AN ONTOLOGY BASED SEARCHABLE REPOSITORY OF GLYCAN STRUCTURES

by

DELARAM RAHBARINIA

(Under the Direction of KRZYSZTOF J. KOCHUT)

ABSTRACT

In the field of Chemistry and more specifically Biochemistry, carbohydrates play a key role in numerous scientific studies. The complex structure of carbohydrates makes it a challenging task to study them. Thus, categorizing, storing and managing the information about these structures in a consistent fashion that is useful and acceptable by the scientific community have become a matter to tackle. In this research, the goal of this research has been to design and create an ontology based data store and search system to provide the scientists with an efficient, consistent, flexible and easy to use system to study and examine the molecular structure of carbohydrates.

INDEX WORDS: Ontology, RDF, OWL, SPARQL, Glycan, Carbohydrate, PDB

AN ONTOLOGY BASED SEARCHABLE REPOSITORY OF GLYCAN STRUCTURES

by

DELARAM RAHBARINIA

Assoc. Prog., University of Industries and Mines, Iran, 2007

BSc, Sadra Institute of Higher Education, Iran, 2010

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2016

AN ONTOLOGY BASED SEARCHABLE REPOSITORY OF GLYCAN STRUCTURES

by

DELARAM RAHBARINIA

Major Professor:	Krzysztof J. Kochut
Committee:	Robert J. Woods
	Lakshmish Ramaswamy

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
August 2016

DEDICATION

This thesis is dedicated to my loving parents and dearest brother for providing me a lifetime of endless support and unconditional love.

ACKNOWLEDGEMENTS

I would like to thank my major advisor, Dr. Krzysztof Kochut for his positive thoughts, valuable input and all the guidance throughout my research work. I am grateful for my committee member, dear Dr. Robert Woods, for his patience, inspiration and support all the time. Also, I would like to thank my other committee member, dear Dr. Lakshmish Ramaswamy, for always encouraging and believing in me, and everything I learned from him during master's studies.

My sincere appreciation to Dr Lachele Foley for trusting in me and for everything she has taught me.

Last but not least, many thanks to my lab mates and friends who helped me throughout this journey.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

In the field of Chemistry and more specifically Biochemistry, carbohydrates play a key role in numerous scientific studies. The complex structure of carbohydrates makes it a challenging task to study them. Thus, categorizing, storing and managing the information about these structures in a consistent fashion that is useful and acceptable by the scientific community have become a matter to tackle. In this research, the goal of this research has been to design and create an ontology based data store and search system to provide the scientists with an efficient, consistent, flexible and easy to use system to study and examine the molecular structure of carbohydrates.

CHAPTER 2

BACKGROUND

2.1. Biological Background

In this research, the primary data files that have been processed and used are in the category of PDB (Protein Data Bank) formatted files [1]. A PDB file mainly contains the list of atoms in a molecule and their 3D coordinates in space. Other chemical information about the molecular structures can also be found in PDB files, such as the specific residue and chain that an atom belongs to or insertion code and alternate location of an atom in a molecule. But the main factor is the geometric coordinate of the atoms as shown in Figure 1.

```
ATOM   9542  CZ  ARG B 585      61.706  22.363  35.749  1.00 42.21           C
ATOM   9543  NH1 ARG B 585      62.393  23.421  36.230  1.00 43.46           N
ATOM   9544  NH2 ARG B 585      60.439  22.539  35.321  1.00 40.03           N
ATOM   9545  OXT ARG B 585      60.180  16.715  39.497  1.00 27.77           O
TER    9546      ARG B 585                                                    
HETATM 9547  C11 BCD A 601      37.[illegible]  1.00 49.61           C
HETATM 9548  C21 BCD A 601      37.790  58.453  51.518  1.00 48.70           C
HETATM 9549  O21 BCD A 601      37.531  59.846  51.581  1.00 50.53           O
HETATM 9550  C31 BCD A 601      36.593  57.694  52.087  1.00 48.23           C
HETATM 9551  O31 BCD A 601      35.450  57.947  51.275  1.00 53.18           O
HETATM 9552  C41 BCD A 601      36.852  56.179  52.159  1.00 46.59           C
```

X/Y/Z COORDINATES

Atom Name

Figure 1: A Small Portion of a PDB File Content

The PDB archive of RCSB [2] is growing and more PDB files are being added to it continuously. These PDB files have been produced in laboratories by molecular modelers and biochemists based on their research and projects.

Sometimes, the molecular structures in the PDB files represent carbohydrate-protein structures rather than only protein. Such structures might contain sugar(s) in them. A monosaccharide makes the simplest form of a sugar in carbohydrates and an oligosaccharide is the combination of monosaccharides. The most basic component of the structure of a monosaccharide is called a ring which consists of atoms that are attached to each other and create a cycle. The rings usually have one oxygen atom and four or five carbon atoms. The rings with six atom members are referred to as pyranoses and five membered ones as furanoses. Rings should have certain non-ring atoms (side atoms) attached to them to be considered as a sugar. Monosaccharides might be linked to one another through a side oxygen or in some cases a nitrogen atom. "In chemistry, a glycosidic bond or glycosidic linkage is a type of covalent bond that joins a carbohydrate (sugar) molecule to another group, which may or may not be another carbohydrate" [3]. Another terminology that is used in the field of biochemistry is "Glycan". Glycan refers to multiple monosaccharides that are attached to each other [4].

2.2. Semantic Web and Ontologies

Ontologies have been used in the field of philosophy to deal with the concept of existence of entities and their hierarchies. Ontologies are considered as a sub field of information science. There are multiple definitions for describing the concept of ontology. In 1993, Tom Gruber defined ontology as "explicit specification of a conceptualization" [5]. Ontologies provide representation of knowledge coming from one or more domains. Ontology is a collection of concepts that are organized into hierarchies and it also represents the interrelationships among them. The represented knowledge can provide a common vocabulary for a group of people working on specific domains.

Therefore, ontologies are very powerful in terms of re-using the knowledge and changing the common assumption into explicit terms. Concepts and their instances are known as resources in ontologies and they can be uniquely identified using Uniform Resource Identifier (URIs) [6].

Tim Berners-Lee, defines the term “Semantic Web” as a “web of data that can be processed by machines” [7]. The aim of Semantic Web is to create a web of data from the current web based on the data formats and data exchange protocols described by World Wide Web Consortium (W3C) [8]. Semantic Web as the new extension to the current web creates an understandable web of data for both humans and machines. Semantic web heavily leverages ontologies to model and formalize the resources of web and to build vocabularies from different domains knowledge.

2.3. RDF and OWL Ontologies

The content of ontologies can be encoded with specific frameworks and languages such as Resource Description Framework (RDF) [9] and Web Ontology Language (OWL) [10]. “RDF is a general method for conceptual description or modeling of information that is implemented in web resources” [11]. RDF is a model that adds meta-data to the data to make statements about resources through a set of syntax notations. The building block of RDF is referred to as a triple which is in the form of subject-predicate-object or equivalently, resource-property-value. An example of triple can be shown as follows in which the resource “GCC” is related to different objects by a property called “Compiles”:

<GCC> <Compiles> <C>

<GCC> <Compiles> <C++>

The RDF graph representation of the described triples is shown in Figure 2.

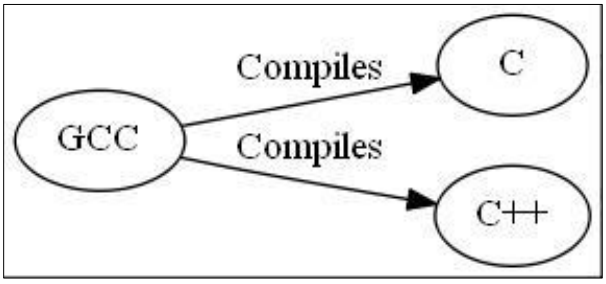

Figure 2: RDF Graph

RDF provides meta-data annotation, however, it does not provide any meaning to the data. By using the RDF Schema (RDFS) [12], classes, sub-classes, properties and sub-properties can be defined to add basic semantic to an RDF graph. “RDFS is the most basic schema language commonly used in the Semantic Web technology stack” [13]. OWL, on the other hand, provides a more expressive way to define the schema of the ontology and brings semantics upon the data. OWL gives broader set of vocabularies to model the data such as equivalences, set operations and restrictions. With OWL it is possible to express more complex properties and relationships between classes. Three flavors of OWL exist: OWL lite uses only a subset of all the capabilities of the language, OWL DL is more expressive than OWL lite. It uses all the capabilities, however, with some restrictions. OWL Full uses all capabilities without any restrictions but there is no guarantee on validity of the statements.

2.4. SPARQL

SPARQL [14] stands for SPARQL Protocol and RDF Query Language. SPARQL is one of W3C’s recommendations and is used for matching query triple patterns to the triples patterns of ontologies. The matched triples are returned as the result set. SPARQL

allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. Unlike querying relational databases, SPARQL queries can ask for the relationship between concepts as well.

2.5. Ontology Storage and Search System

"Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system" [15]. Open Link Virtuoso is the open source edition of Virtuoso Universal Server which is used in Semantic Web applications. Virtuoso triple store provides an endpoint which can be used for querying the ontologies via SPARQL. Virtuoso has been used by large datasets such as DBpedia [16] which is based on the information extracted from Wikipedia and Wikidata. There are also other powerful RDF triple stores such as RDF4J (Sesame) [17] and TDB data set [18]. Sesame is a Java framework that offers creating, storing and querying of RDF data. TDB is a storage and query system that works with Jena APIs [19]. For the purpose of this research, Virtuoso has been chosen as the triple store because of its high performance, ease of use and also due to the constraint of not using Java-based systems for this particular project.

CHAPTER 3

MOTIVATION

Scientists are conducting various research on the carbohydrates. For example, they study carbohydrates that are involved in diseases, such as cancer, Alzheimer's, viral infections, and metabolism diseases. Carbohydrates are not derived from a template, and can form very diverse and highly complex structures versus DNA and Protein structures that can easily be sequenced.

Glycoprotein structures in PDB are heavily used by the domain's scientists in different studies and experiments. However the main issue is that as the nomenclature is inconsistent, there is no easy way for scientists to check and verify their new structures with the existing structures in the database, or find the similar structures to a specific molecular structure. For example, a specialist finds a particular antibody binds to a particular carbohydrate, manually searching PDB and finding the similar structures is cumbersome and might take weeks or even months of work. The aforementioned issues and conditions had increased the demand for a system that can automate the process and provide an easy to use, scalable platform for the community.

3.1. Problem Definition

Inconsistency in nomenclatures and lack of a general system to be used as a common source for scientists has led us to design a solution for storing and managing the information within PDB carbohydrate structures. There might be monosaccharides with

the same nomenclature that refer to completely different structures or vice versa. Also, there might be mislabeled structures in the PDB files that are so hard to be identified and fixed considering the enormous number of files in the database.

The demanded system should be able to automatically parse and process all the structures in PDB files, identify the sugars only based on their 3D structure to avoid any dependency to ambiguous and sometimes misleading data in files, import the structures into a data storage system to create a knowledge management platform, and finally, the system should provide a search system than can query the generated data in different levels of granularities, from atomistic level to residues, monosaccharide, oligosaccharides and the entire PDB file itself.

CHAPTER 4

SYSTEM DESCRIPTION

4.1. Introduction

As a knowledge management system, an ontology has been designed to represent the information extracted from PDB files about sugars and their derivatives and modification patterns. The ontology has been stored in an RDF triple storage and is searchable at different levels.

4.1.1. Glycam Molecular Modeling Ontology (GMMO)

The schema plays a critical role in answering the questions of scientists, thus, it should be designed carefully to be comprehensive enough to contain all the necessary details of the data and to be able to represent the interrelation between the concepts in a meaningful way. At the same time, the schema should be scalable and flexible to possible extensions in the future as the knowledge evolves. Glycam Molecular Modeling Ontology (GMMO) has been designed specifically to model, manage and store the data of the glycam domain. Furthermore, once this ontology is populated, provides inference and analysis capabilities by answering queries. The complete design details of GMMO will be discussed in section 4.2.

4.1.2. Data Extraction

In order to provide the data for the ontology to be populated, PDB files should be processed and relevant information about glycan and moiety structures should be extracted from them. Glycam Molecular Modeling Library (GMML) which is an open

source C++ library, has been used to parse the input files and simulate the 3D structure of molecules. Moreover, Sugar ID process of GMML operates on the simulated structures and identifies the monosaccharides and other modification and derivative structures. Furthermore, in order to develop the structure of oligosaccharides, the attachments between identified monosaccharides and their hierarchies is determined by the process, as well.

4.1.3. Ontology Population

In this project, Java based APIs or libraries have not been considered since most of the other programs and libraries used by the users have been implemented in C/C++ programming language. In order to keep the dependencies of the system at the minimum level, a customized API have been implemented for populating the ontology. The process of ontology population from preparing the extracted knowledge to converting each statement into triple format has been implemented completely in GMML. GMMO is populated from the extracted information from over 114,000 PDB files processed by the Sugar ID process.

4.1.4. Search System

The populated ontology is stored in Virtuoso triple store and it can be queried via SPARQL. Various query generators have been implemented in GMML that can answer questions for scientists. These generators are basically functions that can automatically and transparently generate the suitable SPARQL queries based on the specific input arguments. The system serializes and sends the generated queries to the Virtuoso data store and returns the result set to the user. Users can choose the format of result set (e.g. JSON [20], CSV [21] or XML [22]).

4.2. Glycam Molecular Modeling Ontology (GMMO)

GMMO has been designed to model complex molecular structures at different levels of granularity. It contains different types of classes, from atoms and their connections to oligosaccharides. GMMO schema has been designed using Protégé [23] - ontology editor. The produced schema content can be found in Appendix A. The overall design of GMMO which is represented in UML [33] is shown in Figure 3.

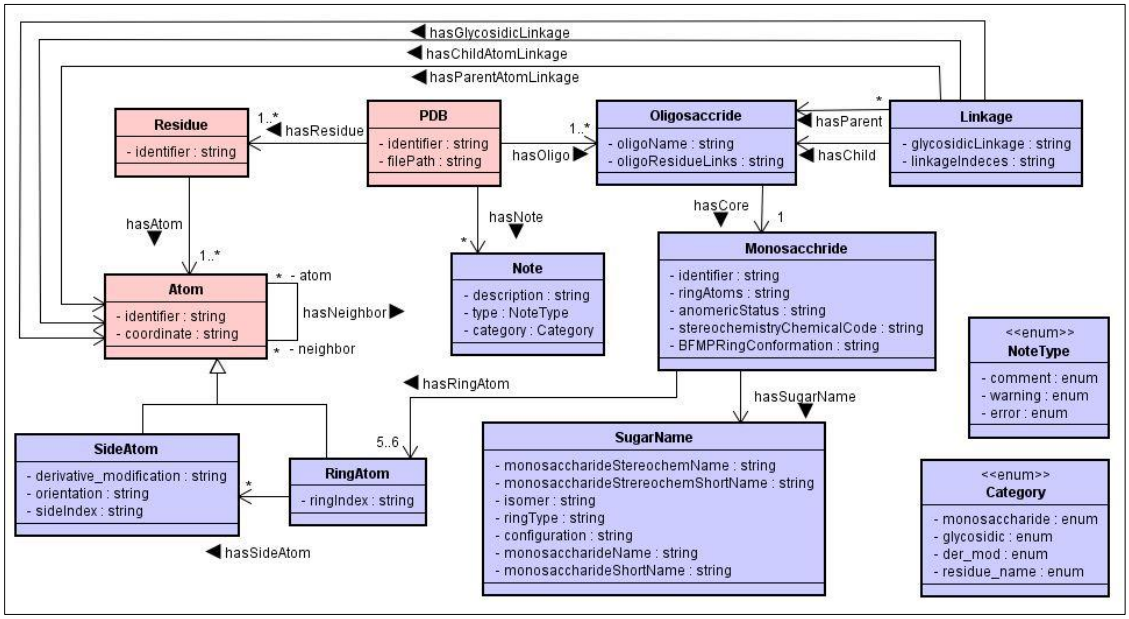

Figure 3: GMMO Schema in UML [33]

In Figure 3, the classes with the pink color (PDB, Residue and Atom) represent the information gained from parsing and initial simulation of the input file structures in GMML and the blue colored classes are designed based on the information gained from data extraction (Sugar ID) process. The PDB class represents the input PDB file structure that has been processed by the system. It is described by the input file's path and a unique identifier (PDB ID) as data properties. It is connected to classes Oligosaccharide, Residue

and Note with *hasOligosaccharide*, *hasResidue* and *hasNote* accordingly. Each PDB can have one or more residues and each residue can have one or more atoms.

The identifier for the Residue class is created from the combination of attributes of a residue. A residue identifier has the following format:

PDBID_ResidueName_ChainID_ResidueSequenceNumber_InsertionCode_AlternateLocation_AssemblyID

e.g. 1EQU_NAP_B_328_n_n_1

The Note class has been designed to store certain notes and/or issues that can be found within the structure of input PDB file. The category of a note identifies that to which concept the issue is related to (e.g. residue naming, monosaccharide). Note type represents the level of the note. The level can be error, warning or a comment.

The Atom class models the simulated graph structure of the molecule in which nodes are the atoms and edges are the bonds between atoms. Each atom individual has a relationship with its geometric coordinate and a unique identifier as data properties. It can be linked to other atom individuals via the *hasNeighbor* object property. Atom identifier is created from combination of attributes of an atom and has the following format:

PDBID_AtomName_SerialNumber_ResidueName_ChainID_ResidueSequenceNumber_InsertionCode_AlternateLocation_AssemblyID

e.g. 1EQU_C1B_4436_NAP_B_328_n_n_1

These attributes are assigned based on the data in the input PDB file. By parsing the input file, a unique identifier is assigned to each atom in the GMML system. GMML will be discussed in more details in section 4.3.

The Oligosaccharide class represents all the sugar structures that have been identified by Sugar ID process. The combination of Oligosaccharide and Linkage classes models the

tree like structure of glycans. Figure 4 demonstrates sample individuals of the aforementioned classes, and the glycan representation of the sample structure.

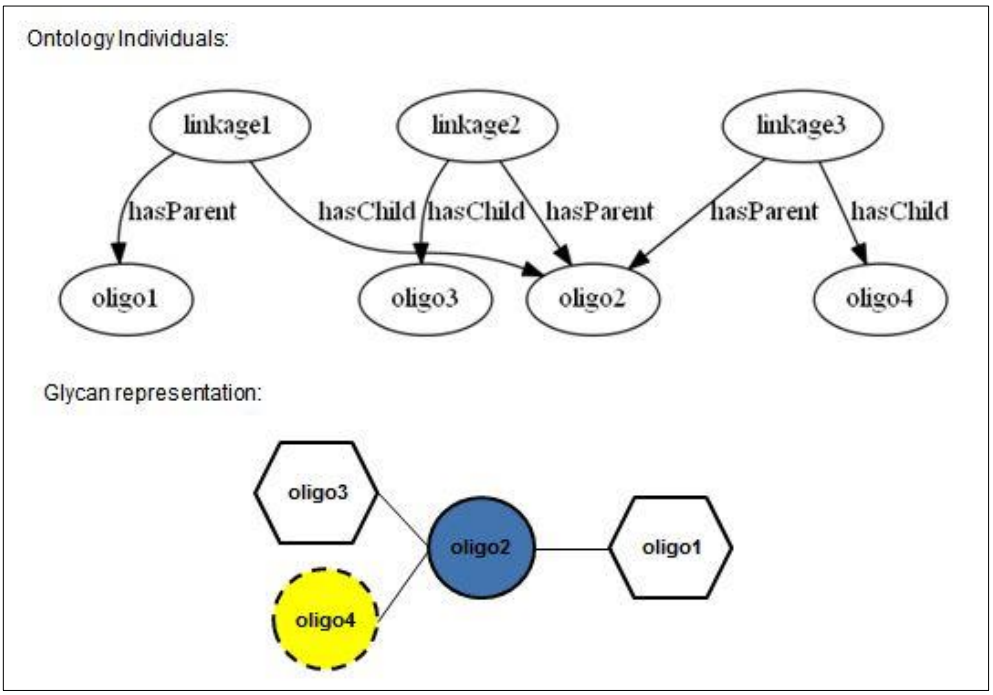

Figure 4: Oligosaccharide Linkages

Each oligosaccharide in this system is connected to its core which is a monosaccharide by *hasCore* object property. For each glycan which is a tree like structure of connected oligosaccharides, only one of the involved oligosaccharides has the data properties *oligoName* and *oligoResidueLinks*. Such oligosaccharide is the root of the glycan. In another words, the purpose of the Oligosaccharide class along with the Linkage class and their object properties is to model the hierarchies and relationships of the glycan structures. The purpose for the Monosaccharide class is to depict the internal information about each oligosaccharide.

The Linkage class also models the atoms that are involved in the linkages between oligosaccharides. Such information about the atoms is represented both as a literal value using *glycosidicLinkage* data property, and as an Atom class individual. *hasChildAtomLinkage*, *hasParentAtomLinkage* and *hasGlycosidicLinkage* object properties are predicates that are used for this matter. A sample representation of individuals involved in a linkage is shown in Figure 5.

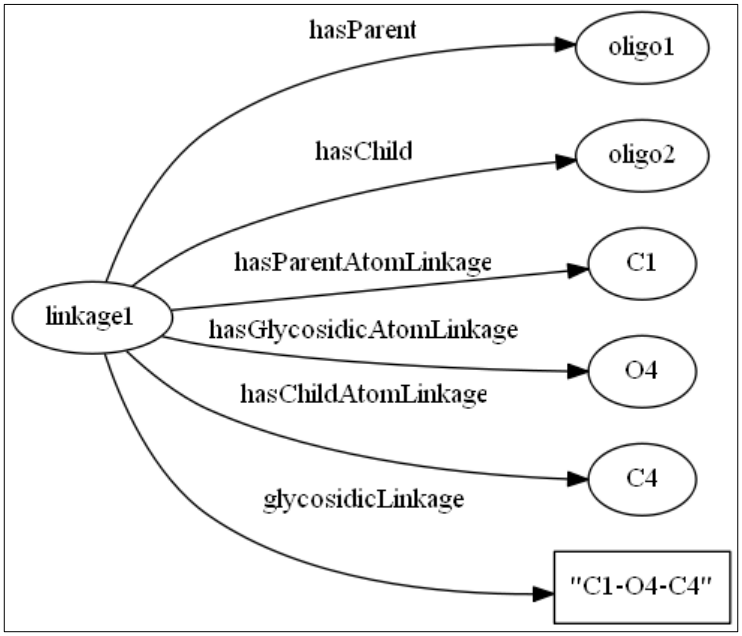

Figure 5: Glycosidic Linkage Atoms

Monosaccharide instance triples are described by literal values such as ring atoms string version, information about the identification of the anomeric carbon and the string version of the chemical code [31] assigned to their structure. These data properties are *ringAtoms*, *anomericStatus* and *stereochemistryChemicalCode*, respectively. *BFMPRingConformation* represents information about the ring shape of the sugar (e.g.

boat, chair). Monosaccharide is also described by object property *hasSugarName* which links it to the SugarName class.

The SugarName class has relationships with literals that represent the information about the different name formats for a sugar (Condensed/Long, Stereochemistry/Complete) and also various part of the name. For example, *ringType* data property represents whether the monosaccharide is a pyranose or a furanose, *isomer* indicates that if it is a “D” or “L” sugar, *configuration* can have values “Alpha” or “Beta”.

RingAtom and SideAtom classes represent the atoms that are involved in the molecular structure of a monosaccharide, the atoms of the ring and the exocyclic atoms that are attached to the ring atoms. *ringIndex* property indicates the index of the ring atoms that have been processed starting from anomeric carbon to the oxygen of the ring. Anomeric carbon gets the lowest index and the oxygen the highest. Each RingAtom individual might be attached to another exocyclic SideAtom individual. SideAtom class has an *orientation* data property which models the geometric orientation of the side atom with respect to the ring. It can be above or below the plane that ring is in. SideAtom individuals can have modifications or derivatives attached to them. This information is represented by *derivative_modification* data property. *sideIndex* data property determines that each SideAtom individuals is attached to which one of the RingAtom individuals. A sample demonstration of relation between Monosaccharide, RingAtom and Side atom classes is shown in Figure 6. One of the SideAtom individuals also has a *derivative_modification* data property:

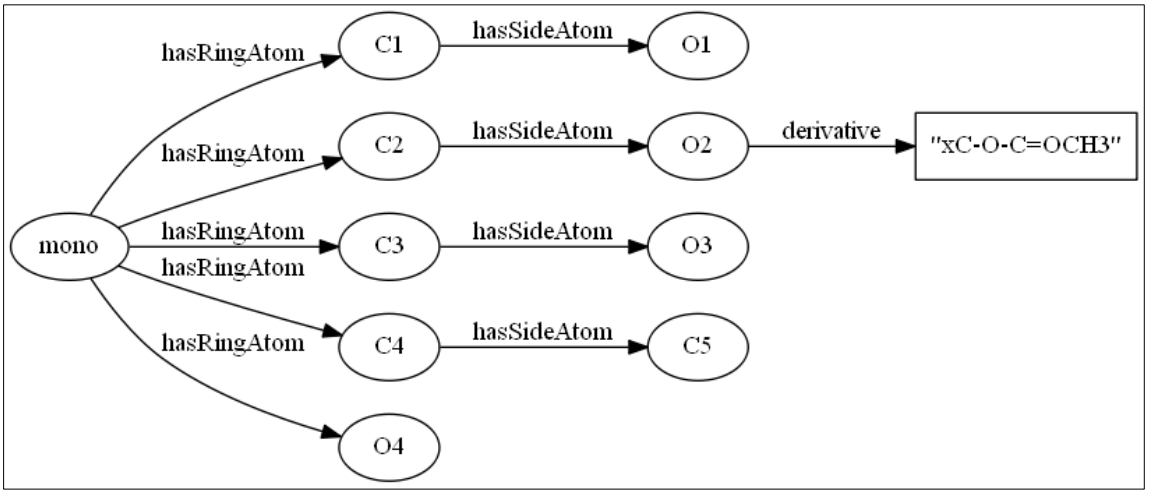

Figure 6: An Example of Monosaccharide, RingAtom, SideAtom Instances and Derivative Property

4.3. Data Extraction

4.3.1. Glycam Molecular Modeling Ontology (GMMO)

Prior to implementation of Sugar ID process certain architectural prerequisites are needed to create a system for reading, parsing, importing, exporting and writing of molecular structures. GMML has been implemented as a library that reads and parses various formats of input files, creates a 3D simulated graph of the structure and imports it in the Central Data Structure (CDS). Figure 7 shows an overall overview of functionalities of GMML.

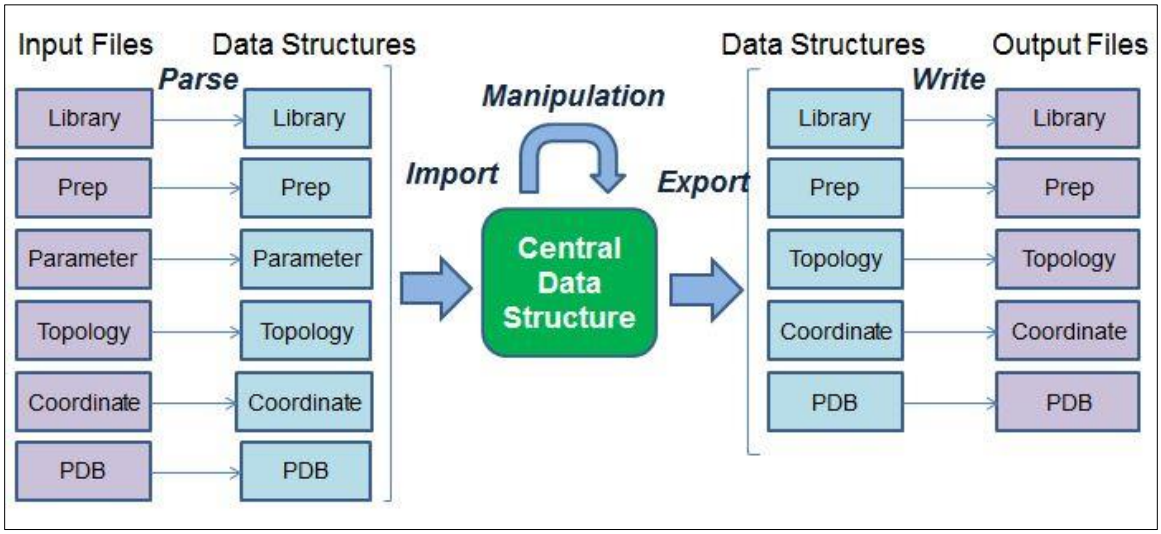

Figure 7: GMML

Structures in CDS have a recursive nature. Initially each molecule is encoded as a concept called an Assembly. An assembly is a class which contains chemical attributes of the molecule and also the graph structure of the molecule. Each assembly has a list of Residues and each residue has a list of atoms. Each atom has an adjacency list to keep record of atoms it is linked to. One assembly can be divided to multiple different assemblies to represent different molecules and also different assemblies can be combined together and become one assembly. Figure 8 demonstrates this concept.

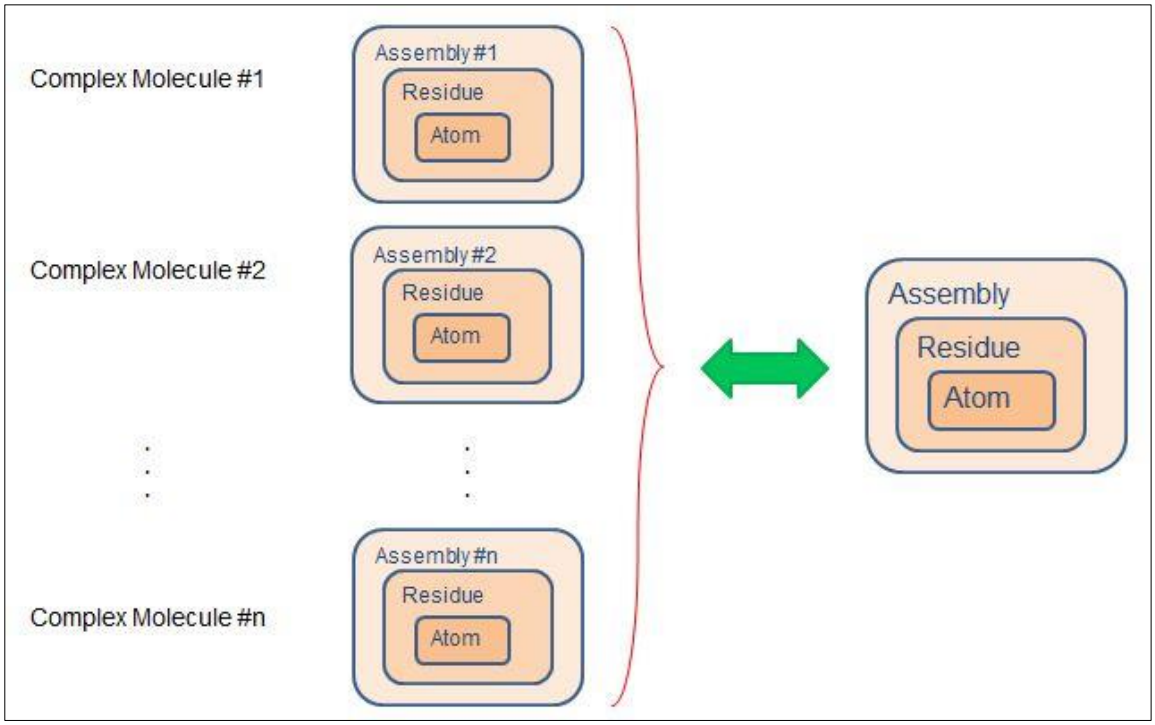

Figure 8: Concept of Assemblies

Moreover, the simulated structures can be exported and written in different file formats. Many other functionalities and features have been implemented in GMML which are not discussed or used in this research but can be used by scientists to perform a variety of additional operations and analysis on the structures.

4.3.2. Glycam Extensible Modeling System (GEMS)

Glycam Extensible Modeling System (GEMS) provides a scripting language layer that has been implemented on top of GMML. The purpose of this layer is to facilitate and simplify the communication with GMML. All the functionalities of the GMML are accessible through web interface APIs, external applications APIs and command line interface using GEMS middleware. This feature makes the integration of GMML with other systems feasible.

"SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages" [24]. Examples of such high level languages are Perl, Python, Ruby and Tcl. SWIG shares some similarities with the Interface Definition Language (IDL) [25] and compilers such as CORBA [26] and Microsoft's COM [27] technology, but the main purpose of SWIG is to provide rapid development and prototyping tool. In this research, SWIG has been used to generate Python wrapper codes based on the declarations of the GMML header files.

The overall architecture of the system is shown in Figure 9.

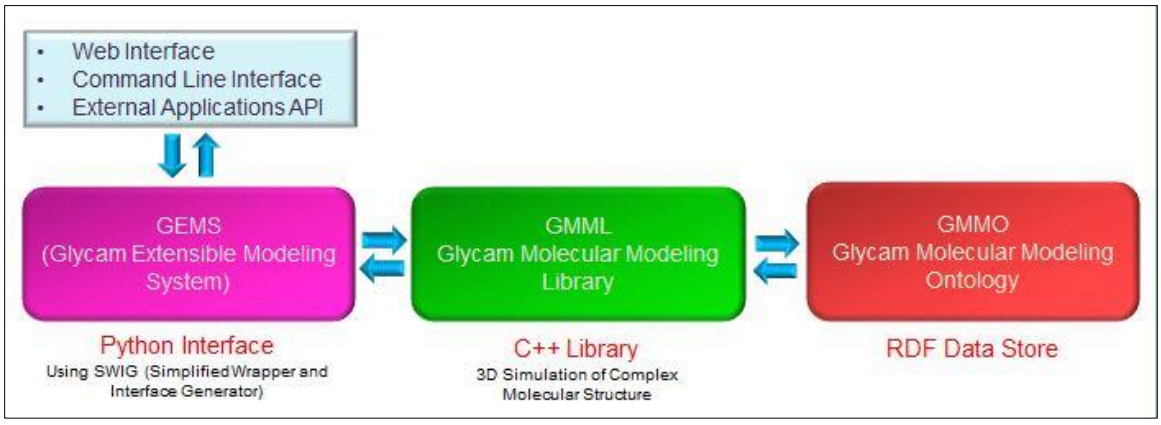

Figure 9: Overall System Architecture

4.3.3. Building Molecular Graph Structure

The PDB files are plain text files and in order to simulate the 3D structure of the molecules the bonding between atoms should be identified. There are several approaches to detect the existence of atom links. The bonds can be identified from the information within the PDB files from the sections called CONNECT cards. However, not all of the existing PDB files contain this section. Even if all the files had this section, based on the opinion of the domain experts, there might be some inconsistencies between the

information within the file and what the 3D structure is supposed to be. Therefore, the most trusted and accurate approach is to build the structure of the molecular graph based on the distance of the atoms from each other. The 3D coordinates of the atoms can be extracted from PDB files and the distance between each and every pair can be calculated. If the distance falls under a certain cutoff threshold, then the atoms are considered as bonded.

Some PDB files might contain a large number of atoms in them (e.g. 25,000+), therefore, a brute force method for calculating the distance for each pair of atoms (taking one atom and compare its distance with the rest of the atoms) is not an optimal approach. Thus, in this system, multithreading methods have been used. First thread based approach is to divide the atoms into equal chunks, assign each chunk to a thread, in each thread compare the atoms of that chunk to the complete list of atoms. Then create or update the adjacency lists for compared atoms, and at the end merge the results from all threads.

Figure 10 shows an example in which ‘n’ is the number of atoms and there are three threads to calculate the distance for equal sized chunks of atoms. The first thread initially calculates the distance of A1 to A2, A1to A3 and so on to A2 to An, then moves to A2 and perform the calculations with A3, A4 to An. The other threads perform the same process on their chunks. Finally, the distances for all pairs have been calculated and the results of the threads are merged.

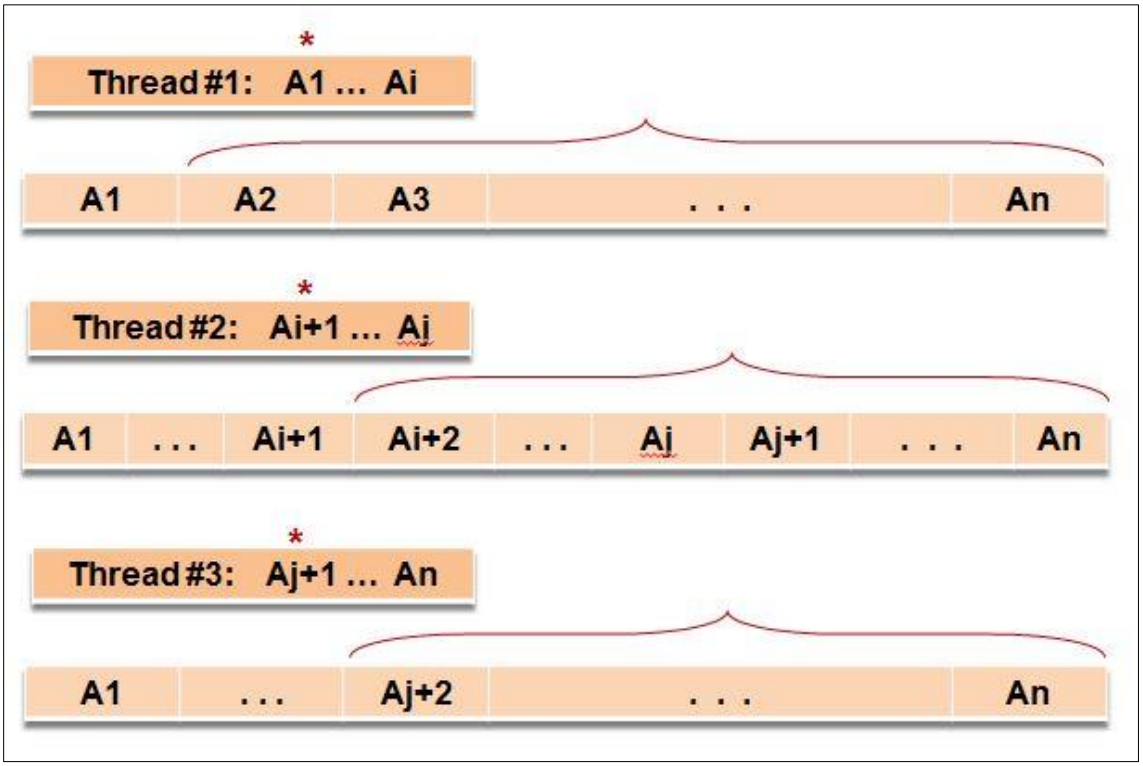

Figure 10: Thread-based Atom Distance Calculation, Method 1: equal chunks, different workloads

In the first approach the workloads of threads are not equal since in the first thread, each atom of the chunk should be compared to almost the entire list of atoms and in the last thread, atoms of the chunk are compared only with the small list of remaining atoms proportional to the size of the chunk itself. This might cause some imbalance and make the merging process slow, since all threads should complete their tasks before this step. Thus, second approach has been designed to create chunks of atoms with different sizes such that the first chunk has a very small size since the distance of its atoms should be calculated with almost the entire list of atoms. The size of atom chunks for the next threads gradually gets increased such that the last chunk has the largest size since its

atoms are compared to a small portion of the entire list of atoms. The second approach provides more similar workloads for threads, thus, the merging process will be faster. In both first and second approach the user can specify the number of threads to be used. Figure 11 shows the visual representation of an example of the second approach.

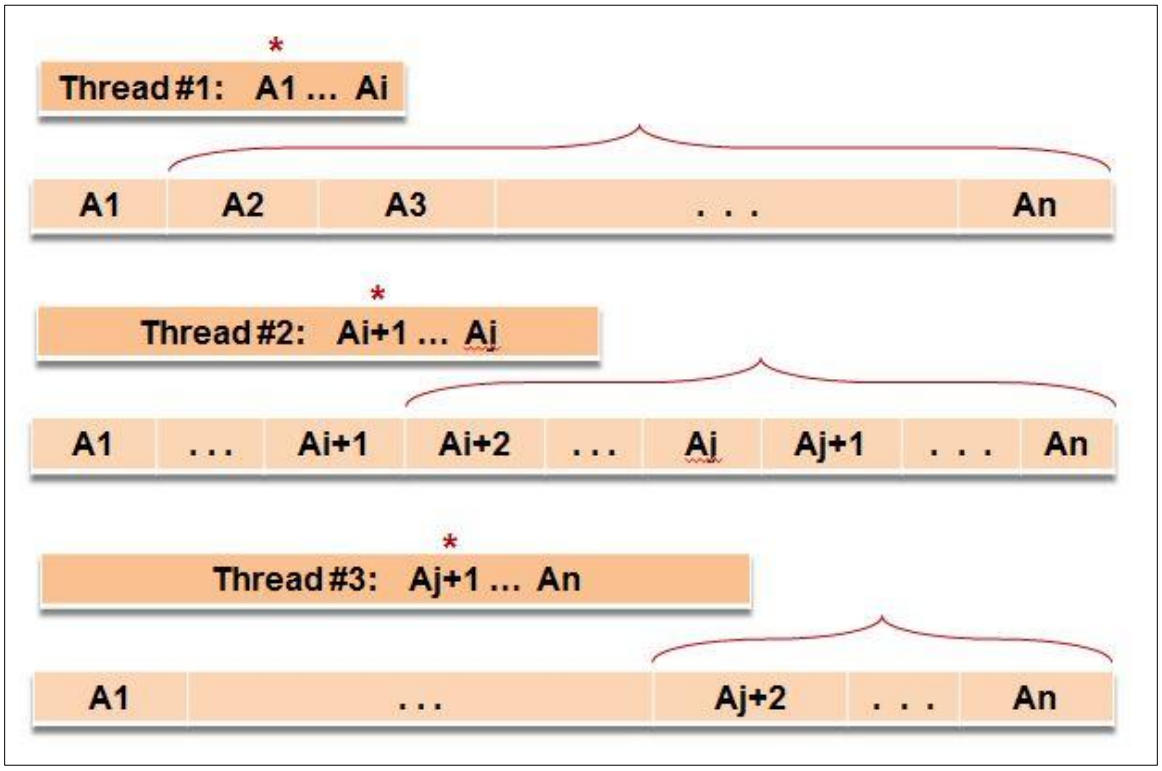

Figure 11: Thread-based Atom Distance Calculation, Method 2: unequal chunks, similar workloads

A third approach has been designed and implemented as well. This method is based on matrices. The atoms are distributed equally in matrix cells. In order to keep the workload of threads equal, two types of threads has been implemented. One for calculating the distance of non-diagonal cells of the top half of the matrix, and the other for the diagonal cells. The former takes two chunks of atoms and calculates the distance of each atom in

the first chunk to the atoms of second chunk. The latter takes two diagonal chunks and calculates the distance of atoms of first chunk to the rest of the atoms in first chunk itself. It does the same for the second chunk as well. Assuming that atoms are evenly distributed in the matrix cells M, Figure 12 shows a visual representation of distribution of 40 atoms (A) in a 4*4 matrix:

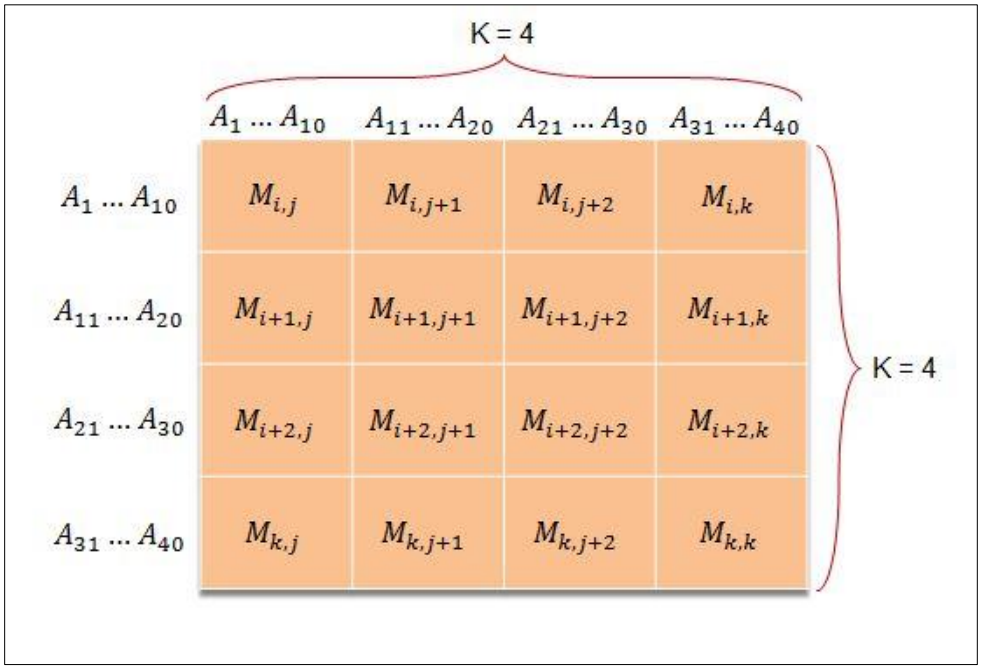

Figure 12: Distribution of Atoms in a 4x4 Matrix

Based on Figure 13, for the atoms in the non-diagonal cells the following represents the way first chunk and second chunk are paired to be sent to the threads:

$M_{i,j}$ pairs with $M_{i,j+1}$, $M_{i,j+2}$ and $M_{i,k}$

$M_{i+1,j+1}$ pairs with $M_{i+1,j+2}$ and $M_{i+1,k}$

…

$M_{k-1,k-1}$ pairs with $M_{k-1,k}$

For this particular example, 2 diagonal threads and 6 non-diagonal threads are created. Figure 13 shows distance calculation operations inside first thread. The distance of each and every atom of first chunk is calculated with all atoms of second chunk to create or update the adjacency lists.

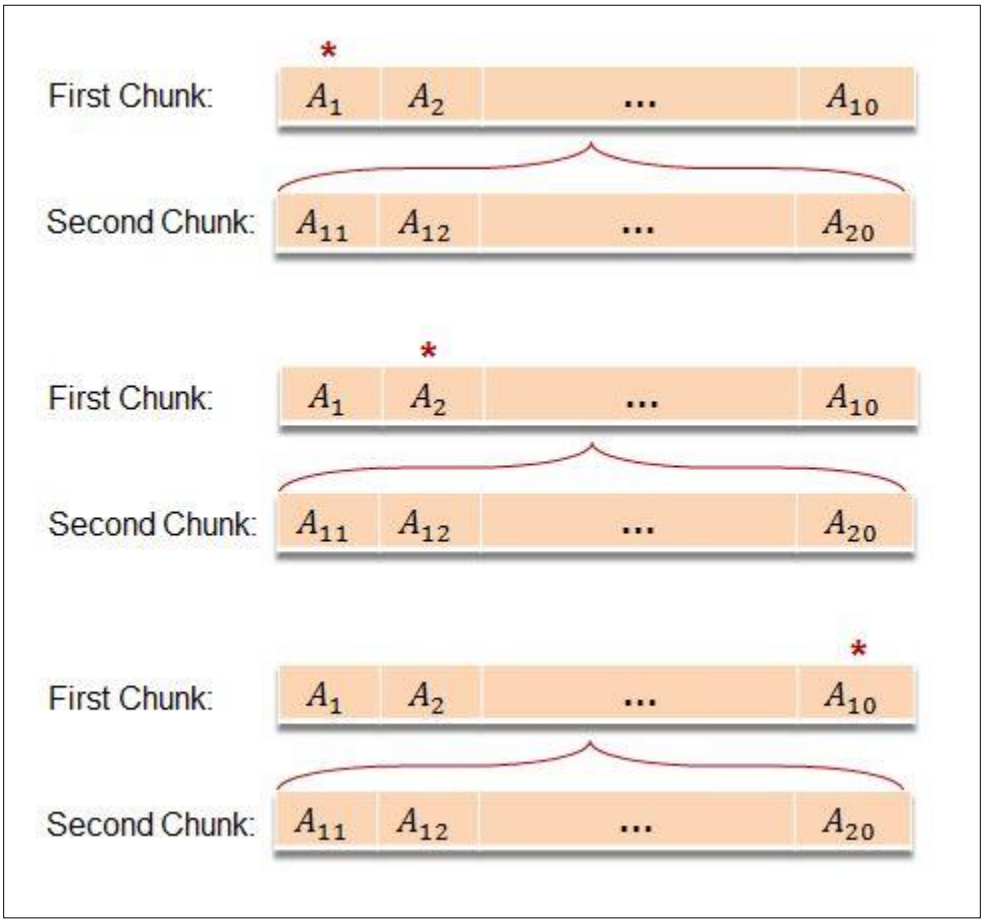

Figure 13: Thread-based Atom Distance Calculation, Method 3: first non-diagonal thread

Figure 14 represents the distance calculation in the first thread of diagonal thread. In this case, all atoms of first chunk are compared to the atoms of the chunk itself (except the

atom that is being compared to other atoms) and the same happens for the second chunk as well.

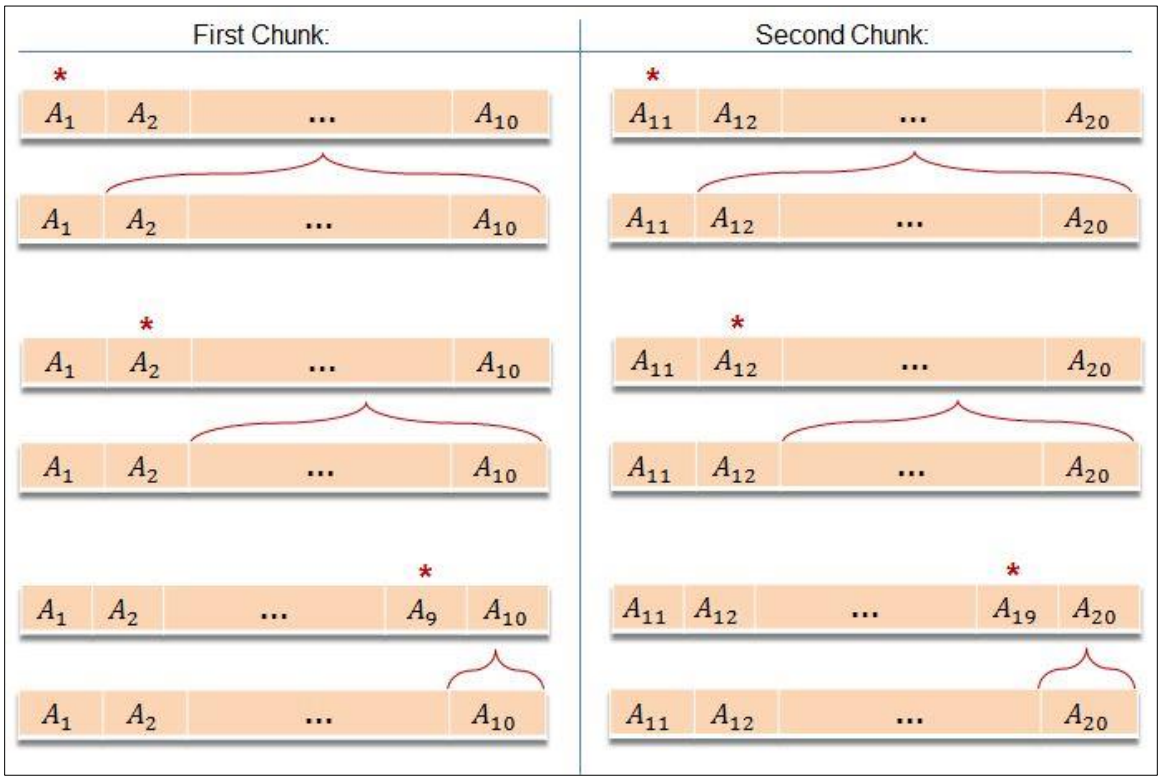

Figure 14: Thread-based Atom Distance Calculation, Method 3: first diagonal thread

The third approach guarantees roughly equal workload for all threads and showed the best performance among other approaches. The only drawback is higher bookkeeping to manage the chunks and more complex implementation. In this approach users can not specify the number of threads since it is dependent to the size of the matrix, however, they can specify the size of the matrix.

4.3.4. Sugar Identification

After building the graph structure of the molecule from the input PDB file based on the distance between each atom pair, the graph can be processed and analyzed to identify the sugar structures, if it has any. The following sections describe the steps of the Sugar ID process.

4.3.4.1. Cycle Detection

The first step to identify the sugars in a molecular structure is to detect the cycles in the graph. Cycles, or rings, make the foundation of a monosaccharide. Detecting cycles identifies the potential sub structures that might be a sugar. Different approaches have been used in this research, such as applying algorithms BFS [28] and DFS [28], using an algorithm called Exhaustive Ring Perception [29] method and using SPARQL queries to match the triples that form a cycle.

4.3.4.1.1. Ring Detection by Exhaustive Ring Perception

Cycle detection is a known problem in the field of computer science and also in chemistry. It is also referred to as ring detection/perception when dealing with molecular structures. The method described in this chapter, has been adopted from the publication [29]. Based on this algorithm, the molecule's graph is traversed and pruned recursively such that after pruning, no atoms with less than two bonds to other atoms remain in the graph. Figure 15 shows a simple graph and the pruning process.

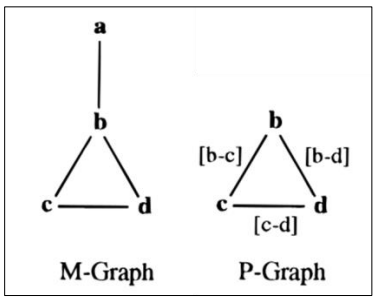

Figure 15 [29]: Pruning the Molecular Graph

The next step of the algorithm after pruning is to convert the "Molecular Graph" into a "Path Graph" as shown in Figure 16. During this process edges of the graph are labeled with the related node names.

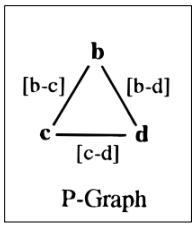

Figure 16 [29]: Converting Molecular Graph into Path Graph

Reducing the path graph is the last step of the algorithm. For applying this step, whenever a path "a-b-c" found in the graph (or a "walk" as stated in the publication), it should be reduced to "a-c" and the newly created edge should be labeled as [a-b-c] to keep track of the reduced nodes in the path. Reducing step continues until there are no more paths in the graph to be reduced. Figure 17 demonstrates this process.

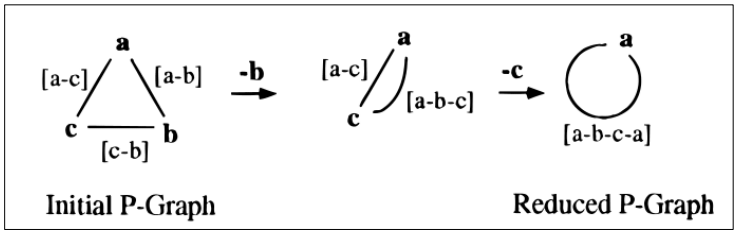

Figure 17 [29]: Reducing the Path Graph

Figure 18 shows all the three steps of the algorithm applied on a sample molecular graph.

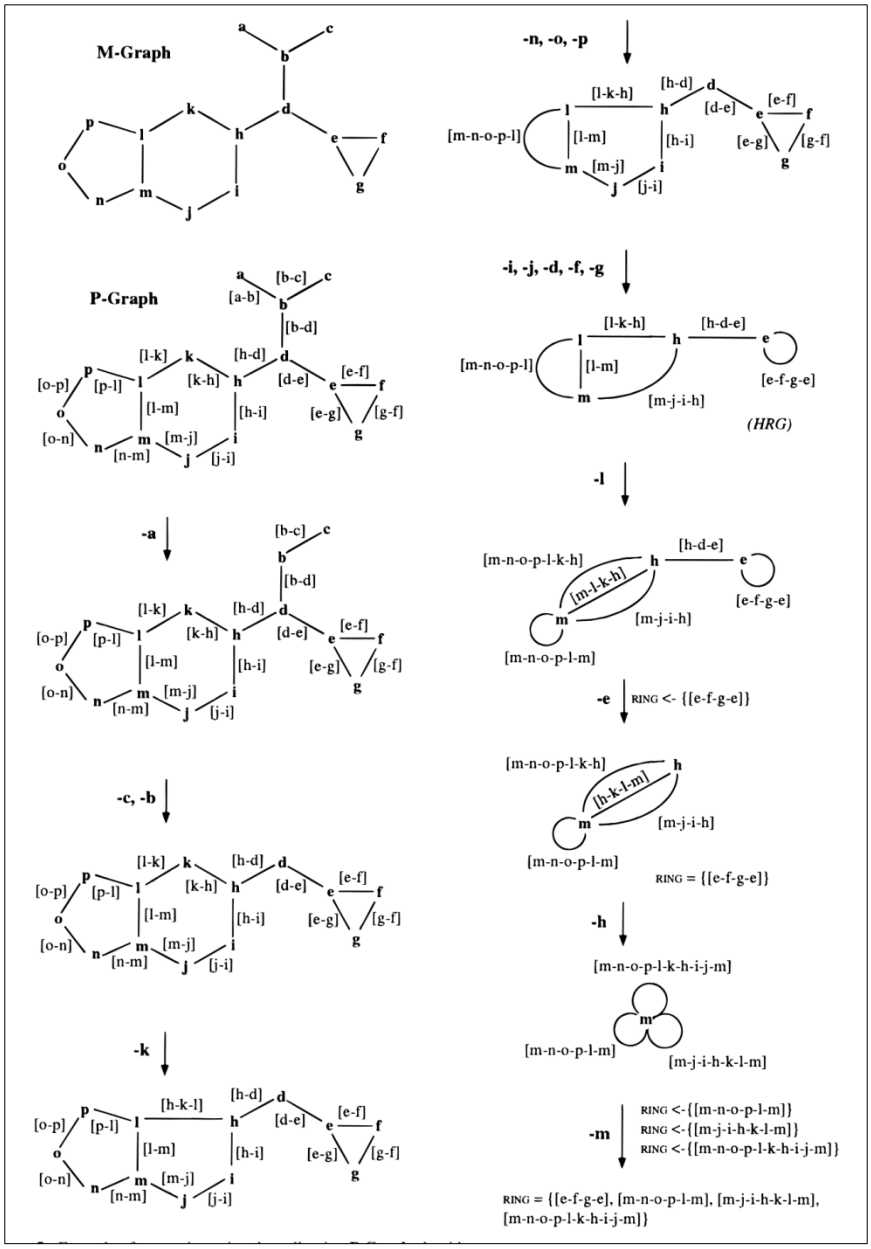

Figure 18 [31]: Exhaustive Ring Perception in Molecular Graphs

By post processing the labels of the reduced path graph, cycles and the atom nodes involved in them can be detected. The labels that have the same start and end nodes are representing a cycle. For the purpose of this research, which finding the six and five

membered rings is the priority, only the cycles with the size less or equal to 6 are retrieved and other cycles are discarded.

Table 1 represents the algorithm for detecting the cycles based on the Exhaustive Ring Perception algorithm [31].

Table 1: Algorithm 1

Cycle Detection by Exhaustive Ring Perception

//Pruning the graph
get the list of atoms
FOR each atom in the list
count the number of neighbors
ENDFOR
WHILE there are no atoms with less than 2 neighbors
REMOVE the atoms with less than 2 neighbors
ENDWHILE
 	
//Converting molecular graph into path graph
CREATE label for each edge based on the node IDs
 	
//Reducing the path graph
WHILE the list of atoms and the list of edges are not empty
SELECT the atom with the lowest number of edges
//A walk is a group of 3 atoms attached to each other, e.g. a-b-c
PROCESS the graph to check if there is a walk passed from the selected atom
IF walk found
//e.g. initial edges: (a,b), (b,c) -> new edge: (a,c)
CREATE a new edge
//e.g. initial labels: [a-b], [b-c] -> new label: [a-b-c]
CREATE a new edge label for the new edge
REMOVE initial edges of the walk //e.g. (a,b), (b,c)
REMOVE initial edge labels of the walk //e.g. [a-b], [b-c]
REMOVE the selected atom
ENDIF
ENDWHILE
 	
//Detect cycles
RETURN the edges and edge labels with the same start and end as cycles

4.3.4.1.2. Ring Detection by Depth First Search and Breath First Search Algorithms

In this method, initially, Depth First Search (DFS) has been used to traverse the graph and detect the cycles. Cycles are detected based on discovering the back edges. "***Back edges*** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree" [28]. Whenever a back edge is discovered it indicates that a cycle has been detected and to retrieve the nodes involved in the cycle, starting from the current node, the graph should be back tracked until the source node is reached. The source node is the same node which has been approached by the back edge from the current node. The way that standard DFS algorithm works is that each node is visited only once during the graph traversal and this property causes an issue with detecting all the cycles within the graph structure. If there are cycles containing smaller cycles, and if the algorithm discovers the bigger cycle first, not all the minimum sized cycles can be detected since revisiting the nodes is not allowed. Figure 19 represents this issue. The red lines show the traversing path, green lines represent the discovery of a back edge and crosses shows the cycles that are detected. In this particular example DFS does not detect the top minimum sized cycle.

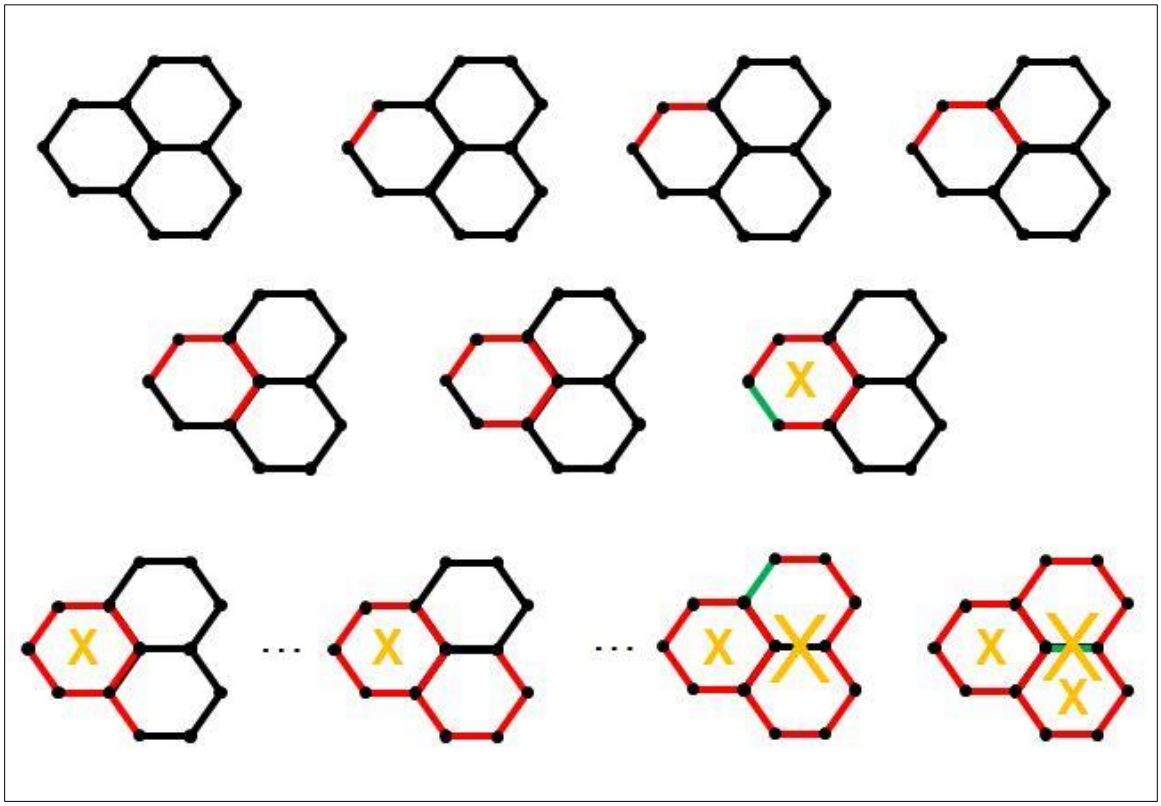

Figure 19: Missing Detection of Minimum Sized Cycles by DFS

Post-processing the detected cycles might help to detect the minimum sized cycles but it is a costly operation. In all of the detected cycles, all of the atoms and the neighbor atoms in their adjacency list should be examined again to differ the containment cycles from the minimum sized ones. In this research, the aim is to identify the minimum sized cycles rather than the cycles containing smaller cycles. Having containment cycles in the list of the detected cycles make the post processing of the cycles a more complex task in the further steps of the Sugar ID process. Applying BFS will cause similar problems. Although traversing the graph with DFS and BFS algorithms offers a high performance but due to the discussed issues it has been decided for Sugar ID not to proceed with these methods any further.

4.3.4.1.3. Ring Detection by SPARQL Query

Despite the constraint of the system on not using Java based systems, Apache Jena API [19] has been used in this method as an experiment. In this approach, GMML produces a CSV file based on the simulated molecule of the input PDB file. The CSV file contains the list of residues and atoms within the structure of the molecule. An external Java application has been implemented to take the generated CSV output as its input to create an RDF graph from its information. This ontology population is performed using Apache Jena API. The RDF graph has the atom and residue individuals such that each atom has links to other atoms that it is bonded to using the object property called "nodeNeighbor". Each residue individual also has relation with its atoms using the "hasAtom" object property. In this case, specific SPARQL queries had to be executed against the RDF graph to retrieve the cycles with five and six nodes. Due to the unavoidable multiple filtering operations in the queries, and also, as the number of atoms of a molecule grows, the running time of this approach increases drastically. The object property that represents neighbor atoms is a symmetric property. Therefore, each time a triple pattern is used in the query, for example, "?a2 <nodeNeighbor#> ?a3", it should be checked that the matched individuals for "?a3" are different from the "?a1" itself due to symmetry. Also, "?a3" should be different than the matched atoms of the previous queries to guarantee not to go backward in the cycle path. Figure 20 shows a sample query for retrieving six-membered rings.

```
base <http://www.semanticweb.org/de1aram/ontologies/2015/1/gmml>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/22-rdf-syntax-ns#>
            select ?a1 ?a2 ?a3 ?a4 ?a5 from <onePDB.rdf> where {
            ///cycles with 5 nodes
            ?a1 <#nodeNeighbor> ?a2.
            ?a2 <#nodeNeighbor> ?a3.
            filter (?a2 != ?a1)
            filter (?a3 != ?a1)
            ?a3 <#nodeNeighbor> ?a4.
            filter (?a3 != ?a2)
            filter (?a4 != ?a2)
            ?a4 <#nodeNeighbor> ?a5.
            filter (?a4 != ?a3)
            filter (?a5 != ?a3)
            filter (?a5 != ?a2)
            ?a5 <#nodeNeighbor> ?a1.
            filter (?a5 != ?a4)
            filter (?a4 != ?a1)    }
```

Figure 20: Cycle Detection by SPARQL Query

Based on the aforementioned performance inefficiency, unnecessary back and forth interaction of GMML with the external application, and the constraint of the system about not using Java based systems this method has been left as an experiment.

Rings that are candidates to be identified as sugars should have an oxygen (or sometimes a nitrogen) atom. Therefore, in this of the Sugar ID process, regardless of the algorithm that has been used to detect cycles, the rings are post processed to retain only the ones that structurally have a potential to be considered as a sugar. Post processing operation discards rings without oxygen or nitrogen, and rings that are sharing an edge (fused cycles). Figure 21 shows an example of retained and discarded rings.

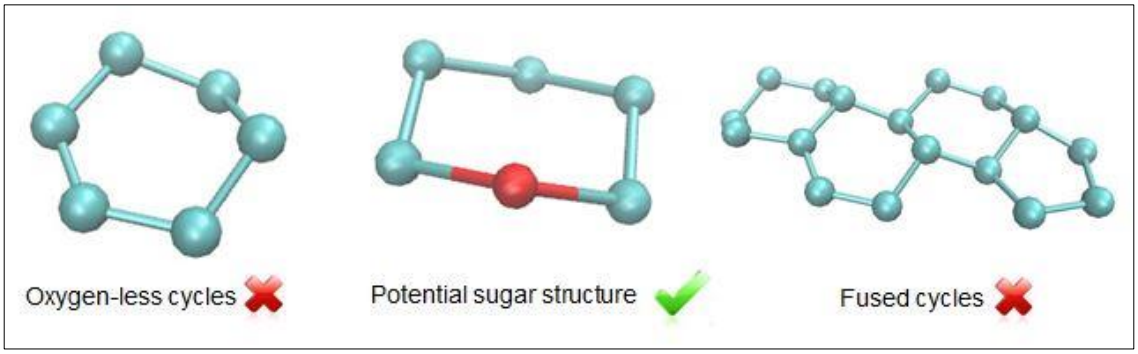

Figure 21: Post Processing Cycles

4.3.4.2. Anomeric Carbon Detection

The carbon atom next to the oxygen of the ring that has another bond to an exocyclic oxygen is called an anomeric carbon. Detecting anomeric carbons is crucial to correctly identifying the sugars. Indexing and creating the stereochemistry chemical code for the geometric structure of the sugars is directly related to detecting the anomeric carbon. Anomeric carbon is considered the first carbon of the ring and indexing starts from there. In Figure 22 anomeric carbon of the ring is shown with a green circle shade.

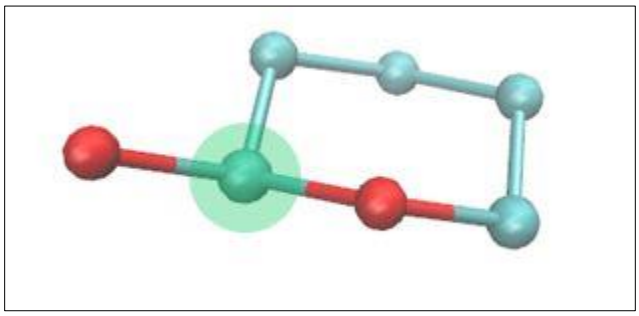

Figure 22: Anomeric Carbon

Sometimes detecting anomeric carbons gets complicated if the exocyclic oxygen that supposed to be attached to the ring carbon is missing. In these situations, different actions might be taken. First method is to detect the anomeric carbon based on the atom indices

that comes with the names of the ring atoms. In this method, the carbon of the ring with the lowest index is chosen as the anomeric carbon. The second method is based on the fact that usually the last carbon of the ring is attached to an exocyclic carbon atom. Thus, between the two carbon neighbors of the ring oxygen the one that is not attached to an exocyclic carbon is the anomeric carbon. The third method is used when the indices are not helpful and also none of the carbon neighbors of the ring oxygen have a bond to an exocyclic carbon atom. In this case, the system randomly chooses one of the ring carbons next to the oxygen of the ring. In all the aforementioned special situations, the user is warned about the issue. A status and a note are stored in the ontology, as well. Figure 23 represents the three methods.

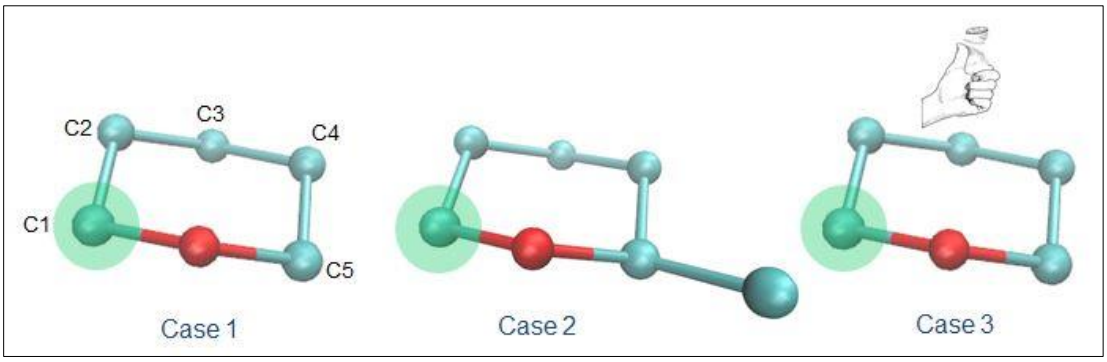

Figure 23: Anomeric Carbon Detection, Special Situations

Table 2 represents the algorithm to detect the anomeric carbon of the ring and the special cases discussed in this section.

Table 2: Algorithm 2

Anomeric Carbon Detection
IF there is an oxygen in the ring //RULE 1

```
        get the two neighbors of the oxygen
        FOR each of the two neighbors of the oxygen
                get the neighbors
                FOR each neighbor
                        IF the atom is a non-ring oxygen
                                set the atom as anomeric carbon
                                RETURN anomeric carbon
                        ENDIF
                ENDFOR
        ENDFOR

        //RULE 2
        Check the order of the ring atoms based on their name indices
        set the ring atom with lowest index to anomeric carbon
        IF anomeric carbon is set
                RETURN anomeric carbon
        ENDIF

        //RULE 3
        FOR each of the two neighbors of the oxygen
                get the neighbors
                FOR each neighbor
                        IF the atom is a non-ring carbon
                                set the other oxygen neighbor as anomeric carbon
                                RETURN anomeric carbon
                        ENDIF
                ENDFOR
        ENDFOR

        //RULE 4
        RETURN one of the two oxygen neighbors randomly

ELSE
        RETURN null
```

4.3.4.3. Side Atoms Orientation Calculation

Sugars might have different names if they have an exact same ring structure but different exocyclic (side) atom orientations. The orientation of a side atom that is attached to a ring atom is calculated based on the position of the side atom with respect to the ring. The orientation can be either above or below the ring. To calculate the

orientation of each side atom of each of the ring atoms, firstly, two vectors are created from previous and next ring atom neighbors of the ring atom. As shown in Figure 24, a plane is calculated based on the coordinates of these two vectors. Moreover, the normal vector of the plane, and the normal vector of the vector created from the side atom's geometric coordinate are also created. The angle between these two normal vectors indicates the orientation of the side atom with respect to the ring. If it is less than 180 degrees, the side atom is above the ring and otherwise it is below. The orientation of the side atoms are used to generate the stereochemistry chemical codes for monosaccharides in the next step of the Sugar ID process.

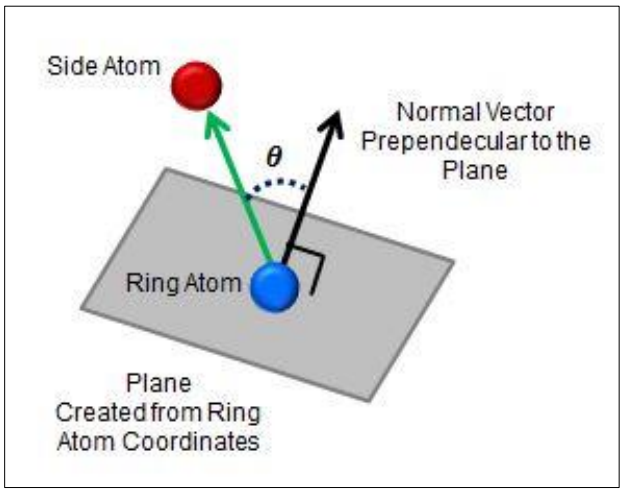

Figure 24: Side Atom Orientation Calculation

4.3.4.4. Chemical Code Generation

Glycobiologists usually look at the structure of the sugar ring from an angle such that the oxygen of the ring is on the right side of the ring and the anomeric carbon is toward the viewer. Also, as shown in Figure 25, if a vertical line cuts the ring in two halves, the

oxygen, anomeric carbon and the last carbon of the ring are on the right half and the rest on the left.

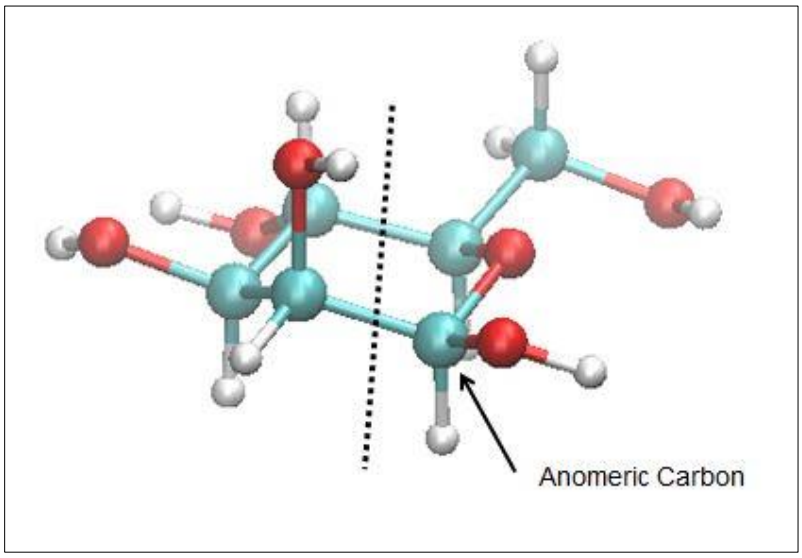

Figure 25: Dividing Ring Atoms

Based on this common technique, a chemical code [31] is assigned to each ring and its side atoms that have been identified in previous steps. An example of a ring in two different views, and its chemical code is shown in Figure 26.

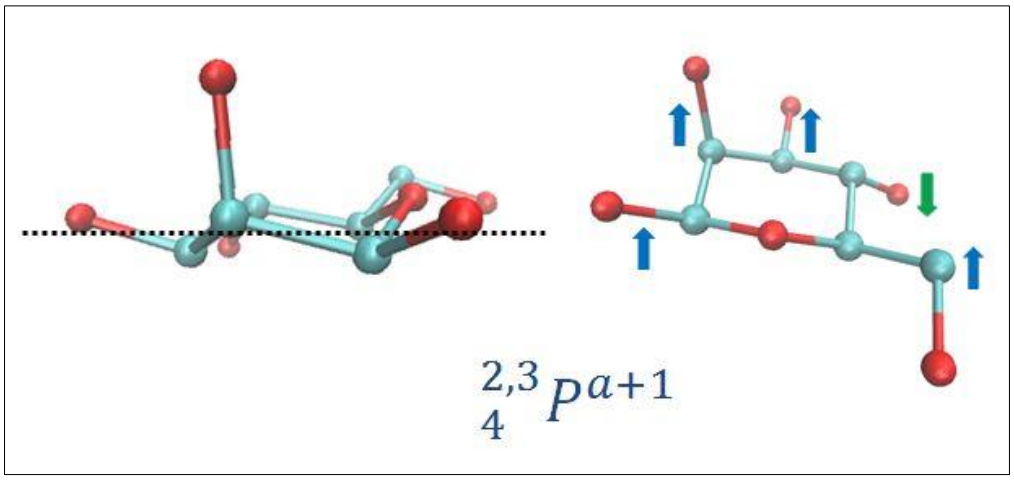

Figure 26: Chemical Code

The center of this chemical code indicates whether the sugar is pyranose or a furanose and it can be “P” or “F” with respect to the number of ring atoms. There are indices at the left and right side of this center element. Indices can be written at top, bottom or middle position on each side of the center element. The middle position is used in cases that the side oxygen/nitrogen is missing for a particular atom. Therefore, no orientation has been assigned to it. Indices represent the index of the atoms in the ring and the positions they are written in, represent their orientations with respect to the ring. By considering the vertical line that has been discussed at the beginning of this section, atoms of the right half (oxygen, anomeric and the last carbon of the ring) are written on the right side of the center element and the rest on the left. As an example, in Figure 27, the anomeric carbon of the ring which is represented by “a” is written on the right side the “P” and its position is up. “+1” notion is used to refer to the side atom of the last carbon of the ring. In this example, second, third and fourth ring carbons are written at the left side of the center element. The fourth carbon is the only carbon that has a side atom with the position below the ring. The chemical code facilitates encoding the geometric structure of the sugar, and it will be used to assign a name to the detected structure.

4.3.4.5. Stereochemistry Name Assignment

Domain experts has provided a lookup table (Appendix B) that can be used for assigning names to the sugar based on the linear version of the generated chemical code. In this step, the chemical code of the sugar is searched in the table; if an exact match is found, then, a stereochemistry long and condensed name is assigned to the sugar structure. In cases when an exact match cannot be found, the system finds the closest match to the generated chemical code for the sugar and the user will be informed of the

situation. Also, a report as an individual member of the GMMO's note class is stored in the ontology. Figure 27 shows an example of the name assignment for a sample monosaccharide.

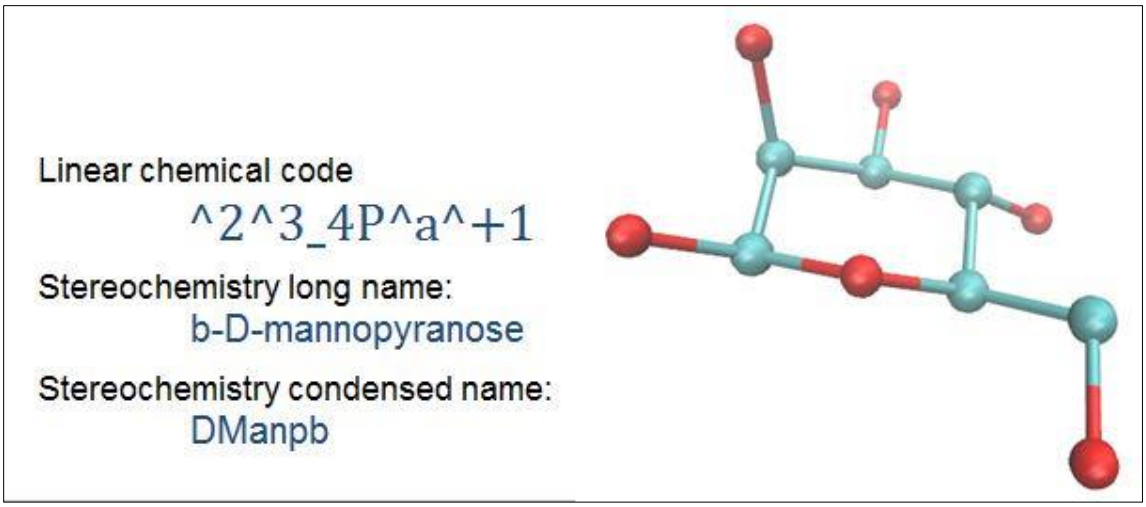

Figure 27: Stereochemistry Name Assignment

4.3.4.6. Modification/Derivative Pattern Matching

At this point, the ring and the immediate side atoms of the ring has been identified and based on the chemical code of the structure, the stereochemistry names have been assigned to the monosaccharide. Existence of some atomic patterns (derivatives or modifications) might change the name of the monosaccharide. Thus, it is important to analyze the subgraphs that can be detected by traversing the molecule's graph structure. Traversal starts from the carbon atoms of the identified monosaccharides. A list of certain patterns and their possible effects on the name of the monosaccharide has been provided by the domain experts (Appendix B). GMML creates subgraph patterns for each entry of the provided list of modification/derivatives. Another set of subgraph patterns are also collected and created from traversing the graph starting from the carbon atoms of the

monosaccharides. The patterns are compared to each other and whenever a match is found, based on the defined effects of that particular derivative/modification, the name of the monosaccharide is updated. Figure 28 depicts examples of subgraph pattern matching. In this specific monosaccharide, one of the sub graphs matches the subgraph that is attached to the monosaccharide structure.

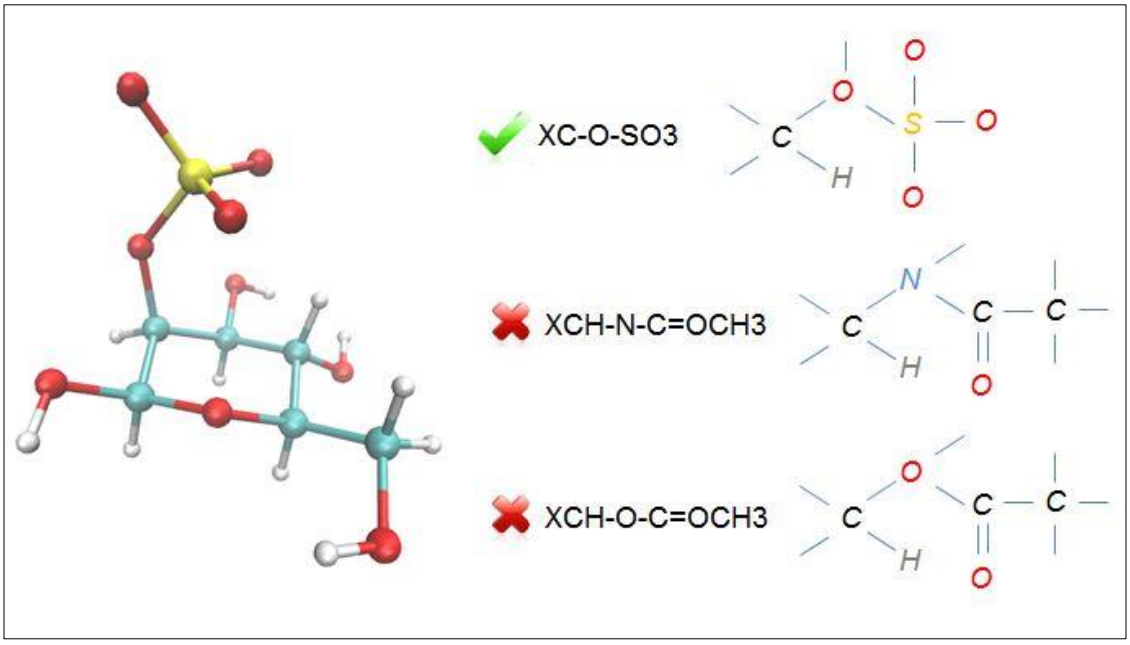

Figure 28: Subgraph Pattern Matching

In Figure 29, the process of updating the name is shown for the particular example represented in Figure 28.

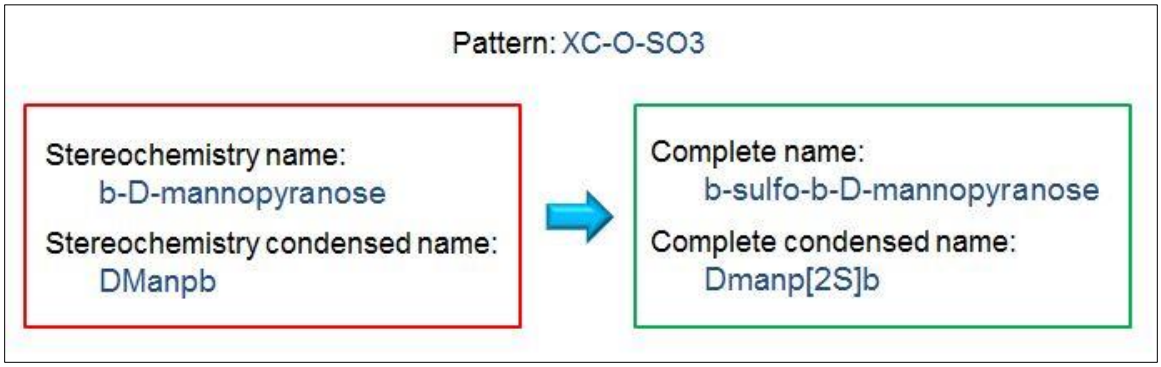

Figure 29: Updating Names of the Monosaccharide Based on the Discovered Pattern

4.3.4.7. Root Identification

Monosaccharides might be attached to each other in a sequence. As shown in Figure 30 the sequence might have branches as well which are the sequences of monosaccharides.

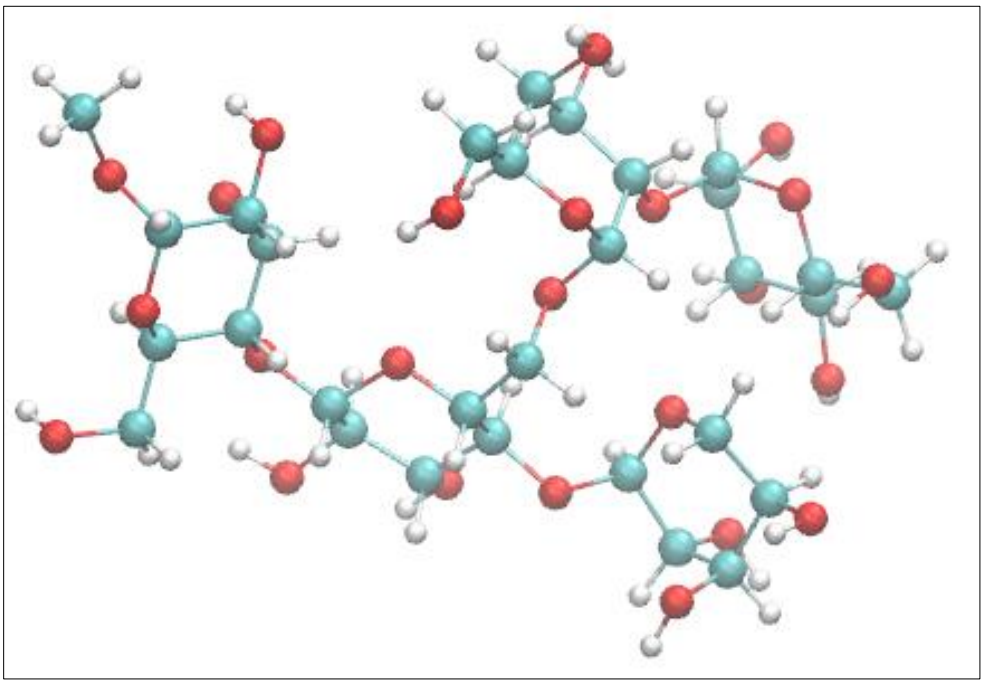

Figure 30: Linked Monosaccharides of an Oligosaccharide

In previous sections, the system has identified monosaccharides individually and without considering their linkages to each other. Detecting these linkages and their properties are important to assign a proper name to the oligosaccharide. However, prior to assign a name to the oligosaccharide, one of its monosaccharides should be selected as the root of the oligosaccharide. The next section discusses using this root as a starting point to create the name of the oligosaccharide. Based on some rules defined by the domain experts, the graph structure of each oligosaccharide is processed and the root monosaccharide is identified for each oligosaccharide. The first rule which has the highest level of priority is applied when there are two monosaccharides that are link to one another by their anomeric positions. In this case, the monosaccharide that has no other attachments to other monosaccharides is chosen as the root. The second rule is called “Directed Graph Rule” by domain experts. The monosaccharide linkages are examined by traversing the graph structure of the oligosaccharide and if there is a monosaccharide *A* attached to monosaccharide *B* through its anomeric position, then monosaccharide *A* is discarded from the list of potential root monosaccharides. This process runs until there is only one monosaccharide left in the list. The remaining monosaccharide is chosen as the root. Figure 31 shows application of this rule on the structure of an oligosaccharide. All the monosaccharides that are crossed in the figure are attached to another monosaccharide through their anomeric atoms.

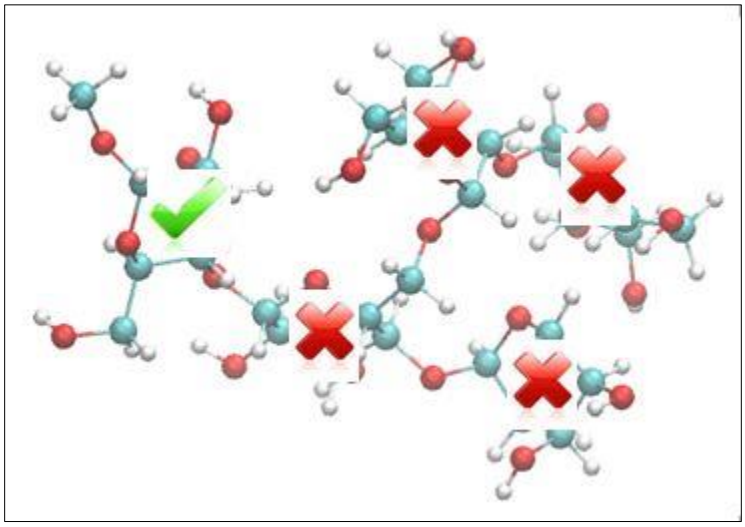

Figure 31: Root Identification by Applying Directed Graph Rule

In situations that the root could not be identified by applying the previous rules, the "non-sugar attachment" rule should be applied. In these situations, the monosaccharide that has a linkage to a non-sugar structure (e.g. a terminal residue or a protein) is identified as the root.

4.3.4.8. Oligosaccharide Name Assignment

There are many different nomenclatures that can be applied for assigning a name to an oligosaccharide. Based on the opinion of the domain experts, one of the most understandable and popular nomenclature disciplines is to use the condensed notation of sugar names, and square brackets to represent the branches of the sugar. As shown in the following, the indices of the carbon atoms that are involved in the monosaccharide linkages are placed in between the condensed notations.

[CondensedNotation1][CarbonIndex]-[CarbonIndex][CondensedNotation2]

Example: DGlcp1-4DAllpa

With respect to the aforementioned disciplines, the graph structure of the oligosaccharide should be traversed in order to create the oligosaccharide name sequence. To make the process of traversing the graph and creating the oligosaccharides name sequence more convenient, a new graph is generated based on the graph structure of the oligosaccharide. Monosaccharides become the new nodes and their linkages become the edges of the new graph versus the initial graph which nodes were the atoms and their bonds the edges. Figure 32 represents the information acquired by applying previous steps of the Sugar ID process and also creating the new graph based on such information.

Monosaccharide	Name
M1 (Root)	DAllpa
M2	DGlcpb
M3	DArapa
M4	LGalpa
M5	DManpa

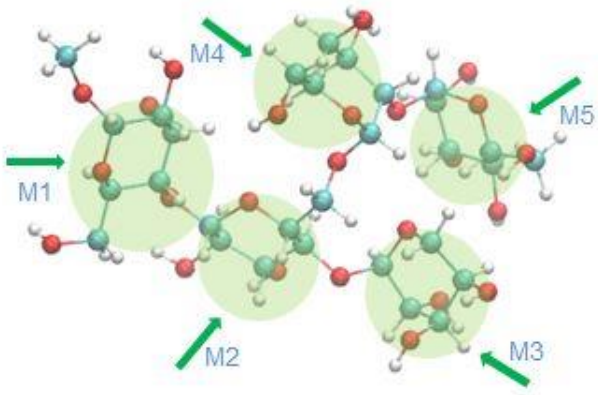

Figure 32: Oligosaccharide Structure Information

Based on the new graph, the structure is traversed starting from the root monosaccharide. The name sequence of the oligosaccharide builds up gradually and in the backward direction. At the end, the root monosaccharide should be at the end of the sequence and the rest of the monosaccharide and branches are built up on its left side toward the beginning of the name sequence. Figure 33 shows an example of the new graph and the created name sequence for the oligosaccharide.

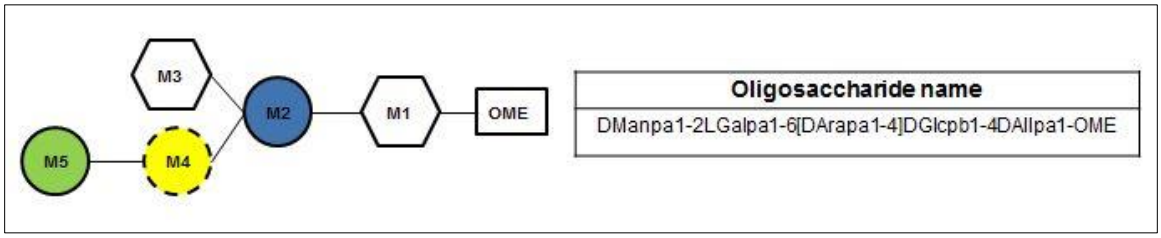

Figure 33: Oligosaccharide Name Assignment

4.4. Ontology Population

The main aim of this system is to provide a searchable data store consist of glycan and moiety structures. The Sugar ID process extracts relevant information from an input PDB file at runtime. But the obtained information is not stored in any type of database persistently. Therefore, by processing PDB files, the ontology should be gradually populated using the resulted information from the Sugar ID process. This chapter discusses the process of ontology population from the extracted data.

4.4.1 Triple Generation Process

Since GMML is a C++ based library and because of the constraints of the system, Java based APIs cannot be used for generating, editing or deleting the triples, and all such processes should be implemented manually inside GMML. No third party libraries have been used for this process in favor of keeping the system with the minimum dependencies. The result of Sugar ID process is not in the triple format and is not stored in a data store. The result is only accessible from memory while the program is running. For example, the assembly class of GMML has an attribute named OligosaccharideVector which holds a list of oligosaccharide instances created from Oligosaccharide struct in the GMML system. Oligosaccharide struct holds the detailed information (e.g. name, linkages, root information etc.) that have been identified during

Sugar ID process. In pursuance of providing the means to generate triples from the information, an ontology vocabulary has been created which can be found in Appendix C. This vocabulary brings a global, consistent, scalable aid to different functions designed to populate different parts of GMMO. The vocabulary accommodates classes and properties of the ontology, prefixes and some SPARQL query phrases. The extracted information from the Sugar ID process is distributed over multiple functions to be prepared and transformed into the Turtle triple format [35]. Furthermore, the generated triples of PDB files are written into a file. Helper functions such as *CreateURIResource()*, *CreateURI()*, *AddTriple()* and *AddLiteral()* have been implemented to set up a more dynamic environment to generate triples. Some examples from the usage of these helper functions can be found in Appendix D.

4.4.2. Bulk Population of Ontology

The Sugar ID process, takes in PDB files one at a time and process them individually. There are more than 114,000 PDB files to be processed. Concerning to populate the ontology from processing of all the PDB files, a bash script file has been designed and written to take the PDB files from their different directories in the repository, extract and send them as an input to Sugar ID and ontology population processes of GMML, and later, store the result after compressing them. The bash script also logs every step of the bulk population process such that it provides trackable notes about the remaining and processed files while the process is running (e.g. it takes about 9 days to process 114,000+ files using 4 threads). It logs all the errors and issues that happened during the process as well. This bash script can be found in Appendix E. Later

on, the collected turtle-formatted triple sets have been bulk loaded into the Virtuoso data store. Figure 34 shows the entire process of bulk ontology population.

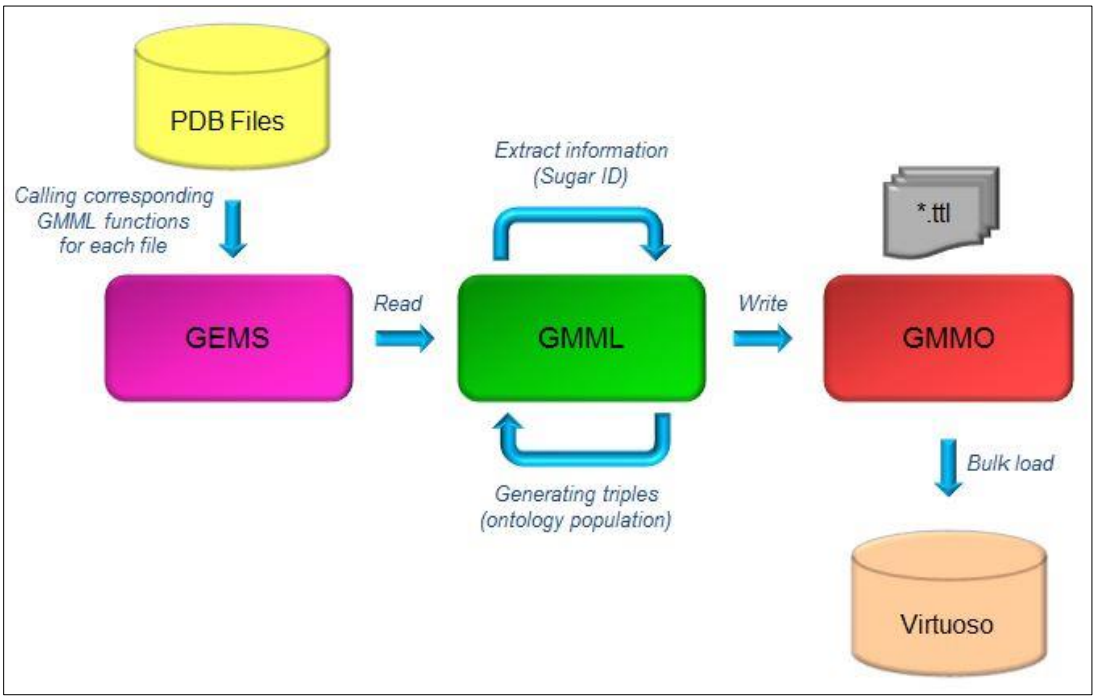

Figure 34: Bulk Ontology Population

Figure 35 shows a small fragment of data converted into the triple format.

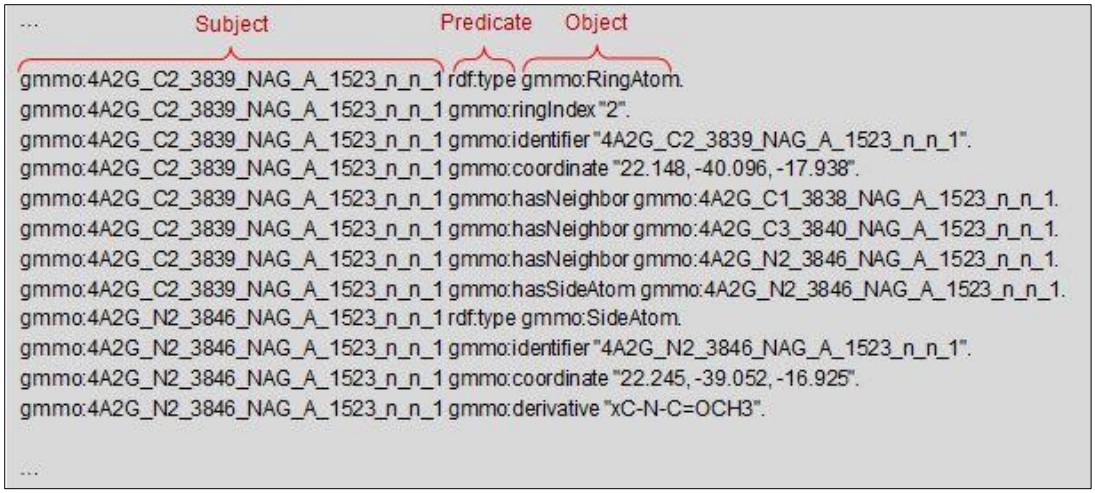

Figure 35: An Example of Extracted Information to Triple Format Conversion

4.5. Search System

In order to create a flexible, easy to use environment for chemists to work with this system, a search platform has been designed and implemented. The Search system enables scientists to query GMMO without the need to learn how to write SPARQL queries.

4.5.1. Dynamic Query Generation

With supervision of domain experts, a number of popular questions and interesting subjects have been gathered to be queried from the ontology (more details in section 4.5.3). Based on the questions, multiple functions are implemented in GMML level that can be triggered from the GEMS level. These functions are designed to create a template query engine. The SPARQL queries are automatically generated within the functions of the GMML system with respect to the input arguments given to these functions. Therefore, similar but yet different queries can be asked using a single function call.

4.5.2. Executing Template Queries

After generating a proper SPARQL query by the query engines, GMML uses cURL [30] to serialize the query and send it to Virtuoso server end point as a URL. “cURL is an open source command line tool and library for transferring data with URL syntax” [30]. It supports variety of protocols such as FTP, HTTP and HTTPS. Virtuoso compares the query triples to the triples of the GMMO. Moreover, if there is any matched triple(s), the result is returned to GMML, GEMS and finally, to the user in the requested data format. Figure 36 demonstrates the search system architecture and Figure 37, an example query execution.

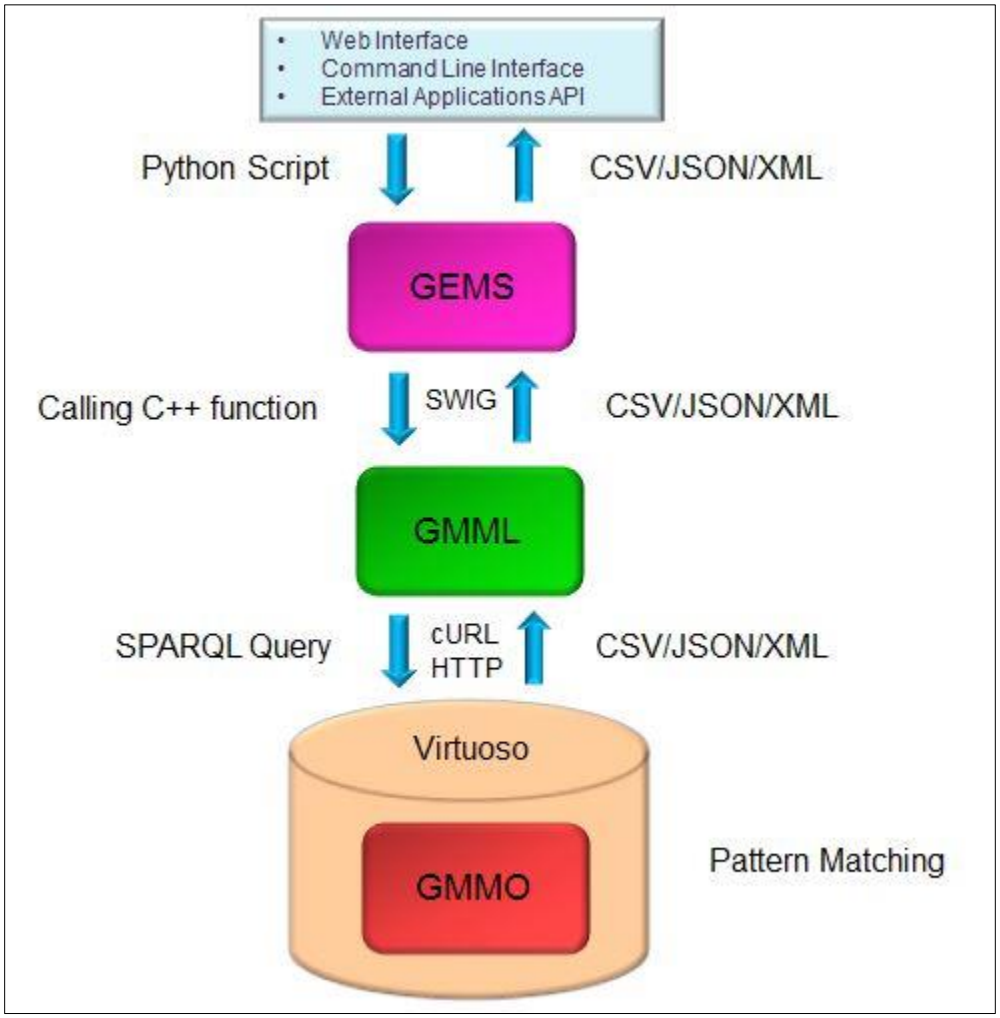

Figure 36: Search System Architecture

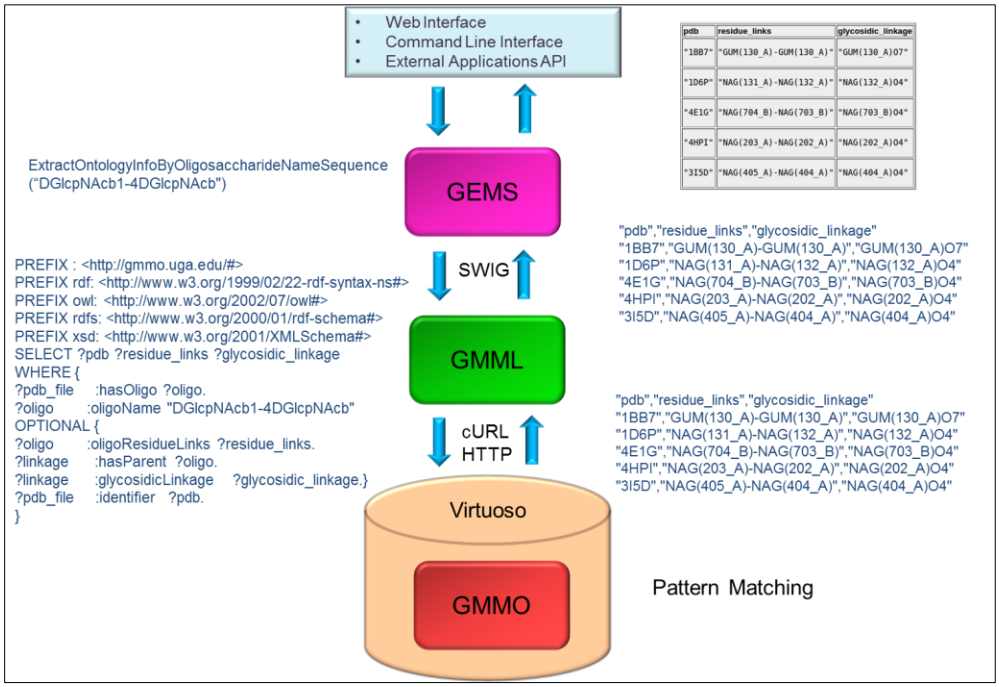

Figure 37: An Example of Query Execution Process

The design of the system takes into account the fact that the users of this system are mainly, chemists who work in computational labs and are comfortable with using scripting languages and APIs. Another point to keep in mind is that this system is also available as a tool that can be used in variety of projects with different specifications. The implemented template query functions can be accessed directly from GMML or more conveniently from the scripting layer, GEMS.

4.5.3. Example Queries

In the following, nine implemented template query functions along with their signature, sample usage and sample generated query are represented (prefixes are all the same for all queries, so they are not shown after Query 1):

Query 1: This query searches the ontology based on names of the glycan and returns the related PDB file and glycan information. The user can specify all four names (stereochemistry/complete/condensed/long) or leave some of them as blank.

Signature: *string ExtractOntologyInfoByNameOfGlycan(string stereo_name, string stereo_condensed_name, string name, string condensed_name)*

Sample Usage: *ExtractOntologyInfoByNameOfGlycan("", "", "", "DGlcpNAcb")*

Sample Generated Query:

```
PREFIX : <http://gmmo.uga.edu/#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?pdb ?residue_id ?stereo_name ?stereo_condensed_name ?name
?condensed_name ?shape WHERE {
?pdb_file       :hasOligo   ?oligo.
?oligo          :hasCore    ?mono.
?mono           :hasSugarName   ?sugarName.
?sugarName      :monosaccharideShortName   "DGlcpNAcb".
?pdb_file       :identifier   ?pdb.
?mono           :hasRingAtom   ?ring_atoms.
?residue        :hasAtom      ?ring_atoms.
?residue        :identifier   ?residue_id.
?sugarName      :monosaccharideStereochemName   ?stereo_name.
?sugarName      :monosaccharideStereochemShortName   ?stereo_condensed_name.
?sugarName      :monosaccharideName   ?name.
?sugarName      :monosaccharideShortName   ?condensed_name.
OPTIONAL { ?mono       :BFMPRingConformation   ?shape.}
}
```

A fragment of the result set is shown in Figure 38.

pdb	stereo_name	stereo_condensed_name	name	condensed_name
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"1X0C"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"2XB6"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"
"2XB6"	"b-D-glucopyranose"	"DGlcpb"	"N-acetyl-b-D-glucopyranose"	"DGlcpNAcb"

Figure 38: A Fragment of Query 1 Result Set

Query 2: This query searches the ontology based on different components of the monosaccharide name.

Signature: *string ExtractOntologyInfoByNamePartsOfGlycan(string isomer, string ring_type, string configuration)*

Sample Usage: *ExtractOntologyInfoByNamePartsOfGlycan("L", "F", "alpha")*

Sample Generated Query:

```
SELECT ?pdb ?stereo_name ?stereo_condensed_name ?name ?condensed_name ?shape
WHERE {
?pdb_file     :hasOligo    ?oligo.
?oligo        :hasCore     ?mono.
?mono         :hasSugarName   ?sugarName.
```

```
?sugarName  :isomer  "L".
?sugarName  :ringType  "F".
?sugarName  :configuration  "alpha".
?pdb_file  :identifier  ?pdb.
?sugarName  :monosaccharideStereochemName  ?stereo_name.
?sugarName  :monosaccharideStereochemShortName  ?stereo_condensed_name.
?sugarName  :monosaccharideName  ?name.
?sugarName  :monosaccharideShortName  ?condensed_name.
OPTIONAL { ?mono  :BFMPRingConformation  ?shape.}
}
```

Query 3: This query searches the ontology based on the identifier of the PDB file, and returns the results specific to that PDB.

Signature: *string ExtractOntologyInfoByPDBID(string pdb_id)*

Sample Usage: *ExtractOntologyInfoByPDBID("4A2G")*

Sample Generated Query:

```
SELECT ?oligo_sequence ?residue_links ?glycosidic_linkage WHERE {
:4A2G  :hasOligo  ?oligo.
?oligo  :oligoName  ?oligo_sequence.
OPTIONAL { ?oligo  :oligoResidueLinks  ?residue_links.
?linkage  :hasParent  ?oligo.
?linkage  :glycosidicLinkage  ?glycosidic_linkage.}
}
```

Query 4: This query is used by scientists who are familiar with the stereochemistry chemical code (GlyCode) that has been used in this system. The query searches the ontology based on the given linear form of the chemical code. "_" in the chemical code is used for atom with positions below the ring and "^" for the ones above it. There is also an option "d" which is used where the ring atom is not attached to any exocyclic oxygen/nitrogen atom. Such atoms are called deoxy atoms.

Signature: *string ExtractOntologyInfoByStringChemicalCode(string chemical_code)*

Sample Usage: *ExtractOntologyInfoByStringChemicalCode("_4^2^3P^a^+1")*

Sample Generated Query:

```
SELECT ?pdb ?name ?short_name ?stereo_name ?stereo_short_name WHERE {
?mono        :stereochemistryChemicalCode        "_4^2^3P^a^+1".
?pdb_file    :hasOligo ?oligo.
?oligo       :hasCore  ?mono.
?mono        :hasSugarName     ?sn.
?sn          :monosaccharideName       ?name.
?sn          :monosaccharideShortName  ?short_name.
?sn          :monosaccharideStereochemName     ?stereo_name.
?sn          :monosaccharideStereochemShortName        ?stereo_short_name.
?pdb_file    :identifier   ?pdb.
}
```

Query 5: This query searches the ontology based on the name of a monosaccharide and returns results specific to that oligosaccharide.

Signature: *string ExtractOntologyInfoByOligosaccharideNameSequence(string oligo_name)*

Sample Usage: *ExtractOntologyInfoByOligosaccharideNameSequence("DGlcpNAcb1-4DGlcpNAcb-ASN “)*

Sample Generated Query:

```
SELECT ?pdb ?residue_links ?glycosidic_linkage WHERE {
?pdb_file     :hasOligo ?oligo.
?oligo        :oligoName        "DGlcpNAcb1-4DGlcpNAcb-ASN"
OPTIONAL { ?oligo       :oligoResidueLinks     ?residue_links.
?linkage      :hasParent        ?oligo.
?linkage      :glycosidicLinkage    ?glycosidic_linkage.}
?pdb_file     :identifier   ?pdb.
}
```

Query 6: This query is one of the most popular queries for the scientists of the field. It searches the ontology based on the given oligosaccharide name pattern. The pattern may contain wild card(s) at different position(s) (at the beginning/middle/end of the sequence) which the function converts to a proper regular expression phrase. By utilizing the "regex" feature of SPARQL a suitable query is generated.

Signature: *string ExtractOntologyInfoByOligosaccharideNameSequenceByRegex(string pattern)*

Sample Usage 1: *ExtractOntologyInfoByOligosaccharideNameSequenceByRegex("*b1-4L*")*

Sample Generated Query 1:

```
SELECT ?pdb ?oligo_sequence ?residue_links ?glycosidic_linkage WHERE {
?oligo        :oligoName       ?oligo_sequence.
FILTER regex(?oligo_sequence, "b1-4L", "i")
?pdb_file     :hasOligo ?oligo.
OPTIONAL { ?oligo       :oligoResidueLinks     ?residue_links.
                ?linkage      :hasParent       ?oligo.
                ?linkage      :glycosidicLinkage    ?glycosidic_linkage.}
?pdb_file     :identifier      ?pdb.
}
```

Sample Usage 2: *ExtractOntologyInfoByOligosaccharideNameSequenceByRegex("DGlcpNAcb*GlcpNAcb")*

Sample Generated Query 2:

```
SELECT ?pdb ?oligo_sequence ?residue_links ?glycosidic_linkage WHERE {
?oligo        :oligoName       ?oligo_sequence.
FILTER regex(?oligo_sequence, "^DGlcpNAcb.+GlcpNAcb$", "i")
?pdb_file     :hasOligo ?oligo.
OPTIONAL { ?oligo       :oligoResidueLinks     ?residue_links.
                ?linkage      :hasParent       ?oligo.
```

?linkage :glycosidicLink age ?glycosidic_linkage.}
?pdb_file :identifier ?pdb.
}

Sample Usage 3: *ExtractOntologyInfoByOligosaccharideNameSequenceByRegex("*DGlcpNAcb*DGlc*")*

Sample Generated Query 3:

SELECT ?pdb ?oligo_sequence ?residue_links ?glycosidic_linkage WHERE {
?oligo :oligoName ?oligo_sequence.
FILTER regex(?oligo_sequence, "DGlcpNAcb.+Glcp", "i")
?pdb_file :hasOligo ?oligo.
OPTIONAL { ?oligo :oligoResidueLinks ?residue_links.
?linkage :hasParent ?oligo.
?linkage :glycosidicLink age ?glycosidic_linkage.}
?pdb_file :identifier ?pdb.
}

Query 7: This function can be used to design creative interactive user interfaces for different projects where the users can specify the orientation of the side atoms with respect to the ring. Or, the function can be simply used by specifying the orientations as “Up”, “Down” or blank. The generated query searches the ontology for the monosaccharides with 3D structures conforming to that of defined orientations.

Signature: *string ExtractOntologyInfoByGlycanStructure(string ring_type, string anomeric_orientation, string minus_one_orientation, string index_two_orientation, string index_three_orientation, string index_four_orientation = "", plus_one_orientation = "")*

Sample Usage: *ExtractOntologyInfoByGlycanStructure("P","Up", "", "Up", "Up", "Down", "Down")*

The generated sample query contains a large number of triples. It can be found in Appendix F.

Query 8: This query searches the ontology based on the given derivative/modification map. Users can create a list of key values such that the key is the index of the carbon atom in the ring or side carbon atom, and the value is the pattern that is attached to that index. The generated query searches for the glycans with the same identified derivative/modification patterns.

Signature: *string ExtractOntologyInfoByDerivativeModificationMap(string ring_type, DerivativeModificationMap derivative_modification_map)*

Sample Usage: derivative_modification_map["2"] = “xC-N-C=OCH3"

ExtractOntologyInfoByDerivativeModificationMap("P", derivative_modification_map)

Generated Sample Query

```
SELECT ?pdb ?stereo_name ?stereo_condensed_name ?condensed_name WHERE {
?mono          :hasRingAtom    ?ring_atom.
?ring_atom     :ringIndex      "2".
?ring_atom     :hasSideAtom    ?side.
?side          :sideIndex      "2".
?side          :derivative     "xC-N-C=OCH3".
?mono          :hasSugarName   ?sn.
?sn            :monosaccharideName       ?name.
?sn            :monosaccharideShortName  ?condensed_name.
?sn            :monosaccharideStereochemName     ?stereo_name.
?sn            :monosaccharideStereochemShortName       ?stereo_condensed_name.
?pdb_file      :hasOligo       ?oligo.
?oligo         :hasCore  ?mono.
?pdb_file      :identifier     ?pdb.
}
```

Query 9: This query is one of the most complicated queries and it can have a lot of triple patterns. The query is generated based on the list of monosaccharide structures specified by their side atom orientations. The defined structures are expected to have linkages to each other. Basically, the user defines 3D structures of monosaccharides and looks for oligosaccharides that contain those monosaccharides such that they are attached to one another.

Signature: *string ExtractOntologyInfoByAttachedGlycanStructures(*

AttachedGlycanStructuresVector attached_structures)

Sample Usage:

```
structure1 = gmml.string_vector()
structure1.push_back("P");
structure1.push_back("Up");
structure1.push_back("");
structure1.push_back("Down");
structure1.push_back("Up");
structure1.push_back("Down");
structure1.push_back("Up");

structure2 = gmml.string_vector()
structure2.push_back("P");
structure2.push_back("Up");
structure2.push_back("");
structure2.push_back("Down");
structure2.push_back("Up");
structure2.push_back("Down");
structure2.push_back("Up");

structures = gmml.dihedral_vector()
structures.push_back(structure1)
structures.push_back(structure2)
ExtractOntologyInfoByAttachedGlycanStructures(structures)
```

The generated sample query contains a large number of triples. It can be found in Appendix F.

CHAPTER 5

RESULTS

In this research, two different machines have been used. First machine which is called CARROLL, which is a server located at the Complex Carbohydrate Research Center of the University of Georgia. The second machine is at the computer science department at the University, and is called MITHRA. The operating system distribution installed on CARROLL is Ubuntu 14.04 and on MITHRA, CentOS 5.10. Table 3 shows the CPU information for each machine.

Table 3: CPU Information of Machines

CARROLL		MITRHRA	
No. of CPU(s):	**2**	**No. of CPU(s):**	**1**
Core(s) per CPU:	**6**	**Core(s) per CPU:**	**4**
Thread(s) per core:	**2**	**Thread(s) per core:**	**2**
CPU clock rate (GHz):	**2.93**	**CPU clock rate (GHz):**	**3.4**
CPU type:	**Intel(R) Xeon(R)**	**CPU type:**	**Intel(R) Core(TM)**

5.1. Ontology Population Statistics

As discussed in Chapter 4 section 4.3.3., three methods have been used to build the structure of the molecule based on the distance of atoms to each other. On both

machines experiments have been run to analyze the performance and to select the best way to process all the PDB files in the repository. Table 4 shows an experiment on CARROLL. It shows the time it takes to build the molecular structure using the three thread-based methods of the system on a PDB file with ID “1RVZ”. This PDB file is one of the largest PDBs; it contains more than 25,000 atoms.

Table 4: Time of Building the Structure of 1RVZ.pdb on CARROLL

CARROLL	**Time of Building Structure By Distance (sec)**		
Running Attempts	**Method 1** **13 Threads:** **different workloads**	**Method 2** **13 Threads:** **roughly equal workloads**	**Method 3** **13 Threads:** **Matrix(5x5)**
Attempt 1	11.15	9.91	9.07
Attempt 2	11.77	9.86	8.28
Attempt 3	12.75	10.80	8.70
Attempt 4	12.20	10.60	7.94
Attempt 5	11.36	10.13	8.6
Average	11.84	10.26	8.51

Table 5 shows the approximate worst case time for building the structure of 100,000 PDB files. However, in reality not all the PDB files are as big as 1RVZ.pdb.

Table 5: Worst Case Time for Building the Structure of 100,000 PDB Files on CARROLL

Time of Building Structure By Distance (days)		
Method 1 13 Threads: different workloads	**Method 2 13 Threads: roughly equal workloads**	**Method 3 13 Threads: Matrix(5x5)**
13.71	11.87	9.84

Table 6 and 7 represent the same experiment on MITHRA but using less number of threads since it has eight threads in total.

Table 6: Time of Building the Structure of 1RVZ.pdb on MITHRA

MITHRA	**Time of Building Structure By Distance (sec)**		
Running Attempts	**Method 1 5 Threads: different workloads**	**Method 2 5 Threads: roughly equal workloads**	**Method 3 5 Threads: Matrix(3x3)**
Attempt 1	28.87	20.81	20.55
Attempt 2	27.60	22.93	21.01
Attempt 3	28.83	22.04	21.78

Attempt 4	28.80	22.81	23.16
Attempt 5	27.16	22.06	20.59
Average	28.25	22.13	21.41

Table 7: Worst Case Time for Building the Structure of 100,000 PDB Files on MITHRA

Time of Building Structure By Distance (days)		
Method 1 **5 Threads:** **different workloads**	**Method 2** **5 Threads:** **roughly equal workloads**	**Method 3** **5 Threads:** **Matrix(3x3)**
32.69	25.61	24.78

In order to create an ontology only for testing purposes and to experiment with, both machines have been used simultaneously to run the Sugar ID and bulk population processes to get the results in the shortest time. RCSB's PDB repository has been cloned in the local system in order to eliminate the latency of processing files over the network. PDB files are organized into directories called "00" through "ZZ". Method #2 of the building structure has been used on CARROLL with 3 threads and method #2 on MITHRA with 5 threads. CARROLL is a server with multiple clients. Therefore, only 3 threads have been used to retain sufficient resources for other users of the server. Table 8

shows the detailed information about the first time of running Sugar ID and bulk population processes on all PDB files. The “Begin/End” column represents the starting and ending directories of the PDB repository for a particular run on each machine.

Table 8: Bulk Ontology Population, First Run

#	Machine	Start Date	End Date	Begin/ End	Processed/ Total	Ontology/Size
1	Mithra	11/09/2015 13:59	11/10/2015 21:48	00/C9	9634/12284	mithra-1-00c9-gmmo.ttl.gz/3GB
2	Carroll	11/16/2015 18:17	11/17/2015 10:33	X0/ZZ	9585/10128	carroll-2-x0zz-gmmo.ttl.gz/6.9GB
3	Carroll	11/20/2015 19:29	11/21/2015 12:37	U0/WZ	11039/11573	carroll-3-u0wz-gmmo.ttl.gz/8.4GB
4	Mithra	11/23/2015 11:50	11/24/2015 20:51	CA/EZ	10728/13536	mithra-4-caez-gmmo.ttl.gz/3.2GB
5	Carroll	11/23/2015 13:20	11/24/2015 10:52	Q0/TZ	14039/14730	carroll-5-q0tz-gmmo.ttl.gz/11GB
6	Mithra	11/25/2015 10:00	11/26/2015 13:15	F0/H9	8559/10837	mithra-6-f0h9-gmmo.ttl.gz/2.7GB

7	Carroll	11/25/2015 10:38	11/26/2015 06:30	N0/PZ	13160/13748	carroll-7-n0pz-gmmo.ttl.gz/9.5GB
8	Carroll	11,26,2015 11:32	11/27/2015 02:44	K0/MZ	13618/14078	carroll-8-k0mz-gmmo.ttl.gz/7.6GB
9	Mithra	11/30,2015 09:07	12/01.2015 16:06	HA/JZ	9697/12232	mithra-9-hajz-gmmo.ttl.gz/2.8GB

The following are the more general information about the first populated ontology:

Total running time = 213h.4m = 8.87 days

Total size of the populated ontology = 55.1 GB

Total number of files = 113,146

Total number of processed files = 100,059

Total number of unprocessed files = 13,087

By experimenting with the first populated ontology, some bugs have been identified in the generated triples and corrected. Also, queries have been improved significantly in terms of performance. There are two main reasons that an input PDB file may remain unprocessed. Firstly, the customized bash script that has been written for the purpose of bulk population has a process time limit for each file. If processing a PDB file takes more than a specific threshold the script kills the process, logs the file status as “unprocessed” and moves on to the next file. Secondly, the error might be due to the format

incompatibility of the input file with either the PDB file reader/parser or the Sugar ID process.

For the second ontology population run, the local PDB repository has been updated to the latest RCSB repository to include the most recent PDB files. The process time limit of the bulk population script has been increased to allow the system to process bigger molecules in the PDB files as well. As Table 9 shows, CARROLL is the only machine that has been used for this experiment using method #2 and with 4 threads.

Table 9: Bulk Ontology Population, Second Run

#	Machine	Start Date	End Date	Begin /End	Processed /Total	Ontology/Size
1	Carroll	01/15/2016 09:18	01/23/2016 05:33	00/ZZ	109707/114388	carroll-1-00zz-gmmo.ttl.gz/79GB

The following are the more general information about the second populated ontology:

Total running time = 200h.15m = 8.34 days

Total size of the populated ontology = 79 GB

Total number of files = 114,388

Total number of processed files = 109,707

Total number of unprocessed files = 4,681

Total number of processed PDB files containing sugar = 25,011

5.2. Experiments on the Queries

Table 10 shows the running time of all nine queries on the second populated ontology.

Table 10: Running Time of Queries on Second Populated Ontology

Dynamic Template Query	Running Time (seconds)					AVG (sec.)
Query 1	2.04	1.99	1.96	1.98	1.97	1.98
Query 2	0.19	0.18	0.18	0.17	0.17	0.17
Query 3	0.09	0.09	0.09	0.10	0.10	0.09
Query 4	0.58	0.58	0.58	0.57	0.56	0.57
Query 5	0.12	0.12	0.13	0.12	0.11	0.12
Query 6	0.18	0.16	0.16	0.15	0.15	0.16
Query 7	1.72	1.72	1.72	1.73	1.73	1.72
Query 8	0.89	0.88	0.86	0.89	0.87	0.89
Query 9	2.29	2.27	2.25	2.26	2.26	2.26

The result indicates that the Virtuoso shows a high performance despite the size of the data that it is operating on (79GB).

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The system we have is able to provide the means for specialists of the field to curate or compare the new structures that they have modeled with the rich information that comes from PDB files in a quick efficient way. This system has a collection of essential information about sugars stored as an ontology in a consistent fashion. This information can be used in various studies related to mortal diseases that involve carbohydrates. The system offers a common vocabulary to be used by the domain experts which helps to remove the ambiguity and inconsistency in communications and studies. Knowledge management is another powerful feature that comes with this system. This system can be used as a web service, or independently, as a tool, by using the implemented library and the populated ontology.

Moreover, by decreasing hurdles to studying glycobiology, this system facilitates future glycobiology research to be conducted by non-glycobiologists using an easy to use platform.

Automating the process of detecting new PDB files added to RCSB repository, processing them, adding their information to the ontology and designing more queries, as the demand increases, will remain as future works. Also, post-processing the query results and extracting statistical information about common sugar structures can be used to check the sanity of the new structures or compare the existing ones with one another.

REFERENCES

1. Berman, Helen M., et al. "The protein data bank." *Nucleic acids research*28.1 (2000): 235-242.
2. http://www.rcsb.org/pdb/home/home.do.
3. https://en.wikipedia.org/wiki/Glycosidic_bond.
4. https://en.wikipedia.org/wiki/Glycan.
5. T. R. Gruber. A Translation Approach to Portable Ontologies. Knowledge Acquisition, 5(2):199–220, 1993.
6. https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.
7. Berners-Lee, T., J. Hendler, and O. Lassila, *The semantic web.* Scientific american, 2001.284(5): p. 28-37.
8. https://en.wikipedia.org/wiki/World_Wide_Web_Consortium.
9. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/rdf-concepts/.
10. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/owl-ref/.
11. https://en.wikipedia.org/wiki/Resource_Description_Framework.
12. https://www.w3.org/2001/sw/wiki/RDFS.

13. https://www.cambridgesemantics.com/semantic-university/rdfs-introduction.

14. https://en.wikipedia.org/wiki/SPARQL.

15. https://en.wikipedia.org/wiki/Virtuoso_Universal_Server.

16. http://wiki.dbpedia.org/.

17. http://rdf4j.org/.

18. https://jena.apache.org/documentation/tdb/.

19. Apache Software Foundation, available from http://jena.apache.org/.

20. http://www.json.org/.

21. https://en.wikipedia.org/wiki/Comma-separated_values.

22. https://www.w3.org/XML/.

23. http://protege.stanford.edu/.

24. Beazley, David M. "SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++." *Tcl/Tk Workshop*. 1996.

25. https://en.wikipedia.org/wiki/Interface_description_language.

26. Object Management Group. *The Common Object Request Broker (CORBA): Architecture and Specification*. Object Management Group, 1995.

27. Williams, Sara, and Charlie Kindel. "The component object model: A technical overview." *Dr. Dobbs Journal* 356 (1994): 356-375.

28. Hanser, Th, Ph Jauffret, and Gérard Kaufmann. "A new algorithm for exhaustive ring perception in a molecular graph." *Journal of Chemical Information and Computer Sciences* 36.6 (1996): 1146-1152.

29. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms, Third Edition 2009: p. 549-611.

30. Curl open software. Available from https://curl.haxx.se/.

31. Lachele B. Foley, http://glycam.org/docs/gmml/2016/03/31/glycode-internal-monosaccharide-representation/.

32. Lachele B. Foley, https://www.ccrc.uga.edu/personnel/index.php?class=60&personnel=Non.

33. Booch, Grady. *The unified modeling language user guide*. Pearson Education India, 2005.

34. Python Software Foundation. Available at http://www.python.org.

35. https://www.w3.org/TR/turtle/

APPENDICES

Appendix A

GMMO Schema

```
@prefix : <http://www.semanticweb.org/owl/owlapi/turtle#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix gmmo: <http://gmmo.uga.edu/#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://gmmo.uga.edu/> .
<http://gmmo.uga.edu/> rdf:type owl:Ontology .
#################################################################
#
#    Annotation properties
#
#################################################################
###  http://gmmo.uga.edu/#description
gmmo:description rdf:type owl:AnnotationProperty ;
                 rdfs:domain gmmo:Note .
#################################################################
#
#    Object Properties
#
#################################################################
###  http://gmmo.uga.edu/#hasAtom
gmmo:hasAtom rdf:type owl:ObjectProperty ;
             rdfs:range gmmo:Atom ;
             rdfs:domain gmmo:Residue .
###  http://gmmo.uga.edu/#hasChild
```

```
gmmo:hasChild rdf:type owl:ObjectProperty ;
          rdfs:domain gmmo:Linkage ;
          rdfs:range gmmo:Oligosaccharide .
###  http://gmmo.uga.edu/#hasChildAtomLinkage
gmmo:hasChildAtomLinkage rdf:type owl:ObjectProperty ;
                  rdfs:range gmmo:Atom ;
                  rdfs:domain gmmo:Linkage .
###  http://gmmo.uga.edu/#hasCore
gmmo:hasCore rdf:type owl:ObjectProperty ;
          rdfs:range gmmo:Monosaccharide ;
          rdfs:domain gmmo:Oligosaccharide .
###  http://gmmo.uga.edu/#hasGlycosidicLinkage
gmmo:hasGlycosidicLinkage rdf:type owl:ObjectProperty ;
                  rdfs:range gmmo:Atom ;
                  rdfs:domain gmmo:Linkage .
###  http://gmmo.uga.edu/#hasNeighbor
gmmo:hasNeighbor rdf:type owl:ObjectProperty ,
                  owl:SymmetricProperty ;
             rdfs:domain gmmo:Atom ;
             rdfs:range gmmo:Atom .
###  http://gmmo.uga.edu/#hasNote
gmmo:hasNote rdf:type owl:ObjectProperty ;
          rdfs:domain gmmo:PDB ;
          rdfs:range :Note .
###  http://gmmo.uga.edu/#hasOligo
gmmo:hasOligo rdf:type owl:ObjectProperty ;
          rdfs:range gmmo:Oligosaccharide ;
          rdfs:domain gmmo:PDB .
###  http://gmmo.uga.edu/#hasParent
gmmo:hasParent rdf:type owl:ObjectProperty ;
          rdfs:domain gmmo:Linkage ;
          rdfs:range gmmo:Oligosaccharide .
```

```
### http://gmmo.uga.edu/#hasParentAtomLinkage
gmmo:hasParentAtomLinkage rdf:type owl:ObjectProperty ;
                    rdfs:range gmmo:Atom ;
                    rdfs:domain gmmo:Linkage .
### http://gmmo.uga.edu/#hasResidue
gmmo:hasResidue rdf:type owl:ObjectProperty ;
            rdfs:domain gmmo:PDB ;
            rdfs:range gmmo:Residue .
### http://gmmo.uga.edu/#hasRingAtom
gmmo:hasRingAtom rdf:type owl:ObjectProperty ;
             rdfs:domain gmmo:Monosaccharide ;
             rdfs:range gmmo:RingAtom .
### http://gmmo.uga.edu/#hasSideAtom
gmmo:hasSideAtom rdf:type owl:ObjectProperty ;
             rdfs:range gmmo:SideAtom ;
             rdfs:domain [ rdf:type owl:Class ;
                      owl:unionOf ( gmmo:RingAtom
                                    gmmo:SideAtom
                                  )
                     ] .
### http://gmmo.uga.edu/#hasSugarName
gmmo:hasSugarName rdf:type owl:ObjectProperty ;
              rdfs:domain gmmo:Monosaccharide ;
              rdfs:range gmmo:SugarName .
#################################################################
#
#    Data properties
#
#################################################################
### http://gmmo.uga.edu/#anomericStatus
gmmo:anomericStatus rdf:type owl:DatatypeProperty ;
                rdfs:domain gmmo:Monosaccharide .
```

```
### http://gmmo.uga.edu/#configuration
gmmo:configuration rdf:type owl:DatatypeProperty ;
                   rdfs:domain gmmo:SugarName .
### http://gmmo.uga.edu/#coordinate
gmmo:coordinate rdf:type owl:DatatypeProperty ;
                rdfs:domain gmmo:Atom .
### http://gmmo.uga.edu/#derivative_modification
gmmo:derivative_modification rdf:type owl:DatatypeProperty ;
                             rdfs:domain gmmo:SideAtom .
### http://gmmo.uga.edu/#filePath
gmmo:filePath rdf:type owl:DatatypeProperty ;
              rdfs:domain gmmo:PDB .
### http://gmmo.uga.edu/#glycosidicLinkage
gmmo:glycosidicLinkage rdf:type owl:DatatypeProperty ;
                       rdfs:domain gmmo:Linkage .
### http://gmmo.uga.edu/#identifier
gmmo:identifier rdf:type owl:DatatypeProperty ;
                rdfs:domain gmmo:Atom ,
                            gmmo:Monosaccharide ,
                            gmmo:PDB ,
                            gmmo:Residue .
### http://gmmo.uga.edu/#isomer
gmmo:isomer rdf:type owl:DatatypeProperty ;
            rdfs:domain gmmo:SugarName .
### http://gmmo.uga.edu/#monosaccharideName
gmmo:monosaccharideName rdf:type owl:DatatypeProperty ;
                        rdfs:domain gmmo:SugarName .
### http://gmmo.uga.edu/#monosaccharideShortName
gmmo:monosaccharideShortName rdf:type owl:DatatypeProperty ;
                             rdfs:domain gmmo:SugarName .
### http://gmmo.uga.edu/#monosaccharideStereochemName
gmmo:monosaccharideStereochemName rdf:type owl:DatatypeProperty ;
```

```
                        rdfs:domain gmmo:SugarName .
###  http://gmmo.uga.edu/#monosaccharideStereochemShortName
gmmo:monosaccharideStereochemShortName rdf:type owl:DatatypeProperty ;
                              rdfs:domain gmmo:SugarName .
###  http://gmmo.uga.edu/#oligoName
gmmo:oligoName rdf:type owl:DatatypeProperty ;
            rdfs:domain gmmo:Oligosaccharide .
###  http://gmmo.uga.edu/#orientation
gmmo:orientation rdf:type owl:DatatypeProperty ;
             rdfs:domain gmmo:SideAtom .
###  http://gmmo.uga.edu/#ringAtoms
gmmo:ringAtoms rdf:type owl:DatatypeProperty ;
            rdfs:domain gmmo:Monosaccharide .
###  http://gmmo.uga.edu/#ringIndex
gmmo:ringIndex rdf:type owl:DatatypeProperty ;
            rdfs:domain gmmo:RingAtom .
###  http://gmmo.uga.edu/#ringType
gmmo:ringType rdf:type owl:DatatypeProperty ;
           rdfs:domain gmmo:SugarName .
###  http://gmmo.uga.edu/#sideIndex
gmmo:sideIndex rdf:type owl:DatatypeProperty ;
            rdfs:domain gmmo:SideAtom .
###  http://gmmo.uga.edu/#stereochemistryChemicalCode
gmmo:stereochemistryChemicalCode rdf:type owl:DatatypeProperty ;
                        rdfs:domain gmmo:Monosaccharide .
###  http://www.semanticweb.org/owl/owlapi/turtle#BFMPRingConfomration
:BFMPRingConfomration rdf:type owl:DatatypeProperty ;
                 rdfs:domain gmmo:Monosaccharide .
###  http://www.semanticweb.org/owl/owlapi/turtle#description
:description rdf:type owl:DatatypeProperty ;
          rdfs:domain :Note .
###  http://www.semanticweb.org/owl/owlapi/turtle#noteCategory
```

```
:noteCategory rdf:type owl:DatatypeProperty ;
        rdfs:domain :Note ;
        rdfs:range [ rdf:type rdfs:Datatype ;
                owl:oneOf [ rdf:type rdf:List ;
                        rdf:first "Anomeric" ;
                        rdf:rest [ rdf:type rdf:List ;
                                rdf:first "GlycosidicLinkage" ;
                                rdf:rest [ rdf:type rdf:List ;
                                        rdf:first "ModificationDerivative" ;
                                        rdf:rest [ rdf:type rdf:List ;
                                                rdf:first "Monosaccharide" ;
                                                rdf:rest [ rdf:type rdf:List ;
                                                        rdf:first "ResidueName" ;
                                                        rdf:rest rdf:nil
                                                      ]
                                              ]
                                      ]
                              ]
                      ]
                ] .
###  http://www.semanticweb.org/owl/owlapi/turtle#noteType
:noteType rdf:type owl:DatatypeProperty ;
        rdfs:domain :Note ;
        rdfs:range [ rdf:type rdfs:Datatype ;
                owl:oneOf [ rdf:type rdf:List ;
                        rdf:first "comment" ;
                        rdf:rest [ rdf:type rdf:List ;
                                rdf:first "error" ;
                                rdf:rest [ rdf:type rdf:List ;
                                        rdf:first "warning" ;
                                        rdf:rest rdf:nil
                                      ]
```

```
                                ]
                            ]
                    ] .
### http://www.semanticweb.org/owl/owlapi/turtle#oligoResidueLinkages
:oligoResidueLinkages rdf:type owl:DatatypeProperty ;
                    rdfs:domain gmmo:Oligosaccharide .
#################################################################
#
#    Classes
#
#################################################################
### http://gmmo.uga.edu/#Atom
gmmo:Atom rdf:type owl:Class .
### http://gmmo.uga.edu/#Linkage
gmmo:Linkage rdf:type owl:Class .
### http://gmmo.uga.edu/#Monosaccharide
gmmo:Monosaccharide rdf:type owl:Class .
### http://gmmo.uga.edu/#Oligosaccharide
gmmo:Oligosaccharide rdf:type owl:Class .
### http://gmmo.uga.edu/#PDB
gmmo:PDB rdf:type owl:Class .
### http://gmmo.uga.edu/#Residue
gmmo:Residue rdf:type owl:Class .
### http://gmmo.uga.edu/#RingAtom
gmmo:RingAtom rdf:type owl:Class ;
            rdfs:subClassOf gmmo:Atom .
### http://gmmo.uga.edu/#SideAtom
gmmo:SideAtom rdf:type owl:Class ;
            rdfs:subClassOf gmmo:Atom .
### http://gmmo.uga.edu/#SugarName
gmmo:SugarName rdf:type owl:Class .
### http://www.semanticweb.org/owl/owlapi/turtle#Note
```

:Note rdf:type owl:Class .

Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net

Appendix B

Monosaccharide Naming Lookup Table [32]

Code	Long Name	Condensed Name
# Alpha D Aldohexapyranoses		
_2_3_4P_a	a-D-ribopyranose	DRibpa
^2_3_4P_a	a-D-arabinopyranose	DArapa
_2^3_4P_a	a-D-xylopyranose	DXylpa
^2^3_4P_a	a-D-lyxopyranose	DLyxpa
_2_3_4P^+1_a	a-D-allopyranose	DAllpa
^2_3_4P^+1_a	a-D-altropyranose	DAltpa
_2^3_4P^+1_a	a-D-glucopyranose	DGlcpa
^2^3_4P^+1_a	a-D-mannopyranose	DManpa
_2_3^4P^+1_a	a-D-gulopyranose	DGulpa
^2_3^4P^+1_a	a-D-idopyranose	DIdopa
_2^3^4P^+1_a	a-D-galactopyranose	DGalpa
^2^3^4P^+1_a	a-D-talopyranose	DTalpa
# Beta D Aldohexapyranoses		
_2_3_4P^a	b-D-ribopyranose	DRibpb
^2_3_4P^a	b-D-arabinopyranose	DArapb
_2^3_4P^a	b-D-xylopyranose	DXylpb
^2^3_4P_a	b-D-lyxopyranose	DLyxpb
_2_3_4P^+1^a	b-D-allopyranose	DAllpb
^2_3_4P^+1^a	b-D-altropyranose	DAltpb
_2^3_4P^+1^a	b-D-glucopyranose	DGlcpb
^2^3_4P^+1^a	b-D-mannopyranose	DManpb
_2_3^4P^+1^a	b-D-gulopyranose	DGulpb
^2_3^4P^+1^a	b-D-idopyranose	DIdopb
_2^3^4P^+1^a	b-D-galactopyranose	DGalpb
^2^3^4P^+1^a	b-D-talopyranose	DTalpb
# Alpha D Ketohexapyranoses		
_2_3_4P^-1_a	a-D-psicopyranose	DPsipa
^2_3_4P^-1_a	a-D-fructopyranose	DFrupa

_2^3_4P^-1_a	a-D-sorbopyranose	DSorpa
^2^3_4P^-1_a	a-D-tagatopyranose	DTagpa
# Beta D Ketohexapyranoses		
_2_3_4P_-1^a	b-D-psicopyranose	DPsipb
^2_3_4P_-1^a	b-D-fructopyranose	DFrupb
_2^3_4P_-1^a	b-D-sorbopyranose	DSorpb
^2^3_4P_-1^a	b-D-tagatopyranose	DTagpb
# D aldotetrafuranoses		
_2_3F_a	a-D-erythrofuranose	DEryfa
^2_3F_a	a-D-threofuranose	DThrfa
_2_3F^a	b-D-erythrofuranose	DEryfb
^2_3F^a	b-D-threofuranose	DThrfb
# D aldopentafuranoses		
_2_3F^+1_a	a-D-ribofuranose	DRibfa
^2_3F^+1_a	a-D-arabinofuranose	DArafa
_2^3F^+1_a	a-D-xylofuranose	DXylfa
^2^3F^+1_a	a-D-lyxofuranose	DLyxfa
_2_3F^+1^a	b-D-ribofuranose	DRibfb
^2_3F^+1^a	b-D-arabinofuranose	DArafb
_2^3F^+1^a	b-D-xylofuranose	DXylfb
^2^3F^+1^a	b-D-lyxofuranose	DLyxfb
# Alpha D aldohexafuranoses		
_2_3F^+1R^+2_a	a-D-allofuranose	DAllfa
^2_3F^+1R^+2_a	a-D-altrofuranose	DAltfa
_2^3F^+1R^+2_a	a-D-glucofuranose	DGlcfa
^2^3F^+1R^+2_a	a-D-mannofuranose	DManfa
_2_3F_+1R_+2_a	a-D-gulofuranose	DGulfa
^2_3F_+1R_+2_a	a-D-idofuranose	DIdofa
_2^3F_+1R_+2_a	a-D-galactofuranose	DGalfa
^2^3F_+1R_+2_a	a-D-talofuranose	DTalfa
# Beta D aldohexafuranoses		
_2_3F^+1R^+2^a	b-D-allofuranose	DAllfb
^2_3F^+1R^+2^a	b-D-altrofuranose	DAltfb
_2^3F^+1R^+2^a	b-D-glucofuranose	DGlcfb
^2^3F^+1R^+2^a	b-D-mannofuranose	DManfb
_2_3F_+1R_+2^a	b-D-gulofuranose	DGulfb
^2_3F_+1R_+2^a	b-D-idofuranose	DIdofb
_2^3F_+1R_+2^a	b-D-galactofuranose	DGalfb

^2^3F_+1R_+2^a	b-D-talofuranose	DTalfb
# D ketopentafuranoses		
_2_3F^-1_a	a-D-ribulofuranose	DRulfa
^2_3F^-1_a	a-D-Xylulofuranose	DXulfa
_2_3F_-1^a	b-D-ribulofuranose	DRulfb
^2_3F_-1^a	b-D-Xylulofuranose	DXulfb
# D Ketohexafuranoses		
_2_3F^+1^-1_a	a-D-psicofuranose	DPsifa
^2_3F^+1^-1_a	a-D-fructofuranose	DFrufa
_2^3F^+1^-1_a	a-D-sorbofuranose	DSorfa
^2^3F^+1^-1_a	a-D-tagatofuranose	DTagfa
_2_3F^+1_-1^a	b-D-psicofuranose	DPsifb
^2_3F^+1_-1^a	b-D-fructofuranose	DFrufb
_2^3F^+1_-1^a	b-D-sorbofuranose	DSorfb
^2^3F^+1_-1^a	b-D-tagatofuranose	DTagfb
# Other		
_2^3(4=5)P(+1A)_a		deltaUAa
_2^3(4=5)P(+1A)^a		deltaUAb

Derivatives and Modifications [32]:

Formula	Type	Long Name	Cond Name	Alt Abbrev Name	No Brackets	Warn Pos	Err Pos
xCH-N	Mod	-osamine	N		2(r:6&!-1)	a	5(r6), 4(r5)
xC-N-C=OCH3	Mod	N-acetyl-	NAc		2(r6&!-1)	a	5(r6), 4(r5)
xC-N-C=OCH2OH	Mod	N-glycolyl-	NGc	NGl	2(r6&!-1)	a	5(r6), 4(r5)
xC-N-SO3	Mod	N-sulfo-	NS		2(r6&!-1)	a	5(r6), 4(r5)
xC-N-PO3	Mod	N-phospho-	NP		2(r6&!-1)	a	5(r6),

							4(r5)
xC-N-CH3	Mod	N-methyl-	NMe		2(r6&!-1)	a	5(r6), 4(r5)
xC-(H,H)	Mod	deoxy-	?			a	5(r6), 4(r5)
+1C-(O,O)	Mod	-uronate	A		1		5(r6), 4(r5)
+1C-(O,OH)	Mod	-uronic acid	AH		1		5(r6), 4(r5)
-1C-(O,O)	Mod	-ulosonate	A			-1	5(r6), 4(r5)
-1C-(O,OH)	Mod	-ulosonic acid	AH			-1	5(r6), 4(r5)
rC-(O,O)	WARN						5(r6), 4(r5)
rC-(O.OH)	WARN						5(r6), 4(r5)
(>+1,<-1)C-(O,O)	Mod		A				eC++
(>+1,<-1) C-(O,OH)	Mod		AH				eC++
xC-O-C=OCH3	Der	acetyl-	Ac				5(r6), 4(r5)
xC-O-C=OCH2OH	Der	glycolyl-	Gc	Gl			5(r6), 4(r5)
xC-O-SO3	Der	sulfo-	S				5(r6), 4(r5)
xC-O-PO3	Der	phospho-	P				5(r6),

							4(r5)
xC-O-CH3	Der	methyl-	Me				5(r6), 4(r5)

Appendix C

Ontology Vocabulary Designed in GMML

```
#ifndef ONTOLOGYVOCABULARY_HPP
#define ONTOLOGYVOCABULARY_HPP

#include <string>

namespace Ontology
{
    const std::string ONT_DOMAIN = "<http://gmmo.uga.edu/#";
    const std::string ONT_PREFIX = "gmmo:";
    const std::string ENTITY_COMMENT = " ###  http://gmmo.uga.edu#";

    const std::string TYPE = "rdf:type";
    const std::string LABEL = "rdfs:label";

    /* Classes */
    const std::string Atom = "gmmo:Atom";
    const std::string RingAtom = "gmmo:RingAtom";
    const std::string SideAtom = "gmmo:SideAtom";
    const std::string Linkage = "gmmo:Linkage";
    const std::string Monosaccharide = "gmmo:Monosaccharide";
    const std::string Oligosaccharide = "gmmo:Oligosaccharide";
    const std::string Note = "gmmo:Note";
    const std::string PDB = "gmmo:PDB";
    const std::string Residue = "gmmo:Residue";
    const std::string SugarName = "gmmo:SugarName";

    /* Object Properties */
    const std::string hasAtom = "gmmo:hasAtom";
    const std::string hasChild = "gmmo:hasChild";
    const std::string hasChildAtomLinkage =
"gmmo:hasChildAtomLinkage";
    const std::string hasParentAtomLinkage =
"gmmo:hasParentAtomLinkage";
    const std::string hasGlycosidicLinkage =
"gmmo:hasGlycosidicLinkage";
    const std::string hasNeighbor = "gmmo:hasNeighbor";
    const std::string hasOligo = "gmmo:hasOligo";
    const std::string hasNote = "gmmo:hasNote";
```

```
    const std::string hasParent = "gmmo:hasParent";
    const std::string hasResidue = "gmmo:hasResidue";
    const std::string hasRingAtom = "gmmo:hasRingAtom";
    const std::string hasCore = "gmmo:hasCore";
    const std::string hasSideAtom = "gmmo:hasSideAtom";
    const std::string hasSugarName = "gmmo:hasSugarName";

    /* Datatype Properties */
    const std::string input_file_path = "gmmo:filePath";
    const std::string anomeric_status = "gmmo:anomericStatus";
    const std::string stereochemistry_chemical_code =
"gmmo:stereochemistryChemicalCode";
    const std::string configuration = "gmmo:configuration";
    const std::string ring_atoms = "gmmo:ringAtoms";
    const std::string derivative = "gmmo:derivative";
    const std::string glycosidic_linkage = "gmmo:glycosidicLinkage";
    const std::string id = "gmmo:identifier";
    const std::string isomer = "gmmo:isomer";
    const std::string mono_name = "gmmo:monosaccharideName";
    const std::string mono_short_name =
"gmmo:monosaccharideShortName";
    const std::string mono_stereo_name =
"gmmo:monosaccharideStereochemName";
    const std::string mono_stereo_short_name =
"gmmo:monosaccharideStereochemShortName";
    const std::string oligo_name = "gmmo:oligoName";
    const std::string oligo_residue_linkages =
"gmmo:oligoResidueLinks";
    const std::string note_type = "gmmo:NoteType";
    const std::string note_category = "gmmo:NoteCategory";
    const std::string note_description = "gmmo:description";
    const std::string orientation = "gmmo:orientation";
    const std::string path = "gmmo:path";
    const std::string ring_index = "gmmo:ringIndex";
    const std::string ring_type = "gmmo:ringType";
    const std::string side_index = "gmmo:sideIndex";
    const std::string x = "gmmo:xCoordinate";
    const std::string y = "gmmo:yCoordinate";
    const std::string z = "gmmo:zCoordinate";
    const std::string coordinate = "gmmo:coordinate";
    const std::string bfmp_ring_conformation =
"gmmo:BFMPRingConformation";

    /* SPARQL Query */
    const std::string PREFIX = "PREFIX :
<http://gmmo.uga.edu/#>\nPREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>\nPREFIX owl:
<http://www.w3.org/2002/07/owl#>\nPREFIX rdfs:
<http://www.w3.org/2000/01/rdf-schema#>\nPREFIX xsd:
<http://www.w3.org/2001/XMLSchema#>\n";
    const std::string SELECT_CLAUSE = "SELECT";
    const std::string WHERE_CLAUSE = "WHERE {\n";
    const std::string END_WHERE_CLAUSE = "}\n";
    const std::string OUTPUT_FORMAT = " \'Accept: text/csv\' "; /* "
```

```
\'Accept: text/csv\' ", " \'Accept: application/json\' " */
    const std::string DATA_STORE_ADDRESS =
"http://192.168.1.52:8890/sparql"; /*
"http://128.192.62.244:8890/sparql",
"http://192.168.1.52:8890/sparql" */
    const std::string CURL_PREFIX = "curl -g -H";
    const std::string QUERY_PREFIX = " --data-urlencode query=\'";
    const std::string QUERY_POSTFIX = "\'";
}

#endif // ONTOLOGYVOCABULARY_HPP
```

Appendix D

Ontology Population Code Snippet

The code snippet demonstrates a sample usage of implemented helper functions and the custom ontology vocabulary to generate the triples.

```
void Assembly::PopulateLinkage(stringstream& linkage_stream,
Oligosaccharide* oligo, string oligo_uri, string id_prefix, int&
link_id, vector<int>& visited_oligos)
{
    string linkage_resource = "";
    string linkage_uri = "";
    string child_oligo_resource = "";
    string child_oligo_uri = "";
    string child_atom_resource = "";
    string child_atom_uri = "";
    string glycosidic_atom_resource = "";
    string glycosidic_atom_uri = "";
    string parent_atom_resource = "";
    string parent_atom_uri = "";
    stringstream linkage str;
    stringstream glycosidic_linkage_str;

    visited_oligos.push_back(oligo->root_->mono_id);

    for(OligosaccharideVector::iterator it = oligo-
>child_oligos_.begin(); it != oligo->child_oligos_.end(); it++)
    {
        int index = distance(oligo->child_oligos_.begin(), it);
        Oligosaccharide* child_oligo = (*it);

        linkage_resource = CreateURIResource(gmml::OntLinkage,
link_id, id_prefix, "");
        linkage_uri = CreateURI(linkage_resource);
```

```
        AddTriple(linkage_uri, Ontology::TYPE, Ontology::Linkage,
linkage_stream);
        AddLiteral(linkage_uri, Ontology::LABEL, linkage_resource,
linkage_stream);

        link_id++;

        AddTriple(linkage_uri, Ontology::hasParent, oligo_uri,
linkage_stream);

        child_oligo_resource =
CreateURIResource(gmml::OntOligosaccharide, child_oligo->root_-
>mono_id, id_prefix, "");
        child_oligo_uri = CreateURI(child_oligo_resource);
        AddTriple(linkage_uri, Ontology::hasChild, child_oligo_uri,
linkage_stream);

        vector<string> linkage_tokens = gmml::Split(oligo-
>child_oligos_linkages_.at(index), "-");

        string parent_atom_id = linkage_tokens.at(0);
        string glycosidic_atom_id = linkage_tokens.at(1);
        string child_atom_id = linkage_tokens.at(2);

        child_atom_resource = CreateURIResource(gmml::OntAtom, 0,
id_prefix, child_atom_id);
        child_atom_uri = CreateURI(child_atom_resource);
        AddTriple(linkage_uri, Ontology::hasChildAtomLinkage,
child_atom_uri, linkage_stream);

        glycosidic_atom_resource = CreateURIResource(gmml::OntAtom,
0, id_prefix, glycosidic_atom_id);
        glycosidic_atom_uri = CreateURI(glycosidic_atom_resource);
        AddTriple(linkage_uri, Ontology::hasGlycosidicLinkage,
glycosidic_atom_uri, linkage_stream);

        parent_atom_resource = CreateURIResource(gmml::OntAtom, 0,
id_prefix, parent_atom_id);
        parent_atom_uri = CreateURI(parent_atom_resource);
        AddTriple(linkage_uri, Ontology::hasParentAtomLinkage,
parent_atom_uri, linkage_stream);

        vector<string> child_atom_id_tokens =
gmml::Split(child_atom_id, "_");

        if(child_atom_id_tokens.at(3).at(0) == gmml::BLANK_SPACE)
            linkage_str << child_atom_id_tokens.at(2) << "(" <<
child_atom_id_tokens.at(4) << ")" << child_atom_id_tokens.at(0);
        else
            linkage_str << child_atom_id_tokens.at(2) << "(" <<
child_atom_id_tokens.at(4) << "_" << child_atom_id_tokens.at(3) <<
")" << child_atom_id_tokens.at(0);

        std::vector<std::string> parent_atom_id_tokens =
```

```
gmml::Split(parent_atom_id, "_");
        if(parent_atom_id_tokens.at(3).at(0) == gmml::BLANK_SPACE)
            linkage_str << "-" << parent_atom_id_tokens.at(2) << "("
<< parent_atom_id_tokens.at(4) << ")"  <<
parent_atom_id_tokens.at(0);
        else
            linkage_str << "-" << parent_atom_id_tokens.at(2) << "("
<< parent_atom_id_tokens.at(4) <<  "_" <<
parent_atom_id_tokens.at(3) << ")"  << parent_atom_id_tokens.at(0);

        AddLiteral(linkage_uri, Ontology::linkage_str,
linkage_str.str(), linkage_stream);
        std::vector<std::string> glycosidic_atom_id_tokens =
gmml::Split(glycosidic_atom_id, "_");
        if(glycosidic_atom_id_tokens.at(3).at(0) ==
gmml::BLANK_SPACE)
            glycosidic_linkage_str <<
glycosidic_atom_id_tokens.at(2) << "(" <<
glycosidic_atom_id_tokens.at(4) << ")" <<
glycosidic_atom_id_tokens.at(0);
        else
            glycosidic_linkage_str <<
glycosidic_atom_id_tokens.at(2) << "(" <<
glycosidic_atom_id_tokens.at(4) << "_" <<
glycosidic_atom_id_tokens.at(3)
                        << ")"  << glycosidic_atom_id_tokens.at(0);
        AddLiteral(linkage_uri, Ontology::glycosidic_linkage_str,
glycosidic_linkage_str.str(), linkage_stream);
    }
}


void Assembly::AddTriple(string s, string p, string o, stringstream&
stream)
{
    stream << s << " " << p << " " << o << "." << endl;
}

void Assembly::AddLiteral(string s, string p, string o,
stringstream& stream)
{
    stream << s << " " << p << " \"" << o << "\"." << endl;
}

string Assembly::CreateURI(string uri_resource)
{
    stringstream uri;
    uri << Ontology::ONT_PREFIX << uri_resource;
    return uri.str();
}

string Assembly::CreateURIResource(gmml::URIType resource , int
number, string id_prefix, string id )
{
```

```
    stringstream uri_resource;
    switch(resource)
    {
        case gmml::OntPDB:
            uri_resource << (Split(this-
>GetSourceFile().substr(this->GetSourceFile().find_last_of('/') +
1), ".").at(0));
            break;
        case gmml::OntOligosaccharide:
            uri_resource << id_prefix << "oligo" << number;
            break;
        case gmml::OntNote:
            uri_resource << id_prefix << "note" << number;
            break;
        case gmml::OntLinkage:
            uri_resource << id_prefix << "link" << number;
            break;
        case gmml::OntMonosaccharide:
            uri_resource << id_prefix << "mono" << number;
            break;
        case gmml::OntSugarName:
            uri_resource << id_prefix << "mono" << number << "_sn";
            break;
        case gmml::OntAtom:
            replace( id.begin(), id.end(), '?', 'n'); // replace all
'?' with 'n'
            FindReplaceString(id, "\'", "q");
            FindReplaceString(id, ",", "c");
            FindReplaceString(id, "*", "s");
            uri_resource << id_prefix << id;
            break;
        case gmml::OntResidue:
            replace( id.begin(), id.end(), '?', 'n'); // replace all
'?' with 'n'
            uri_resource << id_prefix << id;
            break;
    }
    return uri_resource.str();
}
```

Appendix E

Bulk Population Bash Script

The script has been designed for bulk processing of PDB files in order to create the ontology and logging the status and steps.

```
count=0
total=0
success="/tmp/success.log"
dest="/home/delaram/Documents/repos/PDB/unprocessed-extract"
rdest="/home/delaram/Documents/repos/PDB/all-reports"
src="/home/delaram/Documents/repos/PDB/all"
unprocessed="/home/delaram/Documents/repos/PDB/all-
reports/unprocessed.log"
for entry in "$src"/*
do
  for file in "$entry"/*
  do
    total=$((total+1))
    cp "$file" "$dest"
    zipfile=$(echo "$file" | cut -c42-56)
    filename=$(echo "$zipfile" | cut -c4-7)
    zipdest=$(echo "$dest/$zipfile")
    entdest=$(echo "$dest/pdb$filename.ent")
    filename=$(echo "$filename" | tr '[:lower:]' '[:upper:]')
    filedest=$(echo "$dest/$filename.pdb")
    gunzip "$zipdest"
    mv "$entdest" "$filedest"
    repdest=$(echo "$rdest/$filename.txt")
    time=1
    python cycle.py -amino_libs
"gmml/dat/CurrentParams/leaprc.ff12SB_2014-04-
24/amino12.lib,gmml/dat/CurrentParams/leaprc.ff12SB_2014-04-
24/aminont12.lib,gmml/dat/CurrentParams/leaprc.ff12SB_2014-04-
24/aminoct12.lib,gmml/dat/CurrentParams/leaprc_GLYCAM_06j-1_2014-03-
14/GLYCAM_aminont_06j_12SB.lib,gmml/dat/CurrentParams/leaprc_GLYCAM_
06j-1_2014-03-
14/GLYCAM_aminoct_06j_12SB.lib,gmml/dat/CurrentParams/leaprc_GLYCAM_
06j-1_2014-03-14/GLYCAM_amino_06j_12SB.lib"  -pdb "$filedest" &>
"$repdest" & pid=$!
    while [ ! -f "$success" ]
    do
        time=$((time+1))
        if [ $time -gt 2400000 ];
        then
           break
        fi
     done
     if [ -f "$success" ];
     then
        count=$((count+1))
        echo "$count/$total, processed: $filedest  =========>
reported: $repdest"
        rm "$success"
        rm "$filedest"
```

```
    else
        echo "$filedest" >> "$unprocessed"
        gzip "$filedest"
    fi
    kill -9 $pid
  done
done
if [ -f "$success" ];
then
  rm "$success"
fi
echo "$count/$total file(s) processed!"
```

Appendix F

Sample Generated Queries

Sample generated query for query 7:

```
SELECT ?stereo_name ?stereo_condensed_name ?condensed_name ?name
?oligo_sequence ?residue_links ?glycosidic_linkage WHERE {
?pdb_file          :hasOligo      ?oligo.
?oligo             :hasCore       ?mono.
?mono              :hasRingAtom     ?anomeric.
?anomeric          :ringIndex         "1".
?anomeric          :hasSideAtom    ?a_side.
?a_side            :sideIndex      "1".
?a_side            :orientation       "Up".
?mono              :hasRingAtom     ?two.
?two               :ringIndex         "2".
?two               :hasSideAtom    ?two_side.
?two_side          :sideIndex      "2".
?two_side          :orientation       "Up".
?mono              :hasRingAtom     ?three.
?three             :ringIndex       "3".
?three             :hasSideAtom    ?three_side.
?three_side        :sideIndex      "3".
?three_side        :orientation    "Up".
```

```
?mono              :hasRingAtom      ?four.
?four              :ringIndex        "4".
?four              :hasSideAtom      ?four_side.
?four_side         :sideIndex        "4".
?four_side         :orientation      "Down".
?mono              :hasRingAtom      ?last_c.
?last_c            :ringIndex        "5".
?last_c            :hasSideAtom      ?plus_one.
?plus_one          :sideIndex        "+1".
?plus_one          :orientation      "Down".
?oligo             :oligoName        ?oligo_sequence.
?pdb_file          :identifier       ?pdb.
OPTIONAL { ?oligo       :oligoResidueLinks     ?residue_links.
?linkage           :hasParent        ?oligo.
?linkage           :glycosidicLinkage    ?glycosidic_linkage.}
?mono              :hasSugarName     ?sn.
?sn                :monosaccharideName       ?name.
?sn                :monosaccharideShortName  ?condensed_name.
?sn                :monosaccharideStereochemName     ?stereo_name.
?sn                :monosaccharideStereochemShortName        ?stereo_condensed_name.
}
```

Sample generated query for query 8:

```
SELECT ?pdb ?linkage_indices ?stereo_short_name0 ?short_name0
?stereo_short_name1 ?short_name1 ?oligo_sequence ?residue_links WHERE {
?oligo0                          :hasCore        ?mono0.
?mono0                           :hasRingAtom ?mono0_anomeric.
?mono0_anomeric                  :ringIndex      "1".
?mono0_anomeric                  :hasSideAtom    ?mono0_anomeric_side_atom .
?mono0_anomeric_side_atom        :sideIndex      "1".
?mono0_anomeric_side_atom        :orientation    "Up".
?mono0                           :hasRingAtom ?mono0_ring_atom2.
?mono0_ring_atom2                :ringIndex      "2".
?mono0_ring_atom2                :hasSideAtom    ?mono0_side_atom2 .
?mono0_side_atom2                :sideIndex      "2".
```

```
?mono0_side_atom2                :orientation    "Down".
?mono0                           :hasRingAtom ?mono0_ring_atom3.
?mono0_ring_atom3                :ringIndex      "3".
?mono0_ring_atom3                :hasSideAtom    ?mono0_side_atom3 .
?mono0_side_atom3                :sideIndex      "3".
?mono0_side_atom3                :orientation    "Up".
?mono0                           :hasRingAtom ?mono0_ring_atom4.
?mono0_ring_atom4                :ringIndex      "4".
?mono0_ring_atom4                :hasSideAtom    ?mono0_side_atom4 .
?mono0_side_atom4                :sideIndex      "4".
?mono0_side_atom4                :orientation    "Down".
?mono0                           :hasRingAtom    ?mono0_last_c.
?mono0_last_c                    :ringIndex      "5".
?mono0_last_c                    :hasSideAtom    ?mono0_last_c_side_atom.
?mono0_last_c_side_atom          :sideIndex      "+1".
?mono0_last_c_side_atom          :orientation  "Up".
?oligo1                          :hasCore        ?mono1.
?mono1                           :hasRingAtom ?mono1_anomeric.
?mono1_anomeric                  :ringIndex      "1".
?mono1_anomeric                  :hasSideAtom    ?mono1_anomeric_side_atom .
?mono1_anomeric_side_atom  :sideIndex      "1".
?mono1_anomeric_side_atom  :orientation    "Up".
?mono1                           :hasRingAtom ?mono1_ring_atom2.
?mono1_ring_atom2                :ringIndex      "2".
?mono1_ring_atom2                :hasSideAtom    ?mono1_side_atom2 .
?mono1_side_atom2                :sideIndex      "2".
?mono1_side_atom2                :orientation    "Down".
?mono1                           :hasRingAtom ?mono1_ring_atom3.
?mono1_ring_atom3                :ringIndex      "3".
?mono1_ring_atom3                :hasSideAtom    ?mono1_side_atom3 .
?mono1_side_atom3                :sideIndex      "3".
?mono1_side_atom3                :orientation    "Up".
?mono1                           :hasRingAtom ?mono1_ring_atom4.
?mono1_ring_atom4                :ringIndex      "4".
?mono1_ring_atom4                :hasSid eAtom    ?mono1_side_atom4 .
?mono1_side_atom4                :sideIndex      "4".
?mono1_side_atom4                :orientation    "Down".
?mono1                           :hasRingAtom    ?mono1_last_c.
?mono1_last_c                    :ringIndex      "5".
```

```
?mono1_last_c                  :hasSideAtom    ?mono1_last_c_side_atom.
?mono1_last_c_side_atom    :sideIndex      "+1".
?mono1_last_c_side_atom    :orientation  "Up".
{
?linkage0                      :hasParent ?oligo0.
?linkage0                      :hasChild ?oligo1.
?oligo0                        :oligoResidueLinks ?residue_links.
OPTIONAL {?oligo0  :oligoName ?oligo_sequence}
} UNION {
?linkage0                      :hasParent ?oligo1.
?linkage0                      :hasChild ?oligo0.
?oligo1                        :oligoResidueLinks ?residue_links.
OPTIONAL {?oligo1  :oligoName ?oligo_sequence}
}
?linkage0                      :linkageIndeces ?linkage_indices.
?mono0                         :hasSugarName ?sn0.
?sn0                           :monosaccharideShortName ?short_name0.
?sn0                           :monosaccharideStereochemShortName
?stereo_short_name0.
?mono1                         :hasSugarName ?sn1.
?sn1                           :monosaccharideShortName ?short_name1.
?sn1                           :monosaccharideStereochemShortName
?stereo_short_name1.
?pdb                           :hasOligo      ?oligo0.
?pdb                           :hasOligo      ?oligo1.
}
```