noticed that the authors do not explain how they deal with such details.

The obtained connectivity information is used to construct an SDF file which contains 2-dimensional information. This file is then fed into the ChemAxon [21] software to generate 3-dimensional coordinates.

The Asterix server is the latest tool to recognise chemical structure diagrams during the time of writing this thesis. However, the approach seems to heavily rely on template matching and the authors did not mention how they deal with several bond types and formations. These include all of the 3-dimensional bonds, both types of bridge bonds and embedded bonds.

4.9 Chemink

This online interpretation of hand sketched chemical diagrams [74, 75] works by first grouping ink strokes into clusters that represent symbols (ink parsing) and then identifying the symbols represented by these individual clusters (symbol recognition).

A symbol recogniser is used to identify symbols represented by all combinations of up to seven consecutive strokes. Chemical knowledge is used to verify the results and resolve any inconsistencies.

Real-time feedback and update of interpretation are provided since the system performs this process as the structure is being drawn.

The identification of symbols is carried out using a support vector machine based classifier. The classifier was trained on valid symbols such as straight bonds, wedge bonds, hash bonds and atoms represented as sets of strokes. The features used included the number of strokes typically required to draw symbols, dimensions of bounding boxes, the density of ink and the distance between strokes in the cluster (group).

The average length of a straight bond is estimated and used in normalising the features used for symbol identification. This step makes the recogniser more insensitive to scale differences that may exist due to variations in drawing styles.

Two independent recognisers are used to perform text recognition. A handwriting recognition system provided by Microsoft is used in combination with a template matching distance based OCR engine.

The environment around all symbols is checked and some domain-specific rules are applied. These include constraints such as all atoms must be connected to other atoms or bonds (no unconnected atoms), digits next to elements should always appear as subscripts and others. Strokes representing multiple connected bonds are segmented using a poly-line approximation algorithm.

Symbols that share one stroke or more are resolved by comparing their recognition and environment scores. Candidates which exhibit the highest improvement in score are chosen. Other scores are saved for later re-interpretation if needed.

Omitted carbon atoms are assumed where appropriate and associated to any nodes which have no element. To fill any vacancies in their valence shells, hydrogen atoms are also assumed to be present at unsatisfied atoms (atoms are satisfied when the number of connections they have equals their valence).

Also, circles used to indicate the existence of aromatic rings are distinguished from oxygen atoms by making sure they are located inside 6-carbon cycles.

The next step is to check the structure for its chemical significance. Elements are checked to make sure they have the correct number of connections. Any inconsistencies arising from incorrectly recognising elements are resolved by reconsidering the scores saved during the previous phases. The same approach is employed to correct mistakes in connectivity and in symbol identification.

ChemInk attempts to recognise hand-drawn chemical structures. However, the authors did not mention how it spots several bond types and arrangements such as wavy bonds, bold bonds or any of the two types of bridge bonds.

4.10 Summary

A detailed overview of how existing chemical diagram recognition approaches work has been presented in this chapter. The reader can notice that some approaches do not state how they deal with certain bond formations. For example, approaches explained in Sections 4.4, 4.3 and 4.7 do not mention whether or not they handle wavy bonds, let alone how they identify them. The same applies for some of the approaches explained in the previous Sections when dative (arrow) bonds, bold bonds, dashed bold bonds and dashed wedge bonds are concerned. In fact, the majority of existing approaches do not mention how they spot embedded nodes and none of them state that they handle hollow wedge bonds.

However, the careful reader should notice that the main drawback of all existing approaches is the fact that none of them provides a precise definition for each bond convention and formation. Another disadvantage of the above mentioned approaches is that their recognition process seem to be too procedural. What this means is that all structures must go through the exact same step even if some of the recognition steps are unnecessary.

Hence, this is what this Thesis provides. It defines every bond convention and formation in a precise and strict manner such that no two or more bond conventions are mistakenly identified. This is advantageous over all existing approaches because not only does it provide a strict and unambiguous definition for different bond types and formations, but it also allows for non-predetermined execution of bond recognition steps since they can be represented as combinations of predicates. This leads to improvement in recognition accuracy as well as execution speed as will be shown in the remaining chapters.

Part II

Rule-based Recognition System

CHAPTER 5

SPECIFICATIONS

In this chapter we shall provide a high level overview of our system as a whole. We will begin by explaining the overall philosophy of our rule-based recognition system and then provide information about its architecture. After that, we will briefly discuss our implementation approach which resulted in a tool that we have used for validation and evaluation. We will refer to this tool as MolRec in the remainder of this thesis. Then, we shall discuss why the rule-based approach is favourable when compared to the traditional procedural approach. Following this discussion, we are going to provide information about our testing and validation methodology.

5.1 The Overall Philosophy

We have explained how existing approaches work in Chapter 4 and we have mentioned that all of them employ the traditional procedural approach. Each of the existing systems follows a certain list of steps every time an attempt is made to identify various bond types. However, we have chosen to use a rule-based approach to address the problem of automatic recognition of chemical structure diagrams. While we explain why our approach is advantageous over the procedural approach in Section 5.4, we shall explain what rule-based systems are in this section.

According to [13], rule-based systems are used for modular representation of human expert's knowledge. They use rules, in the form of condition-action pairs, to encode that knowledge into an automated system. They facilitate ease of understanding even when the amount of knowledge being represented is large.

Typically, a rule-based system consists of three components. Namely, A *Rule Base*, a *Rule Engine* and a *Working Memory* [84]. The rule base contains all of the relevant knowledge encoded as rules which contain condition-action pairs [12]. It encapsulates all actions that should be taken when conditions are satisfied. The rule engine, also denoted inference engine, interprets these rules and applies them whenever the conditions of a certain rule are met. The working memory contains data that is used by the rule engine to activate rules. The rule engine examines the working memory for the presence, or absence, of data elements and attempts to apply rules accordingly [51].

Notice that this approach differs from the commonly known procedural approach in that the

order in which rules are executed is not predetermined. To make this idea clearer, think of how one does not know which thread is going to execute next in a multi-threaded system. Similarly, in a rule-based system one does not know which rule is going to execute next. We will explain the advantages of this behaviour in Section 5.4.

As for our system, its philosophy is similar to that of equation rewrite systems [34]. As rewrite systems obtain the simplest form of a formula by repeatedly replacing its subterms [31], our system repeatedly rewrites one or more objects into graph edges until a termination criteria is satisfied.

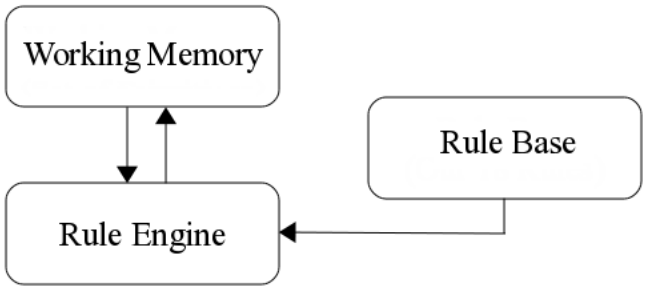


Figure 5.1: Components of a Rule-based System

We split our rule-based system across the three components we have mentioned previously. As Figure 5.1 shows, our system has a *Working Memory* which stores a set of geometric primitives, a *Rule Base* which stores our rules as functions that are of the form of condition-action pairs and a *Rule Engine* which interprets these rules, applies them on subsets of primitives and executes their corresponding actions. We shall demonstrate how these three components work by providing a fully worked example in Section 7.6.

Our working memory, which represents the input to our rule engine contains a set of geometric primitives. As we will explain in Section 7.2, these primitives are a result of a vectorisation step. Each of these primitives can be a character group, circle, line segment, solid triangle or an arrow.

5.2 System Architecture

Figure 5.2 illustrates the components of our system. An image containing a chemical structure diagram is input to our system where the processing starts by binarising the image,

performing optical character recognition and then vectorising the remaining components of the image. The result of this stage is a set of primitives which is used as input to our rule-based recognition engine. The rule-based recognition system attempts to identify bond arrangements and formations. When it terminates, a graph data structure representing the chemical structure is created. This graph is then post-processed and a MOL file is created.

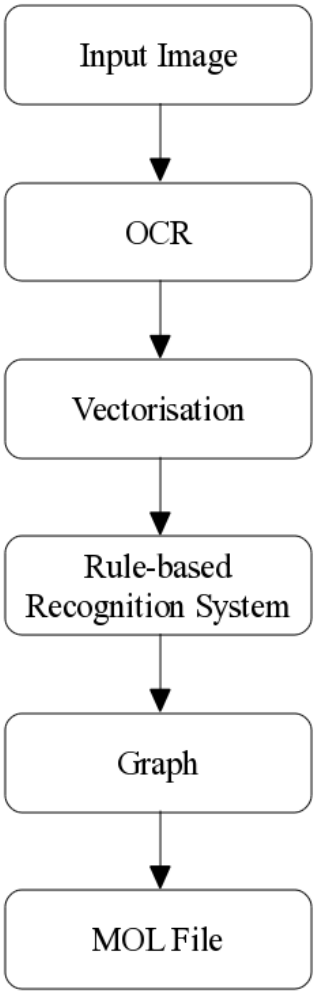


Figure 5.2: Components of Our System

The following is a summary of the purpose of every component of our system.

Optical character recognition (OCR): The first step in our approach is to binarise the image, perform optical character recognition (OCR) and remove recognised characters from the input image. For image binarisation we use Otsu's method [73] because it computes an optimal threshold value from the input image rather than using a static threshold.

As for the OCR, we apply a connected component labelling algorithm and extract a set of structural features from these connected components. We then apply a nearest neighbour classification based on a Euclidean metric to identify characters. We also detect circles at this step since their shape is similar to that of an oxygen atom label, i.e. "O". We then remove all connected components recognised as characters or circles from the image by turning their pixels into background. These characters and circles will form part of the input to our rule-based recognition system. Observe that we have provided a detailed explanation of our OCR method in Appendix A.

Vectorisation: After removing all the recognised characters and circles from the input image, we apply a thinning algorithm to the remaining components of the image which results in a new image where all the components have single pixel width. We then build a polyline representation of the thinned lines. Here we employ a novel adaptation of the Douglas-Peucker line simplification algorithm to determine line segment endpoints. At the same time we determine the average line width and average line length. We use a novel expanding disc technique to identify solid triangles and template matching to identify arrow lines. The result of this step is precise information about line segments, components in the shape of solid wedge and arrow lines. This information is added to the information obtained from the OCR step to complete the required input to our system. We are going to provide a detailed explanation of our vectorisation method in Chapter 6.

Rule-based Recognition System: As we will explain in Section 7.2.1, we create a set of five types of geometrical primitives from the results obtained from the two previous steps. These primitives are rewritten into graph edges by our rules and a graph data structure representing the input chemical structure is created.

As for our rules, we define them in terms of preconditions and consequences and we design them in such a way that they all have the same input and output format. A rule is applicable if there exist geometric objects that satisfy its preconditions (observe that we may refer to the preconditions of a rule as predicates in the remainder of this thesis). The consequence of each

rule is an effect on the input set of primitives which results in the removal of existing geometric objects and the creation of graph edges as well as possibly the addition of new geometric objects. In general, preconditions of different rules are mutually exclusive, and thus the order of rule application is minimised. Our rules work with a number of parameters, both fuzzy and strict, that set certain thresholds, for instance the minimal bond length, under which decisions will be made. These parameters allow for the customisation and adaptation to particular requirements of datasets. We will explain these parameters and provide more details about our rules in Chapter 7.

Graph: After the rule-based recognition system has terminated, we create a graph data structure to represent the input chemical structure diagram as we now have the set of edges. As we have explained in Section 3.1.1, the vertices of this graph represent nodes in the chemical diagram, whereas edges represent bonds. We post-process this graph to resolve superatoms and unknown stereocentres as we show in Section 8.1. We then use this graph to generate a MOL file.

MOL File: We have stated in Section 3.1.5 that MOL files are a common way to represent chemical structure diagrams electronically. As we detail in Section 8.5, we generate a MOL file from the graph data structure as it can be easily used for evaluation by comparing the resulting MOL file to a bench-marked MOL file.

5.3 Implementation Methodology

Our implementation of our rules is easy to understand. We implement each rule as a function that receives a set of primitives and returns a 3-tuple. The function attempts to apply its rule's predicates, have an effect on the set of primitives and then generate an output. As we have mentioned previously, we refer to the effect on the primitive set as consequence.

Inside each rule, or function, we apply the predicates in a waterfall style. In other words, the form of our predicate implementation is as follows:

$$P_1 \rightarrow P_2 \rightarrow \ldots P_n \Rightarrow C$$

We attempt to apply a predicate P_i and pass the data to the next predicate. If all predicates are applied successfully, we apply the consequence C.

To make the idea clearer, we show a pseudocode of a rule in Listing 5.1. Notice that for applying predicates, each function attempts to apply the first predicate on the input set of primitives. If the application of this predicate is successful, it removes the primitives that satisfy this condition from the primitive set and passes them on to the next predicate. This continues from one predicate to another until all preconditions are met and an output is generated. If the application of any predicate is unsuccessful, the function ends immediately without attempting the next predicates.

```
1  rule input =
2    Apply predicate1 to input
3    if application successful then
4      save primitive subset in X1
5      save remaining primitives in Y1
6      Apply predicate2 to X1
7      if application successful then
8        save primitive subset in X2
9        save remaining primitives in Y2
10                ...
11            Create set of graph edges and save in E
12            save Y1, Y2,... in Y
13            Return (success, E, Y)
14     endif
15     else
16       Return (fail, input, empty set)
17   endif
18   else
19      Return (fail, input, empty set)
```

Listing 5.1: Pseudo-code for Rule

Additionally, we place these functions, or rules, in a pool and randomly choose one of them. This rule is attempted on the set of primitives if it does not require any other rule to be attempted before it. Otherwise, we check whether that required rule has been attempted and continue accordingly. Also, we make sure that all rules have an equal chance of being selected by marking each selected rule such that it is not chosen again until all other rules have been chosen. This is going to prevent our system from entering an infinite loop as we explain further in Section 9.1.

Furthermore, we use an abstract data type to represent the primitives as well as represent

a graph edge so that all of our rules, and their predicates, have the same format of input and output. We read the primitives, store them in a data structure and use this data structure as the working memory. The idea here is for the rule engine to constantly access the working memory and examine if any primitive, or subset of primitives, satisfies any rule conditions. If this is true, the rule engine removes this subset of primitives from the working memory and generates corresponding graph edges when the rule executes. As we will explain in Section 7.1, this procedure should continuously run until the termination mechanism is met. Observe that we discuss our termination mechanism in Section 9.1.

5.4 Why Rule-based

Chapter 10 evaluates by comparison, two alternate strategies, rule-based and procedural. The procedural approach uses a mechanical or brute-force system that follows a strict sequence of steps each time (the steps are explained in Section 8.6). The rule-based approach is where steps become independent rules and application of rules is less rigid; order of execution of rules is minimised.

The results in Section 10.2 show that the two systems perform almost equally well. Following from this, it is left to justify why we adopt the system of rules over a traditional procedural approach.

The procedural based approach is clearly imperative, it requires statements on how to perform each action. So in a procedural approach, the action can only be deduced by following paths of the logic in the code. The rule-based approach is declarative: it allows us to write the system as clear statements of what is required. Therefore it becomes clear what the system is doing, allows us to prove aspects of the system and allows us to improve the system by adding support for new constructs.

Consider that the declarative, rule-based system can become imperative by imposing constraints on the rule ordering. This is not ideal as adding new rules requires consideration not only of the guards on the rules, but also the order that is required, making the system unnecessarily fragile. In effect, such rules become a single algorithm, rather than just a set

of independent transformation rules. Not imposing a specific order on the rule execution, allows the developer to simply write the rules declaratively. In this case, only the rules that are applicable, as defined by the guards, will run. Adding a new rule to handle a new construct should not, therefore, cause side effects that cause pre-existing construct handling to break.

5.5 Testing and Evaluation Methodology

In order to evaluate our implementation, we are going to run the tool we have developed on several benchmark chemical structure diagram datasets. Since these datasets contain images of chemical structure diagrams with their corresponding MOL files, we are going to generate our MOL files and compare them against those of the datasets. In order to achieve this, we use OpenBabel [72], which is a freely available tool that provides the ability to compare different MOL files semantically while ignoring unimportant syntactic differences. It is important to notice that, given two MOL files as input, OpenBabel computes a standard International Chemical Identifier (InChI [50]) from each MOL file and checks whether the two InChI keys match. The method we use here is Precision and Recall [82]. Observe that precision and recall are concerned with the relevance of retrieved information whereas in our case we only seek a match or mismatch measure. Therefore, we will only compute the percentage of matched (recalled) and mismatched structures.

5.6 Summary

We have provided specifications of our system in this chapter. We started the chapter by providing information about the philosophy and architecture of our rule-based system and followed that by briefly discussing our implementation methodology. After this discussion, we presented our justification for why our rule-based system is favourable when compared to the traditional procedural approach and we also discussed out testing and evaluation methodology.

Observe that the remaining chapters of this thesis are going to be as follows. Using Figure 5.2 as a reference, we have provided our method for performing optical character recognition in Appendix A. Chapter 6 will contain an explanation of our method to perform

vectorisation. Chapter 7 will provide a description of how our rules work. In it we will explain in detail a selection of four rules and provide a fully-worked example to illustrate how our system works. We explain the remaining rules in Appendix B. After that, we address several issues and aspects of our system in Chapter 9. The final two chapters of this thesis will contain experimental evaluation, conclusions and future work.

CHAPTER 6

VECTORISATION

In this chapter we are going to provide the necessary steps of our suggested mechanism to accurately identify the remaining three types of primitives. Namely, the line segments, solid triangles and arrows. Notice that we are going to assume that Optical Character Recognition has been carried out and all the characters and circles have been removed from the input image.

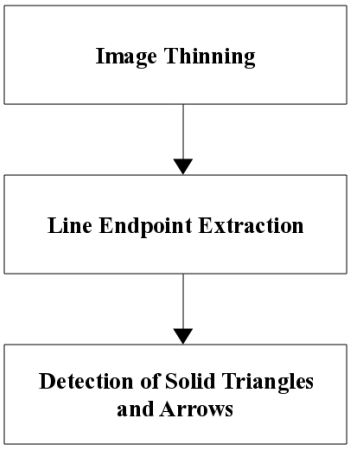


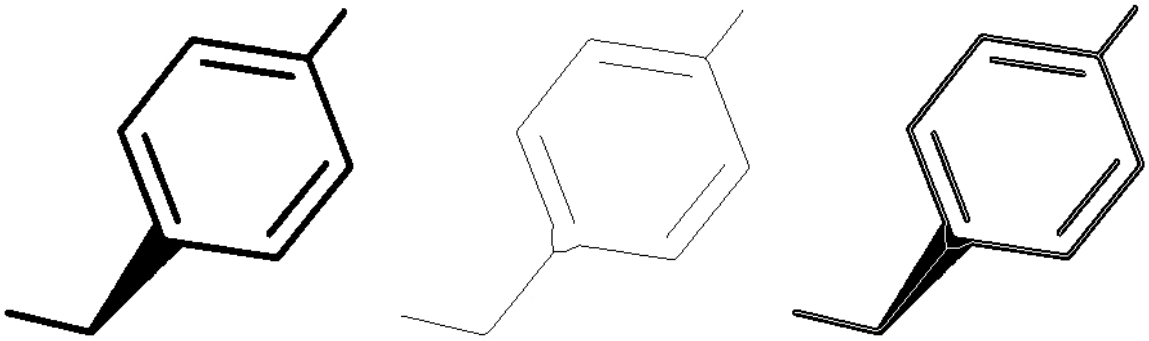
Figure 6.1: Vectorisation Steps

As can be seen in Figure 6.1, the steps start with thinning the image components into unit width thickness and then extracting precise line endpoints. As we will explain later in this chapter, we will perform the line endpoint extraction by using the Douglas-Peucker line simplification algorithm in a novel application. We are going to achieve this by tracing the thinned lines to find and disconnect multi-way junctions (this step may involve an attempt to smoothen the image before thinning it so possible spikes are removed) and then simplifying the resulting poly-lines to find precise line endpoints. Observe that, to our knowledge, this is the first time this algorithm is used to automatically determine endpoints of interconnected lines in images of chemical structure diagrams. We then follow this by detecting solid triangles and arrow lines. We detect solid triangles using a technique which we denote the *expanding disc* technique, and detect arrow lines using template matching.

6.1 Image Thinning

We start processing the input image by thinning its remaining components to unit width. Image thinning can be defined as the process of successively removing selected border pixels

from binary images. The result of thinning an image is another image where all lines are reduced to single pixel thickness. An example image is shown in Figure 6.2(a), its thinned version is shown in Figure 6.2(b). Notice that the thinning algorithm should erode away boundary pixels as much as possible while preserving the topology and connectivity of the original image. It should also preserve pixels at the ends of lines. Additionally, it should ensure that the thinning reduces the image components from all sides so the resulting components, when superimposed on the original components, can be in the middle (as shown in Figure 6.2(c)).



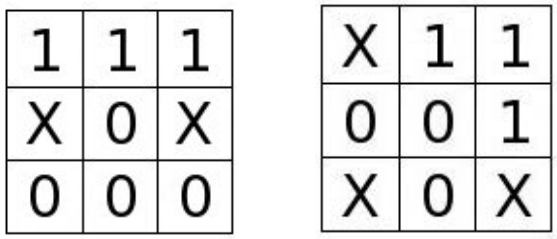
(a) Original Image (b) Thinned Image (c) Medial Axis

Figure 6.2: Image Thinning

Practically, thinning is performed by using a kernel (which is usually a small image on a rectangular grid) and translating its origin to all pixel positions in an image. The kernel, which is sometimes referred to as a structuring element, must always have its origin set to foreground (0 or black). The underlying image pixels are compared with elements of the structuring element (every image pixel is compared with the kernel element over it) and decisions are made accordingly. In more detail, the image pixel below the origin of the structuring element is set to background if the foreground and background pixels in the image match the foreground and background pixels in the structuring element. Choosing a structuring element usually dictates the results of a thinning process. Also, the choice of when to stop performing thinning is another important factor. While usually performing thinning repeatedly until convergence (i.e. until the image stops changing), applying it for a certain number of iterations is common in various applications such as pruning.

We provide two example thinning kernels in Figure 6.3. An input image can be iteratively

scanned using the shown kernels. The kernel shown in Figure 6.3(a) should be used first, the kernel shown in Figure 6.3(b) and then their six 90° rotated versions, making an overall $4 \times 2 = 8$ kernels. This procedure should be repeated in each iteration until none of the kernels causes the image to change. Notice that in the illustrated kernels, we assume a value of 0, or black for the foreground and a value of 1, or white for the background. We use X for the *don't care* case.



(a) (b)

Figure 6.3: Example Thinning Kernels

One of the very common approaches in raster-to-vector conversion of images is to compute the skeleton of the image, i.e. to thin the image to unit width. However, it is often argued that thinning has drawbacks. For example, some information is lost when thinning. Also, it is common to have spikes, or barbs, when the image is irregular or has rough edges. Additionally, line junctions and extremity points are displaced, resulting in unwanted positions as of those intended when generating the original image. These issues were discussed in more detail in [100].

We are going to use a number of solutions to resolve these drawbacks. For example, we are going to overlook the issue of information loss as long as the thinning process maintains the general shape of the image components. Also, feedback can be used to verify the shape of an object at later stages of a recognition, or analysis, process. In addition to this, we will carry out a post-processing step to resolve the issue of unwanted barbs resulting after thinning rough objects. In fact, a preprocessing step such as image blurring can also be used to smoothen the image prior to thinning it. Finally, the problem of junction and endpoint displacement can be easily tackled in many applications where connectivity of image objects is the main focus. For instance, in our context if two bonds in a molecule image are intended to be connected, displacement of their junction is unimportant as long as the result of a molecule recognition

process, that employs thinning, preserves their connectivity.

6.1.1 Types of Thinning Algorithms

We have defined image thinning as the iterative removal of border foreground pixels until a termination criteria is satisfied. Several conditions are usually checked whenever a foreground pixel is encountered and this pixel is deleted if it satisfies those conditions. We will now have a brief overview of two families of thinning algorithms.

6.1.1.1 Sequential Thinning

The concept of *sequential* thinning, as the name suggests, is that the boundary pixels (satisfying removal conditions) of an object in an image are removed sequentially. That is, the border pixels are checked for removal in a constant sequence in every iteration and they are removed one pixel at a time. For example, the removal of a pixel P in the i^{th} iteration depends on two things: it depends on the result of the $(i-1)^{th}$ iteration and on the pixels which have been processed already in the i^{th} iteration. An example sequential thinning algorithm can be found in [54]. An advantage of removing one pixel at a time is that it ensures topology preservation [58], [59].

6.1.1.2 Parallel Thinning

If border pixels are visited in different orders, sequential algorithm may extract different skeletons. This is not the case with *parallel* thinning algorithms [54]. A thinning algorithm is parallel if all the boundary pixels, meeting deletion conditions, are removed at the same time. This means that removing pixels in the i^{th} iteration only depends on the remaining pixels after the previous iteration (i.e. iteration $(i-1)$). Hence, the manner in which pixels are checked is parallel since they are checked independently in every iteration. Example parallel thinning algorithms can be found in [54] and [76].

6.1.2 The Thinning Algorithm we have used

We have implemented Guo and Hall's thinning algorithm [43] where the image is divided into two distinct subfields in a check-board manner and the algorithm runs in a two-subcycle fashion. Using Figure 6.4, a foreground pixel P_0 is erased or kept according to the following conditions:

1. Sub-iteration 1:

   (a) $X_H(P_0) = 1$ where

   $X_H(P_0) = \sum_{i=1}^{4} b_i$

   $$b_i = \begin{cases} 1 & if\ P_{2i-1} = 0\ AND\ (P_{2i} = 1\ or\ P_{2i+1} = 1) \\ 0 & otherwise \end{cases}$$

   (b) $2 \leq min(n_1(P_0), n_2(P_0)) \leq 3$ where

   $$n_1(P_0) = \sum_{k=1}^{4} P_{2k-1} \vee P_{2k}$$

   and

   $$n_2(P_0) = \sum_{k=1}^{4} P_{2k} \vee P_{2k+1}$$

   (c) $(P_2 \vee P_3 \vee \bar{P_8}) \wedge P_1 = 0$

2. Sub-iteration 2:

   (a) Condition a from sub-iteration 1

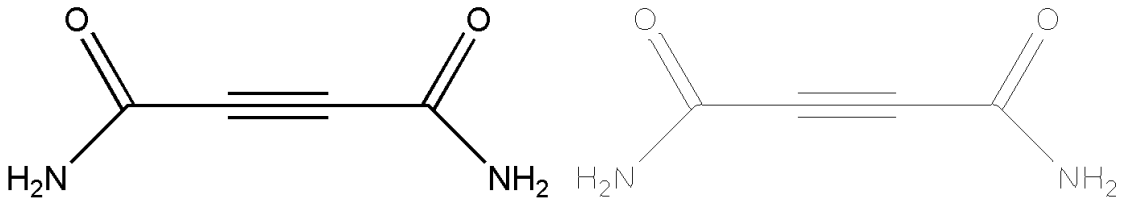
   (b) Condition b from sub-iteration 1

   (c) $(P_6 \vee P_7 \vee \bar{P_4}) \wedge P_5 = 0$

Notice that a single iteration consists of the two sub-iterations. And as we have mentioned previously, to thin the input image to a unit width thickness, the iterations should be run until the resulting image does not change any more. As shown in Figure 6.5(b), we have chosen to use this algorithm because of its robustness and reliable performance.

P4	P3	P2
P5	P0	P1
P6	P7	P8

Figure 6.4: Example 8-Neighbourhood



(a) Original Image (b) Thinning Results

Figure 6.5: Thinning using Guo and Hall's Thinning Algorithm

6.2 Extraction of Precise Line Endpoints

After the image components, (which are a group of connected straight lines), have been thinned, we will now introduce a way for precisely extracting line endpoints. The method we are going to employ is based on traversing the thinned lines and detecting junctions and extremities. Notice that by a junction, we mean the connection point of two or more lines, and by extremities we mean line endpoints which are not connected to any other lines.

Now, we are going to scan the thinned version of the input image from top left until we find a foreground pixel P. As illustrated in Figure 6.6, whenever we find a foreground pixel, we will search its 3×3 neighbourhood in an anti-clockwise direction. We will traverse all connected foreground pixels in this neighbourhood and save their coordinates. We will repeat this process for all connected foreground pixels and we will form the input polylines to the Douglas-Peucker line simplification algorithm. Observe that applying the Douglas-Peucker line simplification algorithm in this domain is a new application. However, before applying the line simplification, we will split these polylines into sub-polylines at various points.

Figure 6.7 shows examples of where we are going to split polylines. As the figure illustrates,

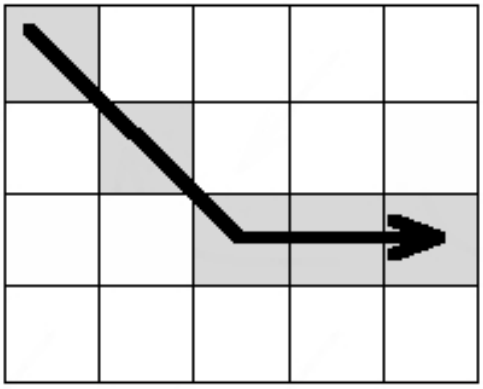


Figure 6.6: Foreground Pixel Tracing

point 1 has only one neighbour, point 2 has four and point 3 has two. As the traversal progresses, the neighbourhood of foreground pixels is checked and the lines are split if one of the following conditions is met:

- If a pixel has only one neighbour and that neighbour has already been visited (this means the pixel is an extremity or it is at the tip of a line).
- If a pixel has more than two neighbours (this means the pixel is at a multi-way junction of more than two lines)
- If a pixel has two neighbours and both neighbours have been visited (this means the pixel is at the end of a cycle)

As we said before, we will save coordinates of all visited pixels as sub-polylines and continue the process until coordinates of all foreground pixels have been processed.

After this step, we are going to detect corners, junctions and free line endpoints by applying a line simplification algorithm to the coordinate points. The algorithm that we are going to use in the context of this work is the Douglas-Peucker algorithm [32].

6.2.1 How Douglas-Peucker Algorithm works

This algorithm is usually used for line simplification algorithms (see Figure 6.8). As the algorithm shown in Appendix C shows, it starts with a list of points that represent the original

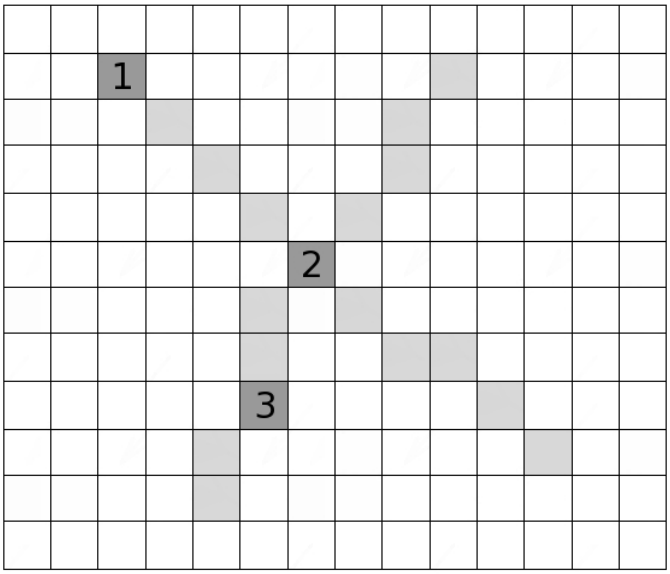


Figure 6.7: Example connected pixels and their neighbours

line and the distance dimension $\epsilon > 0$. Observe that choosing a suitable value of ϵ is application dependant and in our application, we have chosen a value equal to an average line thickness which we compute using the expanding disc as we are going to see in Section 6.3.

The algorithm splits the line recursively and automatically marks the first points and last point to be kept. It then finds the point that is furthest from the line segment formed by the first points and last point. The distance between points and lines is measured using Equation 6.1. Any point whose distance from this line is less than ϵ and that is not currently marked to be kept is discarded. And any point whose distance from this line is equal to or more than ϵ is kept. This is repeated recursively with the first point and the furthest point, and then with the furthest point and the last point. When it completes, a new representation of the line (containing only the points that have been marked to be kept) can be obtained.

Given the coordinates of a straight line segment (x_1, y_1) and (x_2, y_2), and a point (x_0, y_0), we can calculate the shortest distance between them as.

$$d = \frac{|\,(x_2 - x_1)\,(y_1 - y_0) - (x_1 - x_0)\,(y_2 - y_1)\,|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}. \tag{6.1}$$

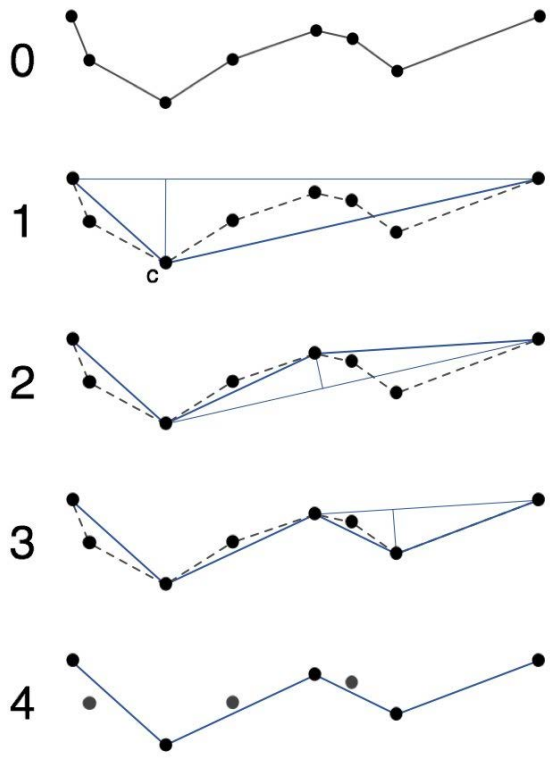


Figure 6.8: How Douglas Peucker Algorithm works (image from [109])

6.2.2 Barb (Spike) Removal

However, if the input image is irregular (i.e. it has rough edges), parasitic components can result after the thinning operation. These components are usually in the form of barbs, or spikes as we have shown in Figure 6.9. Therefore, when the traversal process finishes, these barbs will be considered as valid line segments despite being very short and often causing longer line segments to be split into two or more shorter line segments. We have to handle these barbs correctly in order to obtain precise results. The process to minimise the effect of these components is often denoted *pruning*.

Usually, a pruning algorithm erases all barbs (branches) which are shorter than a predefined number of pixels. The algorithm recursively removes a given number of pixels from each branch, starting at the free endpoints. This step is usually followed by dilating the resulting image with a structuring element (kernel) of a given size and then intersecting the new image with the original image.

We are going to deal with spikes after finalising the thinning and traversal operations. We are going to detect them as very short line segments (usually not longer than the average line width) that are perpendicular, and connected to other straight line segments.

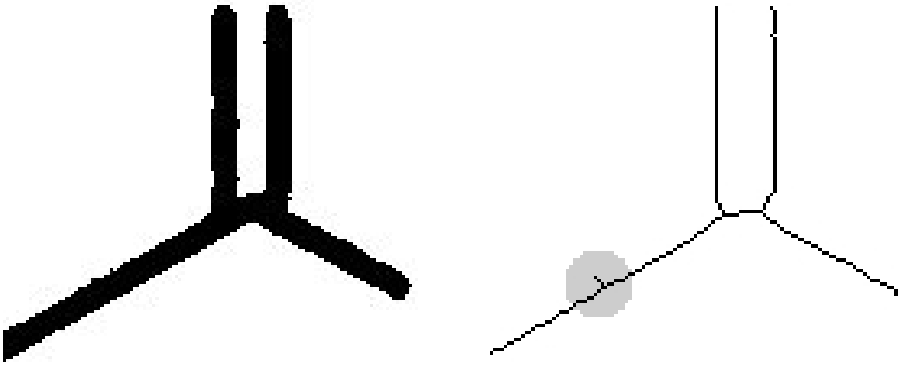


Figure 6.9: Spike resulting after thinning of rough line (shadowed)

If we detect such line segments, we can exclude them from the resulting set of line segments and merge the longer line segments that they split. But, it is obvious that this is not going to successfully work if the image has too many spikes and we need another approach.

In order to obtain better results, we are going to use image *blurring* to smoothen the rough edges. We are going to blur the original (un-thinned) image, binarise it and thin it again. This process guarantees that the thinned image will have fewer barbs than the original image (in fact in many cases the new thinned image can be barb-free). The traversal operation can be performed again and more accurate results can be obtained.

Image blurring is converting a source image X to another image Y such that each pixel in Y is made up of a mixture of surrounding pixels from X. In other words, the destination image Y is generated by spreading over each pixel in the source image X and mixing it into its surrounding pixels. This is traditionally achieved via convolving a kernel over all foreground pixels in the source image.

We are going to use the kernel shown in Figure 6.10 to perform image blurring. We will use this kernel to perform the convolution operation by scanning the original image and superimposing the kernel on it such that the kernel's origin corresponds to all pixel positions of every foreground pixel in the image. We multiply the value of the corresponding pixels from

the kernel and the image pixels they are over and then add together all the results. The result of this addition will be the new pixel value.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 6.10: A Simple Blurring Kernel

As we mentioned previously, we binarise the image, thin it and repeat the traversal process to obtain a list of line endpoints. In the coming sections we are going to introduce our method for detecting solid triangles and arrow lines.

6.3 Detecting Solid Triangles and Bold Lines

We are going to use a new way to extract precise geometric information about bonds in the shape of solid triangles (solid wedges) and thick lines. We are going a disc of a radius larger than a previously measured average line thickness. This disc can fit inside the base of an average solid triangle. The idea is to walk along the thinned lines and use their points as centres of an expanding disc. We are going to have an assumption that thinned lines are on the middle of their original lines (we have illustrated this in Figure 6.2(c)).

At each point, we will grow the disc until it reaches the largest size possible while still covering only foreground pixels in the original image. Then we change the position of the disc to the next point and the process continues.

Since it is relatively straightforward to distinguish between a normal line (planar bond) and a bold line (this can be done by comparing the line thickness against the average line thickness), we are going to the focus here on distinguishing between solid triangles (solid wedge bonds) and thick lines (bold bonds).

To distinguish between solid triangles and thick lines, we are going to use the linear algebra model presented in Algorithm 1 to analyse the values of the radiuses of the discs. Whenever we

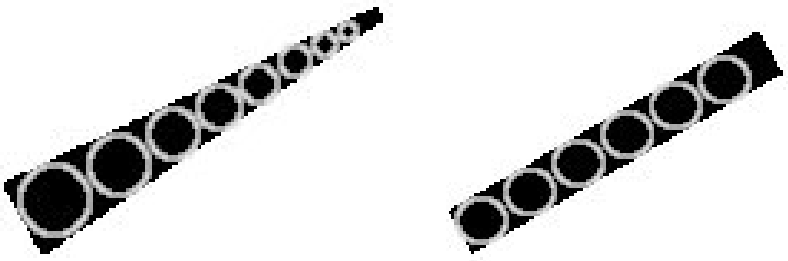


Figure 6.11: Detection of Solid Triangles and Thick Lines

detect a disc whose radius is larger than the average line width, we check the radius values over that line. If we find these values to be increasing or decreasing, then we identify the underlying line as a solid triangle, whereas if we find the values to be relatively stable, then we classify the line as a thick line.

Algorithm 1: Behaviour of Radius Values

Input: Threshold θ
Input: List of sequence of n radius values: $r_1, r_2, \ldots, r_n$
Output: Class of Behaviour c
begin
  Consider this sequence as a set of points in a 2-dimensional plane:
  $l = (1, r_1), (2, r_2), \ldots, (n, r_n)$
  Calculate Slope m of Line of Best Fit for points in l (Equation 6.2)
  if $|m| < \theta$ **then**
    return $c = ThickLine$
  end
  else
    if $m > 0.$ **then**
      return $c = SolidTriangle$
    end
    else
      return $c = SolidTriangle$
    end
  end
end

We are going to consider the sequence of radius values as a set of points in a 2-dimensional

plane and then calculate the slope (Equation 6.2) of line of best fit (the line that best describes these points). We are then going to use the absolute value of this slope to determine the line type.

$$Slope = \frac{n\sum_{i=1}^{n}(x_i y_i) - \big(\sum_{i=1}^{n} x_i\big)\big(\sum_{i=1}^{n} y_i\big)}{n\sum_{i=1}^{n}(x_i^2) - \big(\sum_{i=1}^{n} x_i\big)^2} \tag{6.2}$$

Where n is the number of line points,

$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ are the line points

$\sum xy$ = sum of products $= x_1y_1 + x_2y_2 + \cdots + x_ny_n$

$\sum x$ = sum of values of $x = x_1 + x_2 + \cdots + x_n$

$\sum y$ = sum of values of $y = y_1 + y_2 + \cdots + y_n$

$\sum x^2$ = sum of values of $x^2 = x_1^2 + x_2^2 + \cdots + x_n^2$

Now, to detect such bonds, we are going to use the following procedure:

Let us assume that a disc of a radius larger than the average line thickness is found. And let the sequence of radiuses over a thinned line is: $r_1, r_2, \ldots, r_n$. This sequence can be considered as a set of points in a 2-dimensional plane: $(1, r_1), (2, r_2), \ldots, (n, r_n)$. An example is shown in Figure 6.12, where the values are as illustrated in Table 6.1.

n	radius value
1	4
2	4
3	5
4	6
5	7
6	7
7	8
8	8
9	9

Table 6.1: Example Radius Values

Now, the line that best describes these points can be determined. Notice, this is the best fit line for this set of points.

The equation of a straight line is:

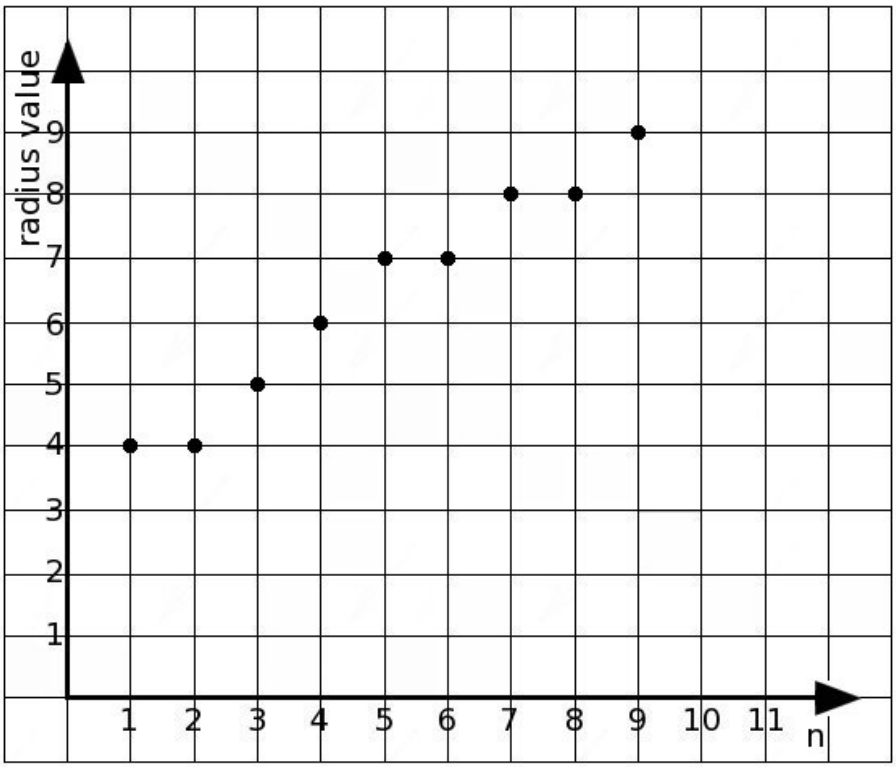


Figure 6.12: Sequence Behaviour

$$y = a * x + b \tag{6.3}$$

Since the equation of a straight line can be formulated by a slope, a, and intercept, b, the slope can be used to determine the line type as follows:

Given θ threshold (which can be determined empirically), if the absolute value of the slope is less than the threshold, then the sequence is *stable*. And, if the slope is positive and greater than the threshold, then the sequence is *increasing*. And, If the slope is negative and less then the negative of the threshold, then the sequence is *decreasing*.

The idea of using the expanding disc can help detecting the direction of the solid triangle since the base can be found at line end with the largest radius value regardless of whether the radius values are increasing or decreasing.

6.4 Detecting Arrows

As explained in Section 2.1.3, a one sided arrow line is used to depict a dative bond. This bond type is used to indicate the existence of a negatively charged atom at the arrow end of the bond. An example structure with dative bonds is provided in Figure 6.13.

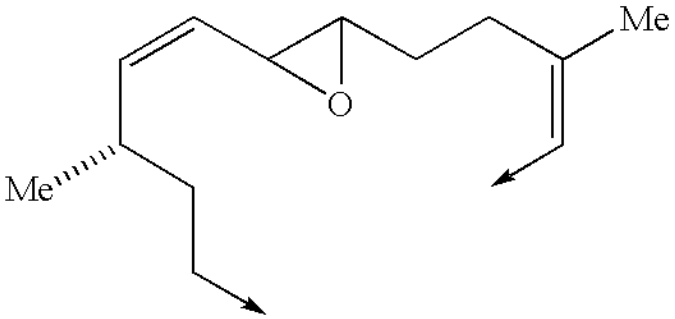


Figure 6.13: Example Structure with Dative Bonds

We are going to use *template matching* to detect arrow heads. Template matching is a technique used to search an image to find areas which are similar to a sub-image. The searched image and the sub-image are denoted the source image and template respectively. The search is done by moving the template in all directions one pixel at a time and comparing it to the source image [19]. To determine how well the match is, a metric that indicates that is computed at every location.

So the idea is to use a small image (i.e. a template) and move it to all possible positions in a larger image. Every time the template is moved, a numerical value that indicates how well the template matches the source image in that location is calculated. In our case, since line endpoints have been obtained, we will use a sub-image that contains an arrow head to search the proximity of these line endpoints in the original image [69]. Observe that we currently only use templates of the arrow type shown in Figure 6.13 and we search around endpoints of line segments which are longer than a pre-calculated average line length. We also use an empirically determined threshold to decide whether a given area of the source image contains an arrow head.

6.5 Summary

In this chapter we have provided a somewhat detailed overview of our suggested method to perform vectorisation of a chemical structure diagram image. The process assumes the image is binarised, and all characters have been removed by an OCR engine. The output of the vectorisation process introduced in this chapter is a list of geometric primitives, each of which can be a line segment, a solid triangle or an arrow. Along with the output of an OCR engine, which is explained in Appendix A, this complements the number of primitives required for our rules system and provides the necessary input for the rule engine. We are going to introduce the rules system in the next chapter. And in Chapter 8, we will discuss our way of forming a mathematical graph data structure representing the chemical structure diagram. This graph can be used to output a suitable textual or tabular format such as a standard MOL file.

CHAPTER 7

A DESCRIPTION OF THE RULES

We shall present a description of some of our rules in this chapter. As we have stated in earlier chapters, the rule-based recognition system consists of three parts. Namely, a rule base, a rule engine and a working memory. We have briefly discussed our working memory in Section 5.1 and we will define its contents, i.e. the geometric primitives, in Section 7.2.1. Our method for extracting these primitives has been explained in Appendix A and Chapter 6. In this chapter, we shall provide more detail about our implementation of the rule base and rule engine. We will also define the geometric primitives, explain some fundamental concepts and define several preliminary parameters which are necessary to understand how our system captures various bond types and architectures. After that, we are going to present a summary of our 18 rules and then explain in detail a selection of four of these rules. We will discuss the predicates of these rules, how they capture and rule out various patterns and provide sample OCAML code to show how we have implemented them. Observe that we are going to provide the remaining 14 rules in Appendix B. At the end of this chapter, we will provide a simple fully worked example to demonstrate how the recognition process works.

7.1 More on our Implementation of Rules

We have explained the structure of a typical rule-based system in Section 5.1. As for our rule base, it contains the implementation of our rules. Since rules contain conditions and actions, we implement each rule as a function that receives a set of primitives as input, combines the rule's corresponding conditions (or predicates) and generates a 3-tuple output. Similarly, we implement each predicate as a function that has the same format of input and output. Having the same input and output format for all predicates as well as the rules themselves is advantageous in the sense that it makes all functions consistent.

The first component of the 3-tuple output is a true/false flag that indicates whether the predicate, or the rule as a whole, applies successfully. The second component of the tuple contains a subset of the primitives set (the primitives that satisfy the predicate, which is passed on to the next primitive). In case of a rule's output, this component will have a set of graph edges created from the subset of primitives which meets all preconditions. The third component

contains the remaining primitives which do not satisfy a given rule's predicates.

We would like to bring to the attention of the reader that our implementation works in a way such that all bonds are captured at once by their relevant rule rather than capturing only one bond at a time.

As we have discussed before, rules are chosen and applied by our rule engine. Since rules are defined in terms of preconditions and consequences, a rule is applicable if there exist geometric objects that satisfy its preconditions. The consequence results in the removal of existing geometric objects and the creation of graph edges as well as possibly the addition of new geometric objects. In general, preconditions of different rules are mutually exclusive, and thus the order of rule application is minimised.

The overall procedure of the rule engine starts by initialising the working memory. We do this by reading the set of primitives into a suitable data structure. We then continue the procedure by placing the rules in a rule pool (which is a list of rules) and randomly choosing one of these rules. We check if the rule does not require any other rule to be attempted before it. If this is the case, then we apply the rule and execute its consequence. As we have mentioned in Section 5.3, we make sure that all rules have an equal chance of being chosen by marking each chosen rule such that it is not selected again until all other rules have been attempted.

As for executing predicates of rules, we attempt to apply the first predicate of a rule on the input set of primitives. If the application of this predicate is successful, it removes the primitives that satisfy this condition from the primitive set and passes them on to the next predicate. This continues from one predicate to another until all predicates are satisfied and an output is generated. If the application of any predicate is unsuccessful, the rule ends immediately without attempting the next predicates.

We apply the consequence of a rule by removing the primitive subset that meets all the conditions and create graph edges from those primitives. However, if the rule consequence involves the creation of new primitives (as in Rule 2 and Rule 3 which are explained in Appendix B.1 and Section 7.5.2 respectively), we add these primitives back to the primitive set. We repeat this selection and application procedure until the termination condition is met. After

that, create a full graph data structure representing the input structure and attempt to resolve any ambiguous graph edges that may result from Rule 4, Rule 5 or Rule 6. We then expand any existing superatoms and guess stereocentres of any dashed, bold or dashed bold bonds. We end the algorithm by generating a MOL file as we have shown in Section 8.1.

7.2 Preliminaries

After detailing the components of our rule-based system, we will now explain in detail our five types of geometric primitives which we use as input. Additionally, we shall define several concepts and parameters on which we base the rules. Observe that these concepts and parameters are going to be used in the implementation of the predicates.

7.2.1 Geometrical Primitives

The first assumption we are going to make is that the work will be with a restricted number of geometrical primitives that are the result of an initial vectorisation phase of the image. A set of these primitives will represent our *working memory*. We are going to distinguish the following five primitives:

Definition 3 ***Line Segment.*** *We will define a line segment l as a rectangle with measurable width and length, a unique centre point $c(l)$, and two endpoints $e_1(l), e_2(l)$ representing the middle points on the edge of the short sides of the rectangle.*

As Figure 7.1, it is not significant how the line segment's attributes are measured so long as they are measured accurately. The two endpoints are at the centre of the shorter edges, the centre point can be determined as the middle point of the line connecting the two endpoints, the line segment's length can be calculated as the length of the line connecting the two endpoints and so on. Observe that our

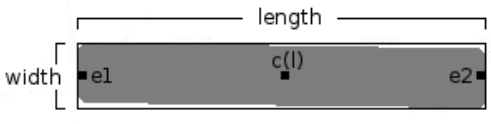


Figure 7.1: Line Segment with its measurable attributes

vectorisation method reduces the width of all line segments to unit width but we calculate the actual width of these line segments using the expanding disc as we explained in Chapter 6.

Notice that the vectorisation step should result in having all connected paths of line segments split into sets of simple line segments at their corners. In particular, any intersecting set of line segments should be split at the junction point. Thus line segments in the form of a sans serif "T" should be split into three separate but *connected* line segments, and in the form of a sans serif "V" into two line segments. We will explain the *connected* concept in the next section.

Definition 4 ***Arrow.*** *We define an arrow as a line segment with an arrow head as one end point (Figure 7.2).*

In other words, this primitive has exactly the same attributes as the line segment primitive with an extra characteristic. It has an arrow head as one of its points. As we will explain in more detail later, this arrow head gives this primitive a direction. This direction is important when determining the chemical significance of a bond of this form.

Figure 7.2: An Arrow Primitive

Definition 5 ***Solid Triangle.*** *We define a solid triangle as a solid wedge shaped with a discernible base line.*

Figure 7.3 illustrates an example solid triangle primitive. Notice that the triangle is filled (solid), and its tip and base are unambiguously distinguishable. Knowing the tip-to-base direction is vital for extracting the chemical significance of such objects.

Figure 7.3: A Triangle Primitive

Definition 6 ***Circle.*** *We define a circle as a ring consisting of the circumference (of some unspecified thickness) of a circle.*

In molecule diagrams this is found as the indicator of an aromatic ring inside multi-sided cycles (very often inside a pentagonal or hexagonal carbon cycle).

As shown in Figure 7.4, a circle has equal width and height. Its centre point can be calculated if necessary. Also, another important point to notice is that a circle should be distinguished from the letter "O" and the digit "0" as some fonts draw them as completely spherical. This is left to how the approach is implemented but some useful hints such using a size filter and examining the environment around suspect objects should be considered as we explained in Section A.5.

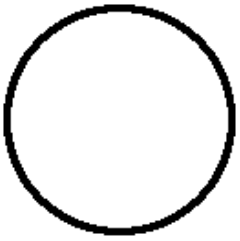


Figure 7.4: A Circle Primitive

Definition 7 ***Character Group.*** *We define it as a group of letters, digits or symbols. They have a small inter-distance and are considered one entity.*

Chargroup primitives can consist of a single character, two characters or multiple characters as shown in Figure 7.5. Characters can be letters, digits or symbols. The digits are usually used as subscripts and they indicate repetition of the character, or substructure, they follow. The grouping of characters is left to the implementation although, as we have explained in Section A.4.1, certain conditions based on rules-of-thumb must be applied in order to produce correct character groups.

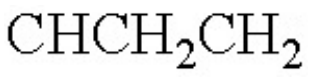


Figure 7.5: A Chargroup Primitive

7.2.2 Concepts

We need a number of general concepts to define the rules, based on parameters that can be instantiated later. We will design the rules so that they accommodate a certain level of approximation in attributes of components of the diagram such as slopes of lines, collinearity of sets of points and lengths of lines. This is due to possible imprecision in the drawing of the images and inevitable digitisation artifacts in images of chemical diagrams.

Therefore, we are going to use *fuzzy parameters* for this purpose. These are pairs of values that record the minimum and maximum allowable values for their attribute of interest. For instance, for a fuzzy parameter $f = [f_{min}, f_{max}]$ and a value v, we say that v is approximately f, or $v \in f$, if and only if v is between the minimum and maximum values allowed by f. We are going to use the following concepts for rule definitions:

Definition 8 ***Separation Distance.*** *The distance between two graphic objects defined as the closest distance between points of the different objects. For objects p and q, the separation distance will be abbreviated as $dist(p, q)$ in the remaining sections of this thesis.*

Definition 9 ***Connected.*** *Two line segments are said to be connected if they have exactly one endpoint in common.*

Definition 10 ***Dash-Neighbouring.*** *Let x be a fuzzy parameter. An element of a set of objects is said to* dash-neighbour *(wrt. x) another element of that set if the two elements are distinct and the closest distance between points of the different elements is approximately x.*

Definition 11 ***Approximate Collinearity.*** *Let x be a strict (i.e. non-fuzzy) parameter. A set of points is defined to be* approximately collinear *with respect to* Radius of Collinearity *x, if there is a line such that the orthogonal distances from the line to each of the points is less than or equal to x. A set of geometrical objects is said to be approximately collinear if the set of all points of the objects are approximately collinear.*

Definition 12 ***Approximately Parallel with separation x and overlap y, $\|_x^y$.*** *Let x, y be strict parameters. Let l_1, l_2 be line segments. Then $l_1 \|_x^y l_2$ if and only if*

(i) for the lines L_1, L_2 fixed by the end points of l_1, l_2, respectively, every point $p \in l_1$ has $dist(p, L_2) \leq x$ and every point $p \in l_2$ has $dist(p, L_1) \leq x$.

(ii) there exist line segments $l_1' \subseteq l_1$, $l_2' \subseteq l_2$ with $length(l_1') = length(l_2') = y$ and for every $p \in l_1'$, $dist(p, l_2') \leq x$, and for every $p \in l_2'$, $dist(p, l_1') \leq x$.

Observe that the definition of approximately parallel with separation and overlap ensures not only that line segments can vary slightly from being truly parallel and must have an approximate orthogonal separation from each other but also that the orthogonal projection of one onto the other must intersect over a length corresponding to the overlap parameter. This rules out collinear line segments as if they overlapped to that extent they would have been vectorised into a single line segment. In the remainder this concept will be referred to as approximately parallel, only.

7.2.3 Parameters

Now, we are going to define several of both strict (i.e. non-fuzzy) and fuzzy parameters. We are going to use these parameters in the rule system. Additionally, these are the parameters on which the concepts introduced in the previous section are based.

Radius of collinearity rc**.** A strict parameter used in the definition of *approximate collinearity*.

Dash length dl**.** A fuzzy parameter indicating the acceptable length of a single dash in a dashed line.

Dash separation ds**.** A fuzzy parameter indicating the acceptable distance between consecutive dashes in a dashed line. It is used for determining dash-neighbouring elements.

The three previous parameters are illustrated in Figure 7.6. Observe that it is required that rc is less than the minimum values of both dl and ds. This is to ensure strictness in the collinearity of dashes forming the same dashed line.

Figure 7.6: Radius of collinearity, Dash length and Dash separation

Bold dash length bdl**.** A fuzzy parameter indicating the acceptable length of a single line segment representing a bold dash in a dashed bold line.

Bold dash width bdw**.** A fuzzy parameter indicating the acceptable width of a single line segment representing a bold dash in a dashed bold line.

The previous two parameters are illustrated in Figure 7.7. Notice that bdl may have a similar

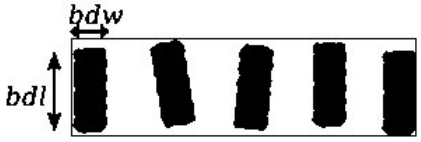
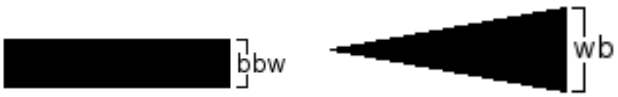


Figure 7.7: Bold dash length and Bold dash width

value to that of dl. The same applies to bdw and the width of a normal line. However, when examining existing chemical diagrams this can be true in some cases and untrue in other cases. Therefore, a separate and independent definition was given to each of them.

Bold bond width bbw**.** As shown in Figure 7.8(a), this is a strict parameter indicating the minimum acceptable width of a single line segment representing a bold bond.

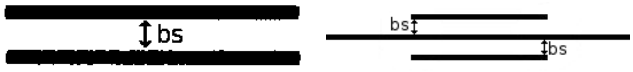
Wedge base wb**.** Illustrated in Figure 7.8(b), this is a fuzzy parameter indicating the acceptable width of a wedge base (it applies for solid, hollow and dashed wedges alike).



(a) Bold bond width (b) Wedge base

Figure 7.8: Bold bond width and Wedge base

Bond separation bs**.** As shown in figure 7.9, this is a strict parameter indicating the acceptable maximum distance between two parallel bonds.



(a) (b)

Figure 7.9: Bond separation

Minimal overlap ol**.** A strict parameter indicating the minimal distance necessary for two line segments to be considered overlapping. Notice that bs and ol are the parameters that generally instantiate the definition of approximately parallel, $\|_{bs}^{ol}$. It is also important to emphasise at this stage that we are only interested in overlap between approximately parallel line segments. Overlapping non-parallel line segments are of no interest because the architecture they form does not have any significance in the context of this work.

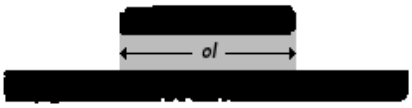


Figure 7.10: Parallel line overlap

7.3 Bond Cutting

Our system has 18 rules that capture various bond types and configurations. Our first three rules deal with regular single, double, and triple planar bonds. However, this is less straightforward than it appears, as these bonds can not only occur separately, as displayed in Figure 7.11(a)–7.11(c), but also can be given implicitly within single line segments.

Figure 7.12 demonstrates this idea, where there is a combination of single bonds with double and triple bonds, respectively. Notice that there is always an overlap between the shorter line segment and the longer line segment in Figure 7.12(a) and 7.12(b). Similarly, the overlap should always be there when there are two short line segments and a longer line segment as in Figure 7.12(c) and 7.12(d) as well as an overlap between the shorter two line segments. It is also essential to notice that the overlap value, can be equal to the length of the short line(s) as shown in Figure 7.10.

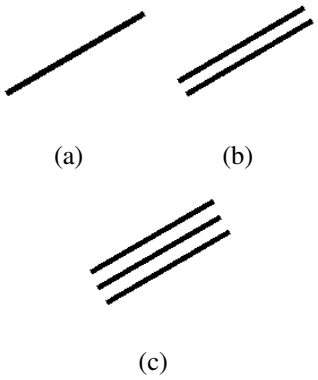


Figure 7.11: Single, Double and Triple Planar Bonds

From a chemical point of view, carbon atoms are understood to be at the shadowed areas separating the bonds (Figure 7.12). Thus we need a mechanism to split up these bonds to be used within the consequences of some rules. We will call this mechanism "*cutting*".

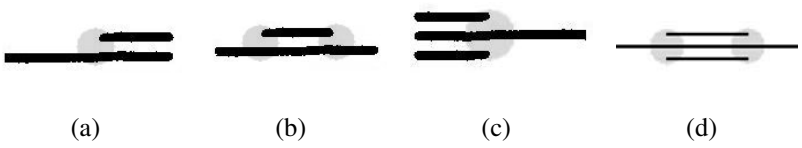


Figure 7.12: Implicit Nodes (shadowed)

Given a set of two or three approximately parallel line segments, a minimal bounding rectangle that contains them can be defined. Also, cut lines (Figure 7.13) orthogonal to the long axis of the rectangle at the end points of the component line segments are defined. Cut lines that are orthogonal and within rc of each other are unified because they are approximately the same. This is done to avoid cutting at the same place more than once (cut lines 1,3 and 2,4 in Figure 7.13).

Figure 7.13: Illustration of Cut Lines

Now the long line segment can be split at the cut lines into separate (collinear) line segments. It is important to notice that this is when new geometric primitives can be added to the working memory (i.e. our set of primitives which is used as input). Only rule R2 and rule R3 use cutting (Appendix B.1 and Section 7.5.2 respectively). If there were two line segments, there will be at most two cut lines and precisely one section corresponding to a double bond. With three line segments there will be at most five sections with precisely one corresponding to a triple bond.

7.4 Summary of Rules

Our current rule engine consists of 18 rules in total. As summarised in Table 7.1, they include rules to capture planar single, double and triple bonds as well as dashed, dashed bold and dashed wedge bonds. They also include rules to recognise two cases of hollow wedge bonds. Additionally, they contain rules that handle wavy, dative, solid wedge and bold bonds. Moreover, there are rules that identify aromatic bonds and two types of bridge bonds.

However, the rules not only cover these bond types, arrangements and configurations, but they also contain rules that deal with ambiguous cases when it is not straightforward to decide whether a pattern represents a dashed bold bond or a planar triple bond, whether a pattern represents a dashed wedge bond or a planar triple bond and whether a pattern represents a dashed wedge bond or a planar double bond. Table 7.1 provides a summary of these rules.

Rule	Summary
Rule 1	Captures planar single bonds and outputs graph edges each of which represents one of the captured single bonds.
Rule 2	Captures planar double bonds and applies cutting when necessary. It outputs graph edges each of which represents one of the captured double bonds.
Rule 3	Captures planar triple bonds and applies cutting when necessary. It outputs graph edges each of which represents one of the captured triple bonds.
Rule 4	Captures a special type of bond which can either a dashed bold bond or a planar triple bond. It outputs graph edges each of which represents this type of bonds. A disambiguation process is carried out after creating a full graph.
Rule 5	Captures a special type of bond which can either a dashed wedge bond or a planar triple bond. It outputs graph edges each of which represents this type of bonds. A disambiguation process is carried out after creating a full graph.
Rule 6	Captures a special type of bond which can either a dashed wedge bond or a planar double bond. It outputs graph edges each of which represents this type of bonds. A disambiguation process is carried out after creating a full graph.
Rule 7	Captures dashed bonds and outputs graph edges each of which represents one of the captured dashed bonds.
Rule 8	Captures dashed bold bonds and outputs graph edges each of which represents one of the captured dashed bold bonds.
Rule 9	Captures dashed wedge bonds and outputs graph edges each of which represents one of the captured dashed wedge bonds. This rule should be attempted before Rule 7.
Rule 10	Captures hollow wedge bonds whose base is not connected from the middle to other bonds. This rule is applied after a full graph has been built and merges edges.
Rule 11	Captures hollow wedge bonds whose base is connected from the middle to other bonds. This rule is applied after a full graph has been built and merges edges.
Rule 12	Captures wavy bonds and outputs graph edges each of which represents one of the captured wavy bonds.
Rule 13	Captures dative (arrow) bonds and outputs graph edges each of which represents one of the captured dative bonds.
Rule 14	Captures solid wedge bonds and outputs graph edges each of which represents one of the captured solid wedge bonds.
Rule 15	Captures bold bonds and outputs graph edges each of which represents one of the captured bold bonds.
Rule 16	Captures aromatic rings. This rule is applied after a full graph has been built as it changes the type of single bond edges forming aromatic cycles to edges representing aromatic bonds.
Rule 17	Captures open bridge bonds. This rule must be applied after R1 as it merges pairs of single bond edges forming open bridge bonds.
Rule 18	Captures closed bridge bonds. This rule is applied after a full graph has been build. It merges single bond edges which are identified to be forming closed bridge bonds.

Table 7.1: Textual Summary of our 18 Rules

7.5 Sample Rules

We are going to discuss a selection of four of our 18 rules in this section. Namely, we will discuss our rule to identify planar single bonds (R1), planar triple bonds (R3), dashed bonds (R7) and wavy bonds (R12). We have chosen these rules so that the reader can compare between R1-R3, and R7-R12 as they share several predicates.

The discussion will include providing the predicates of each of these rules as well as illustrating our OCAML code for each of them. As we mentioned previously, the remaining 14 rules are going to be provided in Appendix B.

7.5.1 R1: Planar Single Bond

The planar single bond is usually drawn as shown in Figure 7.14. It is a line segment with no parallel line segments within close proximity. It can connect either two atoms or atom groups, two nodes, or a node and an atom. We use the following conditions to capture a single planar bond:

1. l is a line segment.
2. $length(l) > wb$
3. $width(l) < bbw$
4. There is no line segment l' such that $l \parallel_{bs}^{ol} l'$.

Consequence: l is a single bond with endpoints of l

Figure 7.14: A Single Planar bond

This rule identifies a single line segment which has no parallel line segment(s) within bond separation. Observe that R1.2 guarantees that the rule does not capture a line segment that can

be a hollow wedge base (R10 and R11) and R1.3 guarantees that the rule does not capture a bold line (R15). Additionally, notice that R1.4 excludes R2 and R3 (Appendix B.1 and Section 7.5.2 respectively).

As for our implementation, we provide our code for R1 in Listing 7.1. To preserve space, we only show the code for the rule and not the predicates. As we mentioned in Section 5.3, the rule receives a list of primitives as input and returns a tuple containing whether it has identified any single bonds, a list of graph edges representing these bonds and a list of the remaining primitives.

```
1  let apply_r1 primitives =
2    match get_normal_lines primitives with
3      | true, all_normal_lines,other_primitives ->
4          (
5            match get_planar_singles (flatten all_normal_lines) with
6             | (true,all_singles,other_normal_lines) ->
7                 (
8                   let single_edges = create_single_edge all_singles in
9                   (true,single_edges,(other_primitives@other_normal_lines))
10                )
11            | (_,_,_) -> (false,[],primitives)
12         )
13     | (false,_,_) -> (false,[],primitives)
```

Listing 7.1: OCAML code for R1

Observe that the `get_normal_lines` function combines predicates R1.1, R1.2 and R1.3 whereas `get_planar_singles` represents predicate R1.4. It is worth mentioning here that it is these predicates who use the parameters we discussed in Section 7.2.3. Also, notice that graph edges representing planar single bonds are created using the function `create_single_edge`.

Additionally, notice that the rule returns the entire set of `primitives` if any of its predicates is not satisfied. This is evident in lines 11 and 13 of Listing 7.1. Also, notice that if the rule successfully executes, it returns all remaining primitives as shown in line 9 (`(other_primitives@other_normal_lines)`).

7.5.2 R3: Planar Triple Bond

As illustrated in Figure 7.15, the planar triple bond is usually drawn as three very close parallel line segments. Cutting can be applied when necessary as explained in Section 7.3. We

identify a triple planar bond if the following conditions are satisfied:

1. $L = \{l_1, l_2, l_3\}$ is a set of three line segments
2. $\forall l \in L : length(l) > wb$
3. x $\forall l \in L : width(l) < bbw$
4. $l_1 \parallel_{bs}^{ol} l_2$ and $l_2 \parallel_{bs}^{ol} l_3$ and $l_1 \parallel_{2*bs}^{ol} l_3$
5. There is no $l \notin L$ such that $l_1 \parallel_{bs}^{ol} l$ or $l_2 \parallel_{bs}^{ol} l$ or $l_3 \parallel_{bs}^{ol} l$

Consequence: Cutting(l_1, l_2, l_3) will yield a triple bond as well as at most four new line segments.

Figure 7.15: A Triple Planar bond

Notice that the two conditions involving $\parallel_{bs}^{ol}$ in each of rules R1, R2 and R3 ensure that the rules are mutually exclusive. This means that the two line segments forming the double bond will not be identified as two separate single bonds. Also, the three line segments forming the triple bond will not be considered as three separate single bonds nor will they be detected as a combination of a single and a double bond.

Notice also that R3.2 guarantees that the rule does not capture any line segment that can be a hollow wedge base (R10 and R11) and R3.3 guarantees that the rule does not capture any bold line (R15).

In addition to that, the rule ensures that a pattern such as the one illustrated in Figure 7.16 will not be identified as a triple bond. Rather, it will be identified as two double bonds (see B.1). This is because rule R3 requires that the two shorter lines are overlapping and approximately parallel.

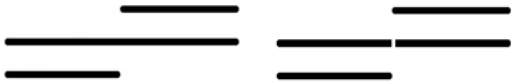


Figure 7.16: Cutting produces two double bonds rather than a triple bond

Observe that this rule is one of the two rules that may add new geometric primitives to the working memory (i.e. our set of primitives which is used as input). The other rule is R2 which is explained in Appendix B.1. As we illustrated in Figure 7.12 in Section 7.3, there are only two cases when cutting can be applied. The first case involves cutting the longer line segment only once which yields one triple bond and one new line segment (Figure 7.12(c)). And in the second case the longer line segment is cut twice which produces one triple bond and two new line segments (Figures 7.12(d)). If cutting is successfully applied, the resulting new line segment(s) will be added to the set of primitives and identified later by rule R1 as planar single bond(s).

As for our implementation, we provide our code for R3 in Listing 7.2 where we only show the code for the rule and not the predicates. As we mentioned in Section 5.3, the rule receives a list of primitives as input and returns a tuple of containing whether it has identified any bonds, a list of edges representing these bonds and a list of the remaining primitives.

```
let apply_r3 primitives =
  match get_normal_lines primitives with
    | (true,normal_lines,o_prmtvs) ->
        (
          match get_planar_triples (flatten normal_lines) with
            | (true,all_triples,o_lines2) ->
                (
                  match attempt_cutting3 all_triples with
                    | (true, tri_clusters,o_lines3) ->
                        let tri_edges = create_triple_edge tri_clusters in
                          (true,tri_edges,(o_prmtvs@o_lines2@(flatten
                             o_lines3)))
                    | (false,tri_clusters,_) ->
                        let tri_edges = create_triple_edge tri_clusters in
                          (true,tri_edges,(o_prmtvs@o_lines2))
                )
            | (false,_,_) -> (false,[],primitives)
        )
    | (false,_,_) -> (false,[],primitives)
```

Listing 7.2: OCAML code for R3

Observe that the `get_normal_lines` function combines predicates R3.1, R3.2 and R3.3 whereas `get_planar_triples` combines predicates R3.4 and R3.5. Observe also that cutting is attempted using the function `attempt_cutting3` after identifying patterns of three adjacent and parallel line segments. If this function finds a cuttable cluster of three line segments, it returns a modified list of three line segments (after being cut) and a list of resulting line segments which should be returned to the set of primitives. If the three line segment cluster does not require cutting, it returns the same cluster. Observe that in both cases graph edges representing planar triple bonds are created using the function `create_triple_edge`.

Additionally, notice that the rule returns the entire set of `primitives` if any of its predicates fails. This is evident in lines 16 and 18 of Listing 7.2. Also, notice that if the rule successfully executes, it returns all remaining primitives as shown in line 11 (`((o_prmtvs@o_lines2@(flatten o_lines3)))`) and line 13 (`((o_prmtvs@o_lines2))`).

7.5.3 R7: Dashed Bond

A dashed bond is usually drawn as short collinear line segments (Figures 7.6 and 7.17). The line segments are evenly spaced apart and their length is similar (their length is usually less than the length of a normal line segment that would be captured by R1, R2 or R3).

Figure 7.17: A Dashed Bond

We identify a dashed bond using the following conditions:

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments
2. $\forall l \in L : length(l) \in dl$ approximately dl
3. L is approximately collinear
4. No two elements of L have a separation distance of less than the minimum of ds

5. Two elements of L, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L. All other elements of L, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L

Consequence: L forms a dashed bond with endpoints given by the endpoints of the minimal line segment that contains $l_1, \ldots, l_n$. The new dashed bond has a down direction with unknown start and end.

Observe that R7.2 guarantees that the captured line segments will not be captured by any of R1, R2, R3, R10, R11, R16, R17 or R18. Also, R7.3 guarantees that the pattern is not captured by R8 and R7.4 guarantees that the pattern is not captured by R12.

```
let apply_r7 primitives =
  let left_overs = ref [] in
  let dashed_edges = ref [] in
  match get_dashes primitives with
    | true,all_dashes,other_lines ->
        ( left_overs := other_lines :: !left_overs;
          match create_col_dash_clusters (flatten all_dashes) with
            | (true,col_dashes,col_others) ->
                ( left_overs := col_others :: !left_overs;
                  iter (
                    fun dashes ->
                      (
                        match create_adjacent_dash_clusters dashes with
                          | (true,adj_dashes,adj_others) ->
                              (
                                let dashed_bonds = map (
                                  fun cls ->
                                    create_dashed_edge cls
                                ) adj_dashes in
                                  dashed_edges := dashed_bonds ::
                                     !dashed_edges;
                              )
                          | (false,_,_) ->
                              left_overs := dashes :: !left_overs;
                      )
                  ) col_dashes;
                  match length !dashed_edges with
                    | 0 -> (false,[],primitives)
                    | _ -> (true,(flatten !dashed_edges),(flatten
                       !left_overs))
                )
            | (false,_,_) -> (false,[],primitives) )
    | false,_,_ -> (false,[],primitives)
```

Listing 7.3: OCAML code for R7

As for our implementation, we provide our code for R7 in Listing 7.3 where we only show the code for the rule and not the predicates. As we mentioned in Section 5.3, the rule receives a list of primitives as input and returns a tuple of containing whether it has identified any bonds, a list of edges representing these bonds and a list of the remaining primitives.

Observe that the `get_dashes` function combines predicates R7.1 and R7.2 whereas `create_col_dash_clusters` represents predicate R7.3. Also, the function `create_adjacent_dash_clusters` combines predicates R7.4 and R7.5. Finally, the rule creates graph edges each of which representing a dashed bond using the function `create_dashed_edge`.

In addition to that, notice that the rule returns the entire set of `primitives` if any of its predicates fails. This is evident in lines 27, 30 and 31 of Listing 7.3. Also, observe that if the rule successfully executes, it returns all the remaining primitives as shown in line 28 (`!left_overs`).

7.5.4 R12: Wavy Bond

Wavy bonds (Figure 7.18) are commonly used in chemical structure diagrams. As the name suggests, they have a *wavy* form although a less commonly used *saw-tooth* format can be encountered in the literature.

Figure 7.18: A Wavy Bond

A vectorisation process will most likely turn a wavy bond into a connected sequence of short line segments arranged in a saw-tooth, or a zig-zag, pattern. As illustrated in Figure 7.19, a straight line can pass through the centre points of these line segments. We will identify a pattern that forms a wavy bond if following conditions are satisfied:

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments

2. $\forall l \in L : length(l) \in dl$

3. All elements of L are connected

4. The centre points of the elements of L are approximately collinear

5. Two elements of L, called the end elements, dash-neighbour precisely one other element of L. All other elements of L, called internal elements, neighbour precisely two other elements of L

6. Two end points that are not connected must be the pair of end points that are furthest apart

Consequence: A wavy bond between the furthest two endpoints. The new wavy bond has an unknown direction and unknown start and end.

Figure 7.19: Dashes in a Wavy Bond

Observe that R12.2 guarantees that none of the captured line segments can be captured by any of the rules that deal with normal line segments (such as R1-3). Also, R12.3 guarantees that the line segments will not be captured by either of R8 or R9.

As for our implementation, we provide our code for R12 in Listing 7.4 where we only show the code for the rule and not the predicates. As we mentioned in Section 5.3, the rule receives a list of primitives as input and returns a tuple of containing whether it has identified any bonds, a list of edges representing these bonds and a list of the remaining primitives.

Observe that the `get_dashes` function combines predicates R12.1 and R12.2 whereas `create_connected_dash_clusters` represents predicate R12.3. Also, the function `create_wavy_dash_clusters` combines predicates R12.4 and R12.5. Finally, the rule creates graph edges each of which representing a wavy bond using the function `create_wavy_edge`.

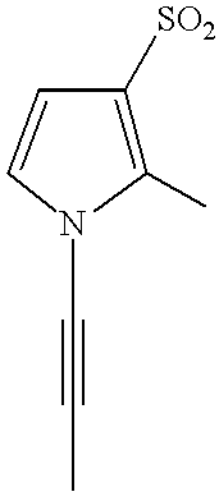
```
1  let apply_r12 primitives =
2    let left_overs = ref [] in
3      match get_dashes primitives with
4        | (true,all_dashes,other_lines) ->
5            ( left_overs := other_lines :: !left_overs;
6              match create_connected_dash_clusters (flatten all_dashes) with
7                | (true,con_dashes,other_dashes) ->
8                    ( left_overs := other_dashes :: !left_overs;
9                      match create_wavy_dash_clusters con_dashes with
10                       | (true,wv_dashes,other_dashes) ->
11                           (
12                             let wavy_edges = map (
13                               fun dashes ->
14                                 create_wavy_edge dashes
15                             ) wv_dashes in
16                             match length wavy_edges with
17                                | 0 -> (false,[],primitives)
18                                | _ -> (true,wavy_edges,(flatten
                                     !left_overs))
19                           )
20                       | (false,_,_) -> (false,[],primitives) )
21               | (false,_,_) -> (false,[],primitives) )
22       | (false,_,_) -> (false,[],primitives)
```

Listing 7.4: OCAML code for R12

Furthermore, observe that the rule returns the entire set of `primitives` if any of its predicates fails. This is evident in lines 17, 20, 21 and 22 of Listing 7.4. Also, notice that if the rule successfully executes, it returns all the remaining primitives as shown in line 18 (`!left_overs`).

7.6 A Fully Worked Example

In this section, we will provide a simple fully worked example to demonstrate how our recognition process works. As stated before, the process starts with vectorising the input image to obtain geometric primitives. An example molecular diagram with information extracted after vectorisation is provided in Figure 7.20. Observe that the actual list of primitives starts at the second line of Figure 7.20(b). The first line shows the image dimensions and coordinates of its centroid which are necessary to create a MOL file as we shall explain in Section 8.5.



(a) Molecular Diagram

```
 1 171;392;208.423354;77.325188
 2 chargroup;N;46;165;66;183
 3 chargroup;SO2;124;5;167;30
 4 linesegment;96;70;30;71;3;0.000000
 5 linesegment;7;134;31;70;3;-2.702761
 6 linesegment;114;134;97;71;3;3.804763
 7 linesegment;74;164;114;135;3;-0.732475
 8 linesegment;160;160;115;135;3;0.565156
 9 linesegment;65;315;65;250;3;1073741823.000000
10 linesegment;49;315;49;250;3;1073741823.000000
11 linesegment;57;382;57;187;3;1073741823.000000
12 linesegment;15;132;36;79;3;-2.630205
13 linesegment;105;130;89;78;3;3.449660
14 linesegment;43;164;6;135;3;0.796805
15 linesegment;97;70;121;28;3;-1.752788
```

(b) Result of Vectorisation

Figure 7.20: A Molecule Diagram with its Corresponding Set of Primitives

Since the primitive list actually starts at the second line of Figure 7.20(b), we observe that there are 14 primitives (two character groups and 12 line segments). Observe that the lines showing character group primitives provide each character group's label and the coordinates of its bounding box. Also, observe that the lines showing line segments provide each line segment's coordinates, width and slope.

It can be seen in Figure 7.20(a) that only three rules are going to execute, namely, the rule that captures planar single bonds (R1), the rule that captures planar double bonds (R2) and the rule that captures planar triple bonds (R3). As we have stated in Section 7.1, our implementation works in a way such that rules attempt to capture all their target bonds at once instead of capturing only one bond at a time.

As for the rule execution sequence, if R1 is executed first then this would generate five graph edges each of which represents a single bond, hence reducing the number of primitives to nine. If R3 executes next, it would generate one graph edge representing a triple bond and add two new line segment primitives to the list of primitives which reduces the number of primitives to eight (notice that R3 removes three line segments and adds two). If R2 executes, it would generate two graph edges each of which represents a double bond, thus removing four

line segments from the list of primitives which decreases its size to four. Now R1 is the only applicable rule to capture the two line segments added by R3 which results in two more graph edges each of which representing a single bond and reducing the size of the list of primitives to two (chargroups only). At this stage, no more rules are applicable and therefore the rule engine terminates and the graph formation process starts.

As shown in Figure 7.21, a different rule execution sequence may start with R3. This would generate one graph edge representing a triple bond and add two new line segment primitives to the list of primitives which reduces the number of primitives to 13 (notice that the initial number of primitives was 14 and R3 would remove three line segments and add two). Now if the next rule to execute is R2, it would generate two graph edges each of which represents a double bond, thus removing four line segments from the list of primitives which decreases its size to nine. Hence, the only remaining applicable rule is R1 which should generate seven graph edges each of which represents a single bond, thus reducing the number of primitives to two (chargroups only). Again, at this stage, no more rules are applicable and therefore the rule engine terminates and the graph formation process starts.

Table 7.2 provides the results of an experiment in which we ran our tool 10000 times on the diagram provided in Figure 7.20(a). The table shows all the possible rule execution orders and the number of their occurrences. Observe that we compared all the resulting MOL files against the benchmark MOL file of the used diagram and they were all matching.

Rule Sequence	# of Occurrences
R2-R1-R3-R1	1672 times
R3-R1-R2	1695 times
R1-R3-R2-R1	1603 times
R2-R3-R1	1673 times
R3-R2-R1	1655 times
R1-R2-R3-R1	1702 times

Table 7.2: Possible Rule Execution Sequences with their Occurrence Frequency

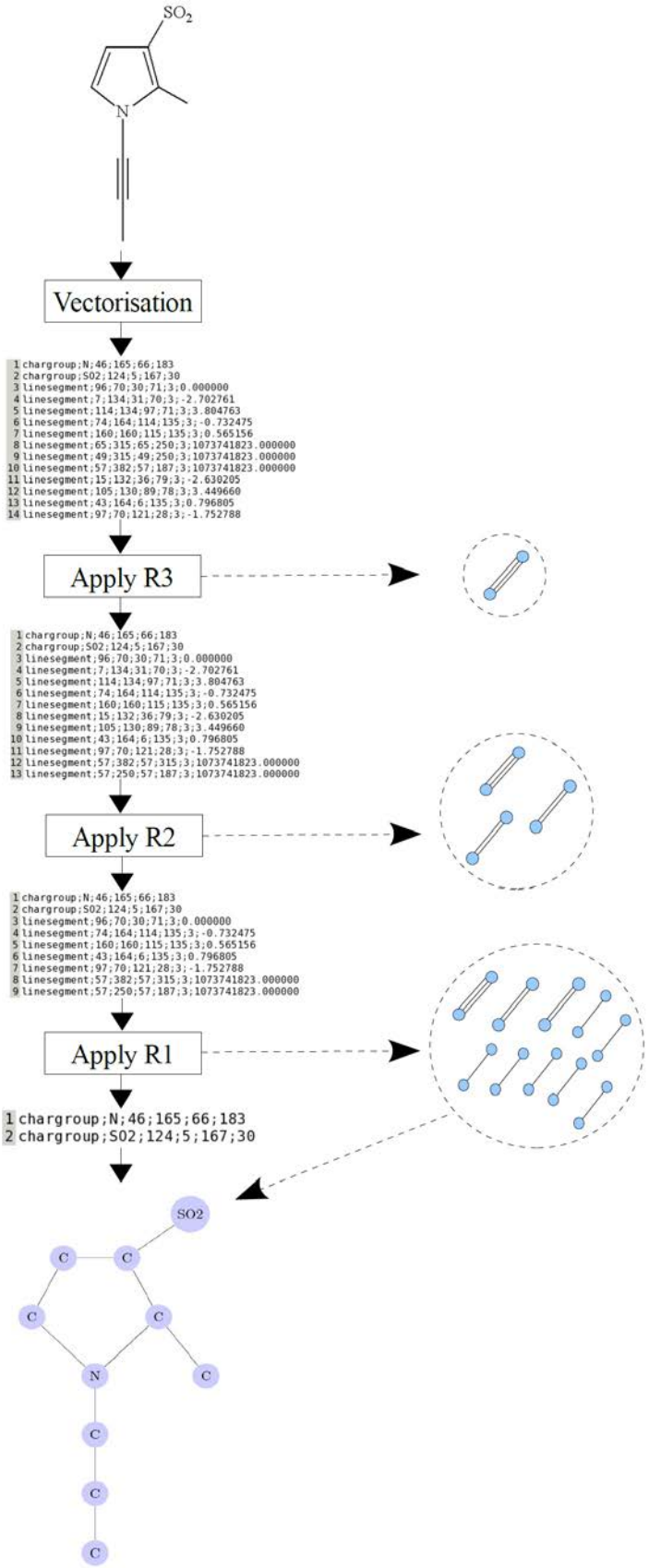


Figure 7.21: A Fully Worked Example of how our Recognition System Works

7.7 Summary

This chapter has provided a description of our rules. We have started by providing more details about our implementation and then defined the geometric primitives, explained some fundamental concepts and defined several preliminary parameters which are needed by the recognition system. After that, we have given a tabular summary of the 18 rules comprising our rule engine. Also, a detailed discussion of four of these rules has been provided in this chapter with the remaining rules explained in Appendix B. As for the four discussed rules, we have explained their predicates and how they capture and rule out various patterns. Not only this, but we have also provided sample OCAML code to show how we have implemented these rules. At the end of the chapter, we have provided a simple fully worked example where we demonstrated how the entire recognition process works. In the next chapter, we shall show how we form a graph data structure and use it to generate a suitable output.

CHAPTER 8

GRAPH FORMATION AND OUTPUT GENERATION

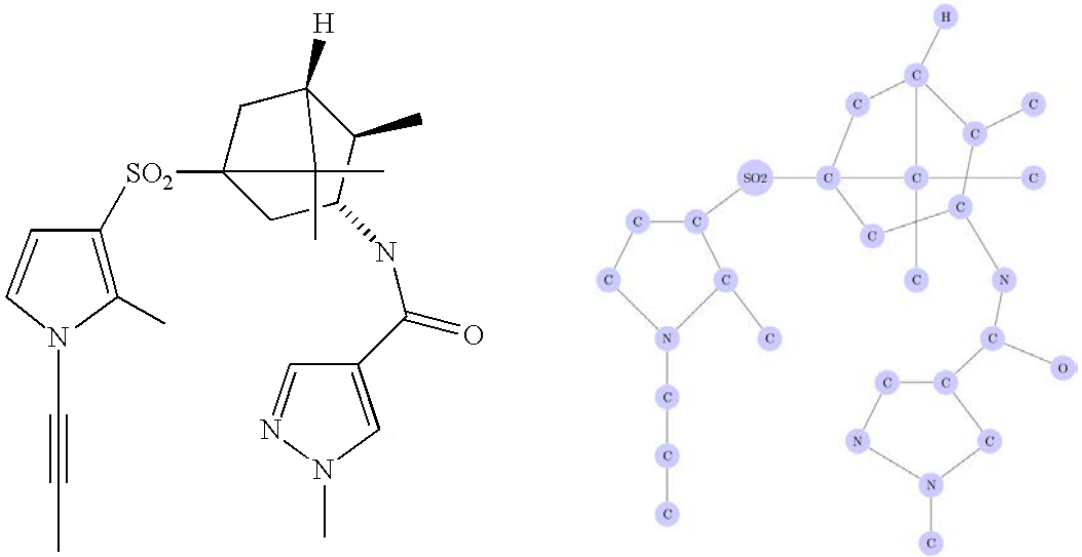
After detailing the components of our rule-based recognition system and providing a description of its embeddings, we shall now explain how we generate a suitable output. As explained in Chapter 7, one or more graph edges are created after a rule successfully executes. We will show in this chapter how we use these edges to construct a graph data structure to represent a chemical structure diagram, post-process it and output a MOL file. Our explanation will include some rules-of-thumb to avoid incorrectly forming a graph as well as our method for expanding superatoms. We will discuss our approach for guessing the stereocentre of a 3-dimensional bond when it is unknown. Additionally, we will illustrate our method of generating a MOL file from the graph. We will end the chapter by listing the steps of a typical procedural approach for recognising chemical structure diagrams.

8.1 Formation of Graph

As we have mentioned in Section 3.1.1, a mathematical graph can be used to represent a chemical structure. A mathematical graph G is a pair (V, E) of sets, the vertex set V and the edge set E. As these sets are application dependent, using a graph data structure to represent a chemical diagram only requires representing nodes as vertices and bonds as edges.

An example chemical structure diagram and its corresponding graph are provided in Figure 8.1 where every node in the graph represents an atom and every edge represents a bond. It can be noticed that there are multiple vertices with the same label, the implied C atom. This may lead to ambiguity in distinguishing atoms (vertices) when processing the graph and therefore we allocate them unique identifiers. Apart from the label, we give vertices another attribute and that is the geometric location of the node in the original image. This location can be computed easily by averaging the x and y components of each line segment endpoint associated with the node. This location is vital for generating a corresponding MOL file as we will explain in Section 8.5.

Notice that rules R16 and R18 (Appendix B.12 and B.14 respectively) require a preliminary graph to operate. This means that their execution must be delayed until all the other rules have finished executing. To form a graph using the output obtained from the rule engine, several



(a) Molecular Structure (b) The Corresponding Chemical Graph

Figure 8.1: Graph Representation of a Chemical Structure

parameters need to be set. Since the output can be represented as a list of graph edges, distances to judge whether edges are connected or not need to be set. Also, chargroup primitives should be used since they represent graph vertices and share one or more edges. Another distance that should be appropriately configured is the distance between an edge and a chargroup that it is attached to. In the absence of an explicit chargroup, a label of C, representing a Carbon atom, should be used. Some heuristics that can be used to precisely build a graph representing a chemical structure are suggested in Section 8.2.

Additionally, one should pay attention to the output of rules R4, R5 and R6. Since the patterns captured by these rules can be one bond in one direction and another bond in the orthogonal direction, we resolve them after building an initial graph. The idea is to examine the position of each pattern and decide what they represent based on the result of that examination.

Two issues should be resolved after the graph has been built. The first issue is that of superatoms. By looking at Figure 8.1, it can be noticed that the node with the SO_2 label is itself a graph since the SO_2 is a superatom (Section 3.1.5.3). As we will explain in Section 8.3, we need to expand the superatom so we can fully generate a graph data structure.

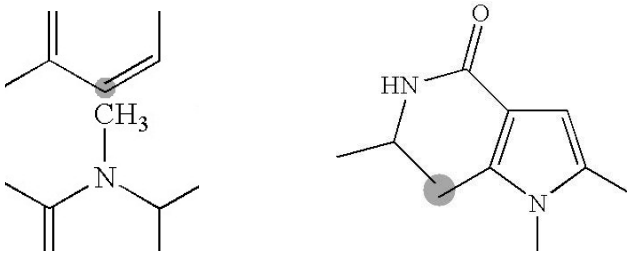
The second issue relates to the direction of dashed, bold and dashed bold bonds. We will

introduce our suggested mechanism for guessing these directions in Section 8.4.

8.2 Heuristics for Graph Formation

The following list of points should be considered when building a graph data structure that represents a chemical structure. These heuristics are based on observation and rules of thumb, and they have been used in the tool developed as part of this work.

- Nodes can either be closed or open in a chemical structure diagram (Figure 8.2). When a node has at least two connected line segments, we denote it as *Closed*. An *Open* node is a node that has no connected line segments. We use this to constraint the identification step so that a closed node can not have an explicit chargroup (atom) even if one is in close proximity. This is significant because it allows us to better classify and identify nodes. For example, as Figure 8.2(a) shows the CH_3 should not be associated with the shadowed node despite being very close to it. On the other hand, if an open node has no explicit atom, lines forming this node must be parallel. As illustrated in Figure 8.2(b), the two shadowed line endpoints do not belong to the same node even though they are within close proximity of one another.



(a) Closed Node (b) Open Node

Figure 8.2: Closed and Open Node Examples

- Lines connected to explicit atoms must be pointing towards them. This means that if a line's endpoint is very close to an explicit atom, then for the line to be connected to that atom, if extended, should pass through the atom's bounding box (Section A.2.2). (i.e. the line must be pointing towards the atom). This is illustrated in Figure 8.3, atom A has two very close line endpoints, one has been shadowed and one has not. When extending the

line, which endpoint has been shadowed, it can be seen that the line will not go through A's bounding box, and therefore it is not connected to it. However, if the other line is extended, it will pass through A's bounding box and therefore it is connected to it.

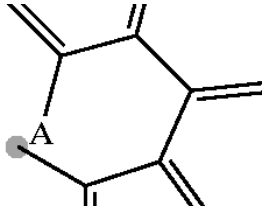


Figure 8.3: Line Close to Atom but not pointing towards it

8.3 Expanding Superatoms

As we have mentioned in Sections 2.2 and 3.1.5.3, superatoms are used as shortcut names of substructures to save display space. Examples of a chemical structure diagrams with superatoms are shown in Figure 8.3.

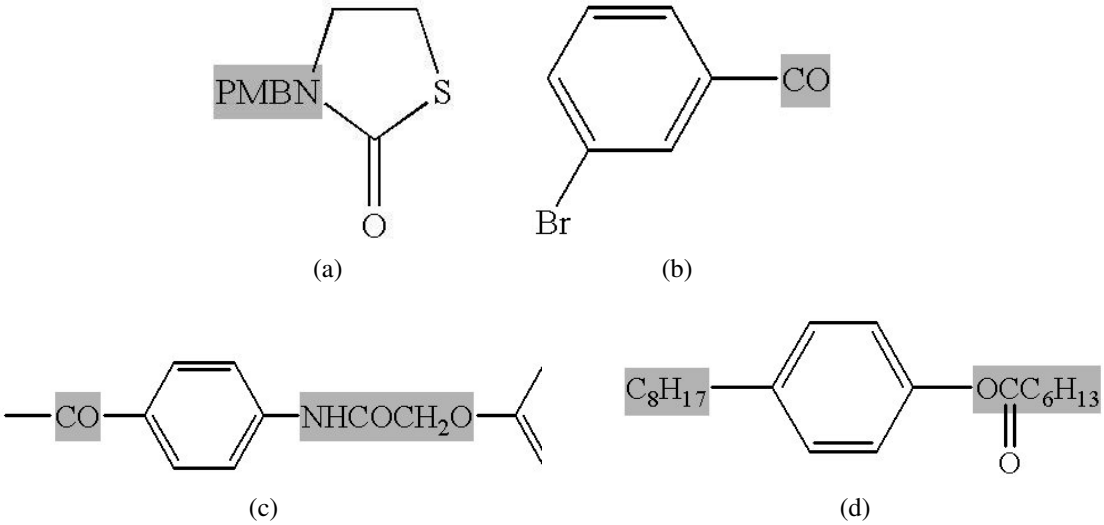


Figure 8.4: Example Superatoms (Highlighted)

We have stated that it is not straightforward to know how the superatom is exactly connected to the main structure. In an attempt to resolve this problem, we suggest using a dictionary — of a large number of superatom names — which should contain superatom names and their corresponding substructures.

Unfortunately, we have not yet been able to find a definitive dictionary of this type, although

a useful one can be found at [64]. Some existing attempts to solve the problem use a dictionary based on mapping names to $SMILES$ strings and generate the corresponding structure from that.

The $SMILES$ expression does not identify which atoms of the superatom connect to the surrounding structure. When there is only one such connecting point, the atom in question can usually be identified by considering valencies on the atoms. However, if there are two or more bonds between the superatom and the surrounding structure, it is unclear how to determine the appropriate permutation of connection possibilities to match the actual chemical structures in question.

Rather than using $SMILES$ expressions, we chose to mine the MOL files of the benchmark dataset made available by OSRA [104] and the training dataset made available by TREC11 [102] to identify more complete information about superatoms than are available from their $SMILES$ strings. Superatoms in benchmark MOL files identify the superatoms in situ, marking each atom of the whole structure that belongs to the superatom, together with the internal bonds in the superatom and the external bonds between the superatom and the surrounding structure.

```
CH2CH2O 2       [(2,3,1);(1,2,1);]      [1;3;]  [(1,C);(2,C);(3,O);]    [(0,0)]
OCH2CH2 2       [(2,3,1);(1,2,1);]      [3;1;]  [(1,C);(2,C);(3,O);]    [(0,0)]
HO2C    1       [(1,3,1);(1,2,2);]      [1;]    [(1,C);(2,O);(3,O);]    [(0,0)]
CO2H    1       [(1,3,1);(1,2,2);]      [1;]    [(1,C);(2,O);(3,O);]    [(0,0)]
CH2CH3  1       [(1,2,1);]      [1;]    [(1,C);(2,C);]  [(0,0)]
Br(CH2)2        1       [(1,2,1);(2,3,1);]      [1;]    [(1,C);(2,C);(3,Br);]   [(0,0)]
NO2     1       [(1,3,1);(1,2,2);]      [1;]    [(1,N);(2,O);(3,O);]    [(1,1);(3,-1)]
O2N     1       [(1,3,1);(1,2,2);]      [1;]    [(1,N);(2,O);(3,O);]    [(1,1);(3,-1)]
OCH3    1       [(1,2,1);]      [1;]    [(1,O);(2,C);]  [(0,0)]
H3CO    1       [(1,2,1);]      [2;]    [(1,C);(2,O);]  [(0,0)]
CN      1       [(1,2,3);]      [1;]    [(1,C);(2,N);]  [(0,0)]
NC      1       [(1,2,3);]      [2;]    [(1,C);(2,N);]  [(1,-1);(2,1)]
COO     2       [(1,3,1);(1,2,2);]      [1;3;]  [(1,C);(2,O);(3,O);]    [(0,0)]
CO2     2       [(1,3,2);(1,2,1);]      [1;2;]  [(1,C);(2,O);(3,O);]    [(0,0)]
```

Figure 8.5: Superatom Dictionary

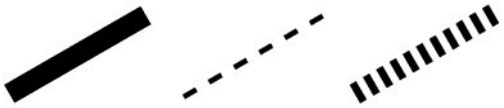
Subsequently, we have built a detailed dictionary of superatoms and their internal information. As Figure 8.5 illustrates, each entry in the dictionary has several fields. The first field represents the superatom's name, the second field provides the number of connections to the main structure, the third field provides an adjacency list corresponding to the superatom substructure, the fourth field shows which atom IDs of the superatom are connected to the main

structure, the fifth field is a list of the atoms forming the superatom's substructure with their unique identifiers and the sixth field shows which atoms has any charge.

This still leaves us with the connection permutation problem, but at least gives us unambiguous internal connection information and identification of the connecting atoms.

8.4 Guessing the Stereocentre

For 3-dimensional bonds with unknown stereochemistry, as in Figure 8.6, one needs to decide the stereo nature of the bond, i.e. which side of the bond is the stereocentre.



(a) Bold bond (b) Dashed bond (c) Dashed bold bond

Figure 8.6: Bonds with Unknown Stereochemistry

Usually, the syntax of the diagram does not give sufficient information to resolve this correctly (Figure 8.7(a) and 8.7(b)) so, in the absence of better domain knowledge, we suggest using a number of heuristics based on observed patterns in diagrams which have been analysed previously. These heuristics can use information about the number of neighbours on each end of the bond in question.

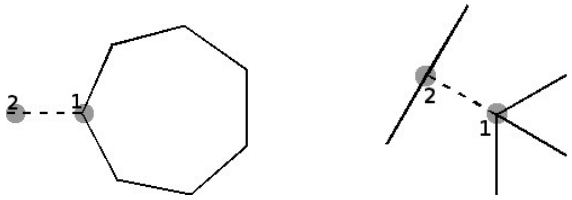
(a) (b)

Figure 8.7: Which atom is the stereocentre?

In more detail, after the graph is built, we examine it using the following procedure:

- If the bond in question has a free end, as in Figure 8.8(a), then the stereocentre is assumed to be the atom at the non-free end.
- If the bond in question does not have a free end, i.e. the bond is connected to other bonds from both sides as in Figure 8.8(b), then the end with the smallest number of neighbours is chosen to be the stereocentre.

- If the two nodes on both sides of the bond have the same number of neighbours, the stereocentre is chosen at random.



(a) Stereocentre is atom number 1

(b) Stereocentre is atom number 2

Figure 8.8: Determining the Stereocentre

8.5 Outputting MOL File

After building a graph data structure that captures details and connectivity information of a given image that contains a chemical structure, one possible output would be a MOL file (Figure 8.9). We have discussed the structure of MOL files in Section 3.1.5.1. The number of atoms, number of bonds, any atom charge information, bond type and bond orientation can be easily obtained from the generated graph.

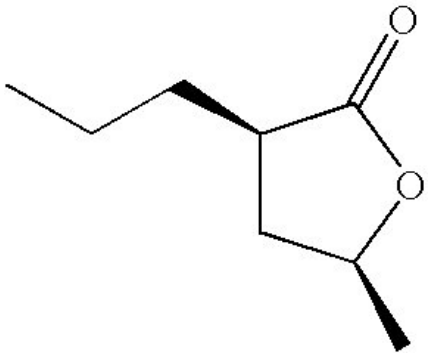
One interesting output that needs to be calculated is the atom coordinates. Since MOL files use Angstroms, using normal coordinates would not be appropriate. However, we use the following procedure to calculate a reasonable estimate of these coordinates:

1. Calculate the input image's centroid (centre of gravity) using Equation A.3.

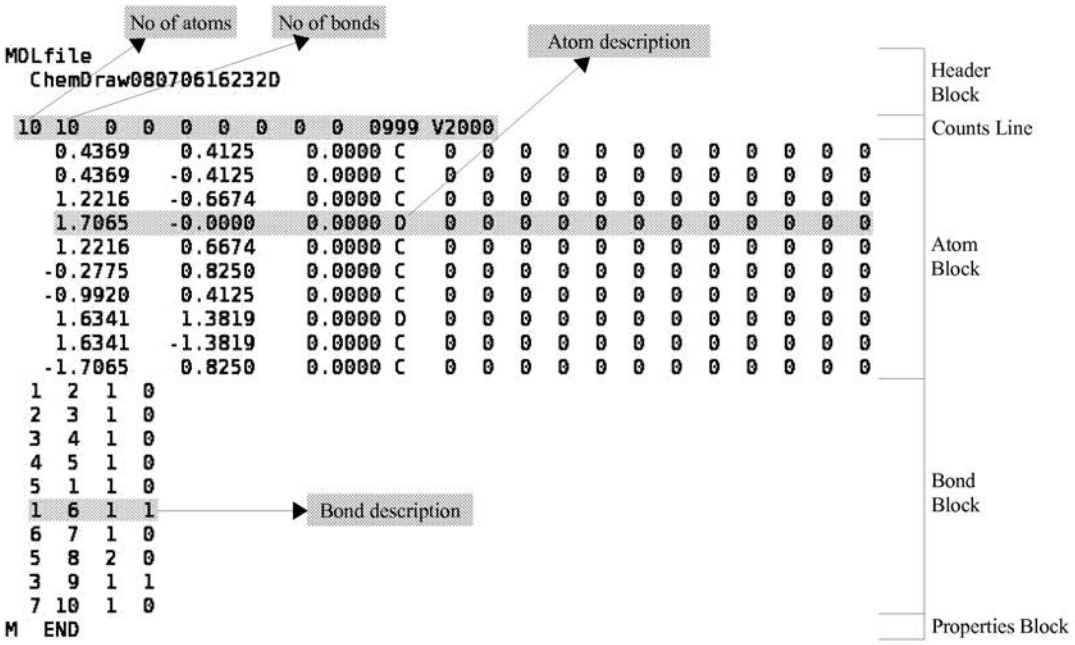
2. The node centres should be available from the generated data structure (a node's centre can be obtained by averaging the x and y components of the endpoints of all the bonds that belong to the node).

3. Now, given $\bar{x}$ and $\bar{y}$ as the two components of the image's centroid, x_c and y_c are a given node's two coordinates (node centre), we compute the approximate X and Y coordinates of the node in the MOL file as follows:

$$X = \frac{x_c - \bar{x}}{100} \qquad and \qquad Y = \frac{y_c - \bar{y}}{100} \tag{8.1}$$



(a) A Chemical Structure



(b) The Corresponding MOL file

Figure 8.9: An Example Structure with its MOL file

8.6 Steps of Procedural Implementation

In the procedural approach we are going make an attempt to extract a molecule diagram from a bitmap image, and construct a graph data structure representing that molecule and generate a MOL file. We are going to carry out the recognition process by a combination of standard techniques and bespoke algorithms. Figure 8.10 presents intermediate results of the recognition procedure for an example molecule image. As the figure illustrates, the steps can be summarised in the following points:

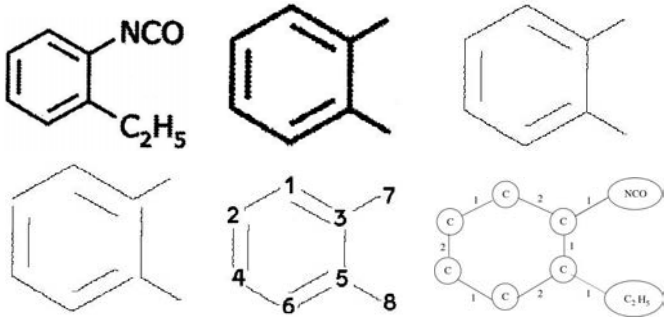


Figure 8.10: Recognition steps for the molecule C_9H_9NO.

- **Preprocessing.** We perform image binarisation using Otsu's method [73] and follow it by identifying connected components. After that, we perform OCR by extracting a set of structural features from connected components and applying a Euclidean metric based classification. We also detect circles and distinguish them from "O" atoms using a size filter. We remove all connected components recognised as characters from the image and use some contextual information to disambiguate difficult cases.

- **Polyline Extraction.** At this point we produce a new copy of the diagram (a character free diagram) and apply a thinning algorithm to connected components to thin them to a single pixel width. Using the thinned lines as a guide, we walk the corresponding paths in the original image to determine the average line width by finding the largest disc that fits wholly with the stroke width of the line. At the same time, we build a polyline representation of the thinned lines. We split polylines, at every junction where three or more polylines meet, into separate polylines. We also identify Closed polylines.

  We then apply the Douglas-Peucker line simplification algorithm [32] to the resulting polylines with the simplification threshold set to between 1 and 2 average line widths as found above. Occasionally, some spikes on the polylines that were produced as artifacts of thinning can remain after applying the line simplification. If these are detected, the un-thinned image is smoothed, binarised, thinned and the line simplification process is repeated.

- **Bond analysis.** Now we calculate a first estimate of average bond length of all the line

segments in the discovered polylines. This estimate is refined as the recognition procedure progresses.

To extract precise geometric information solid wedge and bold bonds we employ the same approach explained in 6.3.

We detect dashed wedge and dashed bold bonds by identifying short lines whose centre points are regularly spaced within a certain tolerance. A more direct approach based on using their slope is not reliable because of the difficulty of accurately finding the slope of such short lines. While we identify dashed bonds as repeated short collinear lines.

Wavy bonds are reduced after thinning and line simplification to a sawtooth pattern polyline of connected short line segments. This is straightforward to identify when following the polyline.

Parallel lines close together where each is of approximately the average bond length indicate a double (or triple) bond. Parallel lines where at least some are shorter than the others indicate a sequence of separate bonds with an atom at the node identified by the end of the shorter lines. We identify all these by clustering lines of the same slope that are within a threshold distance of each other. In the bond sequence case, the long lines are split at the point where the short lines end, so that a node to hold the implied atom is created. Hollow wedge bonds are identified as groups of lines forming isosceles triangles.

- **Graph Creation and Further Bond Analysis.** We construct an initial undirected graph at this point where each bond is an edge and each junction is a node. We achieve this by grouping line segment endpoints by distance and by connectivity to the bounding box of atoms in order to construct each node. We label each vertex of the graph with the atom or molecular sub-formula it represents. In our example in Figure 8.10, a vertex is introduced for each of the carbon atoms as well as one for each of the sub-formulas NCO and C_2H_5. Note that NCO is a superatom and it has to be expanded. Note also that each carbon atom that is normally omitted in the molecule depiction is explicitly represented with a label C

in the graph. We also label the edges of the graph with the type of bond they correspond to.

After forming the initial graph, we detect open bridge bonds by identifying the pattern of three line segments, two of which have the same slope and are aligned with their closest end points within a threshold distance of each other, and where the third segment passes between the endpoints of the first two. On the other hand, closed bridge bonds are identified by examining "X" junctions and checking that the corresponding lines are part of irregularly shaped cycles. Non bridge bond cycles are always regularly shaped polygons. Irregularity, detected by junction angles outside expected ranges, is a clear indication of $2^1/_2$-dimensional perspective drawing of some structure.

We follow this by dealing with aromatic rings. Circles indicating the existence of aromatic rings are identified during the OCR stage. We detect nodes surrounding these circles according to their geometric location and then update the corresponding bonds linking these nodes appropriately.

- **Guessing stereocentres.** As we have built the initial graph and have labelled its edges with the type of bond they correspond to, we make an attempt to guess the stereocentre of dashed, bold and bold dashed bond at this stage of the recognition procedure. We carry out the guessing process using the approach which we are going to explain in Section 8.4.

- **Expanding Superatoms.** The last step we carry out before producing a MOL file is to expand any existing superatoms. Since superatoms have a potentially complex substructure that can contain many atoms and bonds, we expand them and replace them in the molecule graph. We looked them up in our superatom dictionary (see Section 8.3 for more details on this), construct their corresponding graph and plug it into the main graph data structure.

- **Generating MOL file.** At this stage, we use the resulting graph data structure to generate a MOL file using the approach explained in 8.5.

8.7 Summary

This chapter explains how we construct a graph data structure to represent a chemical structure diagram and use it to generate a MOL file. We have provided some rules-of-thumb in order to correctly build such a graph. In addition to that, we have explained how we post-process the graph by expanding superatoms and guessing stereocentres when they exist. We have dedicated the last section of this chapter to list the steps of a procedural approach which we have developed. The next chapter, which is the last of this part, will address several issues about our rule-based recognition system.

CHAPTER 9

EXTENSIBILITY AND FLEXIBILITY OF THE APPROACH

We have provided a tabular summary of our 18 rules and explained in detail a selection of four of these rules in the previous chapter. As for the selected rules, we have discussed their predicates, how they capture and rule out various patterns and provided sample OCAML code to show how we have implemented these rules. We have also demonstrated how our entire recognition process works by providing a simple fully worked example. In this chapter, we are going to discuss a number of issues related to a rule-based recognition system like ours. In particular, we are going to address conflicts, rule execution order, system termination and the issue of circularity.

After that, we will demonstrate the extensibility of our rule-based approach by showing that, despite having 18 rules, the number of rules can grow to widen the scope of handled bond types and bond arrangements.

9.1 Discussion of Rules

Many issues arise when designing a rule-based system for pattern recognition. These issues include rule conflicts, order of rule execution, when, or how, should the recognition system terminate and an issue known as circularity.

9.1.1 Rule Conflicts

Rule conflicts and order of rule execution are closely related in the context of our work. Observe that by rule conflicts we mean a situation when multiple rules can have their conditions satisfied resulting in a subset of rules, usually denoted the *conflict set*, that should execute. If such a scenario happens, a mechanism to resolve such conflicts is vital to choose which rule of the conflict set should execute. Additionally, by rule execution order we mean a situation when it is necessary to execute one or more rules before other rules.

The correct choice of parameter values (Section 7.2.3) helps to minimise rule conflicts. Furthermore, choosing a subset of rules that must execute before other rules improves performance and eliminates rule conflicts.

For example, a single line segment which length is above wedge base length and which

width is less than bold bond width would be captured by R1, R10, R11, R16, R17 or R18. To avoid such conflict, we make sure that R1 is executed at least once before any of R10, R11, R16, R17 and R18. Also, another case is when a dashed wedge bond is formed of very short dashes (often at the pointy end), these dashes can be captured by R7 and interpreted as a dashed bond. This can lead to a dashed wedge bond being interpreted as two bonds (dashed wedge and dashed, or dashed bold and dashed). To avoid this situation, we ensure that R9 is executed at least once before R7.

9.1.2 System Termination

Another issue that may arise is when the rule system terminates. As we explained in previous chapters, the idea of a rule-based system is for the rule engine to constantly access the working memory (i.e. the set of primitives) and examine if any primitive, or subset of primitives, satisfies any rule conditions. If this is true, this subset of primitives is removed from the working memory when the rule executes – bear in mind that R2 and R3 can add new primitives to the primitive set. This procedure should continuously run until a termination mechanism is met.

One suitable termination mechanism is when the working memory becomes empty, i.e. there are no more primitives in the primitive set. However, because the rules do not explicitly make use of chargroup primitives, this mechanism may be modified to terminate the rule system when the primitive set contains only chargroup primitives.

However, a case when the primitive set contains primitives other than chargroups which are not captured by any rule can exist. A typical example is when a circle primitive which is not part of an aromatic cycle exists. Because R16 will not remove this circle from the primitive set (remember it is not part of an aromatic cycle), the previous termination mechanism will not be satisfied and the whole system may enter an infinite loop. To avoid such a scenario, we add another termination mechanism that ensures the system does not go into an endless loop. Namely, we terminate the rule execution if the system goes through all the rules and none of them applies. In other words, we will terminate if the rule engine attempts to execute the 18

rules and no rule executes.

9.1.3 Circularity

Not only that, but one needs to pay attention to another situation that might occur. Because some of our rules (namely R2 and R3) remove as well as generate and add geometric objects to the set of primitives, one needs to realise that the system might enter an infinite cycle of removing and adding geometric primitives. We will refer to such a scenario as *Circularity*. We make sure that our system does not fall into circularity by making sure that these rules apply cutting only when necessary. We control this by choosing a suitable value for the overlap parameter (Section 7.2.3) and by making sure that these rules are not executed again until other rules have been attempted as we have mentioned in Section 5.3. Observe that we have demonstrated the effectiveness of our approach by running our tool 10000 times on a chemical diagram image that contains a cuttable triple bond and our tool terminated successfully in all runs (see Section 7.6 for more details).

9.2 More Bond Types

To provide proof for the extensibility of the approach, we are going to add rules to capture four new bond types. We will explain the necessary conditions to capture each of them and the required changes to any of the previously introduced 18 rules in the next subsections.

9.2.1 R19: Aromatic Bond

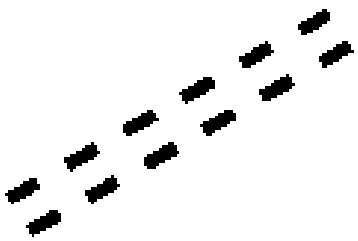


Figure 9.1: Aromatic Bond

This bond is usually illustrated as two parallel dashed lines (a double dashed bond as in Figure 9.1). The two lines have the same number of dashes and every dash on one line appears

as the orthogonal projection of another dash on the other line. Therefore, we will distinguish between this type of bonds and the dashed bond by adding one or more conditions to those of the dashed bond and building a set of rules specifically to identify this type of bonds.

In Section 7.5.3, we had the following conditions in the original rule to identify a dashed bond (**R7**):

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments
2. $\forall l \in L : length(l) \in dl$
3. L is approximately collinear
4. No two elements of L have a separation distance of less than the minimum of ds
5. Two elements of L, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L. All other elements of L, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L

Consequence: L forms a dashed bond with endpoints given by the endpoints of the minimal line segment that contains $l_1, \ldots, l_n$. The new dashed bond has a down direction with unknown start and end.

To distinguish a dashed bond from an aromatic bond, we are going to add the following condition to the conditions we have used to capture a dashed bond (**R7**):

- There is no line segment $l' \notin L$ such that $l' \parallel^{ol}_{bs} l$ where $l \in L$

Observe that this condition ensures that there is not line segment that parallel to and is in close proximity of any dash in the dashed bond.

Now to capture **an aromatic bond**, we are going to use the following conditions:

1. $L_1 = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments
2. $L_2 = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments
3. Cardinality of L_1 equals cardinality of L_2

4. L_1 is approximately collinear as well as L_2

5. L_1 and L_2 are not collinear (notice the difference when compared to the previous predicate)

6. No two elements of L_1 have a separation distance of less than the minimum of ds, the same applies to L_2

7. Every $l \in \{L_1 \cup L_2\}$ has length of approximately dl

8. For every $l_1 \in L_1$, there is one and only one $l_2 \in L_2$ such that $l_1 \parallel_{bs}^{ol} l_2$

   only one element in L_2

9. Two elements of L_1, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L_1. All other elements of L_1, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L_1

10. Two elements of L_2, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L_2. All other elements of L_2, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L_2

Consequence: L forms an aromatic bond with endpoints given by the endpoints of the minimal line segment that contains L_1 and L_2.

9.2.2 R20: Tautomeric Bond

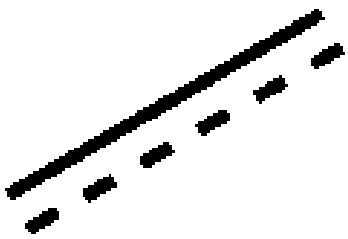


Figure 9.2: Tautomeric Bond

As shown in Figure 9.2, a tautomeric bond is often drawn as a normal line segment parallel to a dashed line, (in other words, as a dashed line parallel to a normal line segment). Every

dash of the dashed line can be fully orthogonally projected on the normal line segment with two dashes having one end point of theirs coinciding with one of the normal line's endpoints.

Therefore, we will distinguish between this type of bonds and the dashed and single bonds by adding more condition(s) to those of the dashed bond and single bond and building a set of rules specifically to identify this type of bonds.

In Appendix 7.5.1, we had the following conditions in the original rule to identify a planar single bond (**R1**):

1. l is a line segment
2. $length(l) > wb$
3. $width(l) < bbw$
4. There is no line segment l' such that $l \parallel_{bs}^{ol} l'$

Consequence: l is a single bond with endpoints of l.

To distinguish a single planar bond from a tautomeric bond, we are going to add the following condition to the conditions we have used to capture a single planar bond (**R1**):

- There is no line segment l' such that l' has length of approximately dl and $l' \parallel_{bs}^{ol} L$

Additionally, in Section 7.5.3, we had the following conditions in the original rule to identify a dashed single bond (**R7**):

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 3$, is a set of line segments
2. $\forall l \in L : length(l) \in dl$
3. L is approximately collinear
4. No two elements of L have a separation distance of less than the minimum of ds
5. Two elements of L, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L. All other elements of L, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L

Consequence: L forms a dashed bond with endpoints given by the endpoints of the minimal line segment that contains $l_1, \ldots, l_n$. The new dashed bond has a down direction with unknown start and end.

To distinguish a dashed bond from a tautomeric bond, we are going to add the following condition to the conditions we have used to capture a dashed bond (**R7**):

- There is no line segment l' such that l' has length of $> dl$ and for every line segment $l \in L$ there is not line l such that $l \parallel_{bs}^{ol} l'$

Now to capture a **tautomeric bond**, we are going to use the following conditions:

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 4$, is a set of line segments
2. L can be split into two distinct sets L_1 and L_2 such that:
   (a) $L_1 = \{l_1\}$ such that length of $>> dl$
   (b) Every $l \in L_2$ has length of approximately dl
3. Every $l \in L_2$ is parallel to, and with overlap of dl, l_1
4. L_2 is approximately collinear
5. No two elements of L_2 have a separation distance of less than the minimum of ds
6. Two elements of L_2, called the *end elements*, dash-neighbour wrt. ds precisely one other element of L_2. All other elements of L_2, called *internal elements*, dash-neighbour wrt. ds precisely two other elements of L_2

Consequence: L forms a tautomeric bond with endpoints given by the endpoints of the minimal line segment that contains L_1 and L_2.

Figure 9.3: Double Bond with Type 1 Stereochemistry

9.2.3 R21: Double Bond with Type 1 Stereochemistry

A double bond with stereochemistry is usually drawn in one of two ways. We will cover the first way in this rule and cover the second way in the next rule (**R22**). As Figure 9.3 shows, it is drawn as a bold line parallel to a normal line. The two lines have the same length and they are very close. In order to capture a bond of this type and avoid having the two lines identified as individual single planar and bold bonds, we are going to modify the rules that we have introduced to capture single planar bonds (**R1** in Appendix 7.5.1) and bold bonds (**R15** in Section B.11).

In Appendix 7.5.1, we had the following conditions in the original rule to identify a planar single bond (**R1**):

1. l is a line segment
2. $length(l) > wb$
3. $width(l) < bbw$
4. There is no line segment l' such that $l \parallel_{bs}^{ol} l'$

Consequence: l is a single bond with endpoints of l.

To distinguish a single planar bond from a double bond with type 1 stereochemistry, we are going to add the following condition to the conditions we have used to capture a single planar bond (R1):

- There is no line segment l' such that $width(l') > bbw$, $len(l') > wb$ and $l' \parallel_{bs}^{ol} l$

Also, in Section B.11, we had the following conditions in the original rule to identify a bold bond (**R15**):

1. l is a line segment
2. $width(l) > bbw$

Consequence: A bold bond. The new bold bond has an up direction with unknown start and end.

To distinguish a bold bond from a double bond with type 1 stereochemistry, we are going to add the following condition to the conditions we have used to capture a bold bond (R15):

- There is no line segment l' such that $width(l') < bbw$, $len(l') < wb$ and $l' \parallel_{bs}^{ol} l$

Now to capture a **double bond with type 1 stereochemistry**, we are going to use the following conditions:

1. $L = \{l_1, l_2\}$, is a set of two line segments
2. $width(l_1) < bbw$
3. $width(l_2) > bbw$
4. $length(l_1) > wb$ and $length(l_2) > wb$
5. $l_1 \parallel_{bs}^{ol} l_2$
6. There is no line segment $l \notin L$ such that $l_1 \parallel_{bs}^{ol} l$ or $l_2 \parallel_{bs}^{ol} l$

Consequence: L forms a double bond with type 1 stereochemistry with endpoints given by the endpoints of the minimal line segment that contains l_1 and l_2.

9.2.4 R22: Double Bond with Type 2 Stereochemistry

As we have shown in Figure 9.4, the second type of a double bond with stereochemistry is drawn as two normal lines crossing at the middle. This results in a pattern of shorter lines of

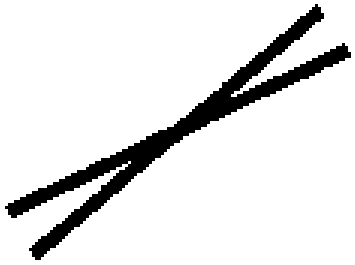


Figure 9.4: Double Bond with Type 2 Stereochemistry

similar length forming an "X" junction. Notice that the free (unconnected) endpoints of these lines can be split into two pairs where the distance between each two endpoints of each pair is small, and, the distance between any endpoint from one pair and any endpoint from the other pair is relatively large. Notice also that the junction can be misinterpreted as part of a closed bridge bond, or as a valid node (junction).

In Appendix B.14, we had the following conditions in **R18** to capture a closed bridge bond:

1. $L = \{l_1, l_2, l_3, l_4\}$ is a set of line segments
2. l_1 and l_2 are approximately collinear, as are l_3 and l_4
3. l_1 and l_2 have a separation distance of 0, as do l_3 and l_4
4. l_1 and l_2 are part of an irregularly shaped cycle
5. There is no character group where these lines connect

Consequence: Replace l_1 and l_2 with the smallest line segment containing them, and similarly for l_3 and l_4.

To distinguish a closed bridge bond from a double bond with type 2 stereochemistry, we are going to add the following condition to the conditions we have used to capture a closed bridge bond (R18):

- The distance between the unconnected endpoints of l_1 and l_3 is $> bs$
- The distance between the unconnected endpoints of l_2 and l_4 is $> bs$

- The distance between the unconnected endpoint of any of l_1 or l_3 and the unconnected endpoint of any of l_2 or l_4 is $> bs$, and vice versa

Now to capture a **double bond with type 2 stereochemistry**, we are going to use the following conditions:

1. $L = \{l_1, l_2, l_3, l_4\}$ is a set of line segments
2. $length(l_1),\ length(l_2),\ length(l_3)\ and\ length(l_4)$ are approximately the same
3. L can be split into two distinct sets $L_1 = \{l_1, l_2\}$ and $L_2 = \{l_3, l_4\}$ such that:
   (a) Elements of L_1 are approximately collinear
   (b) Elements of L_1 have a separation distance of 0
   (c) Elements of L_2 are approximately collinear
   (d) Elements of L_2 have a separation distance of 0
4. The distance between the unconnected endpoints of l_1 and l_3 is within bs
5. The distance between the unconnected endpoints of l_2 and l_4 is within bs
6. The distance between the unconnected endpoint of any of l_1 or l_3 and the unconnected endpoint of any of l_2 or l_4 is $\gg bs$, and vice versa

Consequence: L forms a double bond with type 2 stereochemistry with endpoints given by the endpoints of the minimal line segment that contains L_1 and L_2.

9.3 Summary

We have addressed a number of the issues of rule conflicts, rule execution order, system termination and the issue of circularity in the first section of this chapter. In the second section we have discussed the extensibility of our rule-based approach. Four new bond types have been introduced and the necessary changes to any of the original 18 rules have been provided. In the next chapters we will provide a detailed overview of our experimental evaluation and conclusions.

Part III

Evaluation

CHAPTER 10

EVALUATION

We are going to discuss the experimentation and subsequent evaluation of the designed rules in this chapter. We have implemented a tool to validate and demonstrate the robustness of our rules. In the remaining parts of this chapter, we will refer to the tool as MolRec. It is worth mentioning here that we have developed two versions of MolRec. A brute-force procedural version, where the execution always follows the same list of steps as we have explained in Section 8.6, and a version where we implemented our rules in a way such that the order of rule execution is not predetermined. The first will be referred to as *procedural* MolRec and the second will be referred to as *random* MolRec.

We have used various chemical diagram benchmark datasets to evaluate MolRec. This includes the publicly available dataset provided by OSRA [104], a dataset which we have created that is available at [10], two datasets from TREC11 [102] and two datasets from CLEF12 [27]. Observe that we may refer to the dataset which we have created as the Maybridge dataset. To carry out evaluation, we have used OpenBabel [72], which is a freely available tool that provides the ability to compare different MOL files semantically while ignoring unimportant syntactic differences. However, for cases where generating the MOL file that captures the intended meaning of all symbols and formations available in a structure is difficult, such as when Markush structures are involved, the resulting MOL files were converted into images using ChemAxon's molconvert [65] and the evaluation was done visually.

Observe that before we present a comparison between MolRec and other existing systems, we will first repeatedly run random MolRec on a selection of images from the above mentioned datasets. This will be a method to validate our rules and evaluate their robustness.

10.1 Repeatability on Selection of Images

In order to test our implementation, we have used the above mentioned chemical diagram benchmark datasets which include the OSRA dataset, our dataset, the datasets from TREC11 and the datasets from CLEF12.

As an experiment, we have randomly selected 100 images from the OSRA dataset, 100 images from our dataset, 50 images from the TREC11 datasets and 50 images from the CLEF12

datasets. We have run random MolRec on these images several times and evaluated the results using the method we discussed in Section 5.5 and the results were always the same as shown in table 10.1.

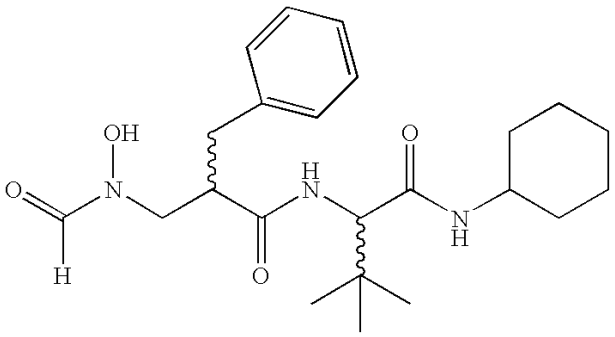
Dataset	# of Images	# of Mis-recognised Images
OSRA	100	12
Ours (Maybridge)	100	8
TREC11	50	2
CLEF12	50	5

Table 10.1: Results of Running our Tool several times on Selection of Images

Not only this, but we have chosen two images from the correctly recognised images from each of these selections and run random MolRec on each of these images 10000 times. We have compared the resulting MOL files against the benchmark MOL files of these images and all of the results were correct.

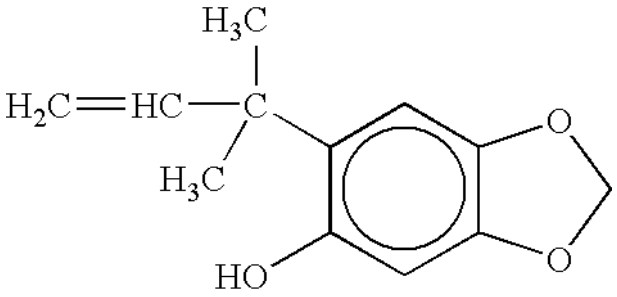
Additionally, we have recorded all the possible rule execution orders for all of these images and counted how many times each possible rule execution order occurred. This is illustrated in the following group of figures and tables.

Figures 10.1 and 10.2 show the two images we have chosen from the OSRA subset. The figures also show the frequency of all possible rule execution orders. Observe that for Figure 10.1, we can have any order of R1, R2 and R12, whereas for Figure 10.2, we can have any order of R1 and R2 followed by R16. This is because we delay the execution of R16 until other rules have executed as it requires an initial graph as we mentioned in Table 7.1 and Appendix B.12.



Rule Sequence	# of Occurrences
R2-R1-R12	1613
R1-R12-R2	1714
R1-R2-R12	1726
R12-R2-R1	1616
R2-R12-R1	1668
R12-R1-R2	1663

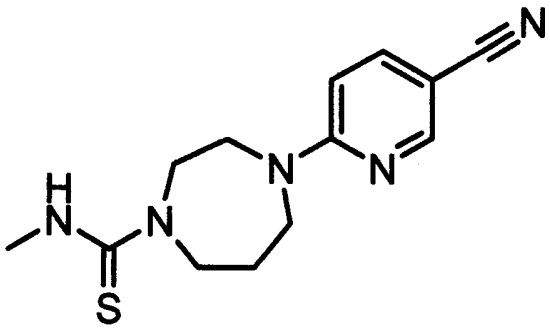
Figure 10.1: An Example image from OSRA Dataset with its Rule Execution Sequence



Rule Sequence	# of Occurrences
R1-R2-R16	4974
R2-R1-R16	5026

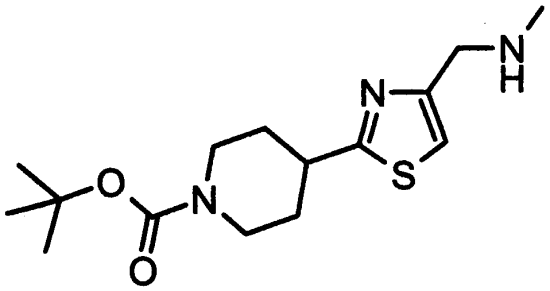
Figure 10.2: Another Example image from OSRA Dataset with its Rule Execution Sequence

The two images we have chosen from the Maybridge subset with the frequency of all possible rule execution orders are provided in Figures 10.3 and 10.4 respectively. Observe that for Figure 10.3, we can have any order of R1, R2 and R3, whereas for Figure 10.4, we can have any order of R1 and R2.



Rule Sequence	# of Occurrences
R3-R1-R2	1702
R1-R2-R3-R1	1699
R1-R3-R2-R1	1637
R2-R1-R3-R1	1680
R3-R2-R1	1622
R2-R3-R1	1660

Figure 10.3: An Example image from Maybridge Dataset with its Rule Execution Sequence

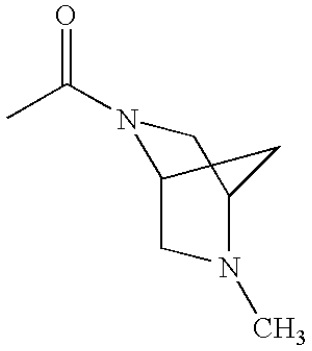


Rule Sequence	# of Occurrences
R1-R2	4982
R2-R1	5018

Figure 10.4: Another Example image from Maybridge Dataset with its Rule Execution Sequence

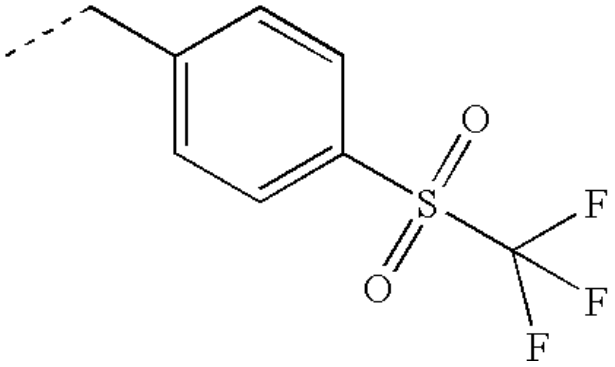
In addition to the previous sample images, we illustrate the two images we have chosen from the TREC11 subset with the frequency of all possible rule execution orders in Figures 10.5 and 10.6 respectively. Observe that for Figure 10.5, we can have any order of R1 and R2, followed by R18. This is because we delay the execution of R18 until R1 has executed and an

initial graph has been built as we have stated in Table 7.1 and Appendix B.14. Notice that in our implementation we effectively delay the execution of R18 until other rules have executed and an initial a graph has been built. For the image in Figure 10.6, we can have any order of R1, R2 and R7.



Rule Sequence	# of Occurrences
R1-R2-R18	4992
R2-R1-R18	5008

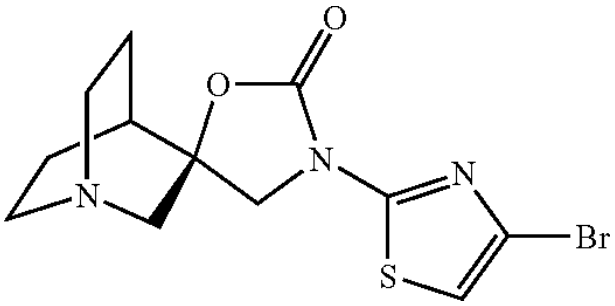
Figure 10.5: An Example image from TREC11 Dataset with its Rule Execution Sequence



Rule Sequence	# of Occurrences
R1-R2-R7	2988
R2-R1-R7	3010
R2-R7-R1	1418
R7-R1-R2	564
R1-R7-R2	1479
R7-R2-R1	541

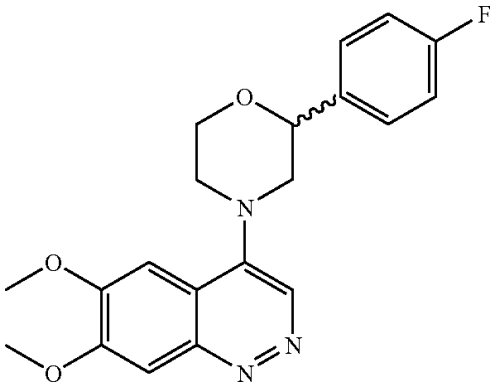
Figure 10.6: Another Example image from TREC11 Dataset with its Rule Execution Sequence

The last two images we are going to display in this section are the images we have chosen from the CLEF12 subset. These are shown with the frequency of all possible rule execution orders in Figures 10.7 and 10.8 respectively. Observe that for Figure 10.7, we can have any order of R1, R2 and R14 followed by R17. This is because we delay the execution of R17 until R1 has executed at least once as we mentioned in Table 7.1 and Appendix B.13. Notice that in our implementation we effectively delay the execution of R17 until other rules have executed and an initial a graph has been built. On the other hand, notice that we can have any order of R1, R2 and R7 for Figure 10.8.



Rule Sequence	# of Occurrences
R2-R1-R14-R17	1622
R1-R14-R2-R17	1608
R14-R2-R1-R17	1712
R1-R2-R14-R17	1683
R14-R1-R2-R17	1643
R2-R14-R1-R17	1732

Figure 10.7: An Example image from CLEF12 Dataset with its Rule Execution Sequence



Rule Sequence	# of Occurrences
R1-R2-R12	1683
R2-R12-R1	1653
R1-R12-R2	1652
R2-R1-R12	1684
R12-R1-R2	1671
R12-R2-R1	1657

Figure 10.8: Another Example image from CLEF12 Dataset with its Rule Execution Sequence

10.2 Comparison with Existing Systems

We shall now provide our comparison and evaluation of the performance of MolRec against the performance of OSRA [38] on two datasets. Additionally, we are going to present performance evaluation results from two international events that we participated in.

10.2.1 Experiments using the OSRA Dataset

OSRA's benchmark set comprises 5735 computer generated diagrams with their corresponding MOL files. Both procedural MolRec and OSRA (version 1.4.0) were run on the datasets and their performance is illustrated in Table 10.2.

	OSRA Succ.		OSRA Failed		Total	
Proc. MolRec Succ.	3420	(59.63%)	1617	(28.20%)	5037	(87.83%)
Proc. MolRec Failed	153	(2.67%)	545	(9.50%)	698	(12.17%)
Total	3573	(62.30%)	2162	(37.70%)	5735	(100%)

Table 10.2: Procedural MolRec vs. OSRA - OSRA dataset

Initial experiments showed OSRA underperforming due to its limited superatom dictionary, therefore, OSRA was enhanced by adding all the superatom descriptions from [64] before doing the comparison. The results of this experiment are given in Table 10.2.

Additionally, random MolRec and OSRA (version 1.4.0) were run on the same dataset and Table 10.3 shows a comparison of their performance:

	OSRA Succ.		OSRA Failed		Total	
Rand. MolRec Succ.	3456	(60.26%)	1642	(28.63%)	5098	(88.89%)
Rand. MolRec Failed	117	(2.04%)	520	(9.07%)	637	(11.11%)
Total	3573	(62.30%)	2162	(37.70%)	5735	(100%)

Table 10.3: Random MolRec vs. OSRA - OSRA dataset

Observe that the previous two tables denote in matrix style the number of images on which: (a) both systems succeed, by returning the correct MOL file as recognition result (top left corner), (b) both systems fail (bottom right), (c) either OSRA succeeds and MolRec fails (bottom left), or (d) MolRec succeeds and OSRA fails (top right).

10.2.2 Experiments using our Dataset

We built a dataset from 5730 scanned molecule images taken from a catalogue of drug compounds by Maybridge [6] and later extended it to 5740 images. The catalogue's pages were scanned at resolution of 600dpi and automatically analysed using a bespoke tool that we have developed. This resulted in images of molecule structures with their corresponding Chemical Abstracts Service Registry numbers. These numbers were used to look up structures in online chemical databases such as [24, 94, 68, 23] to obtain their corresponding International Chemical Identifiers. These in turn were converted to MOL files using OpenBabel. Table 10.4 shows a comparison of recognition results of OSRA (version 1.4.0) and procedural MolRec using our

dataset before extending it. The table can be read in the same way as Tables 10.2 and 10.3.

	OSRA Succ.		OSRA Failed		Total	
Proc. MolRec Succ.	3740	(65.16%)	1068	(18.60%)	4808	(83.76%)
Proc. MolRec Failed	504	(8.78%)	428	(7.46%)	932	(16.24%)
Total	4244	(73.94%)	1496	(26.06%)	5740	(100%)

Table 10.4: Procedural MolRec vs. OSRA - our dataset

In addition to that, the same dataset was used to compare the performance of random MolRec and OSRA (version 1.4.0). This was done after extending the dataset to 5740 images. Table 10.5 shows a comparison of the obtained results.

	OSRA Succ.		OSRA Failed		Total	
Rand. MolRec Succ.	3855	(67.16%)	1100	(19.16%)	4955	(86.32%)
Rand. MolRec Failed	389	(6.78%)	396	(6.90%)	785	(13.68%)
Total	4244	(73.94%)	1496	(26.06%)	5740	(100%)

Table 10.5: Random MolRec vs. OSRA - our dataset

The results provided in the previous two sections demonstrate that MolRec generally outperforms OSRA, even on its own benchmark set.

10.2.3 Experiments using TREC11's Datasets

An Image2Structure evaluation task was carried out at TREC11; participants were given a training set of 1000 images with their corresponding MOL files. They were later given a test set of 1000 images and were asked to run their tools on these images and submit the resulting MOL files. The organisers requested that each participant should submit at most the results of two runs.

In [63] the organisers reported that they received 11 runs from five participants. Table 10.6 shows a list of participants with the results shown in Figure 10.9.

Participant Group	Run name(s) Identifier
University of Birmingham	UoB
SAIC-Frederik	OSRA
GGA Software	GGA
University of Michigan	ChemReader
Fraunhofer SCAI	ChemOCR

Table 10.6: List of Participants at TREC11's I2S Task

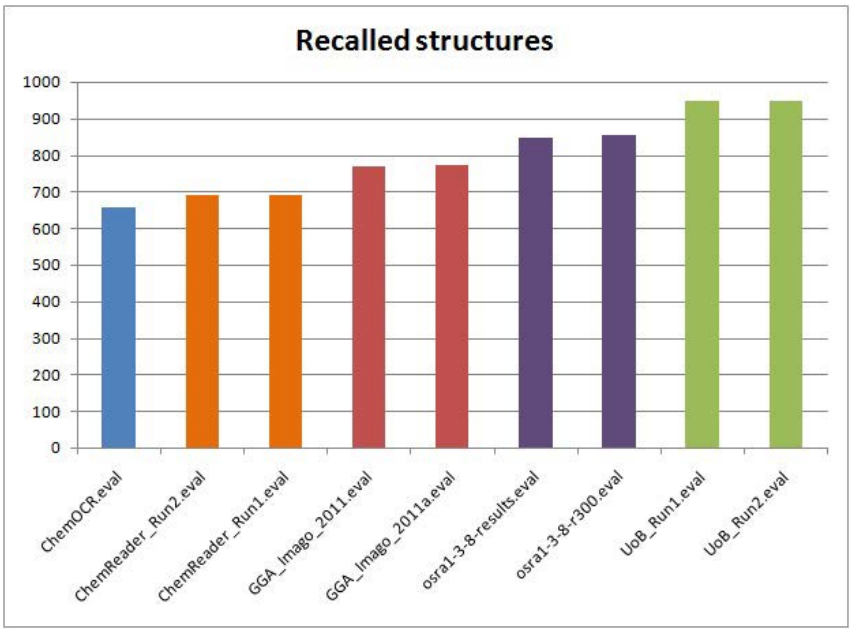


Figure 10.9: Number of Structures Identified in each run at TREC11

As Figure 10.9 shows, MolRec scored the highest success rates with two runs of 94.9% and 95% respectively. On the training set, MolRec scored a success rate of 93%.

10.2.4 Experiments using CLEF12's Dataset

A similar evaluation contest to that of TREC11 was held at CLEF12. The task was different in that the organisers decided to include Markush structures (Section 2.1.4). Handling Markush structures is not covered within the scope of this thesis as there is no way of guessing how to represent a given Markush structure in a MOL file. This is because context information is needed such as searching a full document for what is the equivalent atom(s) for a given R group. Therefore, the evaluation was done in two stages. The first stage was to evaluate the results using OpenBabel, and then to visually evaluate the results which were deemed mis-

recognition. This is because OpenBabel does not recognise symbols such as R and X as valid atom labels. ChemAxon's molconvert was used to generate images of the MOL files and the overall results were as follows:

- The training set has 133 images. Evaluating the performance of MolRec on this set using OpenBabel outputs a success rate of more than 44% (59 images). As for the remaining 74 images, their resulting corresponding MOL files were converted into JPEG images using molconvert. After visually inspecting the output images, 61 of them were found to match the original images, leading to an overall success recognition rate of 90.22%.

- The organisers of CLEF12 published a test set of 961 images. This collection was split into two sets. The first set was of 865 images selected for automatic evaluation by comparison of generated MOL files with the ground truth MOL files using the OpenBabel toolkit [72]. However, there are chemical diagrams whose valid MOL files are beyond OpenBabel's ability to compare (typically because they contain some form of Markush structure) and a second set of 95 such diagrams was selected for manual, visual evaluation. This second set was intentionally included to provide a greater challenge to the participating diagram recognition systems.

  We ran MolRec four times on these sets where we slightly adjusted its internal parameters and MolRec achieved the results illustrated in Tables 10.7 and 10.8. The two tables show the number of correct and incorrect recognitions for the four runs on the manual and automatic evaluation sets respectively. Notice that because most of the diagrams mis-recognised in some runs were also mis-recognised in other runs, there were a total of 52 different diagrams mis-recognised in the manual evaluation set and a total of 46 different diagrams mis-recognised in the automatic evaluation set.

  Again, MolRec scored the highest recognition rate. This has been illustrated in the results table shown in Table 10.9 which was published by the CLEF12 organisers [80].

Run	# Recognitions	# Mis-Recognitions	Accuracy
1	44	51	46.32%
2	56	39	58.95%
3	44	51	46.32%
4	54	41	56.84%

Table 10.7: Four Runs on the Manual Evaluation Set (95 images)

Run	# Recognitions	# Mis-Recognitions	Accuracy
1	832	33	96.18%
2	821	44	94.91%
3	821	44	94.91%
4	832	33	96.18%

Table 10.8: Four Runs on the Automatic Evaluation Set (865 images)

	Automatic Set			Manual Set			Total		
	# structures	Recalled	%	# structures	Recalled	%	# structures	Recalled	%
saic	865	761	(88%)	95	38	(40%)	960	799	(83%)
uob-1	865	832	(96%)	95	44	(46%)	960	876	(91%)
uob-2	865	821	(95%)	95	56	(59%)	960	877	(91%)
uob-3	865	821	(95%)	95	44	(46%)	960	865	(90%)
uob-4	865	832	(96%)	95	54	(57%)	960	886	(92%)

Table 10.9: Chemical diagram recognition results at CLEF12

10.3 Analysis of Results

After carrying out the experiments we have discussed in the previous section, we will now analyse the recognition errors. We shall focus on recognition errors caused by our rules and recognition errors caused by vectorisation. Our analysis of recognition errors has yielded the following most common problems:

10.3.1 Recognition Errors due to Rule Mismatch

Ambiguous Open-bridge Bonds: A number of open-bridge bonds were very ambiguous, causing several images to be incorrectly recognised. This ambiguity was because one of the connecting parts of the bond was too short (Figure 10.10).

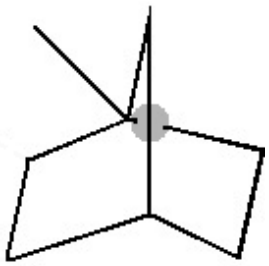


Figure 10.10: An ambiguous Open-bridge Bond (shadowed)

Rule R 17 (Appendix B.13) does not capture an open-bridge bond similar to this. This is because the long line segment is identified as a single bond. However, depending on parameters used to form a graph, this bond can be sometimes interpreted as connected to the node where the short line segment lies.

Ambiguous Closed-bridge Bonds: A number of closed bridge bonds were very ambiguous, causing several images to be incorrectly recognised. Having such patterns causes errors as relevant rules are not executed. As shown in Figure 10.11 a closed bridge bond can be ambiguous and difficult to spot even by a human observer.

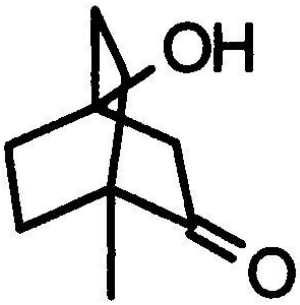


Figure 10.11: An ambiguous Closed-bridge Bond

Rule R 18 (Appendix B.14) does not capture a closed bridge bond similar to this. The reason is that the line segment pointing towards the OH atom is not identified as part of any irregularly shaped polygon, hence, the junction will be seen as a valid node.

R12 Error (Wavy Bond): The image displayed in Figure 10.12, shows a dashed wedge bond with a wavy line crossing it. We are not familiar with this notation and do not know how to

interpret it. An analysis of the corresponding solution MOL file showed the same contents as if the wavy line was not present at all. Our tool interpreted this pattern as 4 bonds connected at the centre: two dashed wedges and two wavy bonds.

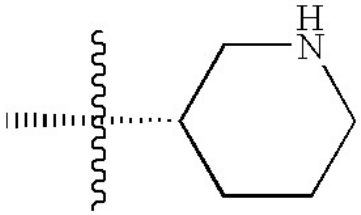


Figure 10.12: R12 Error

R20 Error (Tautomeric Bond): Our current implementation does not yet include the rules we have discussed in Section 9.2. Hence whenever any of those bonds is encountered, it will be mis-recognised. Figure 10.13 shows an image that contains a tautomeric bond. As we have not yet implemented it, it is currently captured as two separate bonds (a single bond and a dashed bond). However, it is worth mentioning that this bond is represented as an aromatic bond in the corresponding solution MOL file.

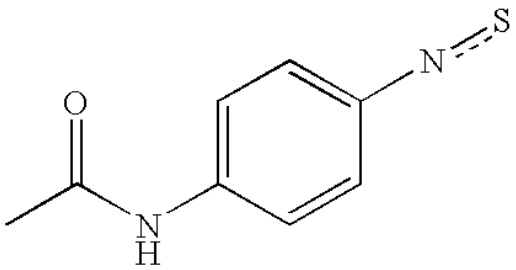


Figure 10.13: R20 Error

Unhandled Bond Type - Dashed Dative Bond: Our rules do not handle the dashed dative bond shown in Figure 10.14. Therefore, whenever a diagram that has such a bond is input to our tool, our tool interprets it as a dashed bond (R7) although the corresponding solution MOL file interprets it as a planar single bond.

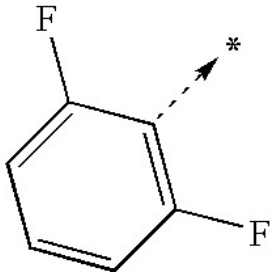


Figure 10.14: Unhandled Bond Type

10.3.2 Recognition Errors due to Vectorisation

Touching Components: It is common to have touching components in images of chemical structure diagrams. These include touching characters (ligatures are very common), characters glued to symbols, characters touching bonds, symbols touching bonds and bonds touching bonds (Figure 10.15). Both OSRA and our dataset were examined for touching components. The OSRA dataset has 1106 (19.29%) images with touching components (including ligatures), and 114 (1.99%) images excluding ligatures. On the other hand, our dataset has 246 images with touching components (4.29%).

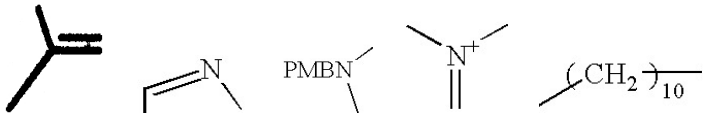


Figure 10.15: Various Touching Components

This causes several vectorisation errors, and subsequently leads to mis-recognition. However, common ligatures can be handled during the OCR process (as explained in Appendix A). A robust segmentation method is required to improve the performance of MolRec in particular and chemical structure diagram recognition tools in general.

Broken Characters: It is not uncommon to have broken components in images of chemical structure diagrams. Currently we do not handle broken characters such as those shown in Figure 10.16. These were mainly encountered in the TREC11 and CLEF12 datasets.

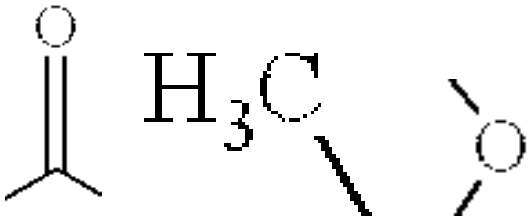
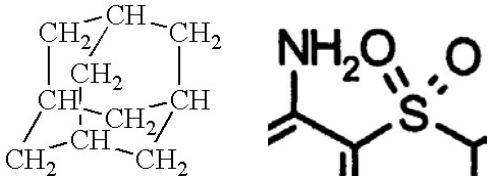


Figure 10.16: Example Broken Characters

This causes several vectorisation errors, and subsequently leads to mis-recognition. A robust segmentation method is required to improve the performance of MolRec in particular and chemical structure diagram recognition tools in general.

Atom Grouping Errors: OCR classification errors and incorrect grouping of atoms, symbols and bonds were also encountered. These also include a few very ambiguous images that are hard to interpret even visually such as the ones shown in Figure 10.17.



(a) (b)

Figure 10.17: Ambiguous Connectivity of Atoms and Bonds

Figure 10.17(a) fails because of difficulty in correctly forming chargroups and in identifying connectivity of different bonds. Figure 10.17(b) illustrates an extremely difficult to interpret chargroup. The NH_2O is actually formed of two chargroups, the NH_2 which is connected to the single bond on the left and the O which is connected to the double bond on the right. It is almost impossible to automatically handle such patterns without using a brute-force approach.

Superatoms: Another common reason for recognition errors is in MolRec choosing an incorrect permutation of superatom connections for superatoms which have more than one connection to the main structure (Figure 10.18).

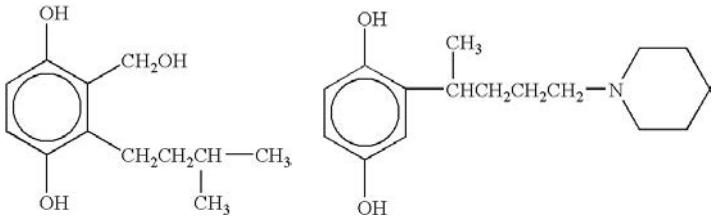


Figure 10.18: Superatoms with 2 or more connections to the main structure

To our knowledge, there exists no robust approach for solving this problem. The use of superatoms in chemical structure diagrams and the difficulty in interpreting their connectivity to the structure they are used in were explained in Sections 3.1.5.3 and 8.3 respectively.

10.4 Summary

This chapter provided a statistical evaluation of MolRec, a tool developed to demonstrate and examine the validation, robustness and performance of our rule-based system. The tool shows that using the rule-based approach to clearly define different bond conventions and arrangements provides a mechanism for accurate identification of such patterns. The provided results show that MolRec outperforms state-of-the-art tools and suggests a need for further research in the field of automatic diagram recognition.

CHAPTER 11

CONCLUSIONS & FUTURE WORK

11.1 Conclusions

In this thesis, we have presented an approach for the recognition of chemical structure diagrams from images. By tackling this challenge in a novel way, we have developed a rule-based system with extremely strict definitions for various bond formations and arrangements. Our approach is not only robust in the sense that it copes with a wide range of bond sketching styles, but also is extremely efficient to develop and easy to extend when needed.

The aim of this thesis was to address the following hypotheses:

- A strictly rule-based approach where diverse bond drawing styles are captured using a list of strict rules can be used to accurately and efficiently recognise chemical structure diagrams.
- The rule framework can be flexible to allow for easy extension to incorporate additional diagram elements.
- The approach can be efficiently implemented in a system that can match the performance of other contemporary approaches in chemical diagram recognition.
- The extraction of geometrical primitives from chemical diagrams can be improved using novel approaches to the detection of line endpoints and geometrical objects in the shape of solid triangles.

We have developed MolRec, a tool that has the implementation of our rules. We have shown through the experimentation with it using several benchmark chemical structure diagram datasets that the use of image thinning, line simplification algorithm and the expanding disc indeed yields higher standard results in terms of determining line endpoints and shapes in the form of solid triangles and bold lines. We have also shown that with our rule-based approach the identification of diverse bond drawing styles is robust, efficient and easy to extend. The high quality results we achieved in two international events demonstrate the superiority of our approach in comparison against leading systems.

11.2 Future Work

After the extensive experimentation of the work in this thesis with several benchmark chemical structure diagram datasets and presenting it to the research community, we have compiled the following list of possible areas where we can improve our system.

11.2.1 Improve OCR

The OCR module in MolRec does not currently handle broken and touching characters. This is a known difficult problem in document analysis. The difficulty of this problem is two fold. Firstly, the correct detection of touching characters is a challenge on its own. And secondly, the accurate identification of the touching characters and precisely segmenting them is another difficult task. We believe that an improved character recogniser and touching and broken symbol correction would considerably improve the system.

11.2.2 Better Bond Segmentation

MolRec does not currently handle touching bonds. In general, touching components in chemical structure diagrams include:

- Touching characters, we discussed this in the previous section
- Characters touching bonds
- Some cases of bonds accidentally touching bonds, usually due to ink bleed between close parallel lines making a connection that should not be there

While a number of solutions have been proposed in the literature to handle touching characters and the segmentation of text and graphics in document images, this problem is notoriously difficult.

11.2.3 Handle Markush Structures

Handling Markush structures is one interesting area of research as it involves several problems. For example, the parentheses used to indicate frequency are often glued to bond

lines and the use of a straight line segment to indicate position variation. So these two are OCR and segmentation problems. Another problem is that MOL files are limited and cannot be used to represent Markush structures [80]. Not having a standard extension or alternative to MOL files suitable for the representation and comparison of diagrams with Markush structures is a severe handicap to further development and testing of chemical diagram recognition systems.

11.2.4 Superatoms and Stereocentre

We have discussed and provided our solutions to the problems of determining the connectivity of superatoms to the main structure and guessing the stereocentre of dashed, bold and dashed bold bonds. We plan to gain more domain knowledge and attempt to solve these two problems more efficiently.

11.2.5 Examine Flexibility of our Approach

We believe that our approach is flexible and can be easily employed in other disciplines. In the coming sections we provide examples of how the rules would capture different components of electronic circuits (Section 11.2.5.1) and flowcharts (Section 11.2.5.2).

11.2.5.1 Electronic Circuit Recognition

Several attempts for the automatic interpretation of electronic circuits have been reported in the literature [71, 35, 36, 33]. An example circuit has been illustrated in Figure 11.1. Notice that the text from the provided figure has been manually removed. Text is usually used within electronic circuit diagrams to provide component values and label components.

By looking at the diagram, it is obvious that several components resemble many bond conventions. For example, wires connecting various components correspond to single bonds (R1). The saw-tooth pattern used to represent resistance correspond to wavy bonds (R12), the pattern used to represent batteries correspond to a dashed bond (R7) placed between two dashed wedge bonds each of which is formed of two dashes (R6) and the arrow used to indicate the existence of relay corresponds to a dative bond (R13). Also, the absence of a *dot* where more than two line segments (wires) on the diagram meet means they are not connected - this is

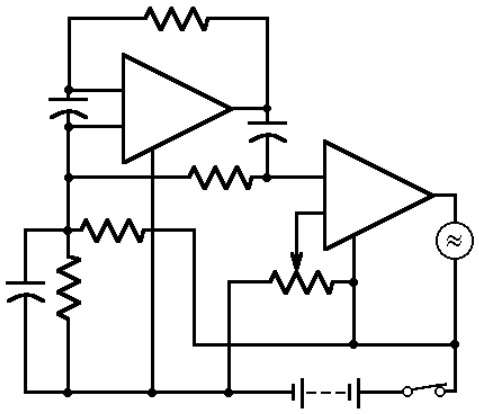


Figure 11.1: Example Electronic Circuit

equivalent to a closed bridge bond (R17).

11.2.5.2 Flowchart Recognition

Another research area is the recognition of flowcharts [112, 99]. Again, there is a clear resemblance between the components of flowcharts and bond conventions and arrangements. Two simple flowcharts have been shown in Figure 11.2.

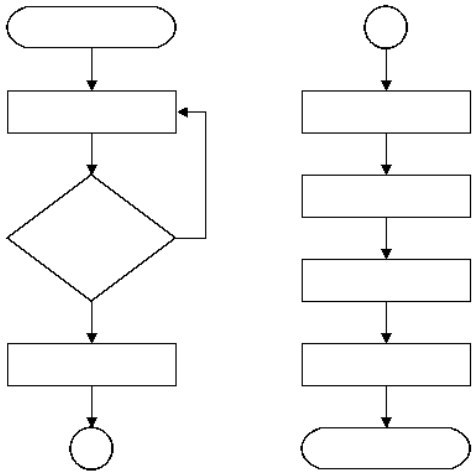


Figure 11.2: Example Flowchart

The circle object used to indicate the existence of an aromatic ring is used as a start, or end, symbol in flowcharts (remember a circle is identified as a primitive in our approach and is used as part of R16). Also, lines connecting different components correspond to single bonds (R1) and arrow used to show flow of control, and signify that flow passes to the symbol the arrow points to, resemble dative bonds (R13). Additionally, diamonds used to represent true/false tests

(decisions) are four sided cycles, whereas rectangles used to illustrate processing steps can be seen as multi-sided cycles (similar to those used in R16).

LIST OF REFERENCES

[1] Chemical Markup Language. http://www.xml-cml.org/. 3.1.4

[2] GREYCstoration: Open-source Algorithms for Image Denoising and Interpolation. http://cimg.sourceforge.net/greycstoration/. 4.4, 4.5

[3] Ocrad - GNU Project - Free Software Foundation (FSF). http://www.gnu.org/software/ocrad/ocrad.html. 4.4

[4] Optical Character Recognition (GOCR). http://sourceforge.net/projects/jocr/. 4.4, 4.5

[5] Special Search Types: Markush Structures. http://www.chemaxon.com/jchem/doc/user/query_markush.html. 2.1.4, 2.1.4

[6] Maybridge Catalogue Compounds for Drug Discovery, 2006. http://www.maybridge.com/default.aspx. 10.2.2

[7] M. E. Algorri, M. Zimmermann, C. M. Friedrich, S. Akle, and M. Hofmann-Apitius. Reconstruction of Chemical Molecules from Images. In *Proc. of EMBS 2007*, pages 4609–4612, 2007. 4.3

[8] M. E. Algorri, M. Zimmermann, and M. Hofmann-Apitius. Automatic Recognition of Chemical Images. In *Proc. of ENC 2007*, pages 41–46, 2007. 4.3

[9] Tetsuo Asano, Danny Z. Chen, Naoki Katoh, and Takeshi Tokuyama. Polynomial-time Solutions to Image Segmentation. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '96, pages 104–113, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics. 4.2

[10] The SDAG Group at University of Birmingham. Benchmark Dataset of Chemical Structures, 2012. Available at: http://www.cs.bham.ac.uk/research/

groupings/reasoning/sdag/uob_sdag_molecule_dataset.zip. 2.1.3, 10

[11] D. H. Ballard. Readings in computer vision: issues, problems, principles, and paradigms. chapter Generalizing the Hough Transform to Detect Arbitrary Shapes, pages 714–725. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. 4.5

[12] A. C. Beerel. *Expert Systems: Strategic Implications and Applications*. Ellis Horwood books in information technology. Ellis Horwood, 1987. 5.1

[13] Samuel J. Biondo. *Fundamentals of Expert Systems Technology: Principles and Concepts*. Computer Engineering and Computer Science. Ablex Publishing Corporation, 1990. 5.1

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. A.3

[15] S. Boyer. Optical Recognition of Chemical Graphics. In *Proceedings of ICDAR'93*, pages 627–631, 1993. 4.1

[16] Jonathan Brecher. Graphical Representation of Stereochemical Configuration (IUPAC Recommendation 2006). *Pure Appl. Chem*, 78(10):1897–1970, 2006. 2.1.3, 2.1.3, 2.1.3

[17] F. K. Brown. Chemoinformatics: What is it and How Does it Impact Drug Discovery. *Annual Reports of Medicinal Chemistry*, 33:375–384, 1998. 3

[18] Theodore L. Brown and Jr H. Eugene LeMay. *Chemistry: the Central Science*. Prentice-Hall, Inc, 1st edition, 1977. 2

[19] Roberto Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009. 6.4

[20] R. Casey, S. Boyer, P. Healey, A. Miller, B. Oudot, and K. Zilles. Optical Recognition of Chemical Graphics. *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 627–631, 1993. 4.1

[21] ChemAxon: Home Page. http://www.chemaxon.com/. 4.8

[22] ChemGuide. Helping you to understand Chemistry, 2012. http://www.chemguide.co.uk/. (document), 2.1.1, 2.1.2.1, 2.2, 2.1.2.1, 2.3, 2.1.2.2

[23] Octavia Chemicals. Find Chemicals in the ChemExper Chemical Directory, 2011. http://chemexper.otavachemicals.com/. 10.2.2

[24] ChemSpider. ChemSpider: the Free Chemical Database, 2012. http://www.chemspider.com/. 3.1, 10.2.2

[25] Q. Chen. Evaluation of OCR Algorithms for Images with Different Spatial Resolutions and Noises. Master's Thesis. 2003. A.1

[26] Kenneth L. Clarkson. Nearest-Neighbor Searching and Metric Space Dimensions. In *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006. A.3.2

[27] Conference and Labs of the Evaluation Forum (CLEF). Chemical Structure Recognition Task 2012, 2012. http://ifs.tuwien.ac.at/~clef-ip/chem.shtml. 10

[28] Joseph M. Cychosz. Graphics gems iv. chapter Efficient Binary Image Thinning using Neighborhood Maps, pages 465–473. Academic Press Professional, Inc., San Diego, CA, USA, 1994. 4.4

[29] Bonchev D. *Chemical Graph Theory: Introduction and Fundamentals*. Mathematical Chemistry. Taylor & Francis, 1991. 3.1.1

[30] A. Jain D. Trier and T. Taxt. Feature Extraction Methods for Character Recognition: A Survey. *Pattern Recognition*, 29(4):641–662, April 1996. 1

[31] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 243–320. Elsevier, Amsterdam, 1990. 5.1

[32] David H. Douglas and Thomas K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica*, 10(2):112–122, 1973. 4.7, 6.2, 8.6

[33] O. Ejofodomi, S. Ross, A. Jendoubi, M. Chouikha, and J.C. Zeng. Online Handwritten Circuit Recognition on a Tablet PC. In *AIPR04*, pages 241–245, 2004. 11.2.5.1

[34] Baader F and Nipkow T. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998. 5.1

[35] C.S. Fahn, J.F. Wang, and J.Y. Lee. A Topology-Based Component Extractor for Understanding Electronic Circuit Diagrams. *CVGIP*, 44(2):119–138, November 1988. 11.2.5.1

[36] G.H. Feng, C. Viard Gaudin, and Z.X. Sun. On-line Hand-drawn Electric Circuit Diagram Recognition using 2D Dynamic Programming. *PR*, 42(12):3215–3223,, December 2009. 11.2.5.1

[37] Igor V. Filippov, Dmitry Katsubob, and Marc C. Nicklaus. Optical Structure Recognition Application entry in Image2Structure task. In *The Twentieth Text REtrieval Conference Proceedings (TREC 2011)*, Gaithersburg, USA, Nov. 2011. National Institute of Standards and Technology (NIST). http://trec.nist.gov/. 4.4

[38] Igor V. Filippov and Marc C. Nicklaus. Optical Structure Recognition Software to Recover Chemical Information: OSRA, an Open Source Solution. *J. Chem. Inf. Model.*, 49(3):740–743, 2009. 4.4, 10.2

[39] L.A. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:910–918, 1988. 4.5

[40] Akio Fujiyoshi, Koji Nakagawa, and Masakazu Suzuki. Robust Method of Segmentation and Recognition of Chemical Structure Images in ChemInfty. In *Proc. of GREC-11*. Springer, 2011. 4.6

[41] J Gasteiger. *Handbook of Chemoinformatics*. Wiley-VCH, 1st edition edition, 2003. 2.1.4, 3.1.3.3, 3.1.5

[42] Stefania Gnesi, Ugo Montanari, and Alberto Martelli. Dynamic Programming as Graph Searching: An Algebraic Approach. *Journal of the ACM*, 28(4):737–751, 1981. 4.6

[43] Zicheng Guo and Richard W. Hall. Parallel Thinning with Two-subiteration Algorithms. *Commun. ACM*, 32(3):359–373, March 1989. 6.1.2

[44] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Number v. 1 in Computer and Robot Vision. Addison-Wesley Pub. Co., 1992. A.2.1

[45] Robert M Haralick and Linda D Shapiro. *Computer and Robot Vision*, volume 1. Reading, Mass. ; Wokingham : Addison-Wesley Pub. Co, 1991. A.2.1

[46] K.A. Hawick, A. Leist, and D.P. Playne. Parallel Graph Component Labelling with GPUs and CUDA. *Parallel Computing*, 36(12):655–678, 2010. A.2.1

[47] E Hill. On a System of Indexing Chemical Literature; Adopted by the Classification Division of the U. S. Patent Office. *Journal of Americal Chemist Society*, 8:478–494, 1900. 3.1.3.1

[48] P. Ibison, M. Jacquot, F. Kam, A. G. Neville, Richard W. Simpson, Christian A. G. Tonnelier, T. Venczel, and A. Peter Johnson. Chemical Literature Data Extraction: The CLiDE Project. *J Chem Inform Comput Sci*, 33(3):338–344, 1993. 4.2

[49] N. Itoh and H. Takahashi. Handwritten Numeral Verification Method using Distribution Maps of Structural Features. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 1258 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 150–159, August 1990. 4.1

[50] IUPAC. The IUPAC International Chemical Identifier (InChI), 2013. `www.iupac.org/inchi/`. 5.5

[51] Peter Jackson. *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999. 5.1

[52] Professor Joao Aires-de Sousa. Introduction to chemoinformatics, 2006. `http://joao.airesdesousa.com/qc/`. 2.1.4

[53] F. Kam, R.W. Simpson, C. Tonnelier, T. Venczel, and A.P. Johnson. Chemical Literature Data Extraction. Bond Crossing in Single and Multiple Structures. In *Proceedings of the 1992 Chemical Information Conference*, 1992. 4.2

[54] Péter Kardos, Gábor Németh, and Kálmán Palágyi. An Order—Independent Sequential Thinning Algorithm. In *Proceedings of the 13th International Workshop on Combinatorial Image Analysis*, IWCIA '09, pages 162–175, Berlin, Heidelberg, 2009. Springer-Verlag. 6.1.1.1, 6.1.1.2

[55] Daniel Karzel, Koji Nakagawa, Akio Fujiyoshi, and Masakazu Suzuki. Inconsistency-Driven Chemical Graph Construction in ChemInfty. In *Proc. of GREC-11*. Springer, 2011. 4.6

[56] S B Kotsiantis. Supervised Machine Learning : A Review of Classification Techniques. *Informatica*, 31(3):249–268, 2007. A.3.2

[57] P. Kral. Chemical Structure Recognition via an Expert System Guided Graph Exploration. Diploma Thesis. 2007. 3.1.3.1, 3.1.3.2

[58] Paul Kwok. A Thinning Algorithm by Contour Generation. *Commun. ACM*, 31(11):1314–1324, November 1988. 6.1.1.1

[59] L. Lam, S.W. Lee, and C.Y. Suen. Thinning Methodologies— a Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:869–885, 1992. 6.1.1.1

[60] Andrew R Leach and V J Gillet. *An Introduction to Chemoinformatics: Revised Edition*. Springer, 2007. 3, 3.1.5

[61] Day Light. SMILES - A Simplified Chemical Language, 2008. http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html. 1.2, 3.1.3.2, 3.1.3.2

[62] V. Lounnas and G. Vriend. Asterix: A Web Server to Automatically Extract Ligand Coordinates from Figures in PDF Articles. *J Chem Inf Model*, 52(2), February 2012. 4.8

[63] Mihai Lupu, Harsha Gurulingappa, Igor Filippov, Zhao Jiashu, Juliane Fluck, Marc Zimmermann, Jimmy Huang, and John Tait. Overview of the TREC 2011 Chemical IR Track. In *The Twentieth Text REtrieval Conference Proceedings (TREC 2011)*, Gaithersburg, USA, Nov. 2011. National Institute of Standards and Technology (NIST). http://trec.nist.gov/. 10.2.3

[64] The Marvin Abbreviation Group Collection. http://www.chemaxon.com/marvin-archive/5.7.0.0/marvin/chemaxon/marvin/templates/default.abbrevgroup. 8.3, 10.2.1

[65] MolConvert: Molecule File Conversion. http://www.chemaxon.com/marvin/help/applications/molconvert.html. 10

[66] Peter Murray-Rust, Joe Townsend, Sam Adams, Weerapong Phadungsukanan, and Jens Thomas. The Semantics of Chemical Markup Language (CML): dictionaries and conventions. *Journal of Cheminformatics*, 3(1):43, 2011. 3.1.4

[67] Koji Nakagawa, Akio Fujiyoshi, and Masakazu Suzuki. Ground-truthed Dataset of Chemical Structure Images in Japanese Published Patent Applications. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, DAS '10, pages 455–462, New York, NY, USA, 2010. ACM. 4.6

[68] ChemNet: Global Chemical Network. Chemnet's CAS Search, 2011. http://www.chemnet.com/cas/. 10.2.2

[69] M.S. Nixon and A.S. Aguado. *Feature Extraction & Image Processing*. Academic Press. Academic, 2008. 6.4

[70] The International Union of Pure and Applied Chemistry. http://www.iupac.org/. 3.1.3.3

[71] A. Okazaki, T. Kondo, K. Mori, S. Tsunekawa, and E. Kawamoto. An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition. *PAMI*, 10(3):331–341, May 1988. 11.2.5.1

[72] Open Babel: The Open Source Chemistry Toolbox. http://openbabel.org/. 5.5, 10, 10.2.4

[73] N. Otsu. A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9:62–66, January 1979. 1.2.1, 5.2, 8.6

[74] T. Ouyang and R. Davis. Chemink: a Natural Real-time Recognition System for Chemical Drawings. In *Proc. of IUI '11*, pages 267–276. ACM, 2011. 4.9

[75] Tom Y. Ouyang and Randall Davis. Recognition of Hand Drawn Chemical Diagrams. In *Proc. of 20th AAAI*, pages 846–851. AAAI Press, 2007. 4.9

[76] Kálmán Palágyi and Gábor Németh. Fully Parallel 3D Thinning Algorithms based on Sufficient Conditions for Topology Preservation. In *Proceedings of the 15th IAPR international conference on Discrete geometry for computer imagery*, DGCI'09, pages 481–492, Berlin, Heidelberg, 2009. Springer-Verlag. 6.1.1.2

[77] G. Paris. Meeting of the American Chemical Society, 1999. Quoted by: http://www.qsarworld.com/cheminformatics-education.php. 3

[78] J. Park, G. Rosania, K. Shedden, M. Nguyen, N. Lyu, and K. Saitou. Automated extraction of chemical structure information from digital raster images. *Chemistry Central*, 3(1), 2009. 4.5

[79] Jungkap Park, Ye Li, Gus R. Rosania, and Kazuhiro Saitou. ImagetoStructure Task by ChemReader. Nov. 2011. http://trec.nist.gov/. 4.5

[80] Florina Piroi, Mihai Lupu, Allan Hanbury, Alan P Sexton, Walid Magdy, and Igor Filippov. CLEF-IP 2012: Retrieval Experiments in the Intellectual Property Domain. Online Working Notes of CLEF 2012 Evaluation Labs and Workshop, http://clef2012.org, September 2012. 10.2.4, 11.2.3

[81] I. Pitas. *Digital Image Processing Algorithms*. Prentice Hall, 1993. A.2.1

[82] David M. W. Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2:37–63, 2011. http://www.bioinfo.in/contents.php?id=51. 5.5

[83] E. N. Ramsden. *A-Level Chemistry*. Stanley Thornes Publishers Ltd, 3rd edition, 1994. 2

[84] G.A. Ringland and D.A. Duce. *Approaches to Knowledge Representation: an introduction*. Knowledge-based and expert systems series. Research Studies Press, 1988. 5.1

[85] John B. Russel. *General Chemistry*. McGrow-Hill, Inc, 1st edition, 1980. 2

[86] Noureddin Sadawi. Recognising Chemical Formulas from Molecule Depictions. In *In Pre-Proceedings of the 8th IAPR International Workshop on Graphics Recognition (GREC 2009)*, pages 167–175, 2009. 1.4

[87] Noureddin M. Sadawi, Alan P. Sexton, and Volker Sorge. Performance of MolRec at TREC 2011 — Overview and Analysis of Results. In *The Twentieth Text REtrieval Conference Proceedings (TREC 2011)*, Gaithersburg, USA, Nov. 2011. National Institute of Standards and Technology (NIST). http://trec.nist.gov/. 1.4

[88] Noureddin M. Sadawi, Alan P. Sexton, and Volker Sorge. Chemical Structure Recognition: A Rule Based Approach. In *19th Document Recognition and Retrieval Conference (DRR 2012)*. SPIE, January 2012. 1.4

[89] Noureddin M. Sadawi, Alan P. Sexton, and Volker Sorge. MolRec at CLEF 2012 — Overview and Analysis of Results. In *The Third Conference and Labs of the Evaluation Forum Proceedings (CLEF 2012)*, Rome, Italy, Sep. 2012. Conference and Labs of the Evaluation Forum (CLEF). http://www.clef2012.org/. 1.4

[90] Peter Selinger. Potrace: a Polygon-based Tracing Algorithm, 2003. 4.4

[91] A Simon. Image Analysis and Content Retrieval of Printed Chemistry Documents, Ph.D. Dissertation, 1996. 4.2

[92] Jack Sklansky and Victor Gonzalez. Fast Polygonal Approximation of Digitized Curves. *Pattern Recognition*, 12(5):327 – 331, 1980. 4.2

[93] Viktor Smolov, Fedor Zentsev, and Mikhail Rybalkin. Imago: Open-source Toolkit for 2D Chemical Structure Image Recognition. Nov. 2011. http://trec.nist.gov/. 4.7

[94] Cambridge Soft. ChemBioFinder: an Online Chemistry and Biology Reference Database, 2011. http://chembiofinder.cambridgesoft.com/About.aspx. 10.2.2

[95] Eduard Sojka. A New algorithm for Detecting Corners in Digital Images. In *Proceedings of the 18th spring conference on Computer graphics*, SCCG '02, pages 55–62, New York, NY, USA, 2002. ACM. 4.5

[96] H. Stephen Stoker. *Introduction to Chemical Principles*. Prentice-Hall, Inc, 6th edition, 1999. 2

[97] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INFTY: an Integrated OCR System for Mathematical Documents. In *Proceedings of the 2003 ACM symposium on Document engineering*, DocEng '03, pages 95–104, New York, NY, USA, 2003. ACM. 4.6

[98] Symyx. CTfile Format, 2010. http://www.symyx.com/downloads/public/ctfile/ctfile.jsp. 1.2, 3.1.5

[99] W. Szwoch. Recognition, Understanding and Aestheticization of Freehand Drawing Flowcharts. In *ICDAR07*, pages 1138–1142, 2007. 11.2.5.2

[100] Karl Tombre and Salvatore Tabbone. Vectorization in Graphics Recognition: To Thin or not to Thin. *Pattern Recognition, International Conference on*, 2, 2000. 6.1

[101] Karl Tombre, Salvatore Tabbone, Loc Plissier, Bart Lamiroy, and Philippe Dosch. Text/Graphics Separation Revisited. In *Workshop on Document Analysis Systems (DAS)*, pages 200–211. Springer-Verlag, 2002. 4.5

[102] Text REtrieval Conference (TREC). Image2structure Task, 2011. http://trec.nist.gov/pubs/trec20/t20.proceedings.html. 8.3, 10

[103] David Tschumperl. Fast Anisotropic Smoothing of Multi-Valued Images using Curvature-Preserving PDE's. In *Research Report 'Les Cahiers du GREYC', No 05/01. Equipe IMAGE/GREYC (CNRS UMR 6072), Fvrier*, 2005. 4.5

[104] US Patent Office Complex Work Units. Chemical Structure Image Benchmark Dataset, 2012. Available at: http://cactus.nci.nih.gov/osra/uspto-validation.zip. 2.1.3, 8.3, 10

[105] Aniko T. Valko and A. Peter Johnson. CLiDE Pro: The Latest Generation of CLiDE, a Tool for Optical Chemical Structure Recognition. *J. Chem. Inf. Model.*, 49(4):780–789, 2009. 4.2

[106] T Venczel. Recognition of Primitives, Ph.D. Dissertation, 1993. 4.2

[107] D. Weininger. SMILES, a Chemical Language and Information System. *J Chem Inform Comput Sci*, 1:31–36, 1988. 1.2, 3.1.3.2

[108] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES 2 Algorithm for Generation of Unique SMILES Notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101, 1989. 3.1.3.2

[109] Wikipedia. Ramer-Douglas-Peucker Algorithm, 1972. [Online; accessed 30-July-2012]. (document), 6.8

[110] R. Yampolskiy. *Feature Extraction Approaches for Optical Character Recognition*. Briviba Scientific Press, 2007. A.1

[111] Mark C. K. Yang, Jong-Sen Lee, Cheng-Chang Lien, and Chung-Lin Huang. Hough Transform Modified by Line Connectivity and Line Thickness. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(8):905–910, August 1997. 4.5

[112] Zhenming Yuan, Hong Pan, and Liang Zhang. Advances in blended learning. chapter A Novel Pen-Based Flowchart Recognition System for Programming Teaching, pages 55–64. Springer-Verlag, Berlin, Heidelberg, 2009. 11.2.5.2

[113] Marc Zimmermann. Chemical Structure Reconstruction with chemoCR. Nov. 2011. http://trec.nist.gov/. 4.3

Appendices

APPENDIX A

OPTICAL CHARACTER RECOGNITION (OCR)

We have mentioned several times throught this thesis that we use optical character recognition to identify two types of geometric primitives (i.e. character groups and circles. In this appendix, we are going to provide a general overview of a simple and yet effective method of how to conduct a robust Optical Character Recognition (OCR). Since it is an essential part of chemical structure diagram recognition, the appendix we will attempt to give enough information to become familiar with different phases of an OCR process. We are not going to use the OCR process to identify characters only, but as we will explain, we will also use it to detect circles.

A.1 Optical Character Recognition

In pattern recognition, optical character recognition (OCR) is an active and important research field. Given digital images representing characters including alphabetic letters, digits and others, the goal of an OCR system is to correctly identify these characters without the intervention of a human [110]. This is typically achieved by extracting features from a training set of characters, and then finding a match for an input character using the same set of features extracted from the input character. This decision making process of finding a match is often denoted *classification*. According to [25], the general components of a typical OCR process are illustrated in Figure A.1.

As figure A.1 shows, the process starts by using a predefined dataset of characters for training. The dataset contains records of known character classes with their distinctive features. When an unknown character is to be identified, its features are extracted and a classification process is carried out to assign it one of the classes from the training set. The selection of a distinctive set of features and decision making to identify an unknown character represent the core of an OCR system.

A.2 Connected Component Labelling

For the identification of objects in a digital image, groups of connected *foreground* pixels must be located. Notice that we are going to use the term foreground pixels to refer to the non-white pixels of the image. Likewise, we are going to refer to white pixels as *background* pixels.

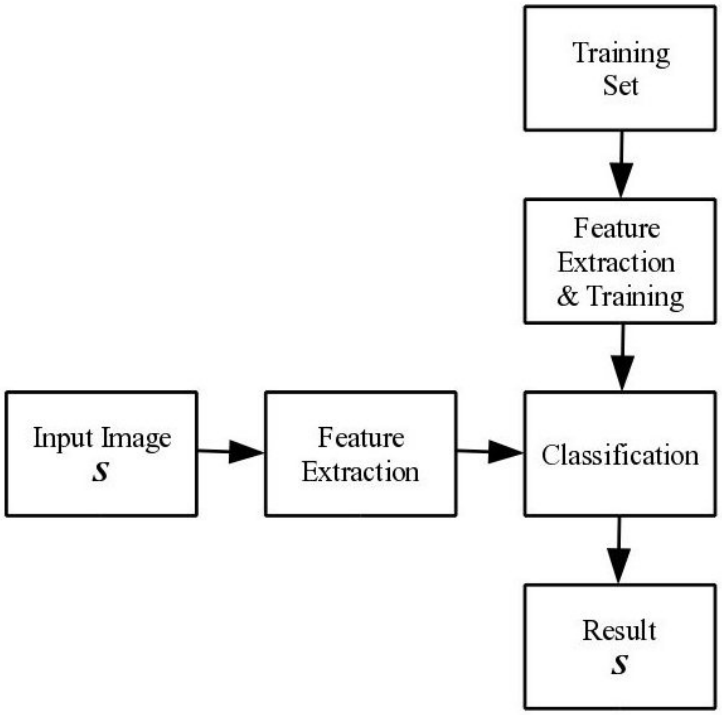


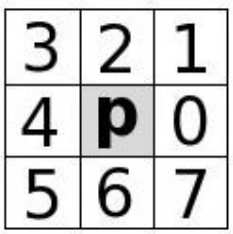
Figure A.1: Components of an OCR Engine

So, the objects in a digital image are the connected components of that image. Therefore, the process of identifying all groups of interconnected foreground pixels is called ***connected component labelling*** or ***CCL***. Formally, a connected component can be defined as follows:

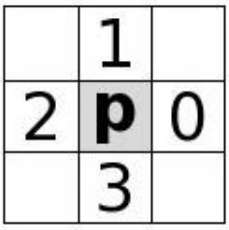
Definition 13 *A **Connected Component**, C, is a set of foreground pixels such that for every pair of pixels p_x, $p_y \in C$, there exists a path $p_x, \ldots, p_y$ which satisfies the following conditions:*

- *All pixels in the path are foreground pixels*
- *Every consecutive pair of pixels in the path are **neighbours***

The neighbourhood, or connectivity, of pixels can usually be either four-connected or eight-connected. As Figure A.2 shows, in a four-connected neighbourhood, pixels that have an edge in common with a pixel p are the pixel's neighbours. Whereas, in an eight-connected neighbourhood, the neighbours of p are the pixels that have an edge or a corner in common with it.



(a) Eight-connected Neighbourhood



(b) Four-connected Neighbourhood

Figure A.2: Two Types of Pixel Neighbourhoods

A.2.1 Calculating Connected Components

There are various ways of computing a connected component and we will summarise them as follows:

Using graph data structure [46]: The foreground pixels of a given image can be represented as a graph where each foreground pixel represents a node in the graph. A list of nodes that are connected to each other can be built, nodes can be considered connected using four, or eight,-connectivity as explained previously, and connected component(s) of the resulting graph can be computed.

Using Flood-fill algorithm: This approach is sometimes referred to as the *grass-fire* approach [81]. It begins by locating a foreground pixel and then searching its neighbours to obtain other pixels that connect to it. The neighbourhood of these pixels is then searched to find pixels that connect to them and so on until the connected component is obtained.

Using a two-pass algorithm with equivalence class resolution [45]: This algorithm works by assigning temporary labels to each foreground pixel. When a foreground pixel is encountered, the neighbours that have already been visited in the scan are examined. If none of these neighbours have been assigned a temporary label, the corresponding pixel is assigned a new label value. If a foreground pixel with two neighbours having different label values is encountered, one of the labels is chosen randomly and then it is recorded that these two label values refer to the same component. Now the temporary labels are examined and the subsets which refer to the same object are identified. This is denoted ***equivalence class resolution***.

Finally, the second pass is performed to re-label the pixels belonging to the same component .

Using Run Length Encoding [44]: In this approach, the image is represented as a collection of sequences, or runs, of foreground pixels. The starting pixel and the number of pixels in a sequence/run are used in analysing the connectivity. Connected sequences can be obtained using the equivalence classes approach. We are going to use this approach in our work because of its simplicity and efficiency. This method is efficient because each pixel in the image is scanned only once. By looking at the example image shown in Figure A.3, we notice that there are only three connected components, corresponding to three equivalence classes. Hence, we can obtain the following set of connectivity pairs:

- Run 1 is connected to Run 2
- Run 3 is connected to Run 4
- Run 5 is connected to Run 6

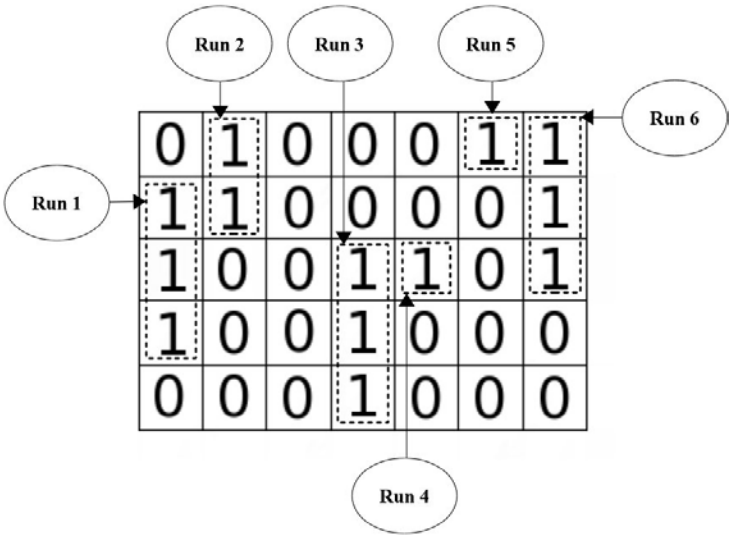


Figure A.3: CCL using Run Length Encoding

A.2.2 Bounding Boxes

The result of connected component labelling is usually the coordinates of the object's bounding box (Figure A.4). Using the definition of a connected component (Definition 13) as a basis, we can define coordinates of a bounding box as follows:

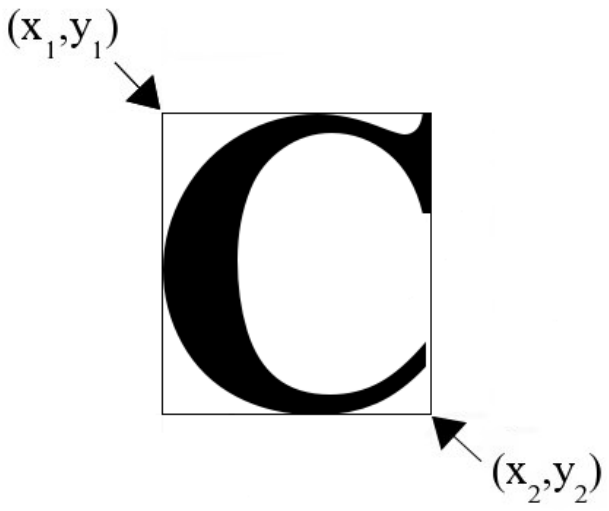


Figure A.4: Coordinates of a Character's Bounding Box

Definition 14 ***Coordinates of bounding box*** *of a connected component, C, are represented as a pair $((x_1, y_1), (x_2, y_2))$, such that.*

- $x_1 = \min_x\{C_x\}$ *where* $C_x \in C$
- $y_1 = \min_y\{C_y\}$ *where* $C_y \in C$
- $x_2 = \max_x\{C_x\}$ *where* $C_x \in C$
- $x_1 = \max_y\{C_y\}$ *where* $C_y \in C$

A.3 Performing OCR and Removal of Characters

As we have mentioned in Section A.1, OCR deals with the identification of any characters. Notice that a character in our context can be a letter, a digit or a symbol. A typical OCR engine finds the connected components (Section A.2.1) of an input image and attempts to associate each component, or group of components, with a predefined character class. Notice that a single connected component can have one or more characters. This can be useful when trying to identify two or more touching characters which is common especially in case of typographic ligatures.

The identification process involves calculating discriminative features from connected components and comparing them with precomputed features. This comparison is known as

classification. In the coming few sections, we will provide a simple method to extract features and carry out classification. Our feature extraction method will use a simple and yet effective and easy to conduct approach. We will extract a small number of features to avoid the *Curse of Dimensionality* phenomenon [14].

A.3.1 Feature Extraction

Unique characteristics that can represent a specific character are called ***features***. Features are unique in the sense that they should be relevant to the formation process of the character and they should be computed feasibly. This step is essential in the context of optical character recognition. Also, features are often calculated as vectors of real numbers. A suggested set of discriminative features and how to compute them includes the following:

1. **Density Features using Grid.** We will extract these features by superimposing a 3 by 3 grid on each character candidate as displayed in Figure A.5, hence zoning the sub-image [30], and then computing the percentage of foreground pixels in each zone.

   Formally, for a sub-image S of width w and height h, this is done by dividing the sub-image into 9 non-overlapping zones $Z_i,\ i = 0 \ldots 8$ each of size $m \times n$ where $m = \frac{w}{3}$ and $n = \frac{h}{3}$.

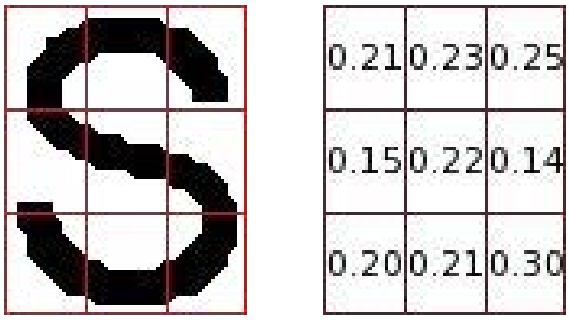
   In this case, the feature f_i is calculated as:

   $$f_i = \frac{N_i}{m \times n} \tag{A.1}$$

   Where N_i is the number of foreground pixels in the i^{th} zone.

   As for the example illustrated in Figure A.5, observe that the feature value in the left most zone of the top row in Figure A.5(b) corresponds to the left most zone of the top row in Figure A.5(a), the feature value in the middle zone of the top row of Figure A.5(b) corresponds to the middle zone of the top row in Figure A.5(a) and so on.

2. **The Aspect Ratio.** The aspect ratio r of an image can be defined as the ratio of the image's longer dimension to its shorter dimension. In other words, it is ratio of the



(a) Character S with 3×3 grid superimposed

(b) Grid foreground pixel percentages f_i

Figure A.5: Zoning of a character image.

image's width to the image's height. However, we are going to calculate it as follows:

$$r = \frac{w}{w+h} \tag{A.2}$$

It can be noticed that the denominator is $w + h$ instead of h. We are going to do this to guarantee that the value of the aspect ratio is always $0 < r < 1$. Also, it will help make feature calculation size invariant.

This results in ten features $(f_1, f_2, ..., f_9, f_{10})$. We are going to use these features in identifying objects as we will explain in the coming sections.

In the next point, we are going to show how we can compute another feature, the centre of gravity. Notice that we will not use it in the OCR process but we will use it in calculating the coordinates of atoms (or nodes) as we will explained in more detail in Section 8.5.

3. **Centre of Gravity:** The centre of gravity, or centroid of a region can be computed as the arithmetic mean of the coordinates of foreground pixels in the x and y directions. For a binary image I, where $I_{(x,y)} \in \{0, 1\}$, the components of the centroid $(\bar{x}, \bar{y})$ can be calculated as:

$$\bar{x} = \frac{\sum\limits_{(x,y)\in I} x}{|I|} \qquad and \qquad \bar{y} = \frac{\sum\limits_{(x,y)\in I} y}{|I|} \tag{A.3}$$

Where $|I|$ is the area of I, which corresponds to the number of foreground pixels in I.

Also, *normalised* components of the centre of gravity can be obtained by using the following equations:

$$\bar{x_c} = \frac{\bar{x}}{w} \qquad and \qquad \bar{y_c} = \frac{\bar{y}}{h} \tag{A.4}$$

where w and h are the image's width and height respectively.

A.3.2 Classification

In object recognition in general, any input should be assigned to one of possible object classes, or models. This process is denoted ***classification***. A predefined library, or dataset of objects including their extracted features and classes, or models is usually employed. Hence, a classifier's job is to search this library and find a match for a given input using a similarity measure.

We are going to use a metric to measure the similarity between two objects A and B in a set U. Namely, we are going to use a Euclidean distance D (Equations A.5 and A.6) to find a best match. According to [26] the Euclidean distance should satisfy the following conditions:

1. Positivity: $D(a, b) \geq 0$;
2. Zero self-distance: $D(a, a) = 0$;
3. Symmetry: $D(a, b) = D(b, a)$;
4. Isolation: $a \neq b \Rightarrow D(a, b) > 0$;
5. The triangle inequality: $D(a, c) \leq D(a, b) + D(b, c)$.

We are going to develop and use a database of characters for training and identification of unknown input. Each entry in the database will consist of a character class name and the precomputed set of features. Table A.1 shows a sample database of features computed using the grid, section A.3.1, where the first field ($f1$) is the density in the top left grid, the second field

($f2$) is the density in the middle top grid and so on. The tenth field ($f10$) represents the aspect ratio.

Format of Features and Classes						
Case	f1	f2	f3	...	f10	Class
	0.41666	0.58333	0.50000	...	0.40000	2
	0.12500	0.50000	0.50000	...	0.40000	2
	0.55681	0.44886	0.51704	...	0.40243	8
	0.33333	0.26666	0.40000	...	0.40740	8
	0.21428	0.14285	0.35714	...	0.47500	C
	0.20000	0.45000	0.55000	...	0.45161	C
	0.32653	0.06122	0.34693	...	0.51162	H
	0.41558	0.00000	0.42207	...	0.43421	H
	0.56666	0.20000	0.50000	...	0.45454	N
	0.68888	0.07222	0.38888	...	0.44444	N
	0.33750	0.33333	0.34583	...	0.47872	O
	0.28571	0.18367	0.26530	...	0.50000	O
	0.36250	0.31250	0.36250	...	0.44444	S
	0.44791	0.42187	0.53645	...	0.43678	S
	0.58333	0.45833	0.62500	...	0.40625	R

Table A.1: An Example Dataset

Then, we are going to use these features in a classification process to assign a character label to each character. We are going to achieve this by calculating the feature vector of the different components of the image and then finding the best match using Euclidean distance (Equation A.5) and a certain threshold that is computed empirically.

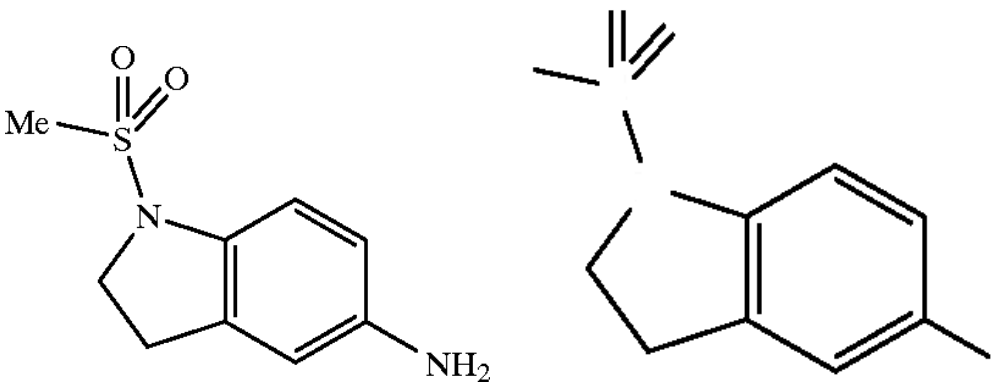
Given two feature vectors $\vec{x} = (x_1, x_2, ..., x_n)$ and $\vec{y} = (y_0, y_1, ..., y_n)$, we are going to calculate the Euclidean distance between these two vectors as.

$$d = \sqrt{\Delta^2} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{A.5}$$

The classification technique we have adopted in this work is often denoted *Instance-based learning* or *Nearest-neighbour search*. Please refer to [56] for an overview of various classification techniques.

A.3.3 Removal of Symbols

We are going to remove all identified characters from the input image. The removal process is simply turning all foreground pixels representing any of the identified characters into background pixels (i.e. turning them into white). As we can see in Figure A.6, the input image is now stripped of all recognised characters and it is ready for vectorisation (we are going to talk about vectorisation in detail in Chapter 6). The characters that we identify from Figure A.6(a) are “M”, “e”, “O”, “O” , “S”, “N”, “H” and “2”. Notice that “M” and “e” form one atom. This is also the case with “N”, “H” and “2”. Additionally, notice that there is a high resemblance between the character “O” and a circle. We are going to talk about joining characters that form the same atom and distinguishing between “O” and a circle in the following subsections.



(a) Molecular Structure Diagram (b) Symbols are Removed

Figure A.6: The Removal of Symbols from a Molecular Structure Image

A.4 Formation of Atoms (Chargroups)

This is the process of grouping characters that are part of the same word. As we have mentioned before, characters can be letters, numerals or symbols. We are going to perform character grouping in a top-down, left-right manner as the characters are usually written. Also, we are going to carry it out according to the inter-character distance, which is the distance between the characters’ borders (the bounding boxes of their corresponding connected components).

For every connected component, C_i where $i = 1, 2, ..., k$, of the k characters found in a chemical structure diagram image, the coordinates, $((x_1, y_1), (x_2, y_2))$ indicating four extreme points in four directions are obtained as explained in Section A.2. However, we are going to refer to these points in a somewhat different way in the remaining sections of this appendix. As we have shown in Figure A.7, we are going to use $L(C_i)$ for extreme left , $R(C_i)$ for extreme right , $T(C_i)$ for extreme top and $B(C_i)$ for extreme bottom.

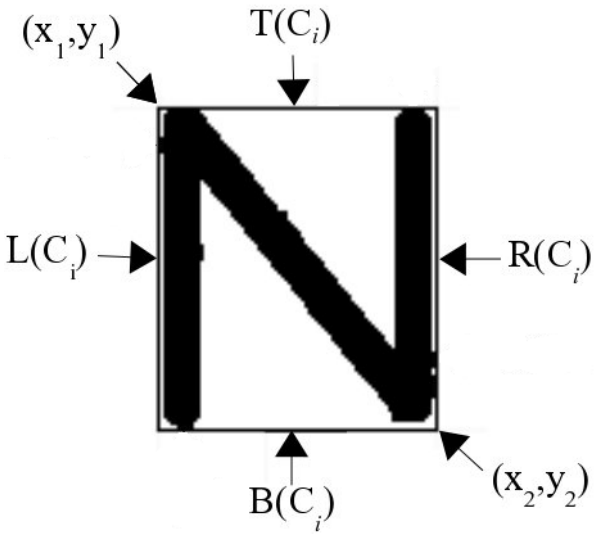


Figure A.7: Coordinates and Extreme Points of a Character

In addition to that, we are going to perform characters grouping according to the following three parameters:

1. An alignment function, F.

2. The consecutivity of characters.

3. A threshold distance σ separating the characters.

A.4.1 Alignment of Characters

Given two connected components C_a and C_b which correspond to any two characters in the image, we will have a function F that should guarantee the alignment of C_a and C_b. Remember that a character can be a letter ($'a', 'b', \ldots, 'z', 'A', 'B', \ldots, 'Z'$), a digit ($'0', '1', '2', \ldots, '9'$) or a symbol ($'(', ')', ' + ', ' - '$).

This function guarantees that characters on the same horizontal line are grouped from left to right, whereas characters on the same vertical line are grouped from top to bottom.

This function means that, a value of 1 is returned when examining the horizontal alignment of:

- a character against a character
- a digit against a digit
- a character against a digit, or vice-versa
- a symbol against a character or against a digit, or vice-versa

However, it only returns a value of 1 when examining the vertical alignment of a character against a character. In other words, we will only allow horizontal alignment between characters, digits or symbols, whereas we will only allow vertical alignment for characters only.

Determining the alignment of characters also helps identifying charge signs. In chemical structure diagrams, the plus sign is usually used to indicate the existence of positive charge and the minus sign is used to indicate the existence of negative charge. Their size is usually smaller than other components so they can be identified using a size filter. Additionally, it is common to place these signs near the top left or top right of other characters. Hence, using a function such as the one described above further assists the accurate spotting of such symbols.

A.4.2 Consecutivity of Characters

The consecutivity of components means that they are in a left-to-right or top-to-bottom order. We are going to use the following function to determine the consecutivity of any two connected components.

Given a connected component C_s, we are going to identify the consecutive component C_q of C_s by using the following function:

$$q = \begin{cases} \min_x\{L(C_x) - R(C_s)\} & \text{where } L(C_x) > R(C_s)) \quad \text{In horizontal direction} \\ \min_x\{B(C_x) - T(C_s)\} & \text{where } T(C_x) > B(C_s)) \quad \text{In vertical direction} \end{cases}$$

A.4.3 Inter-character Distance

Information on inter-character distance is required for the process of atom formation. We will define this distance as the horizontal or vertical distance between any two consecutive characters. We are going to use a predefined value, σ, to decide whether any two characters belong to the same atom. The Euclidean distance can be used to measure the distance between any two characters. Since the extreme points can be obtained as explained previously, they can be used as the parameters as follows:

Given two points (x_1, y_1) and (x_2, y_2), we calculate the Euclidean distance between these two points as.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{A.6}$$

A.4.4 Forming the Atoms

To form atoms, any two identified characters, C_1 and C_2, are grouped if the following three conditions are satisfied:

1. $F(C_1, C_2) = 1$
2. C_1 and C_2 are consecutive
3. The distance d between C_1 and C_2 is $\leq \sigma$

A.5 Identifying Rings/Circles

We have mentioned that circle objects are used to indicate the existence of aromaticity in Section 2.1.2.3. We have also talked about the high similarity between the letter "O" and a circle in Section A.3.3. We are going to exploit this resemblance when trying to identify aromatic rings in images of chemical structure diagrams. Instead of having a separate module to recognise circles, they can be automatically recognised using a training dataset which contains either circle templates or capital O images in font(s) that draw the letter "O" as circular as possible.

An example showing the use of a circle to illustrate the existence of an aromatic ring in aspirin is depicted in Figure A.8. The high similarity between the shapes of the circle and the

"O" atom can be easily noticed.

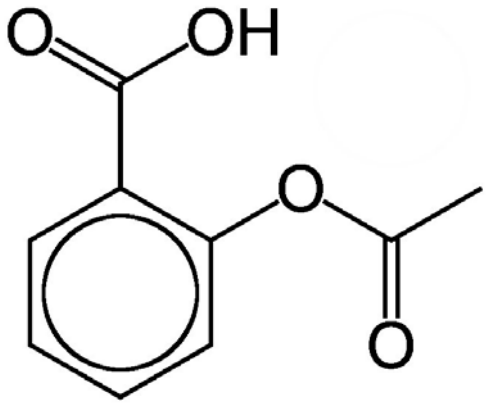


Figure A.8: Circle used to represent aromatic ring in aspirin

It can be error prone to just rely on the results of the previously explained classification approach to distinguish between an "O" and a circle. Therefore, we will identify all such objects as the same character, say O, and will process them further. For instance, it can be noticed from Figure A.8 that the circle has a larger circumference, i.e. a larger bounding box. Hence we are going to use a size filter to separate the two. Spotting a circle which is a part of an aromatic cycle will be explained in Section B.12.

A.6 Summary

We have provided a brief overview of optical character recognition throughout this appendix. We have made an attempt to familiarise the reader with the stages of a typical OCR engine by providing a simple way of building an OCR engine. Not only this, but we have also introduced a method to form atom names in chemical diagrams. The method included various conditions that can be used to avoid incorrectly joining characters. Additionally, we have provided a simple approach to distinguish circles and "O" letters. By doing this, we have introduced a way to identify two of the five geometric primitives. This leaves us with the three remaining primitives. We have already introduced our method for retrieving these in Chapter 6.

APPENDIX B

THE RULES SYSTEM

We have explained four of our 18 rules in detail in Chapter 7 and, as we have mentioned several times throughout this thesis, we are going to explain the remaining 13 rules in this appendix. Observe that, unlike we have done in Chapter 7, we are not going to provide sample source code for any of these rules.

B.1 R2: Double Planar Bond

The planar double bond is usually drawn as two very close parallel line segments. We identify a double planar bond if the following conditions are true:

1. $L = \{l_1, l_2\}$ is a set of two line segments
2. $\forall l \in L : length(l) > wb$
3. $\forall l \in L : width(l) < bbw$
4. $l_1 \parallel_{bs}^{ol} l_2$
5. There is no line segment $l \notin L$ such that $l_1 \parallel_{bs}^{ol} l$ or $l_2 \parallel_{bs}^{ol} l$

Consequence: Cutting(l_1, l_2) will yield a double bond as well as at most two new line segments.

Figure B.1: A Double Planar bond

This rule identifies a double line which is conformed of two parallel line segments of similar length. As shown in Figure B.1, there exists no third line segment of any length that is parallel to and very close to any of these two line segments.

Observe that R2.2 guarantees that the rule does not capture any line segment that can be a hollow wedge base (R10 and R11) and R2.3 guarantees that the rule does not capture any bold line (R15).

Notice that this rule is one of the two rules that may add new geometric segments to the working memory (i.e. our set of primitives which is used as input). The other rule is R3 which has been explained in Section 7.5.2. As we illustrated in Figure 7.12 in Section 7.3, if this rule applies cutting successfully, it can either add one or two new line segments to the set of primitives (Figures 7.12(a) and 7.12(b) respectively).

B.2 R4: Dashed Bold Bond vs Triple Bond

In some drawing styles, a triple bond and a dashed bold bond have similar appearance. As illustrated in Figure B.2, this is because the pattern is drawn as three short parallel line segments. Despite being visually straightforward to interpret, automatically identifying what the pattern represents can be problematic. Therefore, whenever a pattern like this is encountered; it is necessary to use information from the environment around the pattern to accurately decide whether it is a triple bond or a dashed bold bond. We will use the following conditions to capture such pattern and yield a special bond type which is disambiguated later:

Figure B.2: Triple Bond or Dashed Bold Bond?

1. $L = \{l_1, l_2, l_3\}$ is a set of line segments
2. $\forall l \in L : length(l) \in bdl$
3. $\forall l \in L : width(l) \in bdw$
4. Centre points of all $l \in L$ are approximately collinear
5. $l_1 \parallel^{bdl}_{ds} l_2$ and $l_2 \parallel^{bdl}_{ds} l_3$
6. There is no $l \notin L$ such that $l_1 \parallel^{bdl}_{ds} l$ or $l_2 \parallel^{bdl}_{ds} l$ or $l_3 \parallel^{bdl}_{ds} l$

Consequence: Either dashed bold bond or triple bond.

Observe that R4.2 guarantees that the captured line segments are not identified as a triple bond immediately (R3) and R4.6 guarantees that this pattern is not a dashed bold bond (R8).

B.3 R5: Dashed Wedge vs Triple Bond

This is similar to the previous case (rule R4 in section B.2), the only difference is that the line segments forming this pattern have different lengths which makes it appear as a pattern involving one or two implicit nodes (Figure 7.12).

Like the previous case, whenever a pattern like this is encountered, the precise identification of the bond type is postponed until context information is used. We will use the following conditions to capture such pattern and yield a special bond type which needs to be precisely identified later:

1. $L = \{l_1, l_2, l_3\}$ is a set of line segments
2. $length(l_1) > length(l_2)$ and $length(l_2) > length(l_3)$
3. Centre points of all $l \in L$ are approximately collinear
4. $l_1 \parallel_{ds}^{bdl} l_2$ and $l_2 \parallel_{ds}^{bdl} l_3$
5. There is no $l \notin L$ such that $l_1 \parallel_{ds}^{bdl} l$ or $l_2 \parallel_{ds}^{bdl} l$ or $l_3 \parallel_{ds}^{bdl} l$

Consequence: Either dashed wedge bond or triple bond group with implicit nodes.

Observe that R5.2 guarantees that the captured line segments will not be captured by R4, and R5.4 guarantees that the pattern is not captured by R6. Also, R5.5 guarantees that the pattern is not captured by R9.

B.4 R6: Dashed Wedge vs Double Bond

This configuration is similar to that captured in rules R4 and R5 (Sections B.2 and B.3) except that the number of line segments involved is two rather than three. This means the

pattern could be either a double bond or a dashed wedge bond. We will use the following conditions to capture such pattern and yield a special bond type which needs to be precisely identified later:

1. $L = \{l_1, l_2\}$ is a set of line segments
2. $length(l_1) > length(l_2)$
3. $l_1 \parallel_{ds}^{bdl} l_2$
4. There is no $l \notin L$ such that $l_1 \parallel_{ds}^{bdl} l$ or $l_2 \parallel_{ds}^{bdl} l$

Consequence: Either dashed wedge bond or double bond.

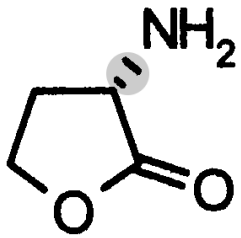


Figure B.3: A Dashed Wedge Formed of only Two Dashes (shadowed)

Observe that this was based on observation and analysis of existing literature. Many chemical structure images were investigated and whenever two very short line segments were used in such a pattern, their lengths were always found to be different as illustrated in Figure B.3. This means it would be straight forward to design a rule to capture the same pattern if the lengths were the same (this would make it a dashed bold bond vs a double bond).

Notice also that R6.2 guarantees that the captured line segments will not be captured by R4 and R6.4 guarantees that the pattern is not captured by R5.

B.5 R8: Dashed Bold Bond

The dashed bold bond is usually drawn as a group of short parallel line segments that can be contained in a rectangle. As Figure B.4 illustrates, the length of each of these line segments is usually shorter than that of a normal line segment and their thickness can be larger than that

of a normal line segment. Also, as shown in (Figure B.4 and B.5), they are evenly spaced apart and a straight line can pass through their centre points.

Figure B.4: A Dashed Bold Bond

We will identify a dashed bold bond if the following rules are satisfied:

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 4$, is a set of line segments
2. $\forall l \in L : length(l) \in bdl$ approximately bdl
3. $\forall l \in L : width(l) \in bdw$
4. The centre points of the elements of L are approximately collinear
5. No two elements of L have a separation distance of less than the minimum of ds
6. Two elements of L, called the end elements, dash-neighbour wrt. ds precisely one other element of L. All other elements of L, called internal elements, dash-neighbour wrt. ds precisely two other elements of L

Consequence: L forms dashed bold bond with endpoints given by the endpoints of the minimal line segment that contains the centre points of the long sides of the line segments $l_1, \ldots, l_n$. The new dashed bold bond has a down direction with unknown start and end.

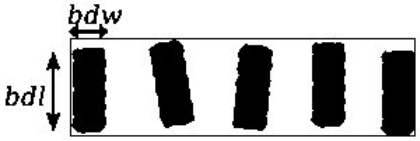


Figure B.5: Dashes in a Dashed Bold Bond

Observe that R8.2 and R8.3 guarantee that the captured line segments will not be captured by R9. Also, 8.4 guarantees that the pattern will not be captured by R7. Also, R8.5 guarantees that the pattern is not captured by R12.

B.6 R9: Dashed Wedge Bond

A dashed wedge bond has the form of a group of line segments which are short, parallel and of increasing length . These line segments are usually evenly spaced apart and they can be contained in a triangle, or a triangular shaped polygon. As Figure B.6 shows, the two end line segments are usually the shortest and longest (or the opposite depending on where one starts). The length of such line segments increases gradually as one moves from the shortest towards the longest line segment.

Figure B.6: A Dashed Wedge Bond

Additionally, similar to the line segments forming a dashed bold bond in rule R8 (Section B.5), the centre points of these line segments are collinear and their thickness can be larger than that of a normal line segment.

Figure B.7: Dashes in a Dashed Wedge Bond

We will capture a dashed wedge bond if the following conditions are met:

1. $L = \{l_1, \ldots, l_n\}$, where $n \geq 4$, is a set of line segments
2. $\forall l \in L : length(l) \in dl$ approximately dl
3. The centre points of the elements of L are approximately collinear
4. No two elements of L have a separation distance of less than the minimum of ds
5. Two elements $e_l, e_u \in L$ and $e_l < e_u \geq wb$, called the end elements, dash-neighbour

precisely one other element of L. All other elements of L, called internal elements, neighbour precisely two other elements of L

6. One neighbour of any internal element must be strictly longer than the internal element, the other neighbour must be strictly shorter

7. Length of elements of L is strictly monotonically increasing from e_l to e_u

Consequence: L forms dashed wedge bond with endpoints given by the endpoints of the minimal line segment that contains the centre points of the long sides of the line segments $l_1, \ldots, l_n$. The new dashed bold bond has a down direction with unknown start and end.

Observe that R9.2, R9.5 and R9.7 guarantee that the captured line segments will not be captured by R8. Also, R9.4 guarantees that the pattern will not be captured by R12.

B.7 R10: Hollow Wedge Bond - case 1

A sample hollow wedge bond is shown in Figure B.8. It consists of three line segments forming an isosceles triangle. Two of the three line segments are of similar length and are longer than the remaining line segment. The third (short) line segment represents the base of the triangle.

Figure B.8: A Hollow Wedge Bond - Case 1

We will identify a hollow wedge bond using the following conditions:

1. $L = \{l_1, l_2, l_3\}$, is a set of line segments

2. $\forall l \in L : width(l) < bbw$

3. $l_1, l_2 \in L : length(l_1) = length(l_2)$

4. $l_3 \in L : length(l_3) \in wb$

5. $length(l_1) > length(l_3)$ and $length(l_2) > length(l_3)$

6. For any $l_i \in L$ and $l_j \in L$, l_i and l_j have exactly one end point in common

Consequence: Hollow wedge bond from joint end point of l_1, l_2 and centre point of l_3. The new hollow wedge bond has an up direction from joint end point of l_1, l_2 to centre point of l_3.

Observe that R10.2 guarantees that none of the captured line segments can be captured by R15. Also, R10.4 guarantees that the line segment will not be captured by R1. Additionally, observe that this rule may conflict with R1 because the line segments l_1 and l_2 appear as valid single bonds and may also conflict with R11. This is a typical situation where these rules can be triggered first and then fired after employing a conflict resolution strategy.

B.8 R11: Hollow Wedge Bond - case 2

Ordinary hollow wedge bonds are captured using rule 10 (section B.7). However, the base of the triangle representing the bond can sometimes consist of two short line segments rather than one. As depicted in Figure B.9, this is due to some drawing styles when a line segment is connected to the middle of the base. Therefore, a slightly different set of conditions can be used to identify such cases.

We will use the following conditions to capture a hollow wedge bond which base is connected from the middle to another line segment:

1. $L = \{l_1, l_2, l_3, l_4\}$, is a set of line segments

2. $\forall l \in L : width(l) < bbw$

3. l_3 and l_4 are collinear and have one end point in common. Let l_5 be the minimal line segment that contains l_3 and l_4, and $L' = \{l_1, l_2, l_5\}$, is a set of lines

4. $l_1, l_2 \in L' : length(l_1) = length(l_2)$

5. $l_5 \in L' : length(l_3) \in wb$

6. $length(l_1) > length(l_5)$ and $length(l_2) > length(l_5)$

7. For any $l_i \in L'$ and $l_j \in L'$, l_i and l_j have exactly one end point in common

Consequence: Hollow wedge bond from joint end point of l_1, l_2 and centre point of l_5. The new hollow wedge bond has an up direction from joint end point of l_1, l_2 to centre point of l_5.

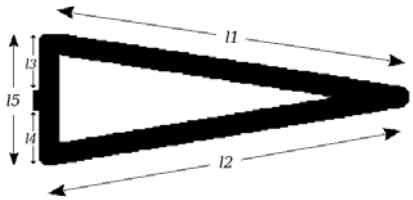


Figure B.9: A Hollow Wedge Bond - Case 2

Observe that R11.2 guarantees that none of the captured line segments can be captured by R15. Also, R11.5 guarantees that the line segment will not be captured by R1. Additionally, observe that this rule may conflict with R1 because the line segments l_1 and l_2 appear as valid single bonds and may also conflict with R10. This is a typical situation where these rules can be triggered first and then fired after employing a conflict resolution strategy.

B.9 R13: Arrow Bond

A one sided arrow line is used to represent what chemists denote as a *dative* bond. This bond type is used to indicate the existence of a negatively charged atom at the arrow end of the bond. An example dative bond is provided in Figure B.10.

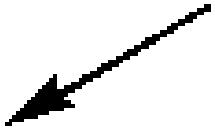


Figure B.10: A Dative Bond

As we have explained in Section 7.2.1, this is one of the five resulting primitives after applying a vectorisation step, and therefore, we will capture this bond type straightforwardly by using the following condition:

1. l is an arrow

Consequence: A dative bond with direction from source to head.

B.10 R14: Solid Wedge Bond

A solid wedge bond in the form of a filled isosceles triangle (as in Figure B.11) is used to depict the existence of an out-of-plane architecture. The base of the triangle indicates the end that is going above the plane, and therefore, the direction of this bond is known.

Figure B.11: A Solid Wedge Bond

As we have mentioned in Section 7.2.1, this should result after a vectorisation phase. Hence, we will identify it by using the following condition:

1. l is a solid triangle

Consequence: A wedge bond with direction from tip to base.

B.11 R15: Bold Bond

A bold bond is used to indicate the existence of out-of-plane architecture with unknown direction (the end that is coming up is unspecified). As illustrated in B.12, a bold bond is represented by a thick line segment in the form of a filled long rectangle. This line segment's thickness is larger than the thickness of a normal line segment. Hence, we only need to filter out line segments resulting from the vectorisation phase. We will perform the filtering based on the thickness of line segments as follows:

Figure B.12: A Bold Bond

1. l is a line segment

2. $width(l) > bbw$

Consequence: A bold bond. The new bold bond has an up direction with unknown start and end

Observe that R15.2 guarantees that this line segment is not captured by any of R1, R3, R3, R10, R11 and R16.

B.12 R16: Aromatic Ring

An older drawing style used large circles inside cyclic structures to represent aromatic rings as shown in Figure B.13. The circle is usually surrounded by a number of single line segments forming a cycle. Usually, the cycles consist of at least five line segments.

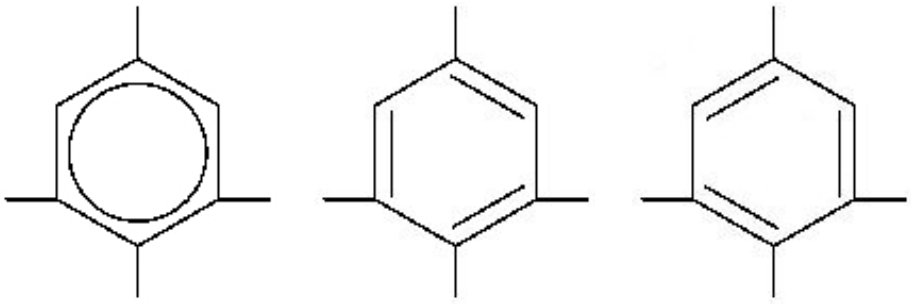


Figure B.13: Aromatic Ring (these structures are equivalent)

Since the circle should result from the vectorisation step (Section 7.2.1), all we need to detect this architecture is to find the group of line segments forming the cycle. Once detected, these line segments are transformed into aromatic bonds. We are going to achieve this using the following conditions:

1. Let C be a circle

2. $L = \{l_1, \ldots, l_n\}$, where $n \geq 5$, is a set of line segments

3. $\forall l \in L : len(l) > wb$

4. $\forall l \in L : width(l) < bbw$

5. All elements of L are within bs of C

Consequence: n aromatic bonds, one for each $l_i \in L$.

Observe that R16.3 guarantees that none of the line segments captured by this rule can be captured by either R10 or R11. Also, R16.4 makes sure that none of these line segments can be captured by R15.

Notice that the single bonds forming the cycle around the circle will have been captured by the single bond rule R1 (Section 7.5.1). However, this can be application dependent as the relevant rules can be triggered first and then executed, or fired, afterwards. Whenever more than one rule is triggered, a conflict resolution mechanism can be applied to decide which rule should execute.

Also, notice that the rule is easy to amend if the cycle surrounding the circle is formed of other types of line segments. Additionally, the rule can be updated to accommodate the less common case when line segments forming the cycle are not connected and have a chargroup in common. When encountering this case, firing the rule can be delayed until chargroups are taken into account.

B.13 R17: Open Bridge Bond

Bridge bonds are 3-dimensional structures where there are multiple different connection paths between different parts of the chemical structure diagram. These are typically presented in a $2^1/_2$-dimensional perspective drawing form. The bond line segments that cross in the drawing but do not intersect in 3 dimensions are sometimes drawn in an *open* form, where the background bond is drawn broken leaving the viewer to deduce that the broken lines are

actually one line. We will detect an open bridge bond such as the one shown in Figure B.14 using the following conditions:

1. $L = \{l_1, l_2, l_3\}$ is a set of line segments
2. $\forall l \in L : length(l) > wb$
3. l_1 and l_2 are approximately collinear
4. $dist(l_1, l_2) \in 2 * bs$
5. l_3 passes between the closest endpoints of l_1 and l_2

Consequence: Replace l_1 and l_2 with the smallest line segment containing them.

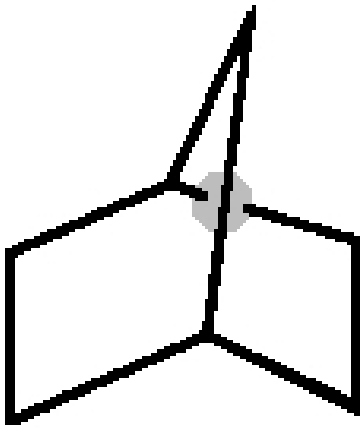


Figure B.14: Open Bridge Bond (Bridge is shadowed)

Observe that R17.2 guarantees that none of the lines captured by this rule will be captured by either R10 or R11. Also, R17.4 guarantees that these lines are not captured by R18.

The three line segments forming the open bridge bond architecture can be identified as single bonds and captured by rule R1 (Section 7.5.1). This can be overcome using the same methodology used in Section B.12. The relevant rules can be triggered first and then executed, or fired, afterwards. Whenever more than one rule is triggered, a conflict resolution mechanism can be applied to decide which rule should execute.

B.14 R18: Closed Bridge Bond

As shown in Figure B.15, closed bridge bonds are another case where 3-dimensional structures are presented in a $2^1/_2$-dimensional perspective drawing form. In this case, the different connection paths between different parts of the chemical structure diagram are drawn in a *closed* form, leaving the viewer to deduce that the crossing point does not correspond to a junction. The conditions we will use to identify closed bridge bonds are as follows:

1. $L = \{l_1, l_2, l_3, l_4\}$ is a set of line segments
2. $\forall l \in L : length(l) > wb$
3. l_1 and l_2 are approximately collinear, as are l_3 and l_4
4. $dist(l_1, l_2) = 0$ and $dist(l_3, l_4) = 0$
5. l_1 and l_2 are part of an irregularly shaped cycle
6. There is no character group where these lines connect

Consequence: Replace l_1 and l_2 with the smallest line segment containing them, and similarly for l_3 and l_4.

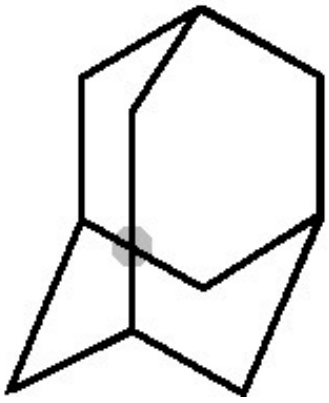


Figure B.15: Closed Bridge Bond (Bridge is shadowed)

Observe that R18.2 guarantees that none of the lines captured by this rule will be captured by either R10 or R11. Also, R18.4 guarantees that these lines are not captured by R17.

Again, the four line segments forming the closed bridge bond architecture can be identified as single bonds and captured by rule R1 (Section 7.5.1). This can be overcome using the same methodology used in Section B.12. The relevant rules can be triggered first and then executed, or fired, afterwards. Whenever more than one rule is triggered, a conflict resolution mechanism can be applied to decide which rule should execute.

B.15 Summary

We have explained the remaining 14 rules in this appendix. As we have stated before, our rule engine has 18 rules and we have explained four of these rules in more detail in Chapter 7.

APPENDIX C

THE DOUGLAS-PEUCKER ALGORITHM

Algorithm 2: Douglas-Peucker Algorithm

Input: ϵ

Input: $L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ is any m point polyline.

Output: $L^{'} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where $n \leq m$

begin

Mark (x_1, y_1) and (x_m, y_m) as keep

Invoke function $douglas\_peucker(1, m)$

Copy points marked as keep into $L^{'}$

Procedure $douglas\_peucker(j, k)$

begin

if *index of* $(x_k, y_k) \leq$ *index of* $(x_j, y_j) + 1$ **then**

return

end

Create straight line SL_{jk} between (x_j, y_j) and (x_k, y_k)

Initialise $maxdist = 0$ as the distance of furthest point from SL_{jk}

Initialise $maxi = j$ as the index of the furthest point from SL_{jk}

foreach $(x_i, y_i) \in \{(x_j, y_j), (x_{j+1}, y_{j+1}), ..., (x_k, y_k)\}$ **do**

Find distance d_{pi} between (x_i, y_i) and SL_{jk} using Equation 6.1

if $d_{pi} > maxdist$ **then**

remember distance and index of (x_i, y_i)

$maxdist = d_{pi}$ and $maxi = i$

end

end

if *maxdist* $> \epsilon$ **then**

Mark (x_{maxi}, y_{maxi}) as keep

Split polyline into two polylines at index of (x_{maxi}, y_{maxi})

Recursively apply Douglas-Peucker to the resulting two sub-polylines

begin

$douglas\_peucker(j, maxi)$

$douglas\_peucker(maxi, k)$

end

end

end

end