COPYRIGHT AND CITATION CONSIDERATIONS FOR THIS THESIS/ DISSERTATION

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

- NonCommercial — You may not use the material for commercial purposes.

- ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

**How to cite this thesis**

CONTENTS

Abstract .................................................. viii
Oprosning .................................................. ix

CHAPTER ONE : Introduction

1.1  Introduction ........................................... 1.1
1.2  Knowledge Representation ............................ 1.1
1.3  Goal ..................................................... 1.2
1.4  Overview ............................................... 1.3

CHAPTER TWO : Chemical Computer Systems

2.1  Introduction ........................................... 2.1
2.2  Molecular Graphics .................................... 2.2
2.3  Retrieval Systems ..................................... 2.3
2.4  Expert Systems ........................................ 2.4
2.5  Conclusion ............................................. 2.5

CHAPTER THREE : The Representation of Three-Dimensional Structures

3.1  Introduction ........................................... 3.1
3.2 Grammars as a Representation Method of Three-Dimensional Structures
3.2.1 3D Plex Grammars
3.2.2 Random Context Structure Grammars
3.2.3 Graph Grammars
3.3 Robotics and Computer Vision
3.4 Major Knowledge Representation Paradigms used to Represent Three-Dimensional Structures
3.4.1 Semantic Network schemes
3.4.2 First Order Logic
3.4.3 Frames
3.4.4 Production Systems
3.4.5 Object-Oriented Systems
3.5 Neural Networks
3.6 Databases and knowledge representation
3.7 CAD/Cam and Other Methods of Three-Dimensional Representation of Structures
3.7.1 Problem-Solving Models
3.7.2 Pyramid Representations
3.7.3 Blackboard Approach
3.7.4 Declarative Knowledge Programming
3.8 Conclusion
CHAPTER FOUR: Structure Graph Grammars

4.1 Introduction ..................................... 4.1
4.2 Structure Graphs ................................. 4.1
4.3 Structure Graph Grammars ...................... 4.3
  4.3.1 Structure Graph Grammar
  Extension 4 (SGG-4) ............................ 4.3
  4.3.2 Structure Graph Grammar
  Extension 6 (SGG-6) ............................ 4.17
4.4 Conclusions ..................................... 4.26

CHAPTER FIVE: Knowledge Representation and Object-Oriented Programming

5.1 Introduction ..................................... 5.1
5.2 Explicit versus Implicit Knowledge .......... 5.1
5.3 General Representation ........................ 5.3
5.4 Object-Oriented Programming (OOP) ........ 5.4
  5.4.1 Data Abstraction .......................... 5.4
### CHAPTER SEVEN: Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Introduction</td>
<td>7.1</td>
</tr>
<tr>
<td>7.2 Three-Dimensional Graphics</td>
<td>7.1</td>
</tr>
<tr>
<td>7.2.1 Points</td>
<td>7.3</td>
</tr>
<tr>
<td>7.2.2 Vectors</td>
<td>7.4</td>
</tr>
<tr>
<td>7.2.3 Homogeneous Coordinates</td>
<td>7.5</td>
</tr>
<tr>
<td>7.3 Affine Transformations</td>
<td>7.6</td>
</tr>
<tr>
<td>7.3.1 Rotation</td>
<td>7.7</td>
</tr>
<tr>
<td>7.3.2 Perspective Projection</td>
<td>7.9</td>
</tr>
<tr>
<td>7.4 Algorithms</td>
<td>7.11</td>
</tr>
<tr>
<td>7.4.1 Algorithm 1</td>
<td>7.12</td>
</tr>
<tr>
<td>7.4.2 Algorithm 2</td>
<td>7.13</td>
</tr>
<tr>
<td>7.5 Conclusion</td>
<td>7.15</td>
</tr>
</tbody>
</table>

### CHAPTER EIGHT: Use and Applications of "Object" Structure

**Graph Grammars**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Introduction</td>
<td>8.1</td>
</tr>
<tr>
<td>8.2 Intelligent Grammars?</td>
<td>8.1</td>
</tr>
<tr>
<td>8.3 Application in Organic Chemistry</td>
<td>8.2</td>
</tr>
<tr>
<td>8.4 Vision Systems</td>
<td>8.6</td>
</tr>
<tr>
<td>8.5 Summary</td>
<td>8.9</td>
</tr>
</tbody>
</table>
CHAPTER NINE : Molecule-By-Grammar

9.1 Introduction ............................................. 9.1
9.2 Hardware .................................................. 9.1
9.3 Limitations on the System ................................. 9.2
9.4 Discussion of the System ................................. 9.2
9.4.1 Load ...................................................... 9.4
9.4.2 Display ................................................... 9.4
9.4.3 Generate ................................................ 9.5
9.4.4 Fire ....................................................... 9.5
9.4.5 Trace ...................................................... 9.5
9.4.6 Quit ....................................................... 9.6
9.5 Example ..................................................... 9.6
9.6 Technical Discussion ....................................... 9.9
9.7 Conclusion .................................................. 9.11

CHAPTER TEN : Conclusion

10.1 Introduction ............................................... 10.1
10.2 Summary and Future Topics of Research .............. 10.1
The purpose of this dissertation is to study methods to represent structures in three-dimensions. Due to the fact that chemical molecules are mostly complex three-dimensional structures, we used chemical molecules as our application domain.

A literature study of current chemical information systems was undertaken. The whole spectrum of information systems was covered because almost all of these systems represent chemical molecules in one way or another.

Various methods of three-dimensional structure representation were found in our literature study. All of these methods were discussed in the context of its own application domain.

Structure graph grammars were examined and explained in detail. A small object-based system with structure graph grammars as the underlying principle was developed. We speculated on the use of such "intelligent" graph grammars in structure interpretation and identification. Further research in this area was also identified.
OPSOMMING

Die doel van hierdie verhandeling is die studie van drie-dimensionele voorstellingsmetodes. Aangesien chemiese molekules oor die algemeen komplekse drie-dimensionele strukture is, is dit as ons toepassingsgebied gekies.

'N Literatuurstudie van huidige chemiese inligtingstelsels is gedoen. Die hele spektrum van inligtingstelsels is gedek aangesien bykans al hierdie stelsels molekules met een of ander metode voorstel.

Verskeie drie-dimensionele voorstellingsmetodes is in die literatuur gevind. Elk van hierdie metodes is in terme van die spesifieke toepassingsgebied, deeglik beskryf en bespreek.

Struktuurgrafiekgrammatikas is beskryf en deeglik ondersoek. 'N Klein objek-georiënteerde stelsel met struktuurgrafiekgrammatikas as onderliggende beginsel is ontwikkel. Daar is ook gespekuleer oor die gebruik van sogenaamde "intelligente" struktuurgrafiekgrammatikas in struktuur interpretasie en struktuur herkenning. Verdere navorsingsgeleenthede in hierdie area is ook geidentifiseer.
1.1 Introduction

In this chapter a brief overview of the goal of this study will be presented. A short discussion on knowledge representation is given after which a short overview of the structure of this dissertation will follow.

1.2 Goal

There are many existing schemes of three-dimensional structure representation [Chapter 3, 4, 5]. A literature study was undertaken to examine three-dimensional structure representation methods that are currently being used. We propose the use of structure graph grammars as a representation method to visualize structures in three dimensions. We will examine structure graph grammars as defined in [BAR91] and show that we are able to generate any molecule that is found in organic chemistry.
1.2

The object-oriented programming paradigm was used to develop a small system that generates three-dimensional organic molecules and display the molecules on a two-dimensional screen. The system uses a grammar of the class structure graph grammar extension 4 as input.

A few artificial intelligence applications, where object-based structure graph grammars could be used, are examined.

1.3 KNOWLEDGE REPRESENTATION

Little is known about how the human stores and retrieves information, thus work in this important field is still in its infancy. For example, none of us has to learn the advanced aspects of calculus to be able to compute where you must move your hands to catch a ball that is moving in your direction.

Yet, to give the computer the same ability to visualise the results of new situations, such as the above, requires that it possesses the same form of knowledge of the physical universe that a human being has by means of his human experience, not by being told, by learning at school, by reading books, or by any other means.

This element of human experience and mental model creation is simply beyond the ability of artificial intelligence, yet is
essential to much of what we call problem-solving. It is not facts, nor rules, or any other form of simple logical or mathematical representation that we have available to us currently.

1.4 Overview

A short overview of each chapter will now be given.

Because our application domain is the generation of chemical molecules, an overview of current chemical systems are given in chapter two. Not only did we examine representation and visualization systems, we also looked at other chemical systems such as three-dimensional chemical databases, molecular graphics and the use of artificial intelligence in chemical computer systems.

Chapter three will give an overview of some methods that have been proposed and used in three-dimensional structure representation. We will examine how grammars are used as a representation scheme to represent three-dimensional structures, and how three-dimensional structures are used and represented in robotics and computer vision. Traditional knowledge representation paradigms are overviewed and we briefly look at neural networks. Pictorial databases are also discussed. CAD/CAM and other methods used to represent three-dimensional structures
are also examined in chapter three. Future research topics in this important research area are also discussed.

The use of structure graph grammars as a three-dimensional representation scheme will be demonstrated in chapter four. Definitions of structure graphs will be given. Structure graph grammars (SGG) as defined in [BAR91], as well as examples, will be given to show that we can generate all aliphatic and aromatic organic molecules. Examples are given in both SGG-4 and SGG-6, structure graph grammars extension 4 and 6 respectively, to show the difference between the two.

In chapter five we show that knowledge can be either explicit or implicit. In this chapter we will discuss when knowledge is implicit or explicit. We will look at the advantages and disadvantages of the two types. Object-oriented programming will be examined as well as some examples of object-based representation models.

All the objects that we use to generate a three-dimensional structure, will be given in chapter six. Each object will be discussed briefly after which it is given in BNF notation. The BNF notation used can be found in Appendix B.

Chapter seven is an extension of chapter six. Here we discuss and explain in detail the algorithms we used and how we used them. An example of how a molecule is generated from the grammar and objects will also be given.
A discussion on the use of this "object" structure graph grammars in expert systems and other applications are given in chapter eight. We speculate on how we can use SGG-4 and SGG-6 together to create a grammar that is much more powerful and how to implement it with the use of objects.

Chapter nine is a brief overview of the system. It contains the limitations on the system and an example run of the system is given.

In chapter ten we give a summary and future research is discussed.

Appendix A is a glossary of chemical terms with definitions.

Appendix B gives the BNF-notation used in chapter six.

Appendix C is a list of production rules used in the discussion in chapter eight.
2.1

CHAPTER TWO

Chemical Computer Systems

2.1 Introduction

Current research is aimed at creating intelligent computer systems to be used in all areas of chemical research and development. Computer support in visualising complex chemical structures, searching databases and assessing reaction information are now vital in determining the potential significance of research findings [ALL89, LAN91]. Chemistry has played a vital role in the development of artificial intelligence. Problems such as synthesis design, spectral interpretation and structure elucidation have resulted in today's artificial intelligence techniques. It is thus not surprising that the use of artificial intelligence techniques are increasingly used to address chemical information problems [HOP89, JOH85].

Chemical computer systems can be summarised in the following categories:

Graphical representation methods and modelling systems,
intelligent retrieval systems which include literature databases as well as reaction retrieval systems, and expert systems for synthesis planning and predicting molecular properties [BAW89].

Graphical representation methods and modelling systems will be discussed in section 2. We will give an overview of intelligent retrieval systems in section 3 and expert systems in section 4. A short summary of the chapter will be given in section 5. Future directions regarding computer research in chemical computer systems, will be given in section 5. A glossary of chemical terminology with explanations can be found in Appendix A.

2.2 Molecular Graphics

Computer graphics is proving to be a valuable tool in all aspects of chemistry. Not only medicinal chemistry, but also inorganic, organic, analytical and physical chemistry are using graphical representation software and modelling systems [HAS85, RAM85, ST091, TRI92, VIN85, VIN99].

The large amount of data used by chemists entails manipulating
raw data to obtain as much meaningful information as possible. Visualization is a means of representing the processed data on graphical devices in an informative and convincing manner [ST091, TOW89].

Through visualization, computer graphics is capable of providing an insight into the spatial arrangement of perfectly crystalline materials as well as less conventional structural phenomena such as intergrowths, epitaxy, surface reorganisation and non-stoichiometry [RAM85].

In medicinal chemistry, molecular graphics can be used very effectively to show what is happening at the molecular or atomic level. Three-dimensional graphical representation and computational chemistry explain specific events that are distinguished in the normal and disease state in terms of molecular processes [RAM88, RAM85].

The most valuable asset of molecular graphics is the ability to view molecular architecture in three dimensions.

The basis of computerised graphical chemistry still revolves around the stick picture and the CPK-like representation of van der Waals volumes [VIN89].

Molecular graphical software packages that are currently available includes Chem3D [COH91] and ChemGraf [VIN85].
A short discussion on various packages follows:

ISIS/Draw: This package helps the chemist to draw chemically correct structures. It "understands" the chemistry behind the reactions and sprouts new bonds where appropriate. It also recognizes valence limits and warns the user when they are exceeded [COH91].

NITRO: This software transforms a personal computer into an interactive 3D graphical terminal. It allows you to manipulate graphics on your screen independently of the host computer. Features that NITRO provides are building three-dimensional sketch functions, interactive docking of two molecules and fitting of molecules. Mono, stereo and orthogonal views are supported. Interactive three-dimensional graphical features include inform, relate and animate options that tie the graphs to each other and to the molecular structure under study [TRI92].

CONCORD: It is a program for rapid generation of high-quality approximate three-dimensional organic molecular structures. These structures can be used for immediate computer graphic display and study of three-dimensional molecular structures. Structure-dependent calculations and predictions of molecular properties,
atomic charges from molecular orbital calculations and solubilities estimated using calculated molecular surface area or volumes, can be performed. It also provides excellent initial guesses for geometry optimization by molecular orbital or molecular mechanics energy minimization [TRI92].

Alchemy II: Through animation, Alchemy II provides a visual tool for working with three-dimensional molecular structures. Features it provides include: building chemically correct three-dimensional models, an energy minimizer is provided to optimize the molecule geometry using molecular mechanics to find a local minimum, crystallographic input and stereochemistry [TRI92].

There are a few shortcomings in molecular graphics. One obvious shortcoming of current molecular visualizing methodology is that of representation [VIN89]. A very large structure or multiple pictures may contain too much information. However, an overall impression of a large structure can be very valuable.

Another major problem is the representation of molecular movement [VIN89]. Molecules are moving, i.e. vibrating, interconverting, tautomerising and internally rotating on time scales that are relevant to a host of molecular processes. The
modeller is faced with the problem of representing the molecule as a moving volume rather than a fixed, single conformation.

The extension of molecular mechanics to molecular dynamics enables a simulation of the natural movement of a molecule along a trajectory to be plotted over a finite timescale.

Computational times for these calculations and the number of generated pictures can be immense.

The next two software packages facilitate molecular dynamics and extend the model building process into the realm of computational chemistry.

SYBYL/Dynamics: SYBYL's molecular dynamics tools integrate the setup, calculation and analysis of dynamic trajectories. It provides numerous methods for the construction of molecular structures. Dynamics simulations can be performed in vacuo or with explicit solvation. Two force fields are available; the Tripos Force Field and the AMBER Force Field. Both include united and all-atom versions. There are three modes for trajectile calculation. They are constant energy, constant temperature and constant pressure. For the analysis of results three types of interactive data graphs are available [TRI92].
SYBYL/Advanced Computation: This module allows the user to select systematic conformational searching based on the goal of the conformational analysis. Features include: analysis based both on intramolecular contacts and energy, user control of viability criteria such as van der Waals contacts, energy cut-offs, extension and resolution of conformational space, ability to constrain searches by known atomic distance ranges and searching of ring conformations. Performing local operations on molecular volumes is an integral part of analysing the shape of molecules for identification of the active site. This feature allows the chemist to compute differences, unions and intersections of volumes for a series of molecules and to contour the results for display as a mesh surface [TRI92].

The last area where molecular graphics, computer graphics and computational chemistry interacts is that of modelling. It is important to be able to simulate and predict exactly what is happening at the atomic and molecular level.

It is thus possible to predict properties of unknown compounds before investing costly resources in the synthesis and analysis of these materials [HAS85, VIN85].
SYBYL/QSPR (Quantitative Structure Property Relationships) is a predictive modeller for polymer scientists.

SYBYL/QSPR: This program provides various predictive models for polymer scientists. Components of QSPR are model management, automatic fragment analysis, built-in models and statistical analysis tools. Models that are included are: two models for the glass transition temperature and a model of molar free volume. Analysis is based on techniques such as partial least squares, cross-validation and cluster analysis [TRI92].

2.3 Retrieval Systems

Retrieval systems can be divided into separate systems. They are literature databases, reaction databases, chemical patent databases and three-dimensional crystallographic databases [ALL89, BEI92, CHE92, RH085, WIN85].

It is possible for a chemist, by using a simple structural representation, to search by structure through millions of compounds cited in chemical literature. The search phase of the operation can take as little as five minutes and most searches can be completed within fifteen minutes total time [RH085].
Two major systems that have the facility to perform such a search are:

CAS ONLINE Registry File produced by Chemical Abstracts Service (CAS), available on STN International and the DARC system available through Telesystémes Questel [RH085].

CAS ONLINE Registry File.

CAS ONLINE Registry File is the world's largest file of substance information, including organic compounds, coordination compounds, polymers, alloys, mixtures and minerals. It also contains CONCORD calculated 3D coordinates for over 4 000 000 organic substances [CHE92, RH085].

The Registry File can be searched by chemical structure, partial structure, chemical dictionary terms such as common names or molecular formulas.

Graphic structure and substructure queries can be made off-line with STN Express. This software from CAS allows structure queries and search strategies offline, before going online in a file on STN. Features of STN Express include: help with queries, predefined search strategies, autologin, scrolling text and graphics and capturing text and graphics in the electronic transcript.
Integration of molecular graphics and retrieval systems is possible by using Alchemy II to model a molecule. The molecule is then uploaded with STN Express to the Registry File. A search will be conducted in the Registry file by accessing chemical literature references. STN Express is then used to download three-dimensional coordinates of substances. Alchemy II can now be used to model and manipulate the molecules [BAW89, CHE92].

Beilstein Database

The Beilstein database contains records for organic substances cited in the Beilstein Handbook, covering the period from 1830 to 1980. The database totals 3.5 million substances, with stereochemical data, and with provision for up to 400 types of critically evaluated property data for each substance, each with a literature reference. The Beilstein database is searchable with STN International and the Dialog system [ALL89, BEI92].

The Beilstein database also provides offline structure query facilities through MOLKICK II. MOLKICK II allows you to search structure databases of several different hosts using the same structure query [BEI92].

Heilbron Dictionary

The Heilbron Dictionary of organic chemistry (DOC) and related databases is searchable on the Dialog system.
Structures of over 30 000 organometallic substances can be displayed, provided the user has a high resolution graphics terminal [BAW89].

Fine Chemicals Directory

The Fine Chemicals Directory (FCD) is a compilation of the catalogues of fine chemical vendors. This is available as ChemQuest from Pergamon-Orbit-Infoline and as a separate database with the Maccs system from molecular Design [ALL89].

Several chemical reaction databases are available. In these databases you will find citations of reactions that have been reported in the literature.

Orac

The Orac system (Organic Reaction Access by Computers) developed at Leeds University includes a core database of 55 000 reactions in addition to a further 42 000 reactions from Theilheimer's synthetic methods of organic chemistry. Theilheimer, as well as data from the Journal of Synthetic Methods and Chiras (asymmetric synthesis) are available on the Reaccs system from Molecular Design. There are already more than 100 000 searchable reactions [ALL89, JOH85].

CASReact

Another reaction database is CASReact. The more than 24 000
citations from over 100 journals correspond to the organic sections of Chemical Abstracts. This database is growing at a rate of 12,000 new citations each year. This includes single and multi step reactions, and corresponds to over 200,000 single step reactions [ALL89].

SYNLIB

Smith Kline and French, a pharmaceutical company, has developed a chemical reaction retrieval system in collaboration with Columbia University. This system allows the scientist to search a database for reaction citations extracted from scientific literature. The library consists of about 17,000 reactions. User-specific databases can be constructed to supplement the general database supplied with SYNLIB. SYNLIB (synthesis library) uses simple graphic instructions to enter chemical structure questions. It is possible to perform a narrow search or a generic search [BO085].

Chemical patent databases are provided by Derwent Publications and International Documentation in Chemistry (IDC). Both use fragmentation codes, the performance compares less well with the immediacy of graphics searching.

International Documentation in Chemistry.

Software developed at Sheffield will permit IDC to
generate fragment codes automatically for generic structure database creation. This will enable them to offer the traditional means of searching their databases [ALL89].

Derwent World Patent Index

Generic chemical structures appearing in patents issued since 1987 have been recorded graphically for search by the Markush-Darc system. This enables the Derwent World Patent Index to express the complexity of generic structures accurately.

Information on three dimensional structures is by comparison with information on two dimensional structures, available from a limited number of experimental techniques and for a relatively small proportion of known chemical compounds.

Three dimensional data are obtained mainly from X-ray and neutron methods. The complete chemical spectrum is now covered by four structural databases. See Table 2.1.

The three-dimensional data of the above mentioned databases must not be confused with the CONCORD approximate three-dimensional organic molecular structures.
Table 2.1 Three dimensional crystallographic databases [ALL89]

<table>
<thead>
<tr>
<th>DATABASES</th>
<th>SPECTRUM COVERED</th>
<th>ENTRIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein Data Bank</td>
<td>Biological macromolecules</td>
<td>427</td>
</tr>
<tr>
<td>Cambridge Structural Database</td>
<td>Organics, organometallics and metal complexes</td>
<td>73,890</td>
</tr>
<tr>
<td>Inorganic Crystal Structure Database</td>
<td>Inorganics and minerals</td>
<td>24,406</td>
</tr>
<tr>
<td>Metals Crystallographic Data File</td>
<td>Metals and alloys</td>
<td>11,000</td>
</tr>
</tbody>
</table>

It is clear that the utility of these databases and retrieval systems in the long run will be dependent on the size and quality of the associated database as well as on the sophisticated search facilities.

2.4 Expert Systems

Artificial intelligence (AI) addresses problems that require symbolic rather than numerical data to be handled [NEW76]. AI solutions rely more on logical reasoning than calculation. The major inroad of AI into chemistry has been the technology of expert systems [JOH85].
These systems are able to manipulate and refine knowledge in order to propose solutions that resist precise description and have no deterministic solution. The ability to organise and use knowledge that is often inexact and incomplete are the characteristics of expert systems. They operate on concepts and models, and use generalised rules as well as specific facts.

Examples of such systems are:

- MOLGEN (planning laboratory experiments in genetics)
- DENDRAL (organic structure elucidation on the basis of spectroscopic data)

There are various areas in chemistry where expert technology has been applied. Some of the most successful expert systems are concerned with synthesis planning and elucidation of unknown molecular structure [HOP89, JOH85, SMI77, WIL89].

Elucidation of unknown molecular structures is a process of systematic posing, testing and rejection of hypotheses [SMI77].

A knowledge-based system that uses comparative modelling of protein structures to determine the three-dimensional structure of protein molecules, based on both the amino acid sequence and the three-dimensionality of experimentally known structures, is COMPOSER [TRI92].
The Composer methodology uses multiple protein structures as the template for generating the conserved structural regions for greater accuracy. Composer has a wide application range for protein research that includes the following: active compound design, X-ray structure determination, multidimensional NMR (Nuclear Magnetic Resonance) spectroscopy, electron density maps, site-directed mutagenesis and de novo protein engineering.

The strategy and process of Composer are as follows:

- Identification of homologous structures,
- Determining of seed residues,
- Aligning of known structures and model sequence,
- Modelling of conserved regions and loop regions.

Examples of synthesis planning systems are:

- LHASA (Logic and Heuristics Applied to Synthetic Analysis),
- SECS (Simulation and Evaluation of Chemical Synthesis),
- CASP and
- SYNCHEM II [JOH85].

There are other similar programs such as EROS and SYNGEN, but they do not make use of an empirical knowledge base [HOP89, JOH85].

The LHASA system is an interactive program where structured information is communicated to the computer through the use of computer graphics.
It operates in a retrosynthetic fashion working backwards from the target through successive levels of precursors to possible starting materials [JOH85]. This is similar to a goal-driven search method.

The main knowledge base used by LHASA comprises a set of transform descriptions, each of which contains a set of rules:

- The keying substructure which must be present in the target for the transform even to be considered.
- The scope and limitations of the reaction under consideration.
- The structural modifications needed to generate the precursor structure [JOH85].

Because it is not practically possible to do an exhaustive search, LHASA uses two stratagems to do an adequate search which yields a good solution.

The first: at each level the user decides which precursors merit further processing.

The second: the user can specify certain constraints to be in operation during the course of the search [JOH85].

These constraints will cut down the number of transforms attempted and thus help to reduce the combinatorial nature of the problem. There are several constrained strategies available. A good choice of a search strategy is critical for a good
solution where the search is narrowed without excluding the best solutions.

Related to the synthesis design are programs which attempt to predict plausible metabolites of a xenobiotic compound [HOP89].

Knowledge of the metabolic fate of compounds is important in understanding pharmacological activity and the design of new medical drugs and agrochemicals. The biotransformations performed by specific enzymes can be encoded in a similar way to the reaction rules used in Lhasa.

Programs such as Case attempt to automatically generate such toxicity rules for chemicals based on the presence or absence of substructures [SMI77].

An increasing use of expert systems is to act as front ends to complex software packages or large databases. For example, molecular modelling is a very complex process, and novice users often find it difficult to select the best methods for a study. Chem-X addresses this problem through the use of a knowledge based front end [HOP89].

One limitation of many expert systems is that their performance deteriorate when they are presented with a problem or situation not explicitly considered in the decision tree encoded in the knowledge base.
2.5 Conclusion

Future and current topics of research are amongst other the extension of molecular mechanics, methods to obtain better heats of formation, prediction of spectroscopic parameters and investigation of solvent effects [VIN85].

All the techniques of molecular graphics can be viewed as a variation on a theme, i.e. that of extending the stick picture to try and represent movement. Many groups around the world are currently trying to represent molecular movements in new ways, but little progress is being made away from the set notions of the stick model [TOW89].

Three-dimensional data sets have traditionally been very hard to represent visually, and thus difficult to interpret and explore. Even when stereoscopic hardware is available, volumetric data sets such as the results of molecular orbital calculations and crystallographic electron density maps are normally simplified to discrete isometric contours. This process discards much of the information and generates phantom surfaces and discontinuities that may mislead the observer [VIN89].

The developments of algorithms and techniques for the search and retrieval of information from three-dimensional databases are now a very important research area. The problem involved is perhaps an order of magnitude more complex and challenging than those involved in two-dimensional databases [ALL89].
Individual structures from the Protein Data Bank are extensively used in modelling applications. Here is much software development aimed at searching and at structural systematics. These methods are now being extended into the areas of structure prediction and automated model building, by using knowledge based approaches based on structural homology [ALL89].

One of the most important research areas is integration of the different computer systems the chemist uses. It is therefore important that a standard data exchange format and a standard user interface are accepted by the majority of manufacturers [BEI92, CHE92, TOW89, TRI92].

Much research is needed to solve the problems of integrated retrieval and integrated representation [BAW89].

The techniques of computational chemistry, molecular graphics, information retrieval systems and expert systems have become firmly established as indispensable to the range of physical and biological tools available to the research chemist.

In this chapter computer systems that are currently used in chemical research and development were described. We found that these systems fall in three categories. Each of the categories, namely molecular graphics, information retrieval and expert systems, were discussed.
In the following chapter the representation of three-dimensional knowledge will be discussed. We will look at various traditional paradigms as well as other not so well-known methods.
CHAPTER THREE

The Representation of Three-Dimensional Structures

3.1 Introduction

Representation methods of three-dimensional structures are a resent research area. Quite a few different methods of knowledge representation have been discussed in various articles over the past few years [BAR91, EHL89, EHL91, LIN84].

This chapter will give an overview of some methods that have been proposed and used in the representation of three-dimensional structures. In section 2 we will examine how grammars are used as a representational method to represent three-dimensional structures. Section 3 will concentrate on how three-dimensional structures are used and represented in robotics and computer vision. Traditional knowledge representation paradigms are overviewed in section 4 and we briefly look at neural networks in section 5. Databases will be discussed in section 6 while CAD/CAM and other methods used to represent three-dimensional structures will be examined in section 7. A summary of all the representation schemes that have been discussed will be given in section 8 as well as a brief discussion of future research topics in this important research area.
3.2 Grammars as a Representation Method of Three-Dimensional Structures

Grammars provide an excellent method to describe and define three-dimensional structures formally. The different types of grammars that we will look at include 3D Plex Grammars [LIN84], Random Context Structure Grammars [EHL89, EHL91] and Structure Graph Grammars [BAR91, AIZ91, DOL91]. A short definition and discussion of each grammar will be given as well as examples in some cases.

3.2.1 3D Plex Grammars

The basic idea of 3D plex grammars [BAR91, LIN84] is to consider each terminal or non-terminal symbol as a primitive or composite surface having an arbitrary number $n$ of attaching curves for joining other surfaces. A structure of this type is called an $n$-attaching curve entity (NACE).

Definition

A three-dimensional plex grammar can be represented by a sextuple

$$G_p = (N, \Sigma, P, S, I, i_o)$$

where:

- $N$ is a finite nonempty set of NACEs. (nonterminals),
- $\Sigma$ is a finite nonempty set of NACEs. (terminals)
- $N \cap \Sigma = \emptyset$
- $P$ is a finite set of production rules,
- $S \in N$ is the initial NACE,
- $I$ is a finite set of symbols (identifier)
\[ \text{In } (N \cup \Sigma) = \emptyset \]

\[ i_o \in I \text{ is the null identifier.} \]

A production rule \( p \in P \) has the form:

\[ \mathit{lT}_l \mathit{V}_l \rightarrow \mathit{rT}_r \mathit{V}_r \]

where:

- \( l \) is the left-side component list,
- \( r \) is the right-side component list,
- \( T_l \) is the left-side intersection list,
- \( T_r \) is the right-side intersection list,
- \( V_l \) is the left-side tie-curve list,
- \( V_r \) is the right-side tie-curve list.

The component lists are strings of the form

\[ \mathit{l} = a_1 \ldots a_i \ldots a_m \text{ and } \mathit{r} = b_1 \ldots b_j \ldots b_n \]

where:

- \( a_i \) and \( b_j \) are single NACES (for \( 1 \leq i \leq n \) and \( 1 \leq j \leq n \)) called components.

The \( l \)- and \( r \)-lists provide an ordering for the groups of connected NACES. The connection of attaching curves of two or more NACES forms an intersection, and \( T_l \) and \( T_r \) specifies the way in which the NACES interconnect. The unordered intersection lists are divided into fields that specify which attaching curves of which NACES connect at each intersection. Each intersection requires exactly one field.

The component and intersection lists \( \mathit{lT}_l \) and \( \mathit{rT}_r \), when taken as pairs, define the structure involved in a
rewriting rule. These structures attach to the remainder of the plex at a finite number of intersections are called tie curves. The tie curve lists are also divided into fields, with exactly one field specifying each tie curve. The number of curves for the left and right sides of a production must be the same. The number of tie curve lists will be the same and the tie curve lists are ordered, with the k-th field on the left corresponding to the k-th field on the right.

Figure 3.1 An object that can be generated from 3D Plex grammar.
3D plex grammars [LIN84] can be used to model three-dimensional objects for computer vision applications. The structural relationship descriptive power of 3D plex grammars can be utilized to perform structural analysis because it provides a hierarchical and systematic paradigm for three-dimensional object recognition by reducing the problem of identifying a three-dimensional object to subproblems of primitive-surface-patch identification.

### 3.2.2 Random Context Structure Grammars

Random Context Structure Grammars (RCSG) generate three-dimensional digital structures using a number of permitting and forbidding contexts [EHL89, EHL91]. A RCSG can only generate connected structures.

**Definition**

A RCSG is a 4-tuple $G = (N, T, S, P)$ where:

- $N$ is a finite set of non-terminal symbols,
- $T$ is a finite set of terminal symbols,
- $S$ is the initial symbol and
- $P$ is the set of production rules.

Productions of $P$ are of the form:

$$A \rightarrow B(U_1; T_1/U_2; T_2/U_3; T_3/U_4; T_4/U_5; T_5/U_6; T_6/U_7; T_7)$$

where:
A \in N, B \in (N U T)^+ and
U_i, T_i \in N, U_i \cap T_i = \emptyset \text{ for } 1 \leq i \leq 7.

Figure 3.2 shows a single symbol in a three-dimensional structure. The lines and planes through the point where the symbol is located, determine the different contexts which must be considered before a production is applied.

The symbol A is associated with seven different contexts:
V_1 (T_1) the permitting (forbidding) horizontal line contexts,
V_2 (T_2) the permitting (forbidding) vertical line contexts,
V_3 (T_3) the permitting (forbidding) depth line contexts,
V_4 (T_4) the permitting (forbidding) horizontal plane contexts,
V_5 (T_5) the permitting (forbidding) vertical plane contexts,
V_6 (T_6) the permitting (forbidding) depth plane contexts,
V_7 (T_7) the permitting (forbidding) global contexts.

Figure 3.2 The context associated with a symbol A in a three-dimensional structure.
The power of RCSG's is that the substitution of a symbol in a structure can be influenced by the very presence or absence of other symbols in seven different contexts in the structure \[\text{[BAR91, EHL89, EHL91]}\].

In the following example a methane molecule will be generated by the production rules of a RCSG \[\text{[EHL89]}\].

**Example 3.2**

The production rules are:

The first symbol is generated:

1. \( S \rightarrow K \),

A vertical "bond" is generated:

2. \( K \uparrow V_1 \) (\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
The sentential bonds are replaced by terminal bond symbols:

(11) \( V_1 \uparrow B_1 \) \( (\{C\};\emptyset;\emptyset;\{V_1, B_2, B_3, B_4\}) \),
(12) \( V_2 \rightarrow B_2 \) \( (\{C\};\emptyset;\emptyset;\{B_3, B_4\};\{B_1\};\emptyset) \),
(13) \( V_3 \leftarrow B_3 \) \( (\{C\};\emptyset;\{B_2\};\{B_4\};\{B_1\};\emptyset) \),
(14) \( V_4 \rightarrow B_4 \) \( (\{C\};\emptyset;\{B_2, B_3\};\emptyset;\{B_1\};\emptyset) \),
(15) \( K \rightarrow H \) \( (\emptyset;\emptyset;\emptyset;\emptyset;\emptyset;\emptyset;\emptyset) \).

This generates only one methane molecule. To generate more complex carbon compounds, the contexts in these productions will have to be less stringent. The \( B \)'s desonate the different bonds of the molecule.

Figure 3.3 A methane molecule generated from the grammar in Example 3.2
3.2.3 **Graph Grammars**

The use of higher-dimensional grammars such as structure graph grammars are increasingly used to describe three-dimensional structures [BAR91, DOL91]. Many models for graph grammars have been proposed as kinds of hyper-dimensional generating systems [AIZ91].

The basis primitive in a graph grammar is a directed (non directed) graph which is a set of labelled vertices and labelled edges.

Although structure graph grammars have been used to describe structures, the most literature examples found are two-dimensional grammars. Since two-dimensional grammars are special cases of three-dimensional grammars and outside the scope of this dissertation, we will not discuss any of them.

Because graph grammars are excellent examples of three-dimensional structure generating systems, we will discuss it in detail in chapter four.

3.3 **Robotics and Computer Vision**

Computer vision systems use knowledge of image formation and the application domain to recover useful information about a scene
from one or more images [RA088]. Knowledge engineering techniques are becoming increasingly popular in computer vision systems because image understanding has proved exceedingly difficult in general domains.

Interpretation requires knowledge. Computer vision systems are using increasingly complex methods of knowledge representation and control to match the problem's true complexity [RA088, DER90, HAR88, ZHA91, BRI90, MOR89, MOR87, ROS91, VAN88, W0089].

Input to a computer vision system consists of a digitized image of the scene to be analysed and is usually represented as an array of numeric values. The input image can either be translated into a symbolic representation or some predefined computational sequence can be performed on the input data.

The required processing is usually divided into two sequential phases; low-level vision and high-level vision. Low-level vision usually consists of a segmented image labelling different regions based on the resolution of a given predicate and features of these regions. High-level vision seeks to attach a consistent interpretation to labels obtained during low-level processing. Most vision systems perform segmentation followed by interpretation [HAR88, MOR87]. Image interpretation can be considered as a mapping between sensory data and the model representing concepts relevant to the image’s domain.
Due to the interaction between iconic and symbolic information in computer vision systems, knowledge representation and management problems are more complex than those of traditional symbolic knowledge-based systems [RA088].

In the following paragraphs we will discuss a few examples of knowledge-based vision systems.

Understanding three-dimensional scene geometry from a sequence of images requires careful selection and management of the information they offer. A suitable approach is based on the REVgraph, which is a data structure designed to represent a three-dimensional model of a scene [HAR88], built up from a stereo pair of two-dimensional images. The most important levels of representation are regions, edges and vertices linked together as a graph structure.

This provides a list-based representation and maintains three-dimensional information for features closely related to those existing in the real world. The REVgraph can be divided into two parts:

the geometry, which contains all the metrical information;
and the topology, which contains information concerning connectivity of points, lines and surfaces [HAR88].
The REVgraph is used to enable reasoning processes to reconstruct a geometrically consistent description of surfaces in the scene from various lower levels of description.

Any piece of data in the REVgraph may be regarded as an assertion or a fact. The nature of such facts will depend on the level of the data. At the edge detection level, a fact will be a particular pixel and at the geometric level, it will be the direction and end points of a straight edge segment. At higher levels, it will be the existence of a vertex or region, or ultimately the identification of a group of regions as a recognized object [HAR88].

A model to detect and track features in a sequence of time varying images requires a specific method of representation. The goal is to track line segments corresponding to the edges extracted from the image being analysed [ZHA91]. It is clear that tracking both end points of each segment will be very difficult, since they are not reliable, due to the fact that segments can be broken from one frame to another [DER90]. For this reason, two types of representation have been considered.

In one representation a line segment having endpoints at P1(x₁, y₁) and P2(x₂, y₂) is characterized by the vector \( \mathbf{v} = [c, d, \sigma, L]^T \). The components of \( \mathbf{v} \) are derived from the endpoints as follows:
The orientation $\sigma$ of the line segment:

$$\sigma = \arctan \frac{y_2 - y_1}{x_2 - x_1}$$

The length $L$ of the line segment:

$$L = \sqrt{\left( (x_2 - x_1)^2 + (y_2 - y_1)^2 \right)}$$

The distance of the origin to the line segment denoted by the parameter $c$:

$$c = \frac{(x_2 y_1 - x_1 y_2)}{L}$$

The distance along the line from the perpendicular intersection to the midpoint of the segment denoted by $d$:

$$d = \frac{(x_2 - x_1)(x_2 + x_1) + (y_2 - y_1)(y_2 + y_1)}{2L}$$

The second representation characterizes a line segment by the vector $v = [x_m, y_m, \sigma, L]^T$ where the point $(x_m, y_m)$ defines the coordinates of the midpoint $P_m$ of the segment and are given by:

$$x_m = \frac{(x_2 + x_1)}{2} \quad \text{and} \quad y_m = \frac{(y_1 + y_2)}{2}$$

When using both of the above mentioned line representation methods, the line segments can be broken from one frame to another.
The taxing problem of visually navigating a robot along a road is approached by means of creating and updating a simple representation of the road from a sequence of images. The representation chosen is a 4-parameter model that describes the width, direction and simple curvature of the road in a vehicle centred (X, Y, Z) world coordinate system. This 4-parameter model is capable of representing the position of the robot on the road, the general direction of the road and a simple curvature. An alternative representation would be to use cubic spline approximations or other higher order polynomials to model the road edges [MOR90].
In applications such as robotics and automation, three-dimensional information about the environment is essential for robot movement and object inspection [NAS89]. One method of obtaining depth information is by using a stereo vision system where a pair of images is obtained from two separate cameras that are positioned a small distance from each other.

Although the concept of stereo triangulation for constructing a scene range map is straightforward, this apparent simplicity is quite deceptive. The major problem in stereo vision computation results from the correspondence problem. This arises when trying to identify corresponding points in the two images of the scene to establish a local triangulation. The difference in the positions of the two corresponding points in the respective images is called disparity. Disparity is a function of both the position of the point in the scene and of the position, orientation and physical characteristics of the stereo cameras [NAS89, BRI90, RYG91].

We will now discuss a stereo vision system.

MARVIN (Multiprocessor Architecture for VIsioN) guides a robot arm to pick up various objects in a cluttered scene. The system employs edge based stereo triangulation as a basis for three-dimensional description. Spatial parallelism is employed by dividing both left and right images into eight horizontal slices. A small overlap between adjacent slices is incorporated
to avoid boundary effects and to simplify the processes of recombination [RYG91].

Edge pixels have usually been used as basic units for matching in feature-based stereo. Unfortunately, this leads directly to the correspondence problem and is computationally expensive due to the large number of pixels involved. Recently, line segments have been used to reduce the number of primitives needed to be considered, thus alleviating both problems. However, when dealing with curves, piecewise linear approximations are unstable. The approximations can be significantly different between stereo images and the number of straight line segments required to approximate a curve grows with the curvature of the curve. Consequently, there has been a growing interest in using curves as the match primitive [BRI90].

Each curve is represented as an elastic string [BRI90] which can be locally stretched and shrunk by decreasing or increasing the density of the curve’s underlying parameter. Each curve is represented by an energy function which is composed of a stretching energy term which measures the change in density of the curve’s parameter, and a curvature energy term which encodes shape information by being defined so that points on the curve in higher curvature regions have more energy than those in lower curvature sections.
The density of the curve’s parameter along the curve is obtained by minimizing the integral of the energy over the length of the curve. The effect of this minimization is to reparameterize the curve so that the parameterization is more dependent on the shape of the curve than would have been the case from just using the arc length.

The development of a symbolically represented structural scene description from some set of predefined low-level processes can serve as a suitable source of primitives for stereo matching procedures. A structural description of a scene consists of descriptions of the objects in the scene and their interrelationships. This description is used to concisely represent the information contained in the scene in a symbolic form. For an example and more information on the parametric structural description see [BOY86].

3.4 Major Knowledge Representation Paradigms used to Represent Three-Dimensional Structures

A major problem encountered in three-dimensional structure representation is the choice of an appropriate knowledge representation scheme. We will discuss the various traditional knowledge representational methods from a structure representational viewpoint.
3.4.1 **Semantic Network Schemes**

Quillian and Sharpiro proposed the first major system using semantic networks [RIN88]. A semantic net represents information as a set of nodes interconnected by labelled arcs that represent relationships among the nodes.

Although many types of semantic networks exist, they all share the following features:-

1) All contain a data structure of nodes representing concepts, generally a hierarchy of nodes connected by IS-A and other property links [RIN88, RA088].
2) All contain specialized inferential procedures operating on the data structure of nodes. This involves the inheritance of information from the top level of the hierarchy downwards along IS-A links [RA088].

Semantic networks have some flaws, for example: whether to distinguish information about classes from information about instances of classes and how to handle exceptions.

Ballard has used a semantic network representation to achieve goal-directed image interpretation [RA088]. It views image analysis as a mapping between a model and its image. This represents information in three layers: an image data structure, a model layer and a sketch map. The model is a semantic network that represents appearances of and relations
among objects. The model contains generic information encoded as idealized structural prototypes from low level to high level and holds different kinds of knowledge about the image domain. It includes a relational network of nodes that can be identified with objects and concepts in the domain from which the scene is taken.

3.19

3.4.2 **First Order Logic**

First order logic in a knowledge representational context has been studied because of its usefulness in mechanical theorem proving. The use of the resolution principle as an inference technique and logic-based approaches to knowledge representation have also been investigated.

First order logic has the following advantages:

- One always arrives at a yes-or-no answer for any query because of the formal precision and interpretability of logic.
- There is no ambiguity in decision making and logic supplies an expressiveness that other representation schemes lack.

The database of the logic-based representation is mono-atomic in nature. New facts can be deduced from old facts and can be
added to the database. Thus, database consistency is guaranteed.

There are also a few disadvantages in first-order logic: [RA088, MOR87]

- It lacks an explicit method to index relevant knowledge because each rule is like any other and no hierarchical framework exists in which rules are embedded.
- All rules must be exhaustive, with no way to focus attention on a critical rule or rules.
- The awkwardness of handling incomplete and changing knowledge.

3.4.3 Frames

Knowledge has structure, arising either out of the structure apparent in the domain to be presented or the structures we have to impose to be able to deal usefully with large amounts of knowledge. Even in the cases where frames are logically equivalent to first order logic, it does not follow that the readability and expressive power of the two representations are equivalent [RIN88].
Minsky proposed that a useful knowledge base should be composed of highly modular chunks called frames [RIN88, BHA91, RAN90, RA088].

A frame is a data structure representing a stereotyped situation or class of objects. Attached to each frame are several kinds of information. Frames satisfy several requirements for a knowledge representation scheme. Important facts are made explicit and rarely used information is kept out of sight and can be inferred when required [W0089, ROS91].

A frame can be considered a slot-and-filler structure formed by the following network of nodes and relations:

The upper levels of the frame are fixed and represent unalterable truths about the object and the lower levels consist of terminals or slots which are filled with specific instances or data.

Frame systems are collections of related frames which are linked together by the sharing of slots. For visual scene analysis, a system’s different frames can describe the scene from different viewpoints. Once a frame is proposed to represent a situation, a matching process tries to assign values to each frame’s terminals. This process is controlled by the current goals and information attached to the frame.
A number of knowledge-based vision systems have adopted an underlying representation based on frames. ACRONYM [ROS91] models objects within frames as hierarchies of generalized cylinders. Interpretation proceeds by predicting viewpoint invariant features and merging local matches, subject to image and model constraints, resulting in a full three-dimensional match. VISIONS [RA88] incorporates a frame representation and employs a set of interpretation strategies under the control of a single process whose task is to monitor the performance and operation strategies.

FABIUS [ROS91] is a frame based computer vision system that has been implemented in Prolog. Object recognition consists of four main processes.

Image primitives are extracted from the image. As each model is selected, the leaves of the model hierarchy are instantiated to matching image primitives. A probabilistic updating procedure is applied to each frame and probabilities are propagated up the hierarchy and combined with values from other branches in order to yield a final vote in favour of the model. The data is classified by the model with the largest vote or whose vote exceeds some minimum threshold.
TUPIP's [W0089] world model has two roles. The first is to represent items expected to be encountered in images in order to allow their detection and recognition. The second role is to model the task. The world model is structured so that these roles are combined and is in effect the instantiation of the complete model. This is equivalent to achieving the goals of the task. Goals are represented by model elements which the user indicates explicitly as goal frames. The elements of the goal frames are related directly to one or more levels of abstraction of model elements which represent image content.

The lowest level of model elements in the world model can represent fairly complex image structures. The model does not extend explicitly below what users would normally consider as objects, i.e. edge segments.

3.4.4 Production Systems

Newel advocated production system architecture as a model of human reasoning [RA088]. A set of production rules operates on short-term memory (STM) which constitutes the system's internal description of the world in that particular image.

A rule-based system contains antecedent-consequent pairs called rules [RA088, RIN88, MOR89]. The data are usually
examined by rule antecedents and the consequent is responsible for data modification. Production systems have three main parts:

working memory (a database),
a set of production rules and the interpreter.

The working memory is a store containing objects defined by attribute value lists. Production rules are of the form antecedent \( \rightarrow \) consequent. The interpreter selects rules that match the contents of the working memory and perform the associated actions. The execution of such a production system can be defined as a series of recognize-act cycles.

A system for dimensional analysis of glass containers [MOR89] uses a hybrid between frames and rules as a representation scheme. Each frame consists of two components: rule and fact. A separate frame is created for each type of fact or individual task that will be encountered. For instance, an edge frame will contain rules related to edge detection, a description of how edges may be represented and functions for accessing edge information from other frames. Dividing the system into independent task-oriented frames achieves the goal of minimizing the size of a frame's rulebase, thereby making the match cycle of the inferencing process more efficient.
Nazif and Levine describe a system for low-level image segmentation [RA088]. A rule-based approach provides an expressive coding mechanism that accommodates diverse knowledge sources. This mechanism can capture general-purpose knowledge about image formation and perceptual grouping laws.

The model has three levels of production rules. The first contains knowledge rules encoding information about region properties, lines and areas. The second contains control rules, that are focus-of-attention rules and metarules. The third contains strategy rules, which select the set of control rules.

Blood vessel segmentation is an essential part of solving several practical applications in medical image-based diagnosis [VAN88]. The major application of this system is the automatic localization and quantification of abnormal strictures (steroses) in the blood vessel.

The knowledge-based system consists of three modular sections. The first part uses only geometrical and topological knowledge of line patterns to improve a primal image representation. The second part exploits general blood vessel knowledge. The third part is built around domain specific knowledge available from radiologists; the type of angiogram, important regions in the image, the medical history of the patient and the probability of a certain disease.
3.4.5 Object-Oriented Systems

Object-based systems provide features such as data abstraction, program modularity and inherent concurrency.

Objects encapsulate a private data set and can only be accessed or modified with the activation of their methods. Each object accepts messages that ask it to access or modify data and instantiate some class. Classes can be arranged in a hierarchy in which operations implemented at upper hierarchical levels can be automatically recognized at lower levels.

The object-oriented paradigm will be examined in more detail in chapter five.

3.5 Neural Networks

A parallel computational paradigm involving large interconnected networks of relatively simple neuron-like units is known both as connectionism and neural networking. There are five general entities which completely characterize the design and application of a neural network [JAM88]:

(i) The individual neural unit activation characteristic. The activation characteristic can be a threshold characteristic similar to a relay characteristic.
Conversely, the possibility of external inputs to the neural unit, inhibitory inputs and weighted and nonlinear combinations of inputs are possible.

(ii) The neural unit interconnection strategy or neural network structure. This may be as simple as allowing each neural output to be connected to all other neurons or constraining the neural unit interconnection to be localized. The neural network interconnection strategy may be quite complex and reflect an n-dimensional and hierarchical structure.

(iii) The goal or desired behaviour of the network. This may be reflected in the choice of a numerical performance index, enumeration of a set of stable network states or specification of a desired network output as a function of the network inputs and current state.

(iv) The choice of features used as input to the network, as well as interpretation of the state of the network.

(v) The training or preprogramming of the network. This is an optional aspect of the development of a network, since it is not required that networks have a learning capability in all applications. However, it is often desirable to store new information in the network as new inputs occur.

The local nature of the neural network's computational capability suggests that local image preprocessing are
candidates for neural network implementations. Examples are grey-level filtering, edge detectors and optical flow determination. Many higher-level tasks are also possible, due to a potential hierarchical arrangement of connectionist units [JAM88, BED90, WEI91].

Maximum Entropy (ME) image restoration [BED90] is proposed in the general framework of Lagrangian multipliers. The problem can be resolved by computing the maximum on \( \sigma \) of the minimum on \( f \) of the Lagrangian function \( L(f,\sigma) \). This function is considered as the energy function associated with a graded neural network. An iterative scheme is proposed which uses the powerful capability of neural networks in solving optimization problems.

3.6 Databases and Knowledge Representation

It has been said that a picture is worth a thousand words. Pictorial representations significantly enhance our ability to understand relations and structures [KAM91].

Most modern database management systems are designed to promote sharing of textual information. However, recently there has been much interest in various kinds of database management systems for non-textual information. There have been proposals for multimedia databases, spatial databases and object-oriented databases [GR089].
The type of information to be managed in an integrated image database management system can be classified broadly into five categories:

Iconic data: The images themselves are stored in analog or digitized form in an image store.

Image-related data: This includes registration information, the resolution and various format descriptors.

Information extracted from images: This is the information that results from processing images through a model-based filter. It includes such items as numerical and topological features as well as structural components and their relationships to each other.

Image-world relationships: These are relationships between image components and real-world entities.

World-related data: This is conventional textual data describing the abstracted world pertinent to the given application.

An integrated image database management system must facilitate storage and management of all five types of information. The advantages of data independence, data integrity, data sharing, controlled redundancy and security offered by conventional database management systems for textual data are required here for both image and textual data.
Spatial integrity constraints [PIZ89] can be used to depict unacceptable database states. Spatial integrity constraints specify acceptable database states and state transitions.

Constraint pictures contain arbitrary entities and relationships, as opposed to specific sets of geometric shapes and spatial associations. The data model providing basic picture interpretation semantics, that is associating pictorial primitives with corresponding real-world concepts, is patterned after the Entity-Relationship Model.

Extensions needed to support pictorial data management are provided. Pictorial database entities are associated with specific segments of a picture. Properties that determine the appearance of an entity in the picture are captured in primitive attributes. Properties that can be computed by applying mathematical functions to primitive attributes are captured in derived attributes.

Nonpictorial relationships are associated between entities that exist independent of their positions in the picture. Pictorial or spatial relationships are associations established between pictorial entities because of their relative positions in the picture.

IPAX's [AZZ90] (image processing and archiving system under Unix) main application area is robot vision including
grey-level, colour and stereo images. It was designed to provide a comfortable environment for image processing that allows an easy way of working with images and image processing operations.

Information about the image, such as dimension and type is stored by the system and automatically transferred to the image processing programs. An image archive provides short- and long-term storage of images and the information belonging to them.

The system includes a large number of subroutines for image processing concerning the fields of region- and line-based image segmentation, shape from shading, colour and stereo vision. The image archive is another component of IPAX. Archiving an image does not only mean storing pixel data but also all information kept by the system. It is also possible to store a number of semantically connected images such as stereo or colour images as one unit and have access to it via one name.

3.7 CAD/CAM and Other Methods of Three-Dimensional Representation of Structures

Many forms of engineering are supported by computerized tools that make work easier, for example CAD/CAM workstations. Controlled automation requires the specification of the following knowledge:
Organization of the domain-specific database at appropriate levels of granularity,
Relations among artifacts,
Computerized tools and
Rules regulating the progress of the engineering project.

All this knowledge is specified in MSL (Marvel Strategy Language). MSL is a standard for certain up market CAD systems. The most significant aspect of MSL is its translation and runtime support. This makes it possible to instantiate the generic database manager with the controlled automation desired for the particular engineering environment. The translator and runtime support automatically adjust database structure and behaviour to reflect new definitions [KAI88].

3.7.1 **Problem-Solving Models**

It is a known fact that solving problems in physical worlds requires the representation of large amounts of knowledge. There is much interest in using multiple models to capture the complex and diverse knowledge required during analysis. We can represent physical domains as graphs of models, where the nodes of the graph are models and the edges are assumptions that have to be changed in going from one model to another [ADD91].
Each model is a reformation of domain knowledge that can only be used if its assumptions lead to an acceptable approximation of the world. The main advantage of a model is its specificity. A model addresses a narrow scope of phenomena and is much smaller than the entire domain area.

A crucial aspect of problem-solving based models is that they include methods to automatically and efficiently select and change models. Models are based on sets of assumptions and these assumptions carry much semantic information. Represent this information qualitatively and it is possible to achieve efficient model-changing behaviours [ADD91, CLA89].

3.7.2 Pyramid Representations

Multiresolution (pyramid) image representations in image analysis and computer vision have been used as an approach to image-processing machines [ROS90, SHE89]

A cellular pyramid [SHE89] is a stack of arrays of exponentially tapering sizes. The cells on each level of the stack are connected to their eight neighbours. Each cell is connected to a block of children cells on the level below it where the blocks overlap.
Cellular pyramids have been found useful for performing a variety of image processing and analysis operations. The numbers of the computational steps grow logarithmically with the image.

The pyramid can be used to extract compact regions from an image in $O(\log \text{image size})$ steps. The method is as follows:

(a) A grey-level image is used as input to the bottom level of the pyramid, one pixel per cell.

(b) Each cell on every level above the bottom computes its intensity by pointwise multiplying the grey-level of its
16 children on the level immediately below it, with a four by four kernel.

(c) Each cell then decides whether it is a spot by comparing its intensity with the intensities in its three by three neighbourhood.

(d) Every spot computes the contrast between its intensity and the average of its neighbours' intensities. The contrast of parents and children are compared to select the best spots.

(e) Each spot serves as the root of a tree growing process which links up children with their parents according to their similarity and spatial proximity, using the formula

\[ w_i = d_i (L-1) + 2(L-1) \times k_i \]

where:
- \( w_i \) is the weight of the \( i \)-th parent,
- \( d_i \) is the grey-level difference between the cell and the \( i \)-th parent,
- \( k_i \) is the spatial distance between the cell and the \( i \)-th parent,
- \( l \) is the level number of the cell and
- \( L \) is the maximum level number in the pyramid.

(f) The result of this linking process at the bottom of the pyramid is the set of pixels assigned to the spot. This set defines the compact region associated with the spot.
3.7.3 **Blackboard Approach**

An architecture that has also been used in representations of molecular visualization is a blackboard approach [OLS91]. A blackboard approach emphasizes explicit control and indirect communication. Logic expressed in several different programming paradigms, for example instances, rules, frames and procedural code can be integrated [POM90].

3.7.4 **Declarative Knowledge Programming**

The last knowledge representation paradigm we will discuss is declarative knowledge programming.

Ordinary programming languages traditionally support algorithmic expression of knowledge as procedures, while other types of knowledge are less easily codified. This other types of knowledge can be represented in a declarative style [BUT88].

Declarative knowledge can be expressed as consistency relations and can also represent knowledge of inconsistencies together with strategies to change the knowledge base to maintain consistency. Declarative knowledge can be characterized as knowledge devoid of control information [SCH86, RA088].
Properties of declarative knowledge is:

- Incompleteness of the description,
- The declarative form suppresses detail about when to apply these rules,
- Knowledge about how to use other knowledge, called metaknowledge.
- Declarative knowledge differs in the way that control aspects are handled as described in the next paragraph.
- The usefulness of hidden control increases as the number of interdependencies among modular pieces of knowledge increases.

There are some pitfalls in using the declarative style. It is a direct consequence of the hidden control aspects and involves firing deductive rules with inconsistent data and having loops in rule applications [SCH86]. This makes declaritive knowledge impractical to use in a wide domain where most knowledge is inconsistent and incomplete.

3.8 Conclusion

It is clear that a lot of research is being done in the field of knowledge representation of three-dimensional structures. The traditional knowledge paradigms as well as new methods of representation have had a lot of attention.
Graph-theoretic approaches to object location are currently receiving a lot of attention [DAV91]. This includes association graphs for matching [YAN89]. Research in three-dimensional mathematical morphology and associated special-purpose computer architectures to image processing, target tracking and both three-dimensional data analysis and display is being done [KEN91].

Binary forest segmentation [NIC91] and spatio-temporal reasoning [ORR90] and geometric reasoning [BUX88] are areas in which future research will be conducted.

Chapter four is a discussion of structure graph grammars. We will examine the grammar in detail as it is our underlying method of three-dimensional knowledge representation.
4.1 Introduction

The use of structure graphs as a three-dimensional representation scheme will be demonstrated. Definitions of structure graphs will be given in section 2. Structure graph grammars (SGG) as defined in [BAR91] as well as examples will be discussed in sections 3 and 4.

4.2 Structure graphs

We need to define the concept of a structure parameter in order to define a structure graph [BAR91].

DEFINITION

A structure parameter \( p \) is a real-valued number that describe the geometric properties of an edge of a graph or the relationship between two edges that describe the graph.
DEFINITION

A structure graph $G$ is a 2-tuple $(V, E)$ where $V$ is the vertex set and $E$ the structured edge set of $G$.

An element of $E$ has the form:

$$(v_1 v_j, p_{ij}, p_{ij2}, \ldots, p_{ijm}),$$

$$(v_k v_1, p_{k1}, p_{k12}, \ldots, p_{k1m}),$$

$$p_1, p_2, \ldots, p_r$$

where $v_1, v_j, v_k, v_1 \in V$ ($i, j, k, l \in \{1, 2, 3, \ldots, |V|\}$ and

$$(p_{ij}, \ldots, p_{ijm}, p_{k1}, \ldots, p_{k1m},$$

$$p_1, \ldots, p_r)$$

are structure parameters ($m, r \in \mathbb{N}$).

For a more detailed definition see [BAR91].

The instance of structure graphs that will be referred to in the definition of structure graph grammars, will now be defined.

DEFINITION

An angle-length structure graph is a structure graph $G$ where $p_i = 0$ for all $i \in \{ij_2, \ldots, ij_m, kl_2, \ldots, kl_m, 2, \ldots, r\}$.

The structure parameters $p_{ij}, p_{k1}$ and $p_3$ are then assigned the following geometrical properties:

- $p_{ij}$ gives the length of edge $v_1 v_j$,
- $p_{k1}$ gives the length of edge $v_k v_1$ and
- $p_3$ gives the angle constituted by the edges $v_1 v_j$ and $v_k v_1$ in $G$. 
4.3 Structure graph grammars

Structure graph grammars is a class of graph grammars that will generate a family of structure graphs. We will only consider two extensions of structure graph grammars that have been defined in [BAR91].

4.3.1 Structure graph grammar - extension 4 (SGG-4)

This extension can generate cyclic, acyclic, connected and/or disconnected structure graphs.

**DEFINITION**

Structure graph grammar - extension 4 (SGG-4) is defined as a 4-tuple $G = (N, T, S, R)$ where:

- $N$ denotes a finite set of non-terminal symbols,
- $T$ denotes a finite set of terminal symbols,
- $S$ denotes the start symbol and
- $R$ denotes the set of production rules.

A production rule $r$ in $R$ is a 3-tuple $r = (F, L, C)$ where:

- $F$ is a vertex or an edge in $S$ that must be replaced,
- $L$ is a vertex or an edge that must replace $F$ and
- $C$ is the context.
The context $C$ is defined as a 3-tuple $C = (Ch/V, T)$ where:

$Ch$ is the angle context,

$U$ is the global permitting context and

$T$ is the global forbidding context with $U, T \subset N U T$.

There are four replacement operations.

i) Both $F$ and $L$ are nodes. $L$ will replace $F$ if the context $C$ is valid.

ii) $F$ is a node and $L = (L, 1)$ is an edge, $L = n_1-n_2$. $F$ will be replaced by $L$ only if one of the the nodes of $L$ is equal to $F$ ($F = n_1$ or $F = n_2$).

iii) $F = n_1-n_2$, is an edge and $L$ is a node. $L$ will replace $F$ only if $L$ is one of the nodes of $F$.

iv) Both $F$ and $L$ are nodes. Node $L$ will replace $F$ only if both the edges $F$ and $L$ have a node in common.

See [BAR91] for a more detailed description of the definition and the replacement operations.

[BAR91] showed that it is possible to use structure graph grammars to describe some organic molecules, for instance, a homologous series of alkanes. This is made possible with edge replacement that is supported in SGG-4. We will show
that it is possible to generate all aliphatic and aromatic organic molecules.

Some examples of how a homologous series are generated, are shown in [BAR91].

As the number of atoms increases, so does the number of possible arrangements of the atoms. As we go up the series of alkanes, the number of isomers of successive homologs increases at a surprising rate. There are 3 isomeric pentanes, 5 hexanes, 9 heptanes and 75 decanes. For the twenty-carbon icosane, there are 366,319 possible isomeric structures.

The problem of isomeric compounds is solved with the use of indexes $i$, $j$ and $k$ where:

- $i$ is the number of carbon atoms in the longest chain,
- $j$ is the number of the carbon atom where a branch starts
- $k$ is the number of carbon atoms in the branch,

and thus a specific alkane can be generated. The grammar that will generate a specific branched hydrocarbon and the replacement operations are given in [BAR91].

It is also possible to generate alkenes and alkynes. The
homologous series of alkenes and alkynes have less than the maximum quantity of hydrogen and are referred to as unsaturated hydrocarbons.

In this case, indexes i, j and k are used to show the length of the chain of carbons and between which two carbon atoms the double or triple bond are situated.

The grammar that will generate alkenes and an example is given in [BAR91].

Organic chemistry divide compounds in two groups, the aliphatic and aromatic compounds.

Three of the aliphatic compounds can be generated by SGG-4. They are alkanes, alkenes and alkynes.

It is also possible to generate other aliphatic compounds which has the general formula of R-F where:

R is a alkyl group,
F is a functional group.

Examples of these compounds are:

Alkyl-halides R-C-X
Alcohols R-C-OH
Aldehydes R-C(O)H
Carboxylic acids \( R\text{-COOH} \)

Amides \( R\text{-CONH}_2 \)

Nitriles \( R\text{-C}≡\text{N} \)

Amines \( R\text{-NH}_2 \)

It is possible to generate an alcohol with SGG-4. The grammar that generates the alcohol and a graphic presentation of how it is generated will now be given. We will use the variables \( B \) for the bond length and the variable \( \sigma \) for the angles.

We will generate propanol \( (\text{CH}_3\text{CH}_2\text{CH-OH}) \)

\[ G = (N, T, S, R) \text{ where:} \]

\[ N = \{ s_c, s_h, s_{\text{OH}} \}, \]

\[ T = \{ C, H, \text{OH} \}, \]

\[ S = \{ s_c \} \text{ and} \]

\[ R = \{ \]

(1) \( (s_c, (s_c-s_h, 0.11\text{nm}), (/ \{ s_c \}, \{ s_h, s_{\text{OH}} \}) ) \); 

(2) \( (s_c, (s_c-s_h, 0.11\text{nm}), (((s_c-s_h, 0.11\text{nm}), 109,5^\circ)) / \{ s_c, s_h \}, \{ C, H, \text{OH} \})) ; \)

(3) \( (s_c, (s_c-s_h, 0.11\text{nm}), (((s_c-s_h, 0.11\text{nm}), 109,5^\circ), \)

\( ((s_c-s_c, 0.153\text{nm}), 109,5^\circ)) / \{ s_c, s_h \}, \{ C, H, \text{OH} \}) ; \)

(4) \( (((s_c-s_h, 0.11\text{nm}), (s_c-s_c, 0.153\text{nm}), ( (((s_c-s_h, 0.11\text{nm}), 109,5^\circ), \)

\( (s_c-s_c, 0.153\text{nm}), 109,5^\circ)) / \{ s_c, s_h \}, \{ C, H, \text{OH} \}) ; \)

(5) \( (((s_c-s_h, 0.11\text{nm}), (s_c-s_{\text{OH}}, 0.11\text{nm}), 109,5^\circ), ((s_c-s_{\text{OH}}, 0.11\text{nm}), 109,5^\circ)) / \{ s_c, s_h \}, \{ C, H, \text{OH} \}) ; \)
(6) \((s_c, C, \{\} / \{s_c, s_h, s_{oh}\}, \{H, OH\})\); 
(7) \((s_h, H, \{\} / \{s_h, C, s_{oh}\}, \{s_c, OH\})\); 
(8) \((s_{oh}, OH, \{\} / \{H, C, s_{oh}\}, \{s_c, s_h\})\). 

We will now show how this alcohol is generated.
(4) 
\[ \begin{array}{c}
S_H \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_H \\
\end{array} \]

(3)* 
\[ \begin{array}{c}
S_H \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_H \\
\end{array} \]

(4) 
\[ \begin{array}{c}
S_H \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_H \\
\end{array} \]

(3)* 
\[ \begin{array}{c}
S_H \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_H \\
\end{array} \]

(5) 
\[ \begin{array}{c}
S_H \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_C \\
| \\
| \\
S_OH \\
\end{array} \]
After rules 6, 7 and 8 have been applied:

\[
\begin{array}{cccccc}
\text{H} & \text{C} & \text{C} & \text{C} & \text{OH} \\
\text{H} & \text{H} & \text{H} & & \\
\end{array}
\]

It is not only possible to generate general formula compounds, but specific compounds can also be generated. An example of this is 2-butylamine.

\[
\begin{align*}
( \text{CH}_3 &-\text{CH(NH}_2\text{-CH}_2\text{-CH}_3 ) \\
\end{align*}
\]

We use two indexes, \(i\) and \(j\). The number of carbon atoms in the chain designated by \(i\) and \(j\) will show the position of the amine functional group.

The grammar that will generate 2-butylamine:

\[
G = (N, T, S, R) \text{ where:}
\]

\[
N = \{ s_x (1 \leq x \leq i), s_{H^2}, s_{F} \},
\]

\[
(F = \text{NH}_2)
\]

\[
T = \{ C_x (1 \leq x \leq i), H, \text{NH}_2 \},
\]

\[
S = \{ s_1 \} \text{ and}
\]

\[
R = \{
(1) (s_1, (s_1 - s_{H^2}, 0.11nm), ( / \{ s_1 \}, \{ H \}))
\};
\]
The indexes with values $i = 4$ and $j = 2$ are now given to the grammar. We will now show how the 2-butylamine compound is generated.

The diagram shows the grammatical structure of 2-butylamine compound.
After rules 7, 8 and 9 have been applied the final product looks like this:

R -F-R compounds can be generated with a different set of production rules. The major difference between the R-F and R-F-R aliphatic compounds will be the context of the production rules.

It is thus possible to generate the following compounds.

Ethers \( R-O-R \)
Ketones \[ \text{R-C(O)-R} \]
Esters \[ \text{R-C(O)O-R} \]
Anhydrides \[ \text{R-C(O)OC(O)-R} \]

We will generate a symmetrical ketone \( \text{CH}_2\text{CH}_2\text{COCH}_2\text{CH}_3 \).

\[ G = (N, T, S, R) \text{ where:} \]
\[ N = \{s_c, s_h, s_{co}\}, \]
\[ T = \{C, H, C=O\}, \]
\[ S = \{s_c\} \text{ and} \]
\[ R = \{ \]
\( (1) \) \( (s_c', (s_{c-h}, 0.11\text{nm}), (\{s_c', s_{co}\}) \} \); 
\( (2) \) \( (s_c', (s_{c-h}, 0.11\text{nm}), ((s_{c-h}, 0.11\text{nm}, 109.5^\circ)) / \{s_c', s_h\}, \{C, H, C=0\}) \}; 
\( (3) \) \( (s_c', (s_{c-h}, 0.11\text{nm}), ((s_{c-h}, 0.11\text{nm}, 109.5^\circ), 
\quad ((s_{c-h}, 0.153\text{nm}, 109.5^\circ)) / \{s_c', s_h\}, \{C, H, C=0\}) \}; 
\( (4) \) \( ((s_{c-h}, 0.11\text{nm}, (s_{c-s_c}, 0.153\text{nm}), ((s_{c-h}, 0.11\text{nm}, 109.5^\circ), 
\quad ((s_{c-h}, 0.153\text{nm}, 109.5^\circ)) / \{s_c', s_h\}, \{C, H, C=0\}) \}; 
\( (5) \) \( ((s_{c-h}, 0.11\text{nm}, (s_{c-s_c}, 0.153\text{nm}, ((s_{c-h}, 0.11\text{nm}, 
\quad 109.5^\circ), ((s_{c-h}, 0.153\text{nm}, 0^\circ)) / \{s_c', s_h\}, \{C, H, C=0\}) \}; 
\( (6) \) \( (s_{co}, (s_{co-s_c}, 0.153\text{nm}, ((s_{co-s_c}, 0.153\text{nm}, 0^\circ)) / \{s_{co}, 
\quad s_{co}, s_{co}\}, \{C, H, C=0\}) \}; 
\( (7a) \) \( (s_c', (s_{c-h}, 0.11\text{nm}, ((s_{co-s_c}, 0.153\text{nm}, 0^\circ)) / \{s_c', 
\quad s_h, s_{co}\}, \{C, H, C=0\}) \}; \]
We will now demonstrate how this grammar will generate a ketone.

\[ S_C \xrightarrow{(1)} S_H S_C S_H \]

\[ S_H \xrightarrow{(2)*} S_C \]

\[ S_H \xrightarrow{(4)} S_C \]

\[ S_H \xrightarrow{(3)*} S_C \]
4.16

- (5) - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co}$

- (6) - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co} \rightarrow S_c$

- (7a) - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co} \rightarrow S_c$

- (7b)* - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co} \rightarrow S_c \rightarrow S_H$

- (8) - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co} \rightarrow S_c \rightarrow S_c \rightarrow S_c \rightarrow S_c$

- (7b)* - $S_H \rightarrow S_c \rightarrow S_c \rightarrow S_{co} \rightarrow S_c \rightarrow S_c \rightarrow S_c \rightarrow S_H$
After rules 9, 10 and 11 have been applied the final product look like this.

\[
\begin{array}{c}
\text{H} \\
\text{H} \\
\text{H} \\
\text{C} \\
\text{C} \\
\text{C} \\
\text{CO} \\
\text{C} \\
\text{C} \\
\text{H} \\
\text{H} \\
\text{H} \\
\end{array}
\]

From this example above we see that all aliphatic compounds except cyclic compounds can be generated with SGG-4.

A general grammar to generate all aliphatic compounds can be found in appendix C.

In the next section we will look at the generation of cyclic compounds.

4.3.2 **Structure graph grammar - extension 6 (SGG-6)**

The homologous series of cycloalkanes can not be generated by SGG4. The SGG-6 is capable of generating cyclic molecules [BAR91].

**DEFINITION**

A structure graph grammar - extension 6 (SGG-6) is a 5-tuple \( G = (N, T, S, R, A) \) where:
N denotes a finite set of non-terminal symbols,
T denotes a finite set of terminal symbols,
S denotes the start symbol,
R denotes the set of production rules and
A denotes the edge-pair set.

A production rule \( p \in R \) is a 2-tuple \( p = (F, L) \) where:
F is a vertex or an edge that must be replaced and
L is the vertex or an edge that must replace F.

An element of the edge-pair set A has the form:
\(((b_1, l_1), (b_2, l_2), \sigma)\) where:

- \( b_1 \) and \( b_2 \) are edges in the structure,
- \( l_1 \) and \( l_2 \) the length of \( b_1 \) and \( b_2 \) respectively and
- \( \sigma \) the angle formed by the edges \( b_1 \) and \( b_2 \).

Two of the replacement operations are defined as follows:

i) If \( L \) is a node and the contexts in A allow the label replacement, then \( L \) can replace \( F \).

ii) If \( L \) is a vertex, then \( L \) may replace \( F \) only if \( F \) is one of the two end nodes of \( L \).

The replacement of \( F \) with \( L \) is defined as follows:

a) Remove all vertices in the structure graph.
b) Replace node F with vertex L.

c) Connect all the nodes in the structure graph in such a way that all contexts of the edge-pair set is satisfied.

See [BAR91] for more detail on the replacement operations.

We will now demonstrate how we can generate aromatic compounds. We will generate a simple benzene molecule.

\[
\begin{align*}
&H \\
&C \\
&H-C \\
&C-H \\
&H \\
&H-C \\
&C-H \\
&H
\end{align*}
\]

The following grammar will generate a benzene molecule.

\[
G = (N, T, S, R, A) \text{ where:}
\]

\[
N = \{ s_c, s_h \},
\]

\[
T = \{ C, H \},
\]

\[
S = \{ s_c \},
\]

\[
R = \{
(1) (s_c, s_c - s_h);
(2) (s_c, s_c - s_c);
(3) (s_c, C);
(4) (s_h, H)\}
\] and
\[ A = \{ \]
\[ ((s_C - s_H, 0.11\text{nm}), (s_C - s_C, 0.139\text{nm}), 120^\circ); \]
\[ ((s_C - s_C, 0.139\text{nm}), (s_C = s_C, 0.139\text{nm}), 120^\circ); \]
\[ ((s_C = s_C, 0.139\text{nm}), (s_C - s_H, 0.11\text{nm}), 120^\circ)} \].
We have defined the SGG-4 grammar that will generate 2-butylamine. We will also use two indexes, i and j in the definition of the SGG-6 grammar. The number of carbon atoms in the chain will be designated by i and j will show the position of the amine functional group.
The SGG-6 that will generate the 2-butylamine molecule is given by:

\[ G = (N, T, S, R, A) \]

where:

\[ N = \{s_x (1 \leq x \leq i), s_h, s_f\}, \]

where \( F \) is the functional group \( \text{NH}_2 \),

\[ T = \{C_x (1 \leq x \leq i), H, F\}, \]

\[ S = \{s_j\}, \]

\[ R = \{ \]

1. \((s_x, s_x - s_h) 1 \leq x \leq i;\)
2. \((s_x - s_h, s_x - s_{x+1}) 1 \leq x \leq i-1;\)
3. \((s_x - s_h, s_x - s_f) x = j;\)
4. \((s_x, C) 1 \leq x \leq i;\)
5. \((s_f, F)\)
6. \((s_h, H)\}

and

\[ A = \{ \]

\((s_x - s_h, 0.11\text{nm}), (s_x - s_h, 1.11\text{A}), 109.5^\circ);\)

\((s_x - s_h, 0.153\text{nm}), (s_x - s_{x+1}, 0.153\text{nm}), 109.5^\circ);\)

\((s_x - s_h, 0.11\text{nm}), (s_x - s_f, \beta\text{nm}), \sigma^\circ);\)

\((s_x - s_{x+1}, 0.153\text{nm}), (s_x - s_f, \beta\text{nm}), \sigma^\circ);\)

\((s_x - s_{x-1}, 0.153\text{nm}), (s_x - s_f, \beta\text{nm}), \sigma^\circ).\)
Although the SGG-6 grammar of this molecule is much shorter than the SGG-4 grammar on page 4.9 it is clear that the difference in the generation of the molecule lies in the replacement method.

4.4 Conclusions

Section 4.3.1 has shown us that almost all aliphatic compounds can be generated using SGG-4. By using indexes it is capable to generate specific compounds. In section 4.3.2 we used SGG-6's to generate aromatic compounds as well as aliphatic compounds.

All the aliphatic compounds with formula R-F-R and R-F can be generated with SGG-4 and SGG-6.

This means that we can generate all organic compounds with SGG-4 and SGG-6. In the following chapter we will examine object-oriented methods of three dimensional representation.

Although SGG-4 and SGG-6 can generate complex branched alkanes and other compounds in theory, it is not practical. A solution to this problem must also be found.

In the rest of this dissertation, when we refer to structure graph grammars, we are speaking of structure graph grammar - extension 4.
Knowledge can be either explicit or implicit. In this chapter we will discuss implicit and explicit knowledge. We will look at the advantages and disadvantages of the two types of knowledge. Object-oriented programming will be examined as well as some examples of object-based representation models.

When representing knowledge the knowledge engineer must constantly decide how implicit or explicit to make the knowledge. In conventional programming, the knowledge that is incorporated in a program is nearly always completely implicit [DAV82]. It does not really exist as knowledge in the program other than as knowledge the programmer uses to design its functioning. In AI paradigms such as rule-based programming, the attempt is to make knowledge explicit as general rules so that this knowledge can be applied to new situations that the developer may not have foreseen [RIN88].
So far, the main advantages of making knowledge explicit are the greater generality and flexibility that can be given to applications. Rather than having the knowledge hard wired into a conventional program that cannot compare different pieces of knowledge to reach its own conclusions, it is very useful to provide a way of modelling the general process by which practical results are obtained [DAV82, RAM88, TEL89].

Obviously, it is never possible to make everything the result of a process of explicit reasoning. Certain things must be taken for granted as implicit and used for a limited number of events of explicit analysis or reasoning.

This is then the main distinguishing features of humans as opposed to machines. At any given time and for any reason, we can decide to change something from implicit to explicit status. It is important to sense when it is necessary to question things that are normally taken for granted. In AI systems, the programmer decides what knowledge will be implicit and what knowledge will be explicit [RAM88, DAV82].

The main trade-off is flexibility versus efficiency. The question is how much knowledge can be made explicit without losing adequate performance. The advantage of modular paradigms like rule-based and object-oriented systems is that changes can usually be made by local modifications that do not alter the
functioning of the program as a whole. Often knowledge can be made explicit by adding and subtracting functioning classes that do not require that the application be completely rewritten.

5.3 General Representation

The basic premise is that any generalised reasoning mechanism, including all humans and knowledge-based systems, must possess some internal form of representation which is used not only to carry out its intelligent tasks, but which is also used in order to explain how it performed these tasks to any other intelligent agent [RIN88, FIS91].

If we manage to reduce the problem domain to a simple set of facts and their various relationships, we almost always remove the more important knowledge at the same time. This is the commonsense knowledge that we use all the time, but which must be input into every knowledge based system [RAN90].

Most expert systems cannot solve problems that are less complex or more trivial than the ones they were designed to solve. This means that for every problem we can get the system to solve, there are hundreds simpler problems that it cannot solve.

A general representation method that exists currently is the object-oriented method. In the rest of this chapter we will discuss the object-oriented method.
5.4 Object-Oriented Programming (OOP)

Object-Oriented programming is the most recent development in the programming area [BOR92]. Computer scientists agree that object-oriented programming provides a powerful means for controlling access to data, data abstraction, program modularization and structural knowledge representation.

Unlike procedure-oriented programming, object-oriented programming treats data as primary and functions as secondary. In other words, instead of data being what the functions use, functions are what the data do. Functions are closely associated with the data and do not stand alone.

5.4.1 Data Abstraction

Abstraction can be defined as a general idea which concentrates on the essential qualities of something, rather than on concrete realizations or actual instances [BIS85].

Data Abstraction: a data type is not just a definition of a set of admissible values, but the concept of type is intimately connected with the operations that are meaningful for its objects. The protection of data must be built into the language itself.
5.4.2 Objects and Data Abstraction

The idea of data abstraction can be used to create an object by defining a block of data together with the functions necessary to operate on that data. The data represent the information contained in the object and the functions model the behaviour of the object. The functions thus define the operations that can be performed on that object. The data are not accessible to the outside world. The only way to do anything to an object is by calling one of the functions that implement the object's behaviour [RIN88, RAM88, DAV82, TEL89, BOR92].

Figure 5.1 Data abstraction.
Objects essentially encapsulate a private data set, and can only be accessed or modified with the activation of their methods.

5.4.3 Classes and Methods

In object-oriented programming terminology, the template defining an object's data type is usually called a class. Thus, each object is an instance of a class [RIN88, TEL89, BOR92].

The functions that operate on an object have a special name. They are known as methods because that was the name used in the object-oriented language SmallTalk.

The methods define the behaviour of an object.

5.4.4 Messages

Objects communicate with each other by sending messages to each other. These messages are requests to the other objects to perform a specific method. Class and state methods both describe the order in which actions will be performed when a message is received by the object.

These actions are the sending of messages to other objects,
modifying the state variables and the sending of answers to the original object that has sent the message initially. Messages are the only way to communicate with an object and is also the only way to read and change the state variables [TEL89, BOR92]. Figure 5.2 shows how a message behaves when moving from a sub-class to a super class.

Figure 5.2  Messages
5.4.4 Data Encapsulation

This feature is important to ensure the integrity and changeability of systems and to lessen the dependence of modules on one another. The state of the module is stored in local variables that are only visible to local procedures. Only a local procedure can change the values of these variables. A module can only communicate with other modules through its interface procedures [BIS85].

5.4.6 Inheritance

A real-world object is often an extension of an existing object. For example, we are defining a new object by pointing out how the new object's characteristics and behaviour differ from that of the old one.

Object-oriented programming supports this notion of defining a new object in terms of an old one. The term inheritance [BOR92, TEL89] is used for this concept because one object can be thought of as inheriting the properties from another. It is a class inheriting the behaviour from another class. Inheritance imposes a parent-child hierarchical relationship among classes where a child inherits from its parent. The parent class is often called the super class or base class. See figure 5.3.
5.4.7 Multiple Inheritance

In real-world situations, an object can have the behaviour that may be attributed to more than one immediate ancestor. Figure 5.4 demonstrates how a table can be defined by using two objects.
5.4.8 Polymorphism

Polymorphism is the property that allows an operation to have different behaviour in different objects. Different objects react differently to the same message. The particular version to be executed is determined at run time. This is called late binding [BOR92].

Polymorphism plays an important part in simplifying the syntax of performing the same operation on a collection of objects.
5.5 Object-Based Three-Dimensional Representation

In this section we will discuss two object models. The first one is a general model for object recognition while the second model is used in medical diagnosis.

An object-oriented data model has a hierarchy of abstract data types. An instance of a type is an object that matches the template defined for the type. The values for a particular object's properties can be stated explicitly, inherited or obtained implicitly by the evaluation of a function. Object orientation provides modularity through inheritance and association of functions with object types. The Thin Line Code (TLC) [JAG89] model for image recognition benefits from this modularity in its application diversity.

Lines are often important entities for analysing images. Information is inherently found as lines, as in text, handwriting, diagrams, and fingerprints. In gray-scale images, lines are often a derived feature of interest as a result of such image-processing operations as edge detection or contour tracking. Because of the prevalence of lines across different applications, TLC is a general representation to facilitate processing and pattern recognition.

TLC is structured as a four-level hierarchy of objects. At each level, an object is a particular structural line entity with a
defined template of attributes and associated functions. The levels, from low to high in structural complexity are shown in Table 5.1.

Table 5.1 The Thin Line Code model.

<table>
<thead>
<tr>
<th>Level</th>
<th>Hierarchy</th>
<th>Typical Attributes</th>
<th>Typical Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Structure</td>
<td>Label, bounding box, constituent lines, constituent composites</td>
<td>Delete, TotalLength</td>
</tr>
<tr>
<td>2</td>
<td>Composite</td>
<td>Label, type, constituent lines</td>
<td>OpenLoop</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FindBifurcation</td>
</tr>
<tr>
<td>1</td>
<td>Line</td>
<td>Label, bounding box, length</td>
<td>Delete, JoinPair, LineLength</td>
</tr>
<tr>
<td>0</td>
<td>PCC (chain)</td>
<td>Label, width</td>
<td>Code, Decode</td>
</tr>
</tbody>
</table>

The TLC model is designed to serve as a tool over different image-recognition applications. It was fundamental that generality was not obtained by attempting to build in the most complete set of functions and attributes that could be foreseen. As TLC is applied to new applications, derived object types include new functions and attributes as required. The TLC image model is a skeleton representing levels of composition of image lines.
The second model we discuss is based on the selective inversion-recovery (SIR) method for MRA (Magnetic Resonance Angiograms) [FES91]. Accurate descriptions of arterial trees will be useful for quantitative diagnosis of atherosclerosis, for surgical or treatment planning, for monitoring disease progress or remission, and for comparing efficacies of treatments.

Projection-reconstruction from only a few views is an ill conditioned problem in general. The goal is typically to obtain quantitative descriptions of arterial shape. The parametric approach exploits a priori knowledge of the structure of arteries and translates the reconstruction problem into an object estimation problem.

The method is based on global criterion, to maintain accuracy at the low SNR typical of noninvasive methods; bifurcating arteries are explicitly modelled; there are no empirically determined thresholds; overlapping vessel projections are accommodated and the time-dependence of contrast density is modelled.

A generalized cylinder (GC) object model with slowly varying elliptical cross-sections can be approximated by a set of parallel ellipses. Such a set of ellipses, see figure 5.5, can be parameterized by z. We call the collection a single-valued generalized cylinder. (SGC)
Objects that wind back upon themselves must be represented by more than one SGC, and are called multivalued. Proper patient positioning is essential and the arteries of interest should be aligned as close as possible to the rotational axis.

Each SGC cross-section has three attributes: position, shape and content. An ellipse's position is parameterized by xy coordinates of its centre, denoted by \((c_x, c_y)\). The
ellipse's shape attribute is parameterized by its radius, eccentricity and orientation denoted by $r$, $\sigma$ and $\tau$ (see figure 5.6). The content attribute parameterizes the density within an artery.

Figure 5.6 Elliptical cross-section.
5.6 Conclusion

In this chapter we have looked at how knowledge can be represented in an explicit and implicit way. The object-oriented programming paradigm is well-suited for representing general knowledge in the explicit as well as the implicit way. A few of the most important features of the object-oriented programming paradigm have been discussed with the view on following chapters.

In the next chapter we will give an overview of all the objects used to solve the problem of knowledge representation of three-dimensional structures using structure graph grammars.
CHAPTER SIX

Definition of Grammar Objects

6.1 Introduction

The model that we define here is largely based on how a grammar of the class of Structure Graph Grammar - extention 4 (SGG-4) looks in the real world.

In the following sections classes will be defined using BNF-notation. The BNF-notation used are defined and described in Appendix B.

6.2 Objects

Because of the nature of structure graph grammars, it is quite easy to represent the grammar in an object-oriented language. In figure 6.1 we show the different objects that are combined to form the final grammar object. Figure 6.2 shows how we use composition to enforce data security. From figure 6.3 it is clear that there are interactions between all the objects used.
Figure 6.1 Overview of objects.

Figure 6.2 Enforcing Data Encapsulation.
The following are data structures we use in the classes. This particular structure are used in the generating of the structures.

```c
struct INHOUD
{
  lyn. // Node name or line
  len. // Length of line
  hoek. // Angle of context
};
```

```c
struct GENEREER_DATA
{
  naam.     // Name of line or node F
  inhoud binne[]. // Array of contexts
};
```
All the classes that are used in the prototype system, are defined in the next sections. Table 6.1 is a list of the objects and on which page it is defined.

### Table 6.1 List of classes.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>6.4</td>
</tr>
<tr>
<td>Angle Context</td>
<td>6.5</td>
</tr>
<tr>
<td>P_Rule</td>
<td>6.6</td>
</tr>
<tr>
<td>SGG4</td>
<td>6.10</td>
</tr>
</tbody>
</table>

#### 6.2.1 Class Group

This public class defines the structure that contains the definition of the node or line F and L respectively.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super class</td>
<td>None</td>
</tr>
</tbody>
</table>

State variables:
- Access Public: Group // Line or node
- Length // Length of line

State Member Functions:
- Access Public:
  - Group // Initializer
  - Show

```cpp
Group::Group(New_group, New_length)
Group := New_group.
Length := New_length.
```
6.2.2 Class Angle_Context

This public class uses inheritance to define itself. The only difference between this class and its super class is the state variable angle. The rest of the state variables are inherited.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Angle_Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super class</td>
<td>Group</td>
</tr>
<tr>
<td>State variables :</td>
<td>Access Public: Angle</td>
</tr>
<tr>
<td>State Member Functions :</td>
<td>Access Public:</td>
</tr>
<tr>
<td></td>
<td>Angle_Context // Initializer</td>
</tr>
<tr>
<td></td>
<td>Show</td>
</tr>
</tbody>
</table>

Angle_Context::Angle_Context(New_group, New_length, New_angle) : Group(New_group, New_length)

    Angle := New_angle.

Angle_Context::Show(nr)

    [nr > 0]

    IfTrue:
        Out(Group, Length, Angle).
6.2.3 **Class P Rule**

The production rules will be stored in objects of this class.

A production rule \( p \) in \( P \) is a 3-tuple \( (F, L, C) \) where:

- \( F \) is a node or line in \( S \) that is the replaced,
- \( L \) is a node or line replacement,
- \( C \) is the context.

The context \( C \) is defined as a 3-tuple \( (Ch/U, T_1) \) where:

- \( Ch \) is the angle context,
- \( U \) is the global permitting context,
- \( T_1 \) is the global forbidding context.

This is a composite class because it makes use of other classes as objects in this class.

<table>
<thead>
<tr>
<th>Class name</th>
<th>P_Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super class</td>
<td>None</td>
</tr>
</tbody>
</table>

**State variables:**
- Access Public: Angle_context, Group *L, Group *F, Angle_Context, Permissible, Prohibit

**State Member Functions:**
- Access Public:
  - P_Rule // Constructor
  - P_Rule // Overriding
  - Show
  - ~P_Rule // Destructor
P_Rule::P_Rule()
| Symbol, Distance, Angle, Counter |
Symbol := "".
Distance := 0.
Angle := 0.
Permissible := "".
Prohibit := "".
F := NEW Group(Symbol, Distance).
L := NEW Group(Symbol, Distance).
[Counter := 0, Counter < 5, Counter++]
WhileTrue:
    Ch[Counter] = NEW Angle_Context(Symbol, Distance, Angle).

P_Rule::P_Rule(Nr, Name)
| Symbol, Distance, Angle, Counter, Ch, Filename, File |
String := "".
Symbol := "".
Distance := Angle := Counter := 0.
Permissible := "".
Prohibit := "".
Filename := Name.
ifstream File(Filename). // Create file object
[File = 1] // File open error
IfTrue:
    Out("Can not open for input").
[File <> EOF]

WhileTrue:

File.get(Ch). // File object read
[Ch = '/']

IfTrue: SWITCH (Ch)

CASE 'F':

Do

File.get(Ch).
Symbol := Ch.
File.get(Ch).
Distance := Ch.
F = NEW Group(Symbol, Distance).
Symbol := "".
Distance := "".
WhileTrue [Ch <> '/'].

CASE 'L':

Do

File.get(Ch);
Symbol := Ch.
File.get(Ch);
Distance := Ch.
L = NEW Group(Symbol, Distance).
Symbol := "".
Distance := "".
WhileTrue [Ch <> '/'].

CASE 'C':

Do
6.9

File.get(Ch);
Symbol := Ch.
File.get(Ch);
Distance := Ch.
File.get(Ch);
Angle := Ch.
Ch = NEW Group(Symbol, Distance, Angle).
Angle_context_counter := counter.
Symbol := "".
Distance := "".
Angle := "".
Counter := Counter + 1.
WhileTrue [Ch <> '/'].
CASE 'T':
  Do
    File.get(Ch).
    Permissible := Ch.
    WhileTrue [Ch <> '/'].
CASE 'V':
  Do
    File.get(Ch).
    Prohibit := Ch.
    WhileTrue [Ch <> '\n'].

P_Rule::Show(Nr)
  F->Show('F').
  L->Show('L').
[Counter := 0, Counter < Angle_context_counter, Counter++]

WhileTrue:

Ch[Counter]->Show(Counter).
Out(Permissible).
Out(Prohibit).

P_Rule::-P_Rule()
DELETE F.
DELETE L.
DELETE[] Ch.

6.2.4 Class SGG4

This is our main and most important class. This is also a composite class because all our other classes are contained in this class. An important factor is that this class is a PRIVATE class. The variables and public classes we use are thus protected. Quite a few member functions will be found in the private section of this class.

How we used the PRIVATE access method to protect all the PUBLIC objects are shown in figure 6.2. It is impossible for any other object to alter the data in these objects. The criteria for data security is thus adhered to.
### Class name

<table>
<thead>
<tr>
<th>Class name</th>
<th>SGG4</th>
</tr>
</thead>
</table>

### Super class

<table>
<thead>
<tr>
<th>Super class</th>
<th>None</th>
</tr>
</thead>
</table>

### State variables:

<table>
<thead>
<tr>
<th>Access Private:</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_Rule *Grammar</td>
</tr>
<tr>
<td>Terminal_Symbols</td>
</tr>
<tr>
<td>Non_Terminal_Symbols</td>
</tr>
<tr>
<td>Start_Symbol</td>
</tr>
<tr>
<td>Start_rule</td>
</tr>
<tr>
<td>Terminal_rules</td>
</tr>
<tr>
<td>The_rest // Production Rules</td>
</tr>
<tr>
<td>Rules</td>
</tr>
<tr>
<td>I, J // Index counters</td>
</tr>
</tbody>
</table>

### State Member Functions:

<table>
<thead>
<tr>
<th>Access Private:</th>
</tr>
</thead>
<tbody>
<tr>
<td>First_Rule</td>
</tr>
<tr>
<td>Terminal_Rules</td>
</tr>
<tr>
<td>The_Rest</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Public:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGG4 // Constructor</td>
</tr>
<tr>
<td>Init</td>
</tr>
<tr>
<td>Generate</td>
</tr>
<tr>
<td>Load</td>
</tr>
<tr>
<td>Show</td>
</tr>
<tr>
<td>~SGG4 // Destructor</td>
</tr>
</tbody>
</table>

---

**SGG4::First_Rule()**

```c
| Counter |
Counter := 0.
Do
    [Grammar[Counter]->F->Group = Start_Symbol AND
    Grammar[Counter]->Ch[0]->Group = "None" AND
    Grammar[Counter]->F->Group =
    Grammar[Counter]->Permissible]
IfTrue:
    RETURN (Counter). // Return First rule
```
```
WhileTrue [Counter++ < Rules].
RETURN (-1).  // Return error

SGG4::Terminal_Rules()

| Counter, Teller, I |

Teller := 0.
Counter := 0.
[I := 0, I < 10, I++]

Do
[Grammar[Teller]->L->Group = Terminal_Symbol]
IfTrue:
    Terminal_rules[Counter++] = Teller.
    // Place terminal rules in list
WhileTrue: [Teller++ < Rules].
RETURN (1).  // Return success

SGG4::The_Rest()

| Counter, Stop, I, J, Rules |

Counter = 0;
[I := 0, I < 10, I++]
WhileTrue:
[I < Rules]
IfTrue: The_Rest[I] := I.
IfFalse : The_Rest[I] := 99.
```
The_Rest[Start_Rule] := 99 // Take start rule out
Do
   The_Rest[Terminal_rules[Counter]] := 99.
WhileTrue: [Terminal_rules[Counter++]) <> 99].
// Place after each other
[I := 0, I < Rules, I++]
WhileTrue:
   [The_Rest[I] = 99]
IfTrue:
   [J := I, J < Rules, J++]
   WhileTrue: The_Rest[J] := The_Rest[J+1].
[I := 0, The_Rest[I] < 99, I++] // Count the rules
WhileTrue: Rules := I.
RETURN (Rules).

SGG4::SGG4()
   | String, Rules, II |
String := "".
Terminal_Symbol := "".
Non_Terminal_Symbols := "".
Start_Symbol := "".
Start_rule := 0.
Rules := 0.
[I := 0, II < 10, II++]
WhileTrue:
The_Rest[I] := 99.
I := 0. // Index counter are set to zero
J := 0.
// Production rules
[II := 0, I < RULES, II++]
WhileTrue: Grammar[II] := NEW P_Rule().

SGG4::Init()
    | String, Rules, II |
String := "".
Terminal_Symbol := "".
Non_Terminal_Symbols := "".
Start_Symbol := "".
Start_rule := 0.
Rules := 0.
[I := 0, II < 10, II++]
WhileTrue:
    The_Rest[II] := 99.
I := 0. // Index counter are set to zero
J := 0.
// Production rules
[II := 0, I < RULES, II++]
WhileTrue: Grammar[II] := NEW P_Rule().
SGG4::Generare() {
    | Buffer, g_d // Screen data is g_d
    x, y, R, deel, lank, g_d_teller, binne_teller,
    conflict_set, II, JJ, tel, count, hoek_tel, WAAR, fire
    antwoord_x, antwoord_y, antwoord_z, Radi, kleur |

    // Find first rule
    [Start_rule := Start_Rule() := -1] //Call Start Rule method
    IfTrue:
        Out("Error-1 No first rule").
        RETURN (0). // Exit member function

    // Find Terminal rule
    [Terminal_Rules() := 0] // Call Termination Rule method
    IfTrue:
        Out("Error-2 No terminal rules").
        RETURN (0). // Exit member function

    // The rest
    [The_Rest() < 1] // Call The Rest method
    IfTrue:
        Out("Error-3 No rules").
        RETURN (0). // Exit member functions

    // Use Start Rule, place data in screen structure
    g_d[Start_rule].naam := grammar[Start_rule]->F->Group.
    g_d[Start_rule].binne[0].lyn =
    Grammar[Start_rule]->L->Group.
    g_d[Start_rule].binne[0].len = Grammar[Start_rule] ->L-> Length.
    g_d[Start_Rule].binne[0].hoek = Grammar[Start_Rule]->
    Ch[0]->Angle.
x := GETMAXX() / 7.
y := GETMAXY() / 2.
g_d[Start_Rule].ox := x.
g_d[Start_Rule].oy := y.
R := 2.
lank := g_d[Start_Rule].binne[0].len/deel.
SETCOLOR(WHITE).
SETFILLSTYLE(SOLID_FILL, LightBlue).
SETCOLOR(LightBlue).
CIRCLE(x, y, r + 3).
FLOODFILL(x, y, LightBlue).
MOVETO(x, y - (r + 3)).
y := y - lank.
SETCOLOR(WHITE).
LINETO(x, y).
g_d[Start_Rule].binne[0].x := x.
g_d[Start_Rule].binne[0].y := y.
g_d_teller := Start_Rule.
binne_teller := 0.
[II := 0, I < 5, II++]
SETCOLOR(RED).
CIRCLE(x, y, r).
SETFILLSTYLE(SOLID_FILL, RED).
FLOODFILL(x + 1, y, RED).
// Generate the rest of the molecule
binne_teller := 1.
Do
tel := 0. // count the_rest
count := 0. // count conflict_set
hoek_tel := 0.
Do // Next rule to fire
[Grammar[the_rest[tel]]->F->group = g_d[g_d_teller].naam]
IfTrue: WAAR := 0.
[Grammar[the_rest[tel]]->Ch[hoek_tel++]->group = g_d
[g_d_teller-1].binne[2].lyn AND Grammar[the_rest[tel]]
->Ch[hoek_tel]->group = g_d[g_d_teller-1].binne[3].lyn]
IfTrue: WAAR := 1.
[Grammar[the_rest[tel]]->hoek_konteks_teller = 1]
IfTrue: WAAR := 1.
[WAAR = 0]
IfTrue: conflict_set[count++] := The_Rest[tel].
WhileTrue: [the_rest[tel++] <> 99].
fire := conflict_set[0].
[binne_teller = 1] //z axis
IfTrue:
radi := rad*Grammar[fire]->Ch[0]->Angle.
antwoord_x := (cos(radi)*((g_d[g_d_teller].binne
[binne_teller-1].x-g_d[g_d_teller].ox)*-deel)) +
(-sin(radi)*((g_d[g_d_teller].binne[binne_teller-1].y-
g_d[g_d_teller].oy)*-deel)).
antwoord_y := (sin(radi)*((g_d[g_d_teller].binne
\[ [\text{binne\_teller-1}].x - g_d[g_d\_teller].ox)*-deel) + (\cos(\text{radi})* (g_d[g_d\_teller].binne[\text{binne\_teller-1}].y - g_d[g_d\_teller].oy)*-deel)). \]
\[
\text{antwoord\_z} := 0.
\]

[\text{binne\_teller} = 2] // y axis

If True:

\[
\text{radi} := \text{rad}\*\text{Grammar[fire]}->\text{Ch[0]}->\text{Angle}.
\]
\[
\text{antwoord\_x} := (\cos(\text{radi})* (g_d[g_d\_teller].binne [\text{binne\_teller-1}].x - g_d[g_d\_teller].ox)*deel)).
\]
\[
\text{antwoord\_y} := (g_d[g_d\_teller].binne[\text{binne\_teller-1}].y - g_d[g_d\_teller].oy)*(-deel*0.7).
\]
\[
\text{antwoord\_z} := (\sin(\text{radi})* (g_d[g_d\_teller].binne [\text{binne\_teller-1}].x - g_d[g_d\_teller].ox)*deel)).
\]

[\text{binne\_teller} = 3] // x axis

If True:

\[
\text{radi} := \text{rad}\*\text{Grammar[fire]}->\text{Ch[0]}->\text{Angle}.
\]
\[
\text{antwoord\_x} := -(g_d[g_d\_teller].binne [\text{binne\_teller-2}].x - g_d[g_d\_teller].ox)*(deel*1.5).
\]
\[
\text{antwoord\_y} := (\cos(\text{radi})* (g_d[g_d\_teller].binne [\text{binne\_teller-2}].y - g_d[g_d\_teller].oy)*(-deel*2))).
\]
\[
\text{antwoord\_z} := (\sin(\text{radi})* (g_d[g_d\_teller].binne [\text{binne\_teller-2}].y - g_d[g_d\_teller].oy)*-deel)).
\]
\[
g_d[g_d\_teller].binne[\text{binne\_teller}].x := g_d[g_d\_teller].ox+(\text{antwoord\_x}/\text{deel}).
\]
\[
g_d[g_d\_teller].binne[\text{binne\_teller}].y := g_d[g_d\_teller].oy-(\text{antwoord\_y}/\text{deel}).
\]
If True:

```plaintext
SETCOLOR(WHITE).
[g_d[g_d_teller].binne[binne_teller].x >
g_d[g_d_teller].ox]
```

If True:

```plaintext
[g_d[g_d_teller].binne[binne_teller].y >
g_d[g_d_teller].oy]
```

If True:

```plaintext
SETCOLOR(RED).
SETFILLSTYLE (SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
```

If False:

```plaintext
SETCOLOR(RED).
SETFILLSTYLE (SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
```

```
```
LINE(g_d[g_d_teller].ox+r+3,g_d[g_d_teller].oy-r,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).

IfFalse:
[g_d[g_d_teller].binne[binne_teller].y >
g_d[g_d_teller].oy]

IfTrue:
LINE(g_d[g_d_teller].ox+r-2,g_d[g_d_teller].oy+r-2,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).

IfFalse:
LINE(g_d[g_d_teller].ox+r-2,g_d[g_d_teller].oy-r+2,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).

IfFalse:
SETCOLOR(WHITE).
LINE(g_d[g_d_teller].ox,g_d[g_d_teller].oy,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).

g_d[g_d_teller].binne[binne_teller].lyn :=
            Grammar[fire]->L->group.
g_d[g_d_teller].binne[binne_teller].len :=
            Grammar[fire]->L->Length.
g_d[g_d_teller].binne[binne_teller++].hoek :=
            Grammar[fire]->Ch[0]->Angle.

[binne_teller = 4 AND I <> 1]
IfTrue:
    binne_teller := 3.
tel := 0. // count the_rest
count := 0. // count conflict_set
hoek_tel := 0.
do // Next rule to fire
    [Grammar[the_rest[tel]]->F->group=g_d[g_d_teller].binne
     [binne_teller].lyn]
    IfTrue: WAAR := 0.
[Grammar[the_rest[tel]]->Ch[hoek_tel++]->group =
g_d[g_d_teller].binne[0].lyn AND
Grammar[the_rest[tel]]->Ch[hoek_tel]->group =
g_d[g_d_teller].binne[3].lyn]
IfTrue: WAAR := 1.
[WAAR = 0]
IfTrue: conflict_set[count++] := the_rest[tel].
WhileTrue [the_rest[tel++] <> 99].
fire := conflict_set[0].
g_d[g_d_teller].binne[binne_teller].lyn :=
   Grammar[fire]->l->group.
g_d[g_d_teller].binne[binne_teller].len :=
   Grammar[fire]->l->Length.
g_d[g_d_teller].binne[binne_teller].hoek :=
   Grammar[fire]->Ch[0]->Angle.
g_d[g_d_teller].binne[binne_teller].x :=
   g_d[g_d_teller].ox+(antwoord_x/deel).
g_d[g_d_teller].binne[binne_teller].y :=
   g_d[g_d_teller].oy-(antwoord_y/deel).
[antwoord_z < 0]
IfTrue:
   SETCOLOR(WHITE).
[g_d[g_d_teller].binne[binne_teller].x >
  g_d[g_d_teller].ox]
IfTrue:
   [g_d[g_d_teller].binne[binne_teller].y >
    g_d[g_d_teller].oy]
If True:

SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE(g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL(g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
SETCOLOR(WHITE).
LINE(g_d[g_d_teller].ox+r+2,g_d[g_d_teller].oy+r+2,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).

If False:

SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE(g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL(g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
SETCOLOR(WHITE).
LINE(g_d[g_d_teller].ox+r+3,g_d[g_d_teller].oy-r,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).

If False:

[g_d[g_d_teller].binne[binne_teller].y >
g_d[g_d_teller].oy]

If True:

LINE(g_d[g_d_teller].ox+r-2,g_d[g_d_teller].oy+r-2,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE (SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
If False:
LINE(g_d[g_d_teller].ox+r-2,g_d[g_d_teller].oy-r+2,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE(SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y-1,RED).
If False:
SETCOLOR(WHITE).
LINE(g_d[g_d_teller].ox,g_d[g_d_teller].oy,
g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y).
SETCOLOR(RED).
SETFILLSTYLE (SOLID_FILL, RED).
CIRCLE (g_d[g_d_teller].binne[binne_teller].x,
g_d[g_d_teller].binne[binne_teller].y,r).
FLOODFILL (g_d[g_d_teller].binne[binne_teller].x, \
g_d[g_d_teller].binne[binne_teller].y-1,RED).

    g_d_teller++.
    g_d[g_d_teller].naam := "."
    g_d[g_d_teller].naam := g_d[g_d_teller-1].naam.
    g_d[g_d_teller].ox := g_d[g_d_teller-1].binne[binne_teller].x.
    g_d[g_d_teller].oy := g_d[g_d_teller-1].binne[binne_teller].y.

    binne_teller := 0.
    g_d[g_d_teller].binne[binne_teller].lyn :=
        Grammar[fire]->L->group.
    g_d[g_d_teller].binne[binne_teller].len :=
        Grammar[fire]->L->Length.
    g_d[g_d_teller].binne[binne_teller].hoek :=
        Grammar[fire]->Ch[0]->Angle.
    g_d[g_d_teller].binne[binne_teller].x :=
        g_d[g_d_teller-1].ox.
    g_d[g_d_teller].binne[binne_teller].y :=
        g_d[g_d_teller-1].oy.

    I--.
    binne_teller := 1.

    [binne_teller = 4 AND I = 1]

    IfTrue:
        I--.
        [J <>0]
IfTrue:

tel := -1.


fire := --tel.

kleur := GREEN.

SETCOLOR(kleur).

SETFILLSTYLE(SOLID_FILL, kleur).

CIRCLE(g_d[J-1].binne[2].x, g_d[J-1].binne[2].y, r+1).


SETCOLOR(WHITE).

LINE(g_d[J-1].ox+r+2, g_d[J-1].oy-r-2,


While (I !:= 0).

RETURN (1).

SGG4::Load(name)

| ch, in, II |

filenaam := name.

in := "".

terminal_symbols := "".

non_terminal_symbols := "".

Start_Rule_symbol := "".

ifstream file(filenaam). // Create file object

file = 1]
IfTrue:
    Out("Can not open file for input").
    RETURN 1.
[File <> EOF]
WhileTrue:
    file.get(ch).
[ch = 'F']
    IfTrue: Rules++.
    SWITCH (ch)
    CASE 'N':
        Do
            File.get(ch).
            Non_terminal_symbols = string.
            WhileTrue: [ch <> '}'].
    CASE 'I':
        Do
            File.get(ch).
            I := ch.
            File.get(ch).
            J := ch.
            WhileTrue: [ch <> '}'].
    CASE 'T':
        Do
            File.get(ch).
            Termaaal_symbols := ch.
            WhileTrue: [ch <> '}'].
    CASE 'S':

Do
    File.get(ch).
    Start_symbol := ch.
    WhileTrue: [ch <> ')'].

// Production rules
[II := 0, II < Rules, II++]
    WhileTrue: Grammar[i] := NEW P_Rule(II,name).
RETURN 0.

SGG4::Show()
    | I |
Out("Structure Graph Grammar - 4").
Out(Start Rule symbol : "",Start_Rule_symbol).
Out("Non-terminal symbols : ",non_terminal_symbols).
Out("Terminal symbols : ",terminal_symbols).
[I := 0, I < Rules, I++]
    WhileTrue: Grammar[i]->Show(i).

SGG4::SGG4()
    DELETE[Grammar].

6.3 Conclusion

This chapter defined the classes used in our prototype system. Please note that the complete methods are not given. We only gave
the parts that show the logic we used in the development of the objects.

Chapter seven will explain the three-dimensional mathematics we used in the objects. The generation algorithm and conflict resolution algorithm are discussed in detail in chapter seven.
7.1 Introduction

It is important to remember that although we are representing structures in three dimensions, we are limited to two-dimensional viewing devices. This chapter is concerned with the mathematics we use to determine the relative positions the various atoms will occupy in three-dimensional space and the transformations we will have to do to represent a three-dimensional object on a two-dimensional screen.

Section 2 will discuss three-dimensional graphics and define certain key concepts we will use. In section 3 we will examine the transformations we will perform to display our three-dimensional structure in two dimensions.

7.2 Three-Dimensional Graphics

In two-dimensional graphics, there is general acceptance of the world coordinate system as seen in figure 7.1. As we go to three dimensions, the decision where to place the third dimension, z,
is not uniformly agreed on. If we start by accepting the two dimensional system, we have two options for the positive z direction. These options are known as the left-handed and right-handed coordinate system [ANG90].

Figure 7.1 Two-Dimensional Coordinate System.

Figure 7.2 Left- and Right-Handed Coordinate Systems.
We will use the right-handed system. Another possible source of confusion is the direction of a positive rotation. A positive rotation about an axis will be counterclockwise when we look toward the origin from the positive side of the axis as in figure 7.3.

![Diagram of a right-handed coordinate system with arrows indicating directions of positive rotation around the x, y, and z axes.]

7.2.1 Points

Points are zero-dimensional objects. A point has a location in space, but we can make no length measurement on it. The standard approach to represent a point is either as the set \((x, y, z)\) or as a three-element column matrix or vector [ANG90],
This is usually called a three-dimensional vector, since we are representing a point in a three-dimensional world. The terminology is not inconsistent with the fact that the point itself has no dimensionality.

\[ \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]

7.2.2 Vectors

In computer graphics, points are often associated with and confused with vectors [ANG90]. A vector \( \mathbf{v} \) is defined by the line segment between \((x_1, y_1)\) and \((x_2, y_2)\) as in figure 7.5.

Its magnitude \( \|\mathbf{v}\| \) and direction \( \theta \) are given by:
\[ \mathbf{v} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \] and
\[ \tan \beta = \frac{y_2 - y_1}{x_2 - x_1} \]

A vector is obtained from two points.

![Figure 7.5 Defining a vector]

7.2.3 **Homogeneous Coordinates**

Homogeneous coordinates [ANG90] provide a special four-dimensional representation that allows easy manipulation of three-dimensional entities. We replace the three-dimensional point

\[ \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]
with the four-dimensional point

\[ p = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}. \]

The only restriction on \( w \) is that it must be nonzero. With this restriction we can go back and forth between a point and its homogeneous coordinate representation by multiplying or dividing by \( w \) as necessary. In normal three-dimensional graphics, we will always be able to set \( w \) to 1, and here we will assume that we have made this choice.

If \( p \) is transformed to \( p' \) by an affine transformation, \( p' \) can be represented by

\[ p' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}. \]

### 7.3 Affine Transformations

The primitives we use most often in graphics systems are based on lines. The reasons are multiple and include that lines occur naturally in real-world applications, are easy to generate and are easy to transform.

Consider the problem of transforming a segment \( C \) between points \((x_1, y_1)\) and \((x_2, y_2)\) to a new segment \( C' \) by
a transformation we will denote by $T$. Using our vector representation, any point

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

on $C$ is transformed into a new point $C'$

$$p' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}.$$

We can express this relationship as

$$p' = T(p),$$

where the form of $T$ describes the exact nature of the transformation.

A class of transformations known as affine transformations [ANG90] include rotation, reflection, scaling and translation. We are only concerned with rotation.

### 7.3.1 Rotation

In three dimensions, there are independent rotations about all three axes. We define the operation of rotation as the rotation about the origin of our coordinate system.

Consider a rotation about the $z$ axis. When we rotate a point about the $z$ axis, its $z$ value is unchanged. This observation
allows us to reduce the problem to a two-dimensional $x, y$ rotation in a fixed $z$ plane.

Using polar form $[\text{ANG90}]$, we have

\[ x = r \cos \sigma, \]
\[ y = r \sin \sigma \] and

\[ x' = r \cos (\sigma + \tau), \]
\[ y' = r \sin (\sigma + \tau). \]

Using the trigonometric formulae for the cosine and sine of the sum of two angles, we find:

\[ x' = r \cos \sigma \cos \tau - r \sin \sigma \sin \tau \]
\[ = x \cos \sigma - y \sin \sigma \] and

\[ y' = r \cos \sigma \sin \tau + r \sin \sigma \cos \tau \]
\[ = x \sin \sigma + y \cos \sigma. \]

When we couple these equations with the equations:
z' = z and w' = w, we can express rotation about the z axis in matrix-vector form as:

\[ p' = R_z(\sigma)p, \]

where \( R_z(\sigma) \) is the matrix:

\[
R_z(\sigma) = \begin{bmatrix}
\cos \sigma & -\sin \sigma & 0 & 0 \\
\sin \sigma & \cos \sigma & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotation about the x axis leaves x values unchanged; likewise, rotation about the y axis leaves y values unchanged. Thus, the homogeneous coordinate matrix representations \( R_x \) and \( R_y \) can be derived as for the rotation about the z axis. The matrices are:

\[
R_x(\sigma) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \sigma & -\sin \sigma & 0 \\
0 & \sin \sigma & \cos \sigma & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_y(\sigma) = \begin{bmatrix}
\cos \sigma & 0 & \sin \sigma & 0 \\
0 & 1 & 0 & 0 \\
-\sin \sigma & 0 & \cos \sigma & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

7.3.2 **Perspective Projection**

Since our view surface is two-dimensional, there will be a projection of our three-dimensional primitive onto this surface.
We will make use of perspective viewing [ANG90]. We consider only the simple case, where the centre of projection forms a right pyramid with the projection plane. This case corresponds to most real-world viewing.

Figure 7.8 Perspective Projection

The viewing transformation places the projection plane at \( z = 0 \) and the centre of projection is at \((0, 0, d)\), where \( d > 0 \). A projector connects this point with a point \((x, y, z)\) on the object.

We derive the equations of projection by considering the top and side views as in figure 7.9.

Solving for the projection point \((x_p, y_p, z_p)\) we find:

\[
    x_p = \frac{x}{1-(z/d)}
\]

and
Figure 7.9 Top and Side views of Perspective Projection

$$y_p = \frac{y}{1-(z/d)}$$

The appearance of $z$ in the denominator shows why objects farther from the viewer are smaller in perspective projections and also shows the nonlinearity of perspective viewing.

### 7.4 Algorithms

The next few paragraphs will discuss how we have used the mathematics in our system and other aspects such as conflict resolution.
7.4.1 Algorithm 1

This algorithm calculates the relative positions where the various atoms will be positioned in three dimensional space. It is used in the generation method in the SGG4 class. (See chapter six, section 6.2.4.) We make use of the rotation transformation discussed in section 7.3.1.

The coordinates, \((x_1, y_1, z_1)\) will always be our origin. It does not matter if the real values are not all equal to zero, we will use it as if it is the origin.

After the first line in the molecule has been drawn, the second line is calculated by using the angle context of the current production rule. It will give us the angle or angles that the new line must be rotated. We have found that the following order is the most successful in generating the molecules.

The first transformation will be a rotation around the z axis. The \(p'\) coordinates will be used in the next rotation, which will be y axis. The last rotation that will be performed will be around the x axis. Each time the previous rotation coordinates will be used in the calculation of the new coordinates except when we perform the x axis rotation. Here we will use the z axis coordinates to calculate the new coordinates of the rotation.
As soon as a new production rule is fired, the new position will be used as the origin. The \((x_2, y_2, z_2)\) coordinates will be the new origin. That means that the old origin is the new \((x_2, y_2, z_2)\) position. The method we have used to calculate the new position is quite simple. Multiply the previous \((x_2, y_2, z_2)\) coordinates with -1. This will position the new \((x_2, y_2, z_2)\) in the correct quadrant.

The sequence of the rotation calculations stay the same until the termination rules of the grammar is fired and no more rotation will be necessary.

7.4.2 Algorithm 2

This algorithm explains how we determine which rule fires and how conflict resolution is handled. It is used in the generation method in the SGG4 class. (See chapter six, section 6.4.2.)

The first step after the file with the grammar has been read, is determining which rules are which. The start rule and termination rules are set aside. We call the rest of the rules the generation rules.

Structure graph grammar - extension 4 (SGG-4) is defined as the 4-tuple \(G = (N, T, S, R)\) with:
N a finite set of non-terminal symbols,
T a finite set of terminal symbols,
S the start symbol and
R the set of production rules.

A production rule \( r \) in \( R \) is a 3-tuple \( r = (F, L, C) \) with:

\( F \) a vertex or an edge in \( S \) that must be replaced,
\( L \) a vertex or an edge that must replace \( F \) and
\( C \) the context.

The context \( C \) is defined as a 3-tuple \( C = (Ch, U, T_1) \) with:

\( Ch \) the angle context,
\( U \) the global permitting context and
\( T_1 \) the global forbidding context with \( U, T_1 \subseteq N \cup U \).

We use the angle context, \( Ch \), the global permitting context, \( U \), and the global forbidding context, \( T_1 \), to decide which rule can fire. Each node that is generated and displayed on screen has a data structure associated with it. This data structure contains the context of the node.

After the start rule has fired and its context data structure has been updated we use the generation rules to determine
which rule will fire next. If the data structure contains the vertex or edge \( F \) that must be replaced with \( L \), and if the context requirements are satisfied, it is placed in the conflict set.

When there are more than one rule in the conflict set, we always fire the one that replaces a vertex with an edge. If there are more than one that satisfies this criteria, we use the Periodic Table of the Elements to decide which rule will fire. The rule with the element with the lowest atomic number will fire first.

When no more rules satisfy the criteria to fire, the termination rules are fired to complete the structure. We use a safety device that will stop the generation of the structure after a certain number of rules has been fired. This is necessary to stop generation when the production rules are written so that it is infinite.

7.5 Conclusion

This chapter has given a brief overview of some mathematics used in our system. It has also discussed two algorithms used in our system. Chapter Eight will speculate how the representation method of grammars can be coupled with some intelligence (objects).
8.1 Introduction

In this chapter we will explain why we used objects as our basis for the application of structure graph grammars as a three-dimensional representation scheme. We will also speculate on how it can be used to enhance a three-dimensional chemical molecule generating system. Research in robot vision, in particular vision interpretation is receiving a lot of attention. We will speculate on how "object" structure graph grammars can solve some of the problems that are experienced in that area.

8.2 Intelligent grammars?

In chapter six and seven we discussed how to implement a generating system that uses structure graph grammars as input and then produces three-dimensional structures generated by the structure graph grammar as output. The question remains, why did we use objects as our underlying principle in our system?
The answer to this question is quite simple. Because we used objects each production rule is an independent intelligent object. This means that the production rule is responsible for checking its own context and deciding if it will go ahead and generate its part of the structure. When there are many production rules, they decide themselves which rule is going to fire. This means that each object contains a conflict resolution mechanism that allows the correct rule to fire.

Each production rule is thus embedded with some intelligence and is part of the inference engine. It is thus not necessary to use an elaborate inference engine. The lack of an explanation facility can be overcome by using inheritance to enhance the capabilities of the objects to include such an explanation facility. It is also possible to extend the objects to include any other facility that will be necessary in any other application domain.

This leads us to the next section which will demonstrate how this characteristic of the "object" grammar comes in handy.

8.3 Application in Organic Chemistry

Although structure graph grammars provide an elegant method of representation, they do have some drawbacks. The most important one is that it takes quite a large quantity of production rules to generate a real complex and large three-dimensional
structure. This problem occurs when you have to write down the production rules to generate the molecule in figure 8.1.

This molecule with more than 40 C-atoms, more than 50 H-atoms, and various other atoms imbedded in its body would take more than 120 rules to generate. Not only is it quite a number of rules, but the contexts of the rules can become quite complex.

![Figure 8.1: A complex organic molecule.](image)

When all the rules are embedded in objects, each production rule has the knowledge to generate the part of the structure it represents. This makes it possible to group a few rules
together that will generate a particular organic molecule, for example an alkane or an ester. Each grouping is given the name of the functional group it is able to generate.

This means that you now have a limited number of groupings, named after the functional group they generate, rather than a loose collection of production rules. An expert system can now be built on the knowledge contained in the groupings.

The knowledge of how to generate a certain simple organic molecule is hardwired into the knowledge base. This knowledge is thus explicitly given in the form of a set of production rules.

The user of the expert system can then decide himself on how complex a molecule has to be. This is made possible by the wide range of simple functional groupings he can select and then connect to each other to generate a complex organic molecule. Another advantage of such a system is that the user does not need a lot of computer knowledge because the system is a type of visual programming system.

The complex organic molecule in figure 8.1 can now be generated by choosing 9 groupings and connecting them to each other.

Possible groupings of chemical molecules that can be generated with structure graph grammar extension 4 are:
R-Fu, where:

R is an alkanes, alkenes or alkynes,
Fu is an alkyl-halide -F, -Cl, -Br or -I,
    is an alcohol, -OH
    is an aldehyde, -C(O)-OH,
    is an amide, -C(O)-NH₂,
    is a nitrile, -C≡N,
    is an amine, -NH₂,
    is an ether, -O-,
    is a ketone, -C(O)-,
    is an ester, -C(O)-O- and
    is an anhydride, -C(O)-O-C(O)-.

The group of rules which can be used to generate the functional
groups is given in Appendix C. Each of these grammars will have
the following format:

\[ G = (N, T, S, R) \] where:

\[ N = \{ s_x (1 \leq x \leq i), s_y (1 \leq y \leq j), s_h, s_{fu} \}, \]
\[ T = \{ c_x (1 \leq x \leq i), c_y (1 \leq y \leq j), H, Fu \}, \]
\[ S = \{ s_i \} \] and
\[ R = \{ \text{group of productions as given in Table 8.1} \}. \]

The index i, (Table 8.1) can be one of the following:

The position of a functional group or
the length of the left carbon chain (R-Fu-R).
### Table 8.1 Functional groupings

<table>
<thead>
<tr>
<th>Functional group</th>
<th>Index</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alkanes</td>
<td>i</td>
<td>1, ..., 5, 7, 8.</td>
</tr>
<tr>
<td>Alkenes</td>
<td>i,j</td>
<td>1, ..., 5, 7, 8, 20, ..., 25.</td>
</tr>
<tr>
<td>Alkynes</td>
<td>i,j</td>
<td>1, ..., 5, 7, 8, 26, ..., 31.</td>
</tr>
<tr>
<td>R-F</td>
<td>i,j</td>
<td>1, ..., 9.</td>
</tr>
<tr>
<td>R-F-R</td>
<td>i,j</td>
<td>1, ..., 5, 10, ..., 20.</td>
</tr>
</tbody>
</table>

The index \( j \) can be one of the following:

* The length of the right carbon chain (R-Fu-R) or
* the position of the beginning of the double or triple bond.

Because the contexts are not the same for all the functional groups we use the variables \( B \) for the bond length and the variable \( \sigma \) for the angles.

The grammar that generates the above mentioned groupings can be found in Appendix C.

## 8.4 Vision Systems

Image interpretation and recognition are areas that are receiving a lot of research attention. The representation of image features is an important aspect of vision systems. By choosing a good representation method, a lot of potential problems can be avoided. [RA088]
A problem often found in vision systems, is that of structure recognition. It is not difficult to describe a table to a computer. The problem arises when the table image the computer must recognise, is from a different angle than the one that was used to describe the table in the first place, see figure 8.2 a and b.

A human being will have no difficulty in recognising the second image as a table. The computer will not recognise it. When a situation like this arises, the system must match the description of the table with that of the image. One way to solve this problem is to store each object from a few different angles. This method is not recommendable because it takes up too much space in the knowledge base. The overhead to match the image with all those descriptions will also make it impractical.

A better solution will be one where the table is represented with structure graph grammars. The whole table, as a three-dimensional picture can be rotated as needed without much overhead to achieve a match with that of the image. This approach will also help to keep the knowledge base smaller.

The only problem is that you are matching a three-dimensional picture to a wireframe representation (the structure graph grammar representation). The system will find matches with a greater degree of confidence if there is a low level process that performs a process of segmentation, so that the picture can also be represented as a structure graph grammar.
The use of inheritance can enhance such a system so that it is not only possible to recognize a table from any conceivable angle, but also to perceive the table's type.

This is made possible by adding certain attributes to each object. The super class object will have attributes that all tables have in common, no matter what type it is. This will include attributes such as the number of table legs and the general shape a table can take.

The sub-class will have additional attributes to identify the table uniquely as a certain type. Attributes that include the
length of the table legs and their shape can be sub-class attributes. In other words, a tea table has short legs that can be curved. Figure 8.3 displays a possible hierarchy that such a representation can take.

**TABLE**

<table>
<thead>
<tr>
<th>4 Legs Table Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Legs</td>
</tr>
<tr>
<td>Small Top</td>
</tr>
<tr>
<td>Square or round</td>
</tr>
<tr>
<td>Medium Legs</td>
</tr>
<tr>
<td>Strong, Thick Legs</td>
</tr>
<tr>
<td>Large Top</td>
</tr>
</tbody>
</table>

**TEA TABLE**

**WORK TABLE**

Figure 8.3 Hierarchy of tables.

### 8.5 Summary

In this chapter we speculated on a few applications and uses of structure graph grammars. We examined how it could be used by a chemist to generate complex three-dimensional structures that could prove to be helpful to his research. We also looked at how structure graph grammars could help simplify and solve some problems that are experienced in the field of computer vision.
CHAPTER NINE

Molecule-by-Grammar

9.1 Introduction

A prototype system, Molecule-By-Grammar, was developed during this study to demonstrate the objects and algorithms used to represent three-dimensional structures. This chapter is intended as a brief overview of the system. An example will also be discussed. Hardware requirements, limitations and possible enhancements to the system are discussed.

9.2 Hardware

The system can be used on any IBM compatible system as no special hardware is required. A system with a colour monitor is recommended. The reason is that colour is used to help the user with his depth perception.

The system requires DOS 3.31 or higher, a hard disk or floppy drive and at least 512K bytes of memory and a mouse.

The system was developed on a 80386 PC with a VGA high
9.2

resolution controller card and monitor. The system was not tested on any other monitor system.

9.3 **Limitations on the System**

Because there is a memory limit of 640K bytes that the DOS operating system places on the system, we had to place some limits on the system.

A maximum of 16 rules are allowed.

The next limitation is set because of monitor limitations. We recommend that the indices I and K are never set to more than 7. This is necessary to prevent the structure from being clipped because it is too big for the screen.

9.4 **Discussion of the System**

The system was developed so that the user interface is as simple as possible. The screen is divided into three main viewports. (See figure 9.1)

Viewport A is our main display area. In this area we will display the structures generated by the system using the input grammar.
In viewport B we will display the status of the system and various other messages that need to be displayed. It is also used as our input viewport. We will refer to this viewport as our status viewport.

The menu is displayed in viewport C. The mouse is used to choose an option. Place the mouse cursor on the option you want to select and press the left button.

The menu options are as follows:

- Load is the option that reads the input file.
- Display writes the grammar on the screen.
9.4

Generate does exactly that which its name implies, it generates the structure and displays it on the screen.
Fire lets you select a particular rule you want to fire.
Trace is a facility that shows you in which order the rules are fired and why.
Quit exits the system and returns the user to DOS.

9.4.1 Load

When you select load in the menu viewport, an action message appears in the status viewport. You are given the choice of loading a file or using an editor to enter it yourself. When choosing to load a file it asks the name of the input file. Use the keyboard and enter the file name and press enter. If all is well, it will read the input file into memory. However, it will display an error message if it is not able to open the input file. The editor will allow you to enter a new grammar as input.

9.4.2 Display

When choosing the display option, the contents of the input file, that is the grammar will be displayed on screen. The contents are most probably too much to display at one time. The action message in the status viewport will tell you to either press any key to see the next screen, or the number of a particular rule you wish to see.
9.4.3 **Generate**

This option will only be successful when you have loaded an input file. While it is generating the structure, it will update the trace data, so that it is available when asked for. The structure that it will generate, will be displayed in the main viewport. A code will also appear to explain the icons used in the structure.

9.4.4 **Fire**

When you select this option, the start rule of the present grammar loaded will fire. After the start rule has fired, you may choose any rule to fire. If it can fire it will do so, otherwise it will tell you why it cannot be fired. When firing any terminal rule, all the termination rules will fire and the structure will be completed. A trace will be generated for this structure as well.

9.4.5 **Trace**

The trace facility displays the order in which the rules were fired. This display facility operates on the same principle of the Display option. The only enhancement is that a trace can be saved to a file.
9.4.6 **Quit**

This option closes the system down and frees all the memory that were used by the objects.

9.5 **Example**

This is a small example used to demonstrate the system.

Start the system by entering SGG4 at the DOS prompt. A screen such as in figure 9.2 will appear.

![Main Screen Diagram]

Press Left Button to select option
Select the Display option by placing the mouse cursor in its menu box and pressing the left button. The computer will make a beep sound and an error message will be displayed in the status viewport. The error message prompts the user to load a file for input.

The same response will occur when the Display, Generate, Fire and Trace options are chosen.

When selecting the load option, enter the file name PROPANE.SGG at the prompt. This example file will generate the propane molecule.

Select the generate option. The structure that is displayed, will look like this.
The Fire option will clear the screen and fire rule 0, the start rule. Fire rule 1 twice. The rule will fire both times and the structure looks as follows.

When selecting rule 5 to fire, it gives an error message. The context of the rule is not so that it can be fired. After firing rule 4 the structure looks like this.
After the termination rules have fired you are returned to the main menu.

The Display and Trace facilities are easy to use and will not be explained. To use other facilities the system offers, press the right button while in the menu viewport.

A facility to ask the angle between two lines is one of the other small tasks the system can perform.

9.6 Technical Discussion

The format of the input file is discussed in this section. When creating this file you are free to name it anything you want.

At the top of the file we put the indices. The format for this is \( \{I \ i,j,k\} \) where \( i,j \) and \( k \) is greater than or equal to zero. The PROPAE.SGG file will look like this:

\[
\{I 3,0,0 \}
\]

The non-terminal symbols, termination symbols and the start symbol follow the indices. Each of them start with brackets and a \( N \), \( T \) and \( S \) follows respectively. Separate the symbols with a comma and close the brackets when all symbols are entered. The example file looks as follows:
The production rules follow this. There may only be one production rule on a line. We follow the definition of a production rule closely. Each line will start with a backward slash. The definition states that a production rule $p$ looks like this, $p = (F, L, Ch, T, V)$ where:

- $F$ is the node or vertex to be replaced.
- $L$ is the node or vertex that replaces $F$.
- $Ch$ is the angle context.
- $T$ is the permitting context.
- $V$ is the forbidding context.

Both $F$ and $L$ use the name of the node or vertex, after which the length is entered if it is a vertex, otherwise it is zero. For example, $/L Sc,0$

The angle context starts with the name of the vertex, its length and the angle.

The permitting context and forbidding context name the symbols that may or may not be present.

The complete example file looks as follows:

PROPANE.SGG

$I 3,0$
9.7 Conclusion

The prototype system was briefly discussed in this chapter. An example run of the prototype was given as well as some technical information on the files that the system uses.

The next chapter will be a brief conclusion to this dissertation.
CHAPTER TEN

Conclusion

10.1 Introduction

This chapter gives an overall summary of all the chapters in such a way as to connect all of them in a meaningful manner. We will also identify new areas of research.

10.2 Summary and Future Topics of Research

Future and current topics of research are extensions of the molecular mechanics, methods to obtain better heats of formation, prediction of spectroscopic parameters and investigation of solvent effects [VIN85]

All of the techniques of molecular graphics can be viewed as a variation on a theme i.e. that of extending the stick picture to try and represent movement. Many groups around the world are currently trying to represent molecular movements in new ways, but little progress is being made away from the set notions of the stick model [TOW89].
Three dimensional data sets have traditionally been very hard to represent visually, and thus difficult to interpret and explore. Even when stereoscopic hardware is available, volumetric data sets such as the results of molecular orbital calculations and crystallographic electron density maps are normally simplified to discrete isometric contours. This process discards much of the information and generates phantom surfaces and discontinuities that may mislead the observer [VIN89].

The development of algorithms and techniques for the search and retrieval of information from three dimensional databases is now a very important research area. The problems involved are perhaps an order of magnitude more complex and challenging than those involved in two dimensional databases [ALL89].

Individual structures from the Protein Data Bank are extensively used in modelling applications. There is much software development aimed at searching and structural systematics. These methods are now being extended into the areas of structure prediction and automated model building, by using knowledge based approaches based on structural homology [ALL89].

One of the most important research areas is integration of the different computer systems the chemist uses. It is therefore important that a standard data exchange format and a standard user interface are accepted by the majority of manufacturers [TOW89, TRI92, CHE92, BEI92].
Much research is needed to solve the problems of integrated retrieval and integrated representation [BAW89].

The techniques of computational chemistry, molecular graphics, information retrieval systems and expert systems have become firmly established as an indispensable addition to the range of physical and biological tools available to the research chemist.

A lot of research is being done in the field of knowledge representation of three-dimensional structures. The traditional knowledge paradigms as well as new methods of representation have had a lot of attention.

Graph-theoretic approaches to object location are an important new and future research area [DAV91]. This includes association graphs for matching of objects [YAN89]. Research in three-dimensional mathematical morphology and associated special-purpose computer architectures to image processing, target tracking and both three-dimensional data analysis and display is being done [KEN87].

Binary forest segmentation [NIC91] and spatio-temporal reasoning [ORR90] and geometric reasoning [BUX88] are areas of future research.

It is our view that structure graph grammar is an elegant way to represent structures in three dimensions. Future research can include the union of structure graph grammars extension 4 and
extension 6 for a more powerful way to represent structures. This will open the way for future research into the segmentation of structures that are seen with a video camera to match them with structures in the knowledge base.

The object-oriented programming paradigm is well suited for representing general knowledge in the explicit as well as the implicit way. A few of the most important features of the object-oriented programming paradigm were discussed.

We speculated on various applications and uses of structure graph grammars. We examined how it could be used by a chemist to generate complex three-dimensional structures that could prove to be helpful to his research. We also looked at how structure graph grammars could help simplify and solve some problems that are experienced in the field of computer vision. The correspondence problem that is experienced in stereo vision is a future research area. It could be that with the use of structure graph grammars this problem could be easily overcome.


[BAW89] Bawden, D; "Linking structures and data", Chemistry in Britain, November 1989.

[BEI92] Beilstein Institute; Varrentrappstr. 40-42, D-6000 Frankfurt/Main 90, F.R. Germany.


[CHE92] Chemical Abstracts Service; 2540 Olentangy River Road, P.O.Box 3012, Columbus, OH 43210, USA.


[MOR90] Morgan, A.D, and Dagless, E.L. and Milford, D.A. and Thomas, B.T.; "Road edge tracking for robot road following: a real
time implementation"; Image and vision computing, vol. 8, no. 3 (1990).


[TRI92] Tripos Associates, Inc; 1699 S. Hanley Road, Suite 303, St. Louis, MO 63144, USA.


## APPENDIX A

### Glossary of Chemical Terminology

<table>
<thead>
<tr>
<th>Word</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>amino acid</td>
<td>One of a class of organic acids characterized by substitution of an amino group in the alkyl residue.</td>
</tr>
<tr>
<td>analytical chemistry</td>
<td>That branch of chemistry dealing with the determination of qualitative or quantitative composition of substances and materials.</td>
</tr>
<tr>
<td>conformations</td>
<td>Denotes the geometry of a covalent molecule in terms of the arrangements of its atoms in which more than one arrangement is possible by simple rotations about the single bonds.</td>
</tr>
<tr>
<td>crystallographic input</td>
<td>The crystal intercepts, which are the distances from the origin of the crystallographic axes to the</td>
</tr>
</tbody>
</table>
points where they are cut by the faces bounding the unit cell of the crystal.

de novo protein engineering The engineering of a class of complex compounds of high molecular weight and their immediate derivatives, found associated with living matter.

electron density maps The ratio of the number of valence electrons to the number of atoms in a molecule.

enzymes Soluble, colloidal, organic catalysts which are produced by living organisms and are either simple or conjugated proteins.

homologous series A series of carbon compounds the same type which conform to general formula and in which each member differs from the preceding member by a constant increment of the atomic group CH₂.

inorganic chemistry That branch of chemistry which is concerned with the application of
interconverting chemical facts and principles in the production in industry of chemical materials.

In an electromagnetic transition between two states of an atomic nucleus differing by energy E.

internally rotating Rotating of an atom or group of atoms confined within a molecule.

toxicology That branch of chemistry dealing with the study of tissues, products, processes and other chemical aspects of the body and its parts during disease.

molecular dynamics Those variables in terms of which classical mechanics is built up, and which can be given an operational definition.

molecular mechanics See molecular dynamics.

NMR Nuclear magnetic resonance spectroscopy, used to determine molecular weight and molecular formula,
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-stoichiometry</td>
<td>A compound which has a composition not in accordance with the law of definite proportions.</td>
</tr>
<tr>
<td>organic chemistry</td>
<td>That branch of chemistry which deals exclusively with the compounds of carbon.</td>
</tr>
<tr>
<td>physical chemistry</td>
<td>The subject covers physical properties of chemical substances and relations between energy and chemical change.</td>
</tr>
<tr>
<td>spectroscopic analysis</td>
<td>Analysis of the composition of substances through the use of the spectroscope.</td>
</tr>
<tr>
<td>stereochemistry</td>
<td>That branch of chemistry which is concerned with the structure of molecules.</td>
</tr>
<tr>
<td>surface reorganisation</td>
<td>The occupation of such positions by</td>
</tr>
</tbody>
</table>
certain molecules in the surface of a liquid that one part of the molecule is turned toward the liquid.

**synthesis design**

The combination of elements or compounds to produce compounds of one kind.

**tautomerising**

A phenomenon whereby a substance exists in two isomeric formats, which are in equilibrium and exhibit characteristic reactions.

**van der Waals volumes**

Interatomic and intermolecular forces of attraction, other than valence forces.

**valence limits**

The property of an atom or radical to combine with other atoms or radicals in definite proportions, or a number representing the proportion in which a given atom or radical combines.
BNF Notation

We use this pseudo object-orientated language to define the objects in our system. See chapter six. The BNF notation follow below.

1: <nothing> is a empty string.
2: <space> → ' ' 
3: <figure> → 0|1|...|9
4: <letter> → a|b|...|z, A|B|...|Z
5: <character> → [\[]|{()}|{|}|<[^,|;|:|\]|$|\-|\#|\^|<number>|<letter>|1
6: <unsigned number> → <unsigned integer> | <unsigned real>
7: <unsigned integer> → <figure> {<figure>}
8: <unsigned real> → <unsigned integer>. <figure> {<figure>}
9: <scale factor> → <unsigned integer> | <sign> <unsigned integer>
10: <sign> → + | -
11: <identifier> → <letter> {<letter or figure>}
12: <letter or figure> → <letter> | <figure>
13: <string> → '{<character>}'
14: <comment> → //{character}
15: <variable> → <identifier>
16: <complex statement> → [<statement> {;<statement>}]
17: <statement> → <simple statement>. | <structured statement>.
18: <simple statement> → <assign statement> | <message>
19: <assign statement> → <variable> := <expression>
B.2

20: <expression> → <simple expression> | <simple expression>  
   <relational operator> <simple expression>

21: <relational operator> → = | <> | < | <= | >= | >

22: <simple expression> → <term> | <sign> <term> | <simple expression> <addition operator> <term>

23: <addition operator> → + | - | or

24: <term> → <factor> | <term> <multiplication operator> <factor>

25: <multiplication operator> → * | / | div | mod | and

26: <factor> → <variable> | <unsigned constant> | (<expression>) | <message assignment> | not <factor>

27: <unsigned constant> → <unsigned number> | <constant identifier>

28: <constant identifier> → <identifier>

29: <message assignment> → <message>

30: <message> → <identifier> | <identifier>: <expression>

31: <structured statement> → <complex statement> | <condition statement> | <repeat statement>


33: <repeat statement> → <while statement>

34: <while statement> → [<expression>] whileTrue: <statement>
35: `<while statement> → Do <statement> whileTrue: [<expression>]

36: `<for statement> → [<assign statement>, <expression>,<assign statement>] whileTrue: <statement>

37: `<case statement> → Switch <variable> {case <string> |<variable> | <figure> : <statement>

38: `<member definition> → `<member head> `<local variables> `<block`

39: `<member head` → `<identifier> {<message>}

40: `<local variables` → | `{<variables>}

41: `<block` → `<procedure block` | `<function block`

42: `<procedure block` → `<statement` | `<complex statement`

43: `<function block` → `<procedure block` ^ `<expression`

44: `<class definition` → `<class name definition` `<super class name definition` `<class member variable definition` `<class member definition`

45: `<class name definition` → `class name `<identifier`

46: `<super class name` → `super class `<identifier`

47: `<class member variable definition` → `class member variables `<identifier` | `<nothing`

48: `<class member definitions` → `<class members` `<block`

{(block}) | `<nothing`
APPENDIX C

Functional Group Production Rules

The production rules are:

1. \((s_1, (s_1 - s\_h, 0.11nm), (()) / (s_1, (s\_h)))\);
2. \((s_1, (s_1 - s\_h, 10.11nm), ((((s_1 - s\_h, 0.11nm), 109.5^\circ)) / (s_1, s\_h), ({}))\);
3. \(((s_1 - s\_h, 0.11nm), (s_1 - s\_h, 0.153nm), ((((s_1 - s\_h, 0.110nm), 109.5^\circ)) / (s_1, s\_h), (s_1))\);
4. \(((s\_x - s\_h, 0.110nm), (s\_x - s\_h, 0.153nm), ((((s\_x - s\_h, 0.153nm), 109.5^\circ), ((s\_x - s\_h, 0.110nm), 109.5^\circ)) / (s\_x, s\_h), (s\_x))\) \(2 \leq X \leq i-1;\)
5. \((s\_x, (s\_x - s\_h, 0.110nm), ((((s\_x - s\_h, 0.11nm), 109.5^\circ), (((s\_x - s\_x, 0.153nm), 109.5^\circ)) / (s\_x), (s\_x))\) \(2 \leq X \leq i;\)
6. \(((s\_y - s\_h, 0.110nm), (s\_s - s\_fu, Bnm), ((((s\_y - s\_h, Bnm), \sigma^*), ((s\_y - s\_y, Bnm), \sigma^*), ((s\_y - s\_h, 0.110nm), 109.5^\circ)) / (s\_x, s\_y, s\_h), (C, H, Fu))) 2 \leq X \leq i;\)
7. \((s\_x, C, ((\}) / (s\_x), (H, Fu))) 1 \leq X \leq i;\)
8. \((s\_h, H, ((\}) / (C, s\_h), (s\_x, Fu)) 1 \leq X \leq i;\)
9. \((s\_fu, Fu, ((\}) / (C, s\_h), (s\_h, s\_x)) 1 \leq X \leq i;\)
10. \(((s\_x - s\_h, 0.110nm), (s\_s - s\_fu, Bnm), ((((s\_x - s\_h, Bnm), \sigma^*), ((s\_x - s\_x, Bnm), \sigma^*)) / (s\_x, s\_h), (C, H, s\_x))\) 1 \leq X \leq i;\)
(11) \((s_{fu}, (s_{fu} - s_j, \beta_{nm}), \{(s_{fu} - s_j, \beta_{nm}), \sigma^*\}) / (s_i, s_{fu}, \{s_j\}) \) \(j = 1\);
(12) \((s_{j}, (s_j - s_{h}, 0.11\text{nm}), \{(s_{fu} - s_j, \beta_{nm}), \sigma^*\} / (s_{j}, s_{fu}, \{s_{j+1}\}) \) \(j = 1\);
(13) \((s_{j}, (s_j - s_{h}, 0.11\text{nm}), \{(s_{fu} - s_j, \beta_{nm}), \sigma^*\}, \{(s_j - s_{h}, 0.11\text{nm}), 109.5^\circ\}) / (s_j, s_{fu}, s_{h}, \{s_{j+1}\}) \);
(14) \((\{(s_j - s_{h}, 0.11\text{nm}), (s_j - s_{j+1}, 0.153\text{nm}), \{(s_j - s_{h}, 0.11\text{nm}), 109.5^\circ\}, \{(s_j - s_{f}, \beta_{nm}), \sigma^*\} / (s_j, s_{fu}, s_{h}, \{s_{j+1}\}) \)) \(j = 1\);
(15) \((\{(s_y - s_{h}, 0.11\text{nm}), (s_y - s_{y+1}, 0.153\text{nm}), \{(s_y - s_{h}, 0.11\text{nm}), 109.5^\circ\}, \{(s_y - s_{y}, 0.153\text{nm}), 109.5^\circ\}) / (s_y, s_{fu}, s_{h}, \{s_{y+1}\}) \)) \(2 \leq y \leq j-1\);
(16) \((s_y, (s_y - s_{h}, 0.11\text{nm}), \{(s_y - s_{y}, 0.11\text{nm}), 109.5^\circ\}, \{(s_{y-1} - s_{y}, 0.153\text{nm}), 109.5^\circ\}) / (s_y, s_{fu}, s_{h}, \{s_{y+1}\}) \)) \(2 \leq y \leq j\);
(17) \((s_y, c_y, \{(\} / \{s_y\}, \{h, fu, s_x\}) \) \(1 \leq x \leq i, 1 \leq y \leq i\);
(18) \((s_h, h, \{(\} / (c_x, c_y, s_h), (s_x, s_y, s_{fu})) \) \(1 \leq x \leq i, 1 \leq y \leq i\);
(19) \((s_{fu}, fu, \{(\} / (c_x, c_y, h, s_{fu}), (s_x, s_y, s_{fu}) \)) \(1 \leq x \leq i, 1 \leq y \leq i\);
(20) \((\{(s_y - s_{h}, 0.11\text{nm}), s_y, \{(\} / \{s_y\}, \{s_j=s_{j+1}\}) \)) \(y = j\);
(21) \((\{(s_j - s_{h}, 0.11\text{nm}), (s_j - s_{h}, 0.109\text{nm}), \{(s_{j-1} - s_j, 0.153\text{nm}), 117.5^\circ\}, \{(s_j - s_{j+1}, 0.153\text{nm}), 117.5^\circ\}) / (s_j, s_{j+1}, \{s_j=s_{j+1}\}) \));
(22) \((\{(s_{j+1} - s_{h}, 0.11\text{nm}), (s_{j+1} - s_{h}, 0.109\text{nm}), \{(s_j - s_{j+2}, 0.153\text{nm}), 117.5^\circ\}, \{(s_j - s_{j+1}, 0.153\text{nm}), 117.5^\circ\}) / (s_j, s_{j+1}, \{s_j=s_{j+1}\}) \));
Because the contexts are not the same for all the functional groups we use the variables \( \delta \) for the bond length and the variable \( \sigma \) for the angles.