# Building a database to store Protein-Protein Interactions (PPI) in a rule based format

*Ayush Das*

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2019

# Abstract

The study of Protein-Protein interactions (PPI) involves the analysis and identification of complexes that may form under a variety of reaction conditions. These reactions were initially modeled as Ordinary Differential Equations (ODEs) [26], which is now progressing to a rule-based modeling approach. This is because the interacting biomolecules have the potential to interact in a myriad different ways. The number of possible post-translational modifications and complexes grow exponentially when considering the binary interactions within the reaction network. Using traditional methods like ODEs to model PPI requires large amounts of reaction specific details, and the chemical kinetics of the interactions within network requires explicit mention of the network conditions [10]. A rule-based model on the other hand comprises of, a set of rules where the network specification is implicit. These rules can specified using model specification languages like Kappa [15] or BioNetGen [17]. Software tools enable researchers to model these interactions using different objectives like deterministic or stochastic modeling. Hence, researchers in bioinformatics have spent tremendous efforts in collecting the Protein-Protein interaction rules and the purpose of this project is to create and load a database with the PPI rules stored in a rule-based format. This will enable researchers to readily access PPI rules which when fed to a simulator will enable study of the protein interactions and draw conclusions based on their observations.

## Acknowledgements

I would like to thank my supervisor Oksana Sorokina for her continued guidance and support during all stages of this project. The insightful feedback and guidance helped in improving the project to a large extent. I would also like to thank, Anatoly Sorokin and Douglas Armstrong for their insightful feedback and support during the course of this project.

Finally, I would like to thank my family for their support and guidance.

# Table of Contents

# Chapter 1

# Introduction

Protein is an important component of the cells in the human body. It is an important component of bones, muscles, cartilage and so on. Decades of research in the field of biology have produced a vast repository of knowledge on individual protein molecules. Examples of such knowledge base include UniProt [22]. However, in order to further explore the relationships of complex molecular species it is imperative to understand the interactions that take place between them and their governing rules.

As per [14] Protein-Protein Interactions (PPI) are defined as the physical contacts with molecular docking between the protein molecules that occur in a living organism or cell. PPI interactions play a vital role as they dictate cellular activities which are responsible for good health or diseases. Achieving an in-depth understanding of protein interactions will help researchers improve the existing quality of medicine and health care in general. According to [18] an important source for drug discovery is the study of PPI. This is also evident from the fact that, as per [19] in recent times the study of PPI has gained momentum for research in the field of anti-cancer therapy. All these facts suggest the importance of PPI and it's applications.

Known PPIs are stored in PPI repositories designed to be easily retrievable by the researchers based on the relevant search term. There have been several databases in the past that have tackled the problem of collecting the PPIs. Such databases are of different varieties, based on their method of organizing and structuring the data. These kinds of databases are covered in greater detail, in Chapter 2.

PPI interactions are used to model the dynamics of protein complexes in the cell. Initially they were modeled as Ordinary Differential Equations (ODEs) [26], which have now progressed to a rule-based approach due to their ease and succinctness of expression. Rule-based methods have several applications some of which are assessing

the druggability of proteins [27] and drug effect pathway analysis [20]. These will be dealt in greater detail, in the background section.

This project is aimed at creating a database for Protein-Protein interactions stored in the Kappa rule format [15]. The database would allow the PPI interactions to be retrieved based on certain conditions that are elaborated in Chapter 3. Rule based simulation of protein interaction can either be performed based on Stochastic Simulation Algorithm (SSA) or using Ordinary differential equations [11]. In this project feeding the Kappa rules, to a Kappa simulator will help in visualizing the interactions of protein molecules in the Kappa simulator(KaSim) [16]. KaSim is an implementation, of an algorithm called continuous time Monte-Carlo (CTMC), which is created for systems based on rules.

This work is divided into chapters and we present a brief summary of each of these chapters. Chapter 2 covers, the kinds of PPI database that exist in literature, followed by a description of the kappa rules which encapsulates their syntax and semantics. The application of rule-based methods are also further elaborated in Chapter 2. Chapter 3 covers, the work that has been undertaken. This section elucidates the methodology used to create the database, the python scripts used to extract the relevant information from the assimilation of data collected by researchers. This section also elaborates on the SQL stored procedures used to extract the PPI rules and the user interface for accessing those rules. Furthermore, this chapter elaborates on the steps for deploying the database scripts and the web application. This chapter also walks through the process of adding a new rule to the database, to enable future PPI additions. In Chapter 4, the validation pipeline for the data within the database is defined concisely. In this chapter we retrieve some of the PPI rules and validate the result set with the provided data. Chapter 5 presents the conclusion with future improvements and proposal for work that can be extended from the project.

# Chapter 2

# Background

Protein-Protein interactions play a vital role in the regular functioning of life processes. Hence the study of these interactions, plays a crucial role in improving our understanding of diseases and the life processes.

Historically, PPI interactions were modeled using ordinary differential equations (ODE) [26]. However, as per [10], this traditional approach of modeling PPI through ODE had several limitations due to the following reasons.

- The protein molecules can potentially interact in an exponential number of ways.
- Due to the exponential number of possibilities only large reaction networks can capture them.
- This is a problem because traditional approaches like ODE require explicit network specification.

This problem is overcome by the use of local rules where the network specification is implicit. As a result the specification of the model is concise. These rules can be specified using languages for model specification like Kappa [15] and BioNetGen[17]. Specialized software tools then enable researchers to visualize the PPI interactions and run the simulation of the model in stochastic or deterministic way.

In the following sub-sections we will first explore the kinds of Protein interaction databases, followed by a section that develops on the understanding of rule based specification of protein interactions. The latter is followed by a section that deals with the motivation of constructing the rule based database system and application of protein interaction database in biology.

## 2.1 Protein-Protein Interaction (PPI) Databases

As per [33], the kinds of protein-protein interaction databases can be divided into three types.

- **Pathway:** In such databases researchers and domain experts collect pathway information that are generally agreed upon by the scientific community. This information is manually curated and cover association with diseases, stoichiometry of reactions and so on. Due to the requirement of manual intervention and the aim to achieve a high accuracy, construction of such databases is a laborious process.
  Examples of such databases include KEGG [23] and Reactome [12].

- **Experimentally Verified:** Such databases contain an assimilation of the protein interaction rules that have been experimentally verified. In other words such databases contain experimentally observed (verified) PPI rules. The method of the rule organization and the amount of information carried varies from one database to another.
  Examples of such databases include IntAct [24] and BioGrid [31].

- **Experimentally Verified or Computationally predicted:** Such databases contain PPI rules that are either experimentally observed or are computationally predicted. These PPIs however, assume minor manual curation. Thus, the computationally obtained PPIs may contain false positives and hence, to improve the accuracy a confidence score is often attached to them. In addition to using computational methods to obtain PPI rules, Natural Language Processing and text mining methods are also used in order to extract PPI rules from research literature.
  Examples of such databases include STRING [32] and GeneMANIA [34].

While the three types of databases mentioned above, serve as the primary categorization of PPI databases, there also exists categorization of PPI databases based on diseases, organisms of particular kind and so on.

According to [33], there are over hundreds of databases that aim to collect and store protein interactions. However, none of these databases capture the complexity of biological systems in it's entirety. These kinds of details include - temporal dependencies, spatial dependencies, protein isoforms and so on.

## 2.2 Motivation for constructing the rule based database

As per [8], historically the protein interactions were used to simulte the association of the protein into the complexes through the method of deterministic models , based on differential equations. These models aimed to capture a myriad of information like post-translational, structural information an so on. Such a vast domain of knowledge was susceptible to errors or missing values, which cast a doubt upon their accuracy, as per [8]. Such models also had maintainability issues because besides being difficult to build, they were also difficult to be updated with the ever evolving knowledge base that dictate these protein interactions. The human understanding of protein interactions matures and evolves with time and further research. Hence it is imperative to have a method of expressing PPI rules and building PPI models that make it easy to be read, stored and retrieved. As per [8], kappa rules, used to express the PPI interactions have the ability to encode and express information about the protein molecules (agents) and the sites that take part in the reaction. These rule summarize the pre and post conditions of the protein interaction without venturing into a detailed version of their structural analysis. Hence the rule based format of expressing PPI interaction is more concise, the stochastic models built using Kappa rules enable the description of large protein complexes.

With this motivation in mind we have ventured to create a database of Protein-Protein interactions stored in a rule based format. The database created also has stored procedure routines that enable extraction of PPI rules based on certain conditions like agent name and domain name. The stored procedures are accessible via a web application built in django, with a user interface (UI), that enables the extracted rules to be printed or exported as CSV or Excel files. These files can be further analyzed and processed, fed to the kappa simulator [16] and generate visualizations of the protein interactions.

## 2.3 Understanding rule based specification of protein interactions

### 2.3.1 General Understanding of rule based languages

In rule based languages like kappa [15] and BioNetGen [17], the agent is a conceptualization of the protein molecule. The protein molecules (agents) connect to form site

graphs via the protein sites. The PPI rule as per the rule based format consists of a left hand denoted by $L_r$ side and a right hand side denoted by $R_r$. The $L_r$ and $R_r$ contain the site graphs which mention only the necessary sites for the protein interaction $L_r \rightarrow R_r$.

Furthermore, $\mathfrak{M}$ denotes the state of a system which is also called the reaction mixture. Each disconnected graph denotes one molecular species and the state of the system $\mathfrak{M}$, is a collection of disconnected graphs. The execution of a certain rule 'r' implies the replacement of the mixture matched to $L_r$, with $R_r$, as shown in the Figure 2.1. A model as per [8] a collection of rules. Reasoning at the level of rules helps to introduce a level of compactness essential for the succinct expression of the protein interaction.

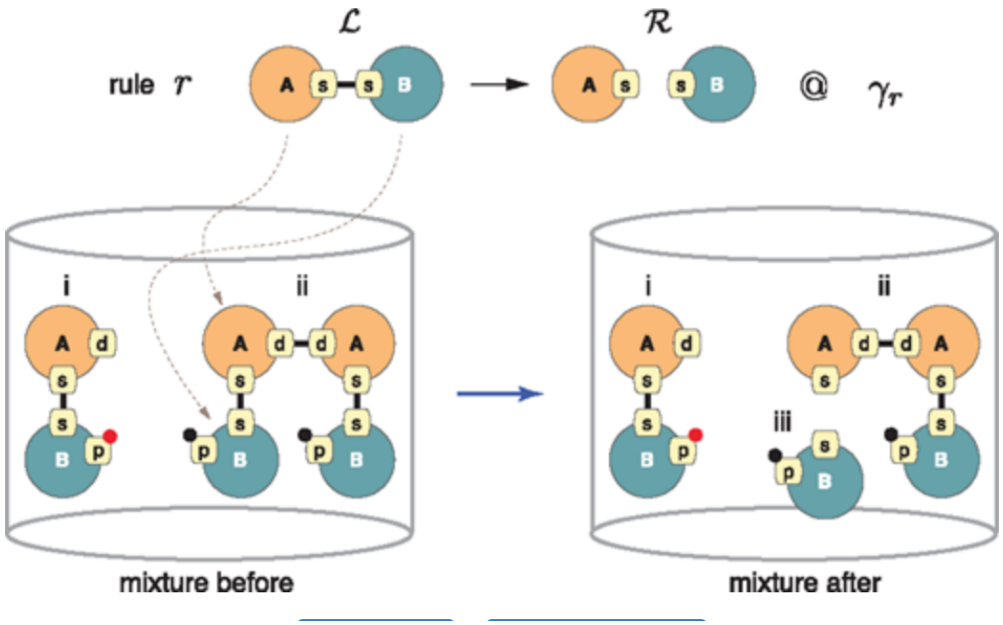



Figure 2.1: Application of rule 'r' to a reaction mixture $\mathfrak{M}$ , comprising of two agents. $L_r$ and $R_r$ represent the state of the site graph before and after the application of rule 'r' respectively. Source: [8]

Modeling tools built using latest technology help to visualize the protein interactions. Some examples of this include Virtual Cell [29] and Kappa Simulator KaSim [16], which provide an environment for the modeling and simulation of cellular interactions. The Kappa rules can be fed into KaSim, which is a protein interaction simulator that implements continuous time MonteCarlo algorithm (CTMC). Similarly, the rules created in BioNetGet [17] format can be simulated with NFsim [30]. KaSim and NFsim employ the process of stochastic modeling that can be used to describe highly complex interactions. Besides them there also exist other stochastic modeling

tools like STOCHSIM [25], which can be used by researchers, to compare the results.

In addition to KaSim, the Kappa simulator, there are tools like KaSA (Kappa Static Analyzer) and KaSTOR (Kappa Story Extractor). As per [8], KaSA helps in analyzing the static properties of the model that can help in debugging, as it can efficiently detect discrepancies between actual and expected behavior. KaSA relies on a technique called 'abstract interpretation' described in [8] to achieve this task. KaSTOR helps by providing an insight as to how an event of interest EOI was obtained in a particular event trace. EOI is obtained by the simulation of a model. KaSTOR works on the concept of mechanistic causality in molecular systems. KaSim generates an output called 'trace' which is a sequence of events generated as a result of the simulation. Algorithms such as the one stated in [13] aid this process. This is dealt in greater detail, in the Analysis of KaSim outputs, subsection.

### 2.3.2 Syntax and semantics of Kappa rules

Kappa rules are formally documented well. Their specification and syntax can be found in [16] and [1]. In rule based specifications graphs are formally specified as objects which have been converted to a notation of textual form, for convenience. As per these rules an agent site denoted by 's', has a binding state 'n' can be specified as s[n]. Here 'n' can be any positive integer or may be replaced by a '.', which indicates the site is not bound. In case 'n' is a positive integer then it implies that the site is bound to another unique site having the same binding site 'n', within the same PPI expression. Hence, a sub expression like, (Agent1 [site1 [1]], Agent2 [site2 [1]] ) indicates that Agent1 is bound to Agent2 through site1 of Agent1 and site2 of Agent2. The protein sites may also have their own internal states which is specified within curly brackets '{}'. Hence Agent1(site1{p}[.]), implies that Agent1 has an unbound site name site1 that is in an internal phosphorylated state (denoted by 'p').

Figure 2.2, depicts the process of obtaining a Protein interaction rule from the textual information of research literature. These rules when passed to KaSim, the Kappa simulator help to visualize the PPI interactions.

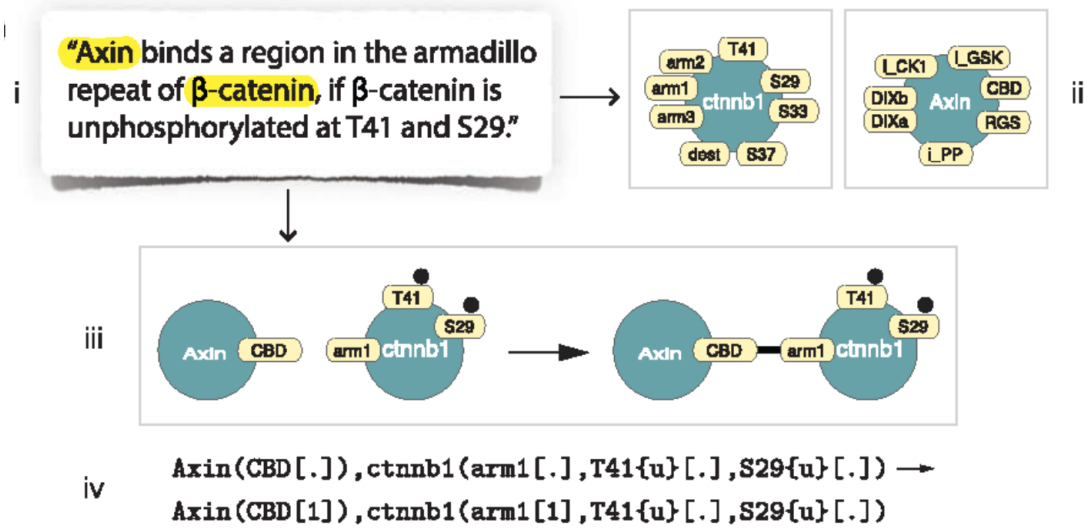



Figure 2.2: Depiction of obtaining a Protein-Protein interaction (PPI) rule from research literature. Source: [8]

### 2.3.3 Analysis of KaSim outputs

As per [8], KaSim generates as it's output the whole sequence of events called trace. The analysis of KaSim output includes discovery of the path by which a particular event of interest (EOI) has occurred. The problem can be formally stated as follows: Provided a trace of events denoted by $\tau$, where $\tau = e_1, e_2, ..e_n$ and $e_n$ denotes an individual event. The problem is to find a suitable explanation for an EOI, where the EOI is an event, $e_n \in \tau = e_1, e_2, ..e_n$. This process is called causal analysis and in Kappa platform KaSTOR, the software agent is utilized for this purpose. The working of this software and the process of causal analysis, is elucidated using the following figure, 2.3

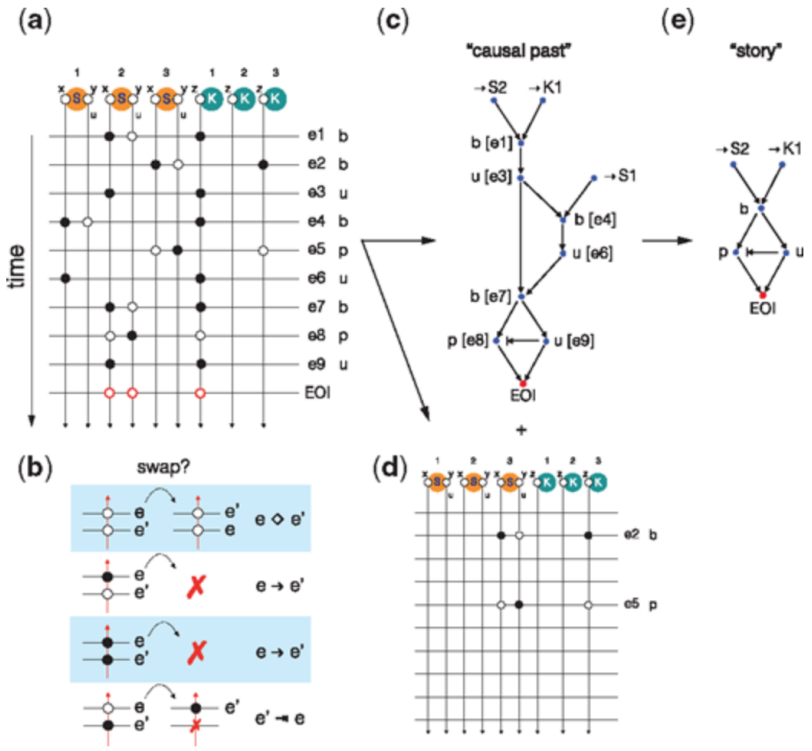



Figure 2.3: Explanation of causal analysis. Source: [8]

In figure 2.3 (a), the vertical lines from each site, of every agent within the mixture denotes a thread. This thread contains the record whether an event tested or modified the state of a site. It should be noted that in kappa terminology a modification of site also implies a test. The events are shown with the horizontal markings. The black discs denote that the thread was modified by its corresponding horizontal event and the white circle denotes that the thread was tested.

As per [8], causality comprises of a relation between events. The algorithm begins by reconstructing the causal past of '*', the identity rule. The figure 2.3 (b) depicts an event 'e' that is followed by event ' e′ '. and iteratively quries if the modification or test of the site in e′ could have occurred before 'e'. There are certain rules for the query which are detailed in [8], under causal linages and compression. This procedure results in a directed acyclic graph (DAG) as shown in 2.3 (c) which represents the precedence structure of the causal past. The procedure depicted in 2.3 (b) also eliminates from within the trace, any events irrelevant to the EOI. This is shown in figure 2.3 (d). The problem with obtaining the precedence structure is that it may violate a property called 'necessity'. A method to tackle this problem involves an algorithm called minimization causal compression elaborated in [13]. KasTOR translates the causal past into a boolean expression where each event is associated with a Boolean variable. The pro-

cessing within KasTOR results to a compressed form of the causal past as shown in the figure 2.3 (e).

## 2.4 Application of Protein-Protein Rule Interaction Database

The databse for PPIs stored in a rule based format will have several applications. Some of these are covered in this section.

- **Building the rule based model in a Kappa format:** The database will allow to build the executable rule based model based on the proteins of interest. For that, the list of proteins submitted to the database would result in a list of the relevant rule PPIs retrived as a csv file. Those rules could be built into Kappa model and simulated by KaSIM to obtain the dynamics of the molecular complex of the interest. This would allow the following more specific applications.

- **Assessment of protein druggability:** As per [28], ruled based PPI stored in a database are useful in the development of a rule-based model for the assessment of protein druggability. Selection of target is an important step for the discovery of drugs. The effectiveness of a drug target depends upon factors like its chemical influence and biological importance. Druggability is defined as the ability of a target to bind a drug-like molecule with an affinity that is at a therapeutically useful level. [28] develops a set of simple rules that govern the process of durggability which can be applied in the future to get an idea of the chemical influence of prospective targets. These rules are based on the property space of druggable pockets like volume, depth and so on.

- **Drug effect pathway analysis:** Pathway analysis involves the study of specific molecular pathways using formalism that are qualitative and quantitative. This domain provides a tremendous computational complexity and experimental approaches that incur a high cost. Research conducted in [21], included extraction of about 200 rules, related to type 2 diabetes obtained from varied sources like literature, pathway databases and conversion from different kinds of models. Using these rules [21] a multi-scale rule-based modeling platform was constructed. This modeling platform was then used for simulation of drug effect pathways of type 2 diabetes drugs and check for it's efficacy. The research concluded that their simulation helped in identifying effective drug combinations and provide a new way to effectively apply existing drugs for new targets.

- **Application of rule based models in biology:** Rule based models can be applied to describe the dynamics of population in a predator-prey ecosystem and simulate rhythm changes that are circadian ([9]).

All these research applications help in substantiating the claim that database for storing PPI in a rule format is an important tool for the study the protein complexes, drug discovery and pathway analysis aimed to improve our understanding of complex biological systems.

# Chapter 3

# Methodology

This chapter includes the methodology adopted for the creation of rule-based Protein-Protein Interaction (PPI) database and the steps to add a new rule to it. It also contains a description of the stored procedures used to extract protein interaction rules based on either agent name or domain name pair. The source code for this project is available on an online repository (Github). This chapter contains a description of the files and folder structure within it.

As a part of this project, a web application using Django [2] was created to access the PPI rules by calling the stored procedures from within it. Chapter 3 includes the design and implementation details for the web application used to access the PPI rules. The PPI rules can be accessed from the web application, either based on the agent name or a domain name pair.

This chapter also includes the steps for deploying the database scripts to a database server and the web application to a web server. During the process of deployment, it is often useful to have the software versions that were initially used to create the software. These are included in Appendix A, under the Software Version section. Having these details will enable developers to debug when deploying the application.

## 3.1 Methodology for creating the database

PPI rules were collected and provided by researchers in the form of an Excel Sheet (rule_baseV0.2.xlsx). To extract the agent , domain names and rules from the dataset, excel sheet parser scripts were created in Python. Python scripts were also written to create and validate entity relations within the database. Description of these scripts are elucidated in Section: Description of the Project Folder and Files.

To construct the PPI database it was decided to structure the data into separate tables consisting of the agent, domain and the PPI rules. In order to connect the data in these three main tables, 2 relationship tables consisting of domain_agent and rule_domain_agent were created.

It should be noted that main tables contain certain 'meta data columns'. These meta data column help in keeping track of the associated data for an entity within the table. For example, the domain table has agent_name column, which keeps a record of all the agent_names that it associates to. However, the correct procedure to obtain the agent_names that associate with a particular domain is to use the domain_agent relationship table. The Entity Relationship (ER) diagram of the database and the description of the database tables are presented in the following sections.

### 3.1.1 Entity Relationship Diagram:

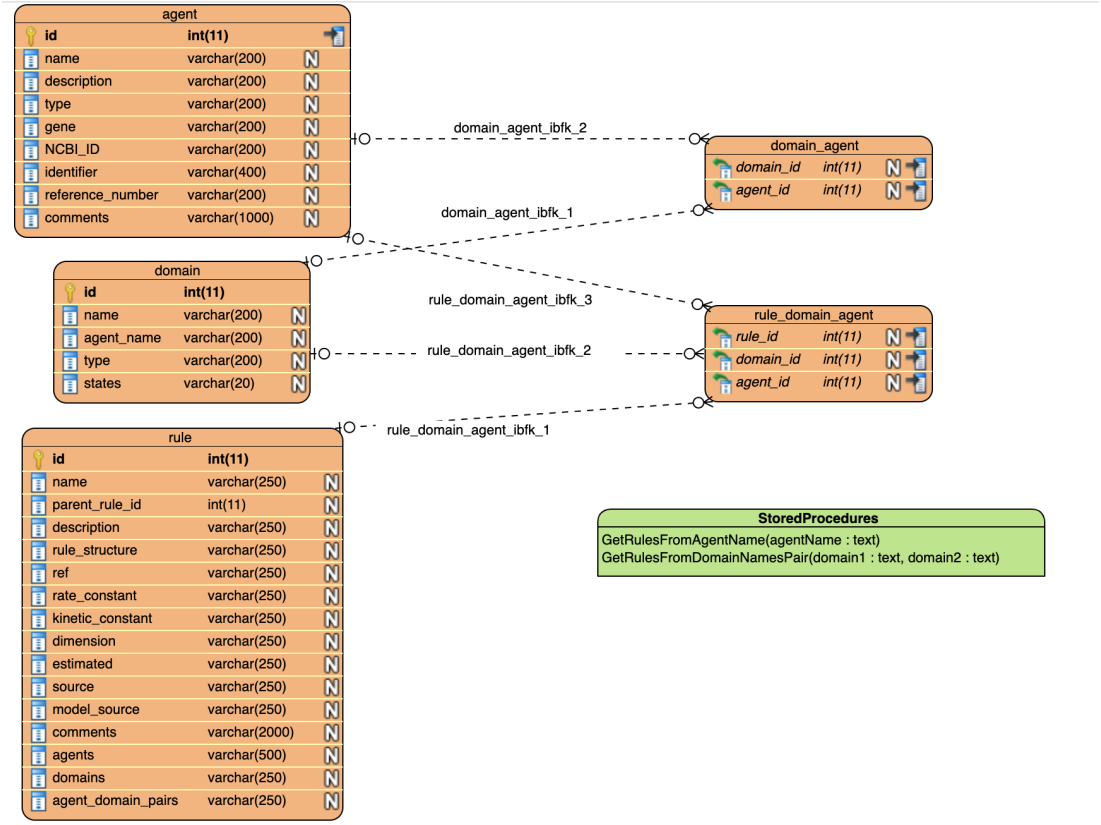



Figure 3.1: Database ER Diagram created using software Visual Paradigm [3]

In Fig 3.1, the dotted lines depict the primary-foreign key relations between the tables. The green box depicts the names of the stored procedures that are part of the database.

### 3.1.2 Description of main tables

1. **agent:-** Provides a description of the protein molecule (the agent). This table contains the following fields: id, name, description, type, gene, NCBI_ID, identifier, reference_number and comments.

   (a) id: The unique identifier for the agent molecule.

   (b) name: The name of the agent.

   (c) description: A description of the agent.

   (d) type: The type/family of the agent.

   (e) gene: The particular gene

   (f) NCBI_ID, identifier, reference_number: Provided within the dataset and contains references to the Uniprot Prosite Database [4] and literature.

   (g) comments: Contains additional comments about the agent.

2. **domain:-** Provides a description of the domains that associate with the agents. This table contains the following fields: id, name, agent_name, type, states.

   (a) id: The unique identifier for the domain name.

   (b) name: The name of the domain.

   (c) agent_name: A 'meta data column' that contains the names of agent molecules that associate with the particular domain.

   (d) type: The type of the domain.

   (e) states: Contains the possible states of the domain.

3. **rule:-** Provides a description of the PPI rules within the database. This table contains the following fields: id, name, parent_rule_id, description, rule_structure, ref, rate_constant, kinetic_constant, dimension, estimated, source, model_source, comments, agents, domains and agent_domain_pairs.

   (a) id: The Unique identifier for the rule

   (b) name: The name of the rule

   (c) parent_rule_id: In some cases the rules can be traced to a parent. This field stored the id of that parent rule. By default it is 0, which signifies no parent rule.

(d) description: A description of the rule.

(e) rule_structure: The structure of the rule.

(f) ref, source: A reference to the rule.

(g) rate_constant: The rate constant for the reaction.

(h) kinetic_constant: The kinetic constant may contain numerical value or a formula to which the rate_constant can be plugged to obtain the result.

(i) dimension: The dimension of the reaction.

(j) estimated: The estimated value of the reaction.

(k) model_source: A reference to the model from which the PPI is obtained.

(l) comments: Any additional comments about the rule.

(m) agents: A 'meta data column' containing the names of the agent molecules that take part in the rule.

(n) domains:A 'meta data column' containing the names of the domains that take part in the rule.

(o) agent_domain_pairs: A 'meta data column' containing the names agent and associated domains that form a part of the protein interaction.

### 3.1.3 Description of relationship tables

1. **domain_agent:-** Each row contains the id of a domain and the id of an agent, obtained by referencing the domain and agent table respectively. By obtaining the domain id from the domain table, the corresponding agent ids that associate with the domain can be obtained. This can then be used to obtain the agent names that associate with that particular domain.

   (a) domain_id: The Id of a domain (from domain table).

   (b) agent_id: The Id of an agent (from agent table).

2. **rule_domain_agent:-** Each row contains the id of a rule, id of a domain and the id of an agent, obtained by referencing the rule, domain and agent table respectively. Hence, this table connects the agents and domains to the PPI rules from the rule table in the database.

   (a) rule_id: The Id of a rule (from rule table).

(b) domain_id: The Id of a domain (from domain table).

(c) agent_id: The Id of an agent (from agent table).

### 3.1.4 Advantages and disadvantages of the database structure

#### 3.1.4.1 Advantages

The database structure is optimized for fast read operations. This is because, by using the id structure to uniquely identify entities in the database, we can leverage the power of indexing [5], which can be used to obtain the results faster. As per [5], in section 8.3.1, most MySQL indexes like primary key, unique key are stored in B Trees. Due to this, it is possible to obtain rows matching the where clause quickly and even leverage the power of Multiple-Column Indexes, as per [5] section 8.3.6. It was also verified in MySQL workbench that the relationship tables (domain_agent and rule_domain_agent) had each of the columns indexed as a B Tree. Hence MySQL would internally be able to work more efficiently with this structure than we if we employed a method that included string regex based searching.

#### 3.1.4.2 Disadvantages

The disadvantage of this database design is that it makes the write operations a multi-step process. For example, if we wanted to add a new rule to the database then besides performing the regular checks, several entries may have to be made in different tables within the database. This is however not a major disadvantage as the database is expected to have many more read operations than write, also the next section states the steps to add a new PPI rule to the database. While trying to make a new entry within the database, following these rules will help in reducing the errors.

## 3.2 Steps for adding new entries to the database

### 3.2.1 Agent

Query the agent table and check if an agent exists with the given name. It should be noted that the agent name is case sensitive. In case it exists then there is no need to add a new entry to the database. However, if it does not exist then a new entry can be made in agent table filling in the relevant fields.

### 3.2.2 Domain

Query the domain table and check if a domain exists with the given name. It should be noted that the domain name is case sensitive. In case it exists then there is no need to add a new entry to the database. However, if it does not exist then a new entry can be made in domain table filling in the relevant fields.

### 3.2.3 Domain-Agent Relationship

When the Agent/Domain is added to the database to connect them we have to make entries within the domain_agent table. Based on the entries made in the Agent and Domain table we can obtain the corresponding ids and make entries containing the (domain id, agent id) pairs. If a new agent is added then all its corresponding domains need to be entered as separate rows where the agent id will remain the same. If a new domain is added then all its corresponding agents need to be added as separate rows where the domain id will remain the same. It should be kept in mind that if a new agent is added to the agent table and it has a new corresponding domain added to the domain table, then that pair must be entered only once within the domain_agent table.

### 3.2.4 Rule

To add a new rule to the database, first check if the rule already exists within the database. If not, then obtain the agents and domains that take part in the rule. Verify that the necessary agents, domains and domain_agent entries exist within the database. If not, then first make the necessary entries in these three tables as per guidelines in the previous sections. Then add the PPI rule to the rule table in the database.

### 3.2.5 Rule-Domain-Agent

After following the steps laid out in Rule subsection, the three pair tuple/s consisting of the rule_id, agent_id for each of the agents in the rule and the domain_id for each of the domains that associate with the corresponding agent within rule, need to be entered in the rule_domain_agent table as (rule id, domain id, agent id).

## 3.3 Description of Stored Procedures

In Fig 3.1, the green box displays stored procedures defined within the database. The two stored procedures created are GetRulesFromAgentName and GetRulesFromDomainNamePair. The expected input, output and algorithm for each of the stored procedure is mentioned in the following subsections.

### 3.3.1 GetRulesFromAgentName

**Input:** An agent name (data type: text).
**Output:** PPI rules in which the agent name, provided as an input occurs.
**Algorithm**

1. The agent id corresponding to the agent name is obtained from the agent table.
2. A list of the rules, in the form of rule ids occurring along with the agent id (obtained in step 1) is extracted from the rule_domain_agent table. This is a list of distinct rule ids, which implies that none of the rule ids are repeated in the output.
3. Using the list of distinct rule ids, the rules are extracted from the rule table and provided as output of the stored procedure.

### 3.3.2 GetRulesFromDomainNamePair

**Input:** Two domain names (data type of each being: text).
**Output:** PPI rules in which both the domain names, provided as an input occurs.
**Algorithm**

1. The domain id for each of the domains are extracted from the domain table and stored in two separate variables within the stored procedure, called domain1 and domain2.
2. From the table, rule_domain_agent those rule ids are extracted that co-occur with the domain2.
3. The rule ids are extracted which co-occur with domain1 and also occur in the list of rule ids extracted in the previous step. Hence implying that the rules ids occuring in this step are the rules that co-occur with domain1 and domain2. This is a distinct list of rule ids.

4. Using the list of distinct rule ids, the rules are extracted from the rule table and provided as output of the stored procedure.

## 3.4 Description of the project folder and its files

### 3.4.1 Project folder source

This project is hosted on Github, as a public repository in the link [6], and is open source.

### 3.4.2 Project folder structure and files

The main folder is called PPI_DB, which contains within it the following sub-folders:

1. django-app: Contains files pertaining to the web application for accessing the PPI rules. The relevant files and folder structure are elucidated in the next section.

2. Parser: Contains files pertaining to extracting the agents, domains and PPI rules from the dataset (rule_baseV0.2.xlsx). This folder has the following files and sub-folder.

   (a) PARSER_POPULATE_TABLE_AGENT.py: Accesses the Agents Sheet of the dataset and generates the table insertion script for the SQL table, agent.

   (b) PARSER_POPULATE_TABLE_DOMAIN.py: Accesses the Agents Sheet of the dataset and generates the table insertion script for the SQL table, domain.

   (c) PARSER_POPULATE_TABLE_RULE.py: Accesses the Rules Sheet of the dataset and generates the table insertion script for the SQL table, rule.

   (d) Extractors: This folder contains python scripts used to obtain the agents, domains, agent-domain pairs and kinetic constants for each of the PPI rules, from the Rules Sheet of the dataset. These extracted columns are then appended to the Rules Sheet of the dataset, as additional columns and included into the SQL rule table. The ‘meta data columns’ like agents, domain and agent-domain pairs provide additional information about the rules.

(e) DB_POPULATE_TABLE_DOMAIN_AGENT.py: This script connects to the database using the credentials mentioned within the script and generates the table insertion script for the SQL table, domain_agent.

To summarize, this script works by first obtaining a row from the domain table created in step (b). The agent_name field in the row gives names of all the agents to which the domain associates itself with. The id of each of these agents is obtained from the agents table and the id of the domain is included in the row extracted from the domain table. Then insert table commands of the form (domain_id, agent_id) are generated, to populate the domain_agent table.

(f) DB_POPULATE_TABLE_RULE_DOMAIN_AGENT.py: This script connects to the database using the credentials mentioned within the script and generates the table insertion script for the SQL table, rule_domain_agent.

To summarize, this script works by first obtaining a row from the rule table created in step (c). The agent_domain_pairs field in the row gives all the agent-domain pairs within that PPI rule. The id of each of these agents is obtained from the agents table and the id of the domain is obtained from the domain table (both obtained by querying from the database). Then insert table commands of the form (rule_id, domain_id, agent_id) are generated, to populate the rule_domain_agent table.

3. Stored Procedure: Contains two files that define stored procedures.

   (a) RULES_FROM_AGENT_NAME.sql: Defines the stored procedure that returns the PPI rules based on the agent name as input.

   (b) RULES_FROM_DOMAIN_NAME_PAIR.sql: Defines the stored procedure that returns the PPI rules based on the pair of domain names as input.

4. Table Populate Scripts: Contains database scripts that are used to populate the PPI database. These table insertion scripts are written in MySQL format and tested using the MySQL client called MySQLWorkbench. It contains the following files:

   (a) AGENT.sql: Script to populate the agent table.

   (b) DOMAIN.sql: An initial Script to populate the domain table. (included in the projects file only for reference)

(c) DOMAIN_V2.sql: Script to populate the domain table.

(d) RULE.sql: Script to populate the rule table.

(e) DOMAIN_AGENT.sql: Script to populate the domain_agent table.

(f) RULE_DOMAIN_AGENT.sql: Script to populate the rule_domain_agent table.

5. Tables/Database Creation Scripts: Contains the database scripts used to define and create the database and tables within it. These scripts are written in MySQL format and tested using the MySQL client called MySQLWorkbench. It contains the following files:

   (a) CREATE_DATABASE_PPI.sql: Script to define and create the PPI Database. This serves like a container to store the PPI database tables.

   (b) CREATE_TABLE_AGENT.sql: Script to define and create the agent table.

   (c) CREATE_TABLE_DOMAIN.sql: Script to define and create the domain table.

   (d) CREATE_TABLE_RULE.sql: Script to define and create the rule table.

   (e) CREATE_TABLE_DOMAIN_AGENT.sql: Script to define and create the domain_agent table.

   (f) CREATE_TABLE_RULE_DOMAIN_AGENT.sql: Script to define and create the rule_domain_agent table.

6. Rectified Rules: Verification steps were adopted after the initial population of the database. A report was generated based on any discrepancies within the data in the database. For example, it is possible that while inserting a rule it is observed that the domain that the agent occurs with was not entered in the database before as it was not observed in the Agents sheet, in the dataset. Based on the report and subsequent discussions the table population scripts were updated. Moe information regarding the verification pipeline is presented in the 'Results and Discussion Section'. This folder contains the following two files:

   (a) problematic_agents_domains.txt: The initial report on the the observed discrepancies.

   (b) rule_baseV0.2_fixed.xlsx: The dataset with the updated values for agents, domains and rules based on the discrepancies reported in

problematic_agents_domains.txt. It is an updated version of the dataset contained in the Parser folder.

## 3.5 Web Application for accessing the stored procedures

This section provides a walk-through of the django app file structure and the important files for the project. It also explains the procedure to run the django app and retrieve the PPI rules. We also explain the functionality of the user interface (UI) in this section.

### 3.5.1 Folder Structure and files

Files for the web application are contained in the django-app folder within the source code hosted on Github repository, [6]. The root folder for django app is called djangonautic and within it we find the subfolders called djanjonautic, ppi and templates. This folder also contains the manage.py file which is automatically generated while creating the django app and is used to run tasks like making database migrations and starting the web server.

Within the djangonautic folder contains an important file called settings.py. This contains important information such as the installed apps and database connection details. In case the django-app needs to be extended then the installed app need to be registered in the INSTALLED _APPS section. The database details can also be modified from the settings.py file in the DATABASES section.

The ppi folder contains the files and templates relevant to extracting and displaying the PPI information in the webpage. When the url is invoked, in order to serve the client request the django server first matches the url with the url patterns available in the urls.py file located in this folder. Once the url pattern is matched, the corresponding function from the views.py file is invoked. Within the functions of the views.py file the database call to the relevant stored procedure is made. Once the result is returned by the database the relevant template from the ppi/templates folder is invoked and the result is rendered onto the screen. The static folder within the ppi/templates contains the javascript, css and fonts obtained from the library [7], are used to create and display the PPI rules in the UI.

### 3.5.2 Running the web application

In order to run the web application the following steps need to be performed:

1. The SQL server in which the database is hosted should be in running state. In MAC OSX this can be verified by navigating to the System Preferences → MySQL and ensuring that the MySQL server is running.

2. Navigate to the folder django_app → djangonautic and run the following command to start the server.
   python manage.py runserver

### 3.5.3 Get PPI rules based on agent name

1. In order to access the PPI rules based on Agent Name, enter the following URL in the web browser: http://127.0.0.1:8000/ppi/AgentNameForm/.

2. Enter the agent name in the text box provided and click on submit as shown in the figure 3.2. (Note: The agent name is case sensitive). Internally the django app makes a connection to the database and invokes the GetRulesFromAgentName stored procedure and passes the agent name as argument to it.

3. The retrieved PPI rules will be visible in the next page as shown in the figure 3.3. In case there are no matching PPI rules then a message is displayed on the screen to inform the user.

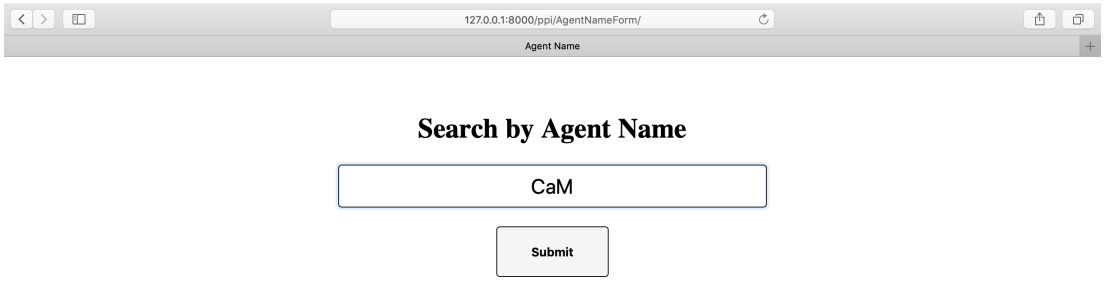
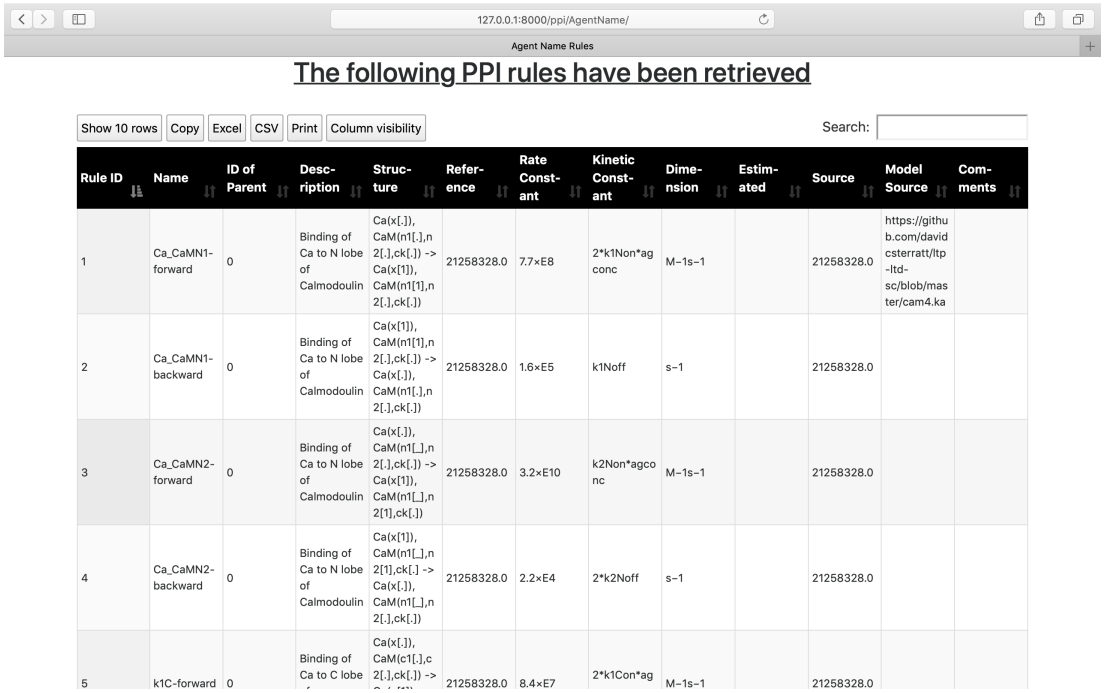



Figure 3.2: Form to accept the agent name.




| Rule ID | Name | ID of Parent | Desc-ription | Struc-ture | Refer-ence | Rate Const-ant | Kinetic Const-ant | Dime-nsion | Estim-ated | Source | Model Source | Com-ments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Ca_CaMN1-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[.],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[1],n2[.],ck[.]) | 21258328.0 | 7.7×E8 | 2*k1Non*agconc | M−1s−1 | | 21258328.0 | https://github.com/davidcsterratt/ltp-ltd-sc/blob/master/cam4.ka | |
| 2 | Ca_CaMN1-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[1],n2[.],ck[.]) -> Ca(x[.]), CaM(n1[.],n2[.],ck[.]) | 21258328.0 | 1.6×E5 | k1Noff | s−1 | | 21258328.0 | | |
| 3 | Ca_CaMN2-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[_],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[_],n2[1],ck[.]) | 21258328.0 | 3.2×E10 | k2Non*agconc | M−1s−1 | | 21258328.0 | | |
| 4 | Ca_CaMN2-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[_],n2[1],ck[.] -> Ca(x[.]), CaM(n1[_],n2[.],ck[.]) | 21258328.0 | 2.2×E4 | 2*k2Noff | s−1 | | 21258328.0 | | |
| 5 | k1C-forward | 0 | Binding of Ca to C lobe | Ca(x[.]), CaM(c1[.],c2[.],ck[.]) -> | 21258328.0 | 8.4×E7 | 2*k1Con*ag | M−1s−1 | | 21258328.0 | | |

Figure 3.3: The rules retrieved based on the domain names provided in the agent name form.

### 3.5.4 Get PPI rules based on domain names

1. In order to access the PPI rules based on a pair of domain names, enter the following URL in the web browser: http://127.0.0.1:8000/ppi/DomainNameForm/

2. Enter the domain names in the text boxes provided and click on submit, as shown in the figure 3.4. (Note: The domain names are case sensitive). Internally the django app makes a connection to the database and invokes the GetRulesFromDomainNamePair stored procedure and passes the domain names as arguments to it.

3. The retrieved PPI rules will be visible in the next page as shown in the figure 3.5. In case there are no matching PPI rules then a message is displayed on the screen to inform the user.

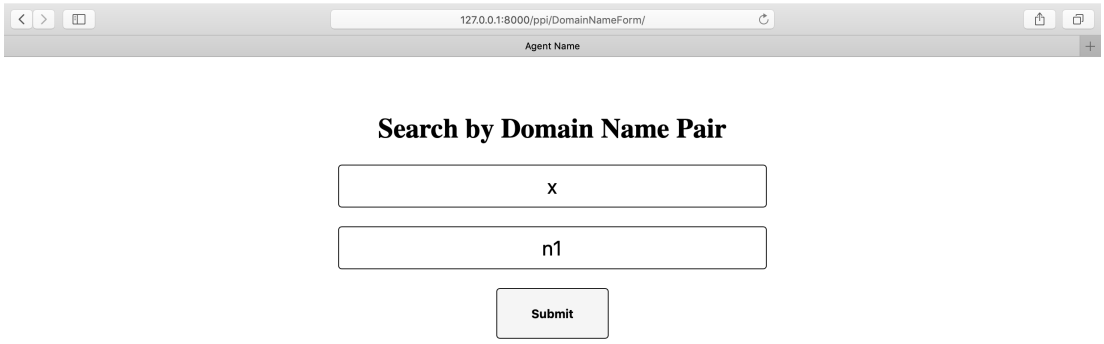



Figure 3.4: Form to accept the domain names.

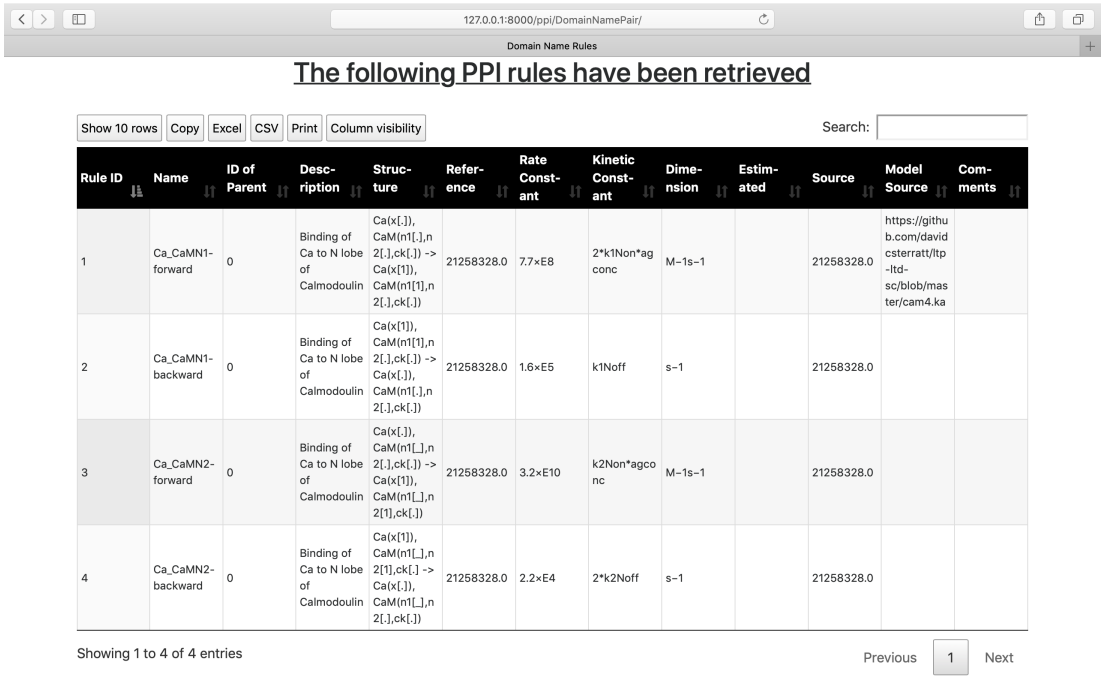



Figure 3.5: The rules retrieved based on the domain names provided in the domain names form.

### 3.5.5 User Interface (UI) functionality

The buttons in the user interface have been created with the help of the DataTables libraray [7].These UI options are helpful, as the retrieved results can be processed and fed into the kappa simulator for visualization of the protein interactions. A description of the buttons in the figure 3.6 is as follows:

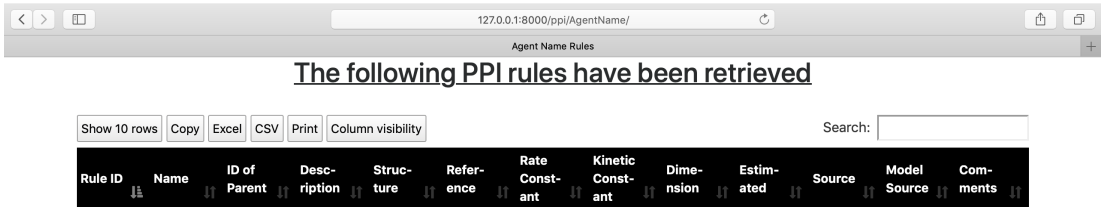



Figure 3.6: A closer view at the user interface (UI) buttons.

1. Show ‘n’ rows: Provides the option of displaying 10, 25, 50, 100 or all the rows.
2. Copy: Allows the table to be copied to the clipboard.
3. Excel: Allows the table to be exported to an excel sheet.
4. CSV: Allows the table to be exported to a comma separated file.

5. Print: Allows the user to print the retrieved results.

6. Column visibility: Allows the user to decide which columns are to be visible. This can be useful while exporting or saving only the required results.

7. Arrow buttons beside the field names: Allow the user to sort the results in numerical order in case of a numerical field or alphabetical order in case of a text field.

8. Search: The search button allows the user to search over the entire table based on a search key.

## 3.6 Deployment steps

### 3.6.1 Database Scripts

The SQL scripts were tested on MySQL server using the client MySQLWorkbench. The respective versions can be found in Appendix A. In order to deploy the database scripts onto a database server the following steps need to be followed.

1. Create a user account within the database server (if not created already).

2. Login to the user account within the database server.

3. Navigate to the Tables:Database Creation Scripts folder located within the project folder hosted on Github.

4. Execute the CREATE_DATABASE_PPI.sql SQL script in the MySQL client, to create the PPI database.

5. Execute the remaining scripts in the following order, CREATE_TABLE_AGENT.sql, CREATE_TABLE_DOMAIN.sql, CREATE_TABLE_DOMAIN_AGENT.sql, CREATE_TABLE_RULE.sql and CREATE_TABLE_RULE_DOMAIN_AGENT.sql.

6. Navigate to the Table Populate Scripts folder located within the project folder hosted in Github.

7. Execute the SQL scripts in the following order AGENT.sql, DOMAIN_V2.sql, RULE.sql, DOMAIN_AGENT.sql and RULE_DOMAIN_AGENT.sql.

### 3.6.2 Web Application

To deploy the web application onto a web server navigate to the django_app folder within the project folder hosted on Github.

1. Navigate to the file django_app/djangonautic/djangonautic/settings.py, update the database credentials with the ones created in the previous section, in the DATABASE section of the code and save it.

2. Navigate to the folder django_app/djangonautic/ and run the following commands to synchronize the data models within the django application, with the database:
   python manage.py migrate
   python manage.py makemigrations

3. After synchronizing the data model, start the web server by running the following command:
   python manage.py runserver
   and acess the URLs for retrieving the PPIs based on agent name or domain names through a web browser.

The web application was tested on the web browsers Safari, Google Chrome and Mozilla Firefox.

# Chapter 4

# Verification and Results

This chapter presents the verification techniques adopted, to ensure correctness in several parts of the project. These include aspects of correctness in database creation, values retrieved by the stored procedures and the displayed results within the UI.

## 4.1 Verification pipeline for database entries

This section details the verification pipeline used to ensure correctness of data stored within the PPI database. This is verified using the following reasoning and python scripts within the project folder.

1. The Agent table is created by parsing the Agents sheet within the provided dataset (Excel sheet).

2. The domain table is then created by parsing the agents sheet within the provided dataset (Excel sheet). Hence the agent table and the domain table is obtained.

3. The rule table was then created by parsing the Rules sheet within the provided dataset (Excel Sheet). The rules were pre-processed to obtain the agents, domains and agent-domain pairs. These were added as additional columns in the Excel sheet and are visible in the Processed Excel Sheets folder. The additional columns are also visible in the rule table of the database.

4. The correctness of parsing the rules was manually verified to ensure that they were free from errors.

5. We next proceed to create the domain_agent table using the script DB_POPULATE_TABLE_DOMAIN_AGENT.py.

The script connects to the database and obtains a domain by querying the domain table. It acquires its corresponding id and all the associated agent names from the domain table.

6. This script then queries the database for each of the agent names associated with that particular domain.

7. In case it does not find the agent name in the agent table of the database, it reports this discrepancy.

8. In case it finds the agent name in the agent table of the database then it proceeds to create the SQL entry for the domain_agent table.
   Note: The domain id was already obtained in step 3.

9. We next proceed to create the rule_domain_agent table using the script DB_POPULATE_TABLE_RULE_DOMAIN_AGENT.py.
   The script connects to the database using MySQL connector and for each rule it obtains the agents and domains present within the rule.

10. It then connects to the database and queries if each of the agent names exist within the database. In case it does not, then it reports the discrepancy with the prefix Problem in AGENT_NAME.

11. For each of the domains extracted it checks whether the domain name exists in the domain database. In case the domain name does not exist, then it reports this discrepancy with the prefix Problem in DOMAIN_NAME. These discrepancies are written to a file named problematic_agents_domains.txt.

12. The script then proceeds to verify that an agent and domain extracted from the rule, have the necessary entry in the domain_agent relation. In case, it does not find this entry, then it reports this discrepancy.

13. In case no discrepancies were observed then, it implies that the agent name occurs in the agent database and the domain name exists in the domain database. Also the relevant connection between the domain and agent is found in the domain_agent relation.

14. The MySQL table entry for the relation rule_domain_agent for the observed rule, domain and agent is created. This procedure is repeated for each of the agents and associated domains within the rule and for every rule.

15. Finally, the discrepancies were examined and discussed with the supervisor. Based on inputs provided the discrepancies were resolved and the database loading scripts were updated to accommodate the changes.

## 4.2 Verification of stored procedure results

Two stored procedures were created as part of the project. These are useful in retrieving the PPI rules based on the criteria defined in the following sub-sections.

### 4.2.1 GetRulesFromAgentName

The purpose of this stored procedure is to return the PPI rules which contains the agent name (provided as an input). To verify that the defined stored procedure behaves as expected, test cases were tried and the outputs were verified. Some of these test cases are as follows:

1. 
   - The stored procedure is called with the agent name 'CaM', using the command, call GetRulesFromAgentName('CaM') from MySQL workbench.
   - The expected output is the PPI rules, containing the agent CaM within it.
   - The output obtained from the database contained 14 rows. A snippet of the output is shown below, in the figure 4.1. Each of these PPI rules were verified to contain the agent name CaM within it.

| id | name | parent_rule_id | description | rule_structure | ref | rat |
|---|---|---|---|---|---|---|
| 1 | Ca_CaMN1-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[.],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[1],n2[.],ck[.]) | 21258328.0 | 7.7 |
| 2 | Ca_CaMN1-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[1],n2[.],ck[.]) -> Ca(x[.]), CaM(n1[.],n2[.],ck[.]) | 21258328.0 | 1.6 |
| 3 | Ca_CaMN2-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[_],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[_],n2[1],ck[.]) | 21258328.0 | 3.2 |
| 4 | Ca_CaMN2-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[_],n2[1],ck[.] -> Ca(x[.]), CaM(n1[_],n2[.],ck[.]) | 21258328.0 | 2.2 |
| 5 | k1C-forward | 0 | Binding of Ca to C lobe of Calmodulin | Ca(x[.]), CaM(c1[.],c2[.],ck[.]) -> Ca(x[1]), CaM(c1[1],c2[.],ck[.]) | 21258328.0 | 8.4 |
| 6 | k1C-backward | 0 | Binding of Ca to C lobe of Calmodulin | Ca(x[1]), CaM(c1[1],c2[.],ck[.]) -> Ca(x[.]), CaM(c1[.],c2[.],ck[.]) | 21258328.0 | 2.6 |
| 7 | k2C-forward | 0 | Binding of Ca to C lobe of Calmodulin | Ca(x[.]), CaM(c1[_],c2[.],ck[.]) -> Ca(x[1]), CaM(c1[_],c2[1],ck[.]) | 21258328.0 | 2.5 |
| 8 | k2C-backward | 0 | Binding of Ca to C lobe of Calmodulin | Ca(x[1]), CaM(c1[_],c2[1],ck[.]) -> Ca(x[.]), CaM(c1[_],c2[.],ck[.]) | 21258328.0 | 6.5 |
| 17 | CaM_CaMKII-forward | 0 | Binding of CaM to CaMKII | CaM(ck[.],n1[_],n2[_],c1[_],c2[_]), CaMKII(cam[.]) -> CaM(ck[1],n1[_],n2[_],c1[_],c… | 1317063.0 | 1.5 |
| 18 | CaM_CaMKII-backward | 0 | | CaM(ck[1],n1[_],n2[_],c1[_],c2[_]), CaMKII(cam[1]) -> CaM(ck[.],n1[_],n2[_],c1[_],… | 1317063.0 | 6.7 |
| 19 | CaMKii_P | 0 | Phosphorylation of CaMKII by CaM | CaM(ck[1]), CaM(ck[2]), CaMKII(cam[1],s{u},a[3]), CaMKII(cam[2],s{u},b[3]) -> C… | 18769913.0 | |
| 21 | CaMKii_auto_P_CaM | 0 | Auto phosphorylation of CaMKII with bo… | CaM(ck[1]), CaMKII(cam[.],s{p},a[2]), CaMKII(cam[1],s{u},b[2]) -> CaM(ck[1]), C… | 18769913.0 | |
| 72 | NR1_CaM | 0 | Binding NR1 to CaM | CaM(h), NR1(c0) <-> CaM(h!0), NR1(c0!0) | | |
| 87 | SAP102_CaM | 0 | Binding SAP102 to CaM | SAP102(H), CaM(h) <-> SAP102(H!1), CaM(h!1) | | |

Figure 4.1: The retrieved rules from the stored procedure called with the argument 'Cam'

2. 
   - The stored procedure is called with agent name 'SynGAPa', using the command, call GetRulesFromAgentName('SynGAPa') from MySQL workbench.

- The expected output is the PPI rules, containing the agent SynGAPa.
- The output obtained from the database contained 6 rows. A snippet of the output is shown in the figure 4.2. Each of these PPI rules were verified to contain the agent name SynGAPa within it.

| id | name | parent_rule_id | description | rule_structure | ref | rate_constant |
|---|---|---|---|---|---|---|
| 35 | PSD95_SynGAP | 35 | Binding od PSD95 to SynGAP | PSD95(PDZ1),SynGAPa(c~u) <-> PSD95(PDZ1!1),SynGAPa(c~u!1) | 27623146.0 | |
| 43 | PSD93_SynGAP | 35 | Binding od PSD93 to SynGAP | PSD93(PDZ3),SynGAPa(c~u) <-> PSD93(PDZ3!1),SynGAPa(c~u!1) | | |
| 81 | SAP97_SynGAP | 35 | Binding of SAP97 to SynGAP | SAP97(PDZ3), SynGAPa(c) <-> SAP97(PDZ3!1), SynGAPa(c!1) | | |
| 90 | SAP102_SynGAP | 35 | Binding of SAP102 to SynGAP | SAP102(PDZ3), SynGAPa(c~u) <-> SAP102(PDZ3!1), SynGAPa(c~u!1) | | |
| 153 | SynGap_Ras(H-RAS) | 0 | Binding SynGAP to RAS(H-RAS) | SynGAPa(RasGAP), H-RAS(gap~GTP) <-> SynGAPa(RasGAP!1),H-RAS(gap~G… | | |
| 166 | SynGap_Ras(Rap1) | 153 | Binding SynGAP to Rap1 | SynGAPa(RasGAP), Rap1(gap~GTP) -> SynGAPa(RasGAP!1), Rap1(gap~GTP!1) | | |

Figure 4.2: The retrieved rules from the stored procedure called with the argument ‘SynGAPa’

### 4.2.2 GetRulesFromDomainNamesPair

The purpose of this stored procedure is to return the PPI rules which contains the domain names (provided as an input). To verify that the defined stored procedure behaves as expected, test cases were tried and the outputs were verified. Some of these test cases are as follows:

1. 
   - The stored procedure is called with the domain name arguments ‘x,n1’, using the command,
     call GetRulesFromDomainNamesPair(‘x’,‘n1’)
     from MySQL workbench.
   - The expected output is the PPI rules, containing the domain names x and n1 within it.
   - The output obtained from the database contained 4 rows. A snippet of the output is shown below, in the figure 4.3. Each of these PPI rules were verified to contain the domain names x and n1 within it.
   - The stored procedure call was also tested with the arguments swapped (n1 and x) and the retrieved rules were verified to be the same.

| id | name | parent_rule_id | description | rule_structure | ref | rate_co |
|---|---|---|---|---|---|---|
| 1 | Ca_CaMN1-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[.],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[1],n2[.],ck[.]) | 21258328.0 | 7.7×E8 |
| 2 | Ca_CaMN1-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[1],n2[.],ck[.]) -> Ca(x[.]), CaM(n1[.],n2[.],ck[.]) | 21258328.0 | 1.6×E5 |
| 3 | Ca_CaMN2-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[_],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[_],n2[1],ck[.]) | 21258328.0 | 3.2×E1( |
| 4 | Ca_CaMN2-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[_],n2[1],ck[.] -> Ca(x[.]), CaM(n1[_],n2[.],ck[.]) | 21258328.0 | 2.2×E4 |

Figure 4.3: The retrieved rules from the stored procedure called with the argument ‘x’ and ’n1’

2. - The stored procedure is called with the domain name arguments ‘GK,GKBD’, using the command,
   call GetRulesFromDomainNamesPair(‘GK’,‘GKBD’)
   from MySQL workbench.
   - The expected output is the PPI rules, containing the domain names GK and GKBD within it.
   - The output obtained from the database contained 9 rows. A snippet of the output is shown below, in the figure 4.4. Each of these PPI rules were verified to contain the domain names GK and GKBD within it.
   - The stored procedure call was also tested with the arguments swapped (GKBD and GK) and the retrieved rules were verified to be the same.

| id | name | parent_rule_id | description | rule_structure | ref | rate_constant | kinetic_constant |
|---|---|---|---|---|---|---|---|
| 36 | PSD95_GKAP | 0 | Binding of PSD95 to GKAP | PSD95(GK), GKAP(GKBD) <-> PSD95(GK!1), GKAP(GKBD!1) | | | 1, 0.01 |
| 39 | PSD95_SPAR | 0 | Binding of PSD95 to SPAR | PSD95(GK), SPAR(GKBD) <-> PSD95(GK!1), SPAR(GKBD!1) | | | 1, 0.01 |
| 44 | PSD93_GKAP | 36 | Binding of PSD93 to GKAP | PSD93(GK), GKAP(GKBD) <-> PSD93(GK!1), GKAP(GKBD!1) | | | 1, 0.01 |
| 47 | PSD93_SPAR | 39 | Binding of PSD93 to SPAR | PSD93(GK), SPAR(GKBD) <-> PSD93(GK!1), SPAR(GKBD!1) | | | 1, 0.01 |
| 85 | SAP97_SPAR | 0 | Binding SAP97 to SPAR | SAP97(GK), SPAR(GKBD) <-> SAP97(GK!1), SPAR(GKBD!1) | | | 1, 0.01 |
| 86 | SAP97_GKAP | 0 | Binding SAP97 to GKAP | SAP97(GK), GKAP(GKBD) <-> SAP97(GK!1), GKAP(GKBD!1) | | | 1, 0.01 |
| 92 | SAP102_SPAR | 0 | Binding of SAP102 to SPAR | SAP102(GK), SPAR(GKBD) <-> SAP102(GK!1), SPAR(GKBD!1) | | | 1, 0.01 |
| 94 | SAP102_GKAP | 0 | Binding SAP102 to GKAP | SAP102(GK), GKAP(GKBD) <-> SAP102(GK!1), GKAP(GKBD!1) | | | 1, 0.01 |
| 114 | S-SCAM_GKAP | 0 | Binding S-SCAM to GKAP | S-SCAM(GK), GKAP(GKBD) <-> S-SCAM(GK!1), GKAP(GKBD!1) | | | 1, 0.01 |

Figure 4.4: The retrieved rules from the stored procedure called with the argument ‘GK’ and ’GKBD’

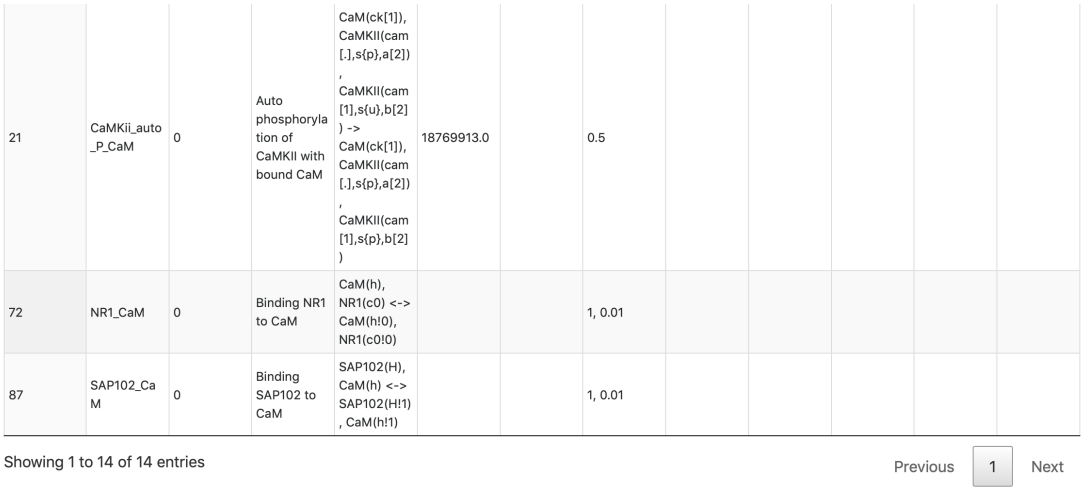
## 4.3 Verification of results displayed in UI

In this section we proceed to verify the outputs displayed in the UI created using django and DataTables. In order to judge the correctness of the results fetched from the database and displayed within the UI, it has to be ensured that the results from the stored procedure are displayed in the UI without any data loss or modifications. This retrieval and display test is performed for obtaining the PPI rules based on both the stored procedures (GetRulesFromAgentName and GetRulesFromDomainNamesPair).

### 4.3.1 Agent Name

In this test, the stored procedure GetRulesAgentName is called from within the django app and the output is displayed in the UI. The displayed output in the UI was matched with the output from the stored procedure in MySQL workbench.
Two examples of such test cases are as follows:

1. The UI is called with the agent name CaM and a snippet of the retrieved rules is shown in the figure 4.5. The number of entries retrieved in the UI was 14 and the number of entries retrieved by the stored procedure was also 14. The rules retrieved were also exactly the same.



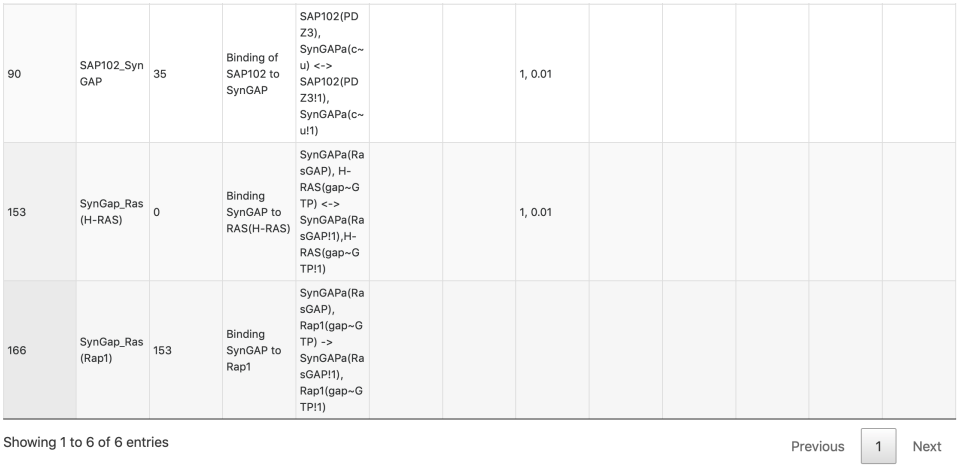
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | CaMKii_auto_P_CaM | 0 | Auto phosphorylation of CaMKII with bound CaM | CaM(ck[1]), CaMKII(cam[.],s{p},a[2]), CaMKII(cam[1],s{u},b[2]) -> CaM(ck[1]), CaMKII(cam[.],s{p},a[2]), CaMKII(cam[1],s{p},b[2]) | 18769913.0 | | 0.5 | | | | | |
| 72 | NR1_CaM | 0 | Binding NR1 to CaM | CaM(h), NR1(c0) <-> CaM(h!0), NR1(c0!0) | | | 1, 0.01 | | | | | |
| 87 | SAP102_CaM | 0 | Binding SAP102 to CaM | SAP102(H), CaM(h) <-> SAP102(H!1), CaM(h!1) | | | 1, 0.01 | | | | | |

Showing 1 to 14 of 14 entries

Previous 1 Next

Figure 4.5: The retrieved rules from the stored procedure called with the agent name 'CaM'

2. The UI is called with the agent name SynGAPa and a snippet of the retrieved rules is shown in the figure 4.6. The number of entries retrieved in the UI was 6 and the number of entries retrieved by the stored procedure was also 6. The rules retrieved were also exactly the same.



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 90 | SAP102_SynGAP | 35 | Binding of SAP102 to SynGAP | SAP102(PDZ3), SynGAPa(c~u) <-> SAP102(PDZ3!1), SynGAPa(c~u!1) | | | 1, 0.01 | | | | | |
| 153 | SynGap_Ras (H-RAS) | 0 | Binding SynGAP to RAS(H-RAS) | SynGAPa(RasGAP), H-RAS(gap~GTP) <-> SynGAPa(RasGAP!1),H-RAS(gap~GTP!1) | | | 1, 0.01 | | | | | |
| 166 | SynGap_Ras (Rap1) | 153 | Binding SynGAP to Rap1 | SynGAPa(RasGAP), Rap1(gap~GTP) -> SynGAPa(RasGAP!1), Rap1(gap~GTP!1) | | | | | | | | |

Showing 1 to 6 of 6 entries

Previous 1 Next

Figure 4.6: The retrieved rules from the stored procedure called with the agent name 'SynGAPa'

### 4.3.2 Pair of Domain Names

In this test, the stored procedure GetRulesFromDomainNamesPair is called from within the django app and the output is displayed in the UI. The displayed output in the UI was matched with the output from the stored procedure in MySQL workbench. For each of the domain name pair inputs, the entries in text boxes of the web form were interchanged the and results were verified to be the same.
Two examples of such test cases are as follows:

1. The UI is called with the domain names x,n1 and a snippet of the retrieved rules is shown in the figure 4.7. The number of entries retrieved in the UI was 4 and the number of entries retrieved by the stored procedure was also 4. The rules retrieved were also exactly the same.

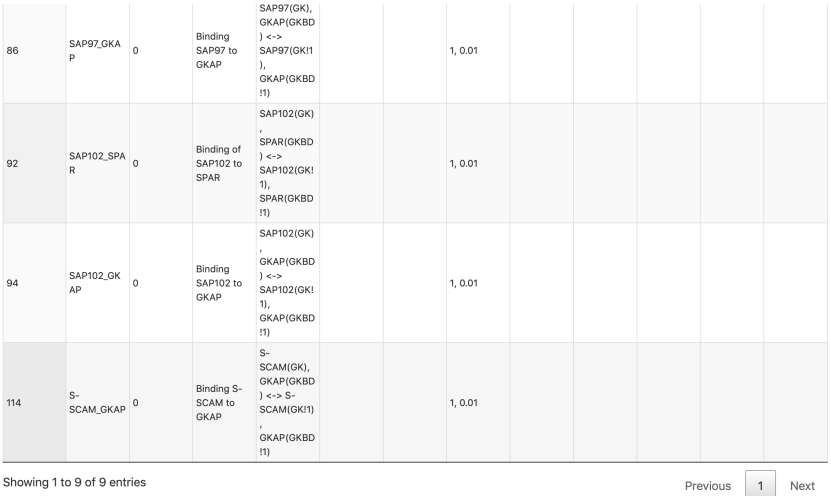
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Ca_CaMN1-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[.],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[1],n2[.],ck[.]) | 21258328.0 | 7.7×E8 | 2*k1Non*agconc | M−1s−1 | | 21258328.0 | https://github.com/davidcsterratt/ltp-ltd-sc/blob/master/cam4.ka | |
| 2 | Ca_CaMN1-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[1],n2[.],ck[.]) -> Ca(x[.]), CaM(n1[.],n2[.],ck[.]) | 21258328.0 | 1.6×E5 | k1Noff | s−1 | | 21258328.0 | | |
| 3 | Ca_CaMN2-forward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[.]), CaM(n1[_],n2[.],ck[.]) -> Ca(x[1]), CaM(n1[_],n2[1],ck[.]) | 21258328.0 | 3.2×E10 | k2Non*agconc | M−1s−1 | | 21258328.0 | | |
| 4 | Ca_CaMN2-backward | 0 | Binding of Ca to N lobe of Calmodoulin | Ca(x[1]), CaM(n1[_],n2[1],ck[.] -> Ca(x[.]), CaM(n1[_],n2[.],ck[.]) | 21258328.0 | 2.2×E4 | 2*k2Noff | s−1 | | 21258328.0 | | |

Showing 1 to 4 of 4 entries

Previous 1 Next

Figure 4.7: The retrieved rules from the stored procedure called with the domain names ‘x’ and ‘n1’

2. The UI is called with the domain names GK,GKBD and a snippet of the retrieved rules is shown in the figure 4.8. The number of entries retrieved in the UI was 9 and the number of entries retrieved by the stored procedure was also 9. The rules retrieved were also exactly the same.



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 86 | SAP97_GKAP | 0 | Binding SAP97 to GKAP | SAP97(GK), GKAP(GKBD) <-> SAP97(GK!1), GKAP(GKBD!1) | | | 1, 0.01 | | | | | |
| 92 | SAP102_SPAR | 0 | Binding of SAP102 to SPAR | SAP102(GK), SPAR(GKBD) <-> SAP102(GK!1), SPAR(GKBD!1) | | | 1, 0.01 | | | | | |
| 94 | SAP102_GKAP | 0 | Binding SAP102 to GKAP | SAP102(GK), GKAP(GKBD) <-> SAP102(GK!1), GKAP(GKBD!1) | | | 1, 0.01 | | | | | |
| 114 | S-SCAM_GKAP | 0 | Binding S-SCAM to GKAP | S-SCAM(GK), GKAP(GKBD) <-> S-SCAM(GK!1), GKAP(GKBD!1) | | | 1, 0.01 | | | | | |

Showing 1 to 9 of 9 entries

Previous 1 Next

Figure 4.8: The retrieved rules from the stored procedure called with the domain names ‘GK’ and ‘GKBD’

# Chapter 5

# Conclusions

# Bibliography


[1] https://kappalanguage.org/documentation.

[2] https://docs.djangoproject.com/en/2.2/.

[3] https://www.visual-paradigm.com/.

[4] https://www.uniprot.org/database/DB-0084.

[5] https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html.

[6] https://github.com/ayushdas/PPI_DB.

[7] https://datatables.net/.

[8] Pierre Boutillier, Mutaamba Maasha, Xing Li, Hctor F Medina-Abarca, Jean Krivine, Jrme Feret, Ioana Cristescu, Angus G Forbes, and Walter Fontana. The Kappa platform for rule-based modeling. *Bioinformatics*, 34(13):i583–i592, 06 2018.

[9] Álvaro Bustos, Ignacio Fuenzalida, Rodrigo Santibáñez, Tomás Pérez-Acle, and Alberto JM Martin. Rule-based models and applications in biology. In *Computational Cell Biology*, pages 3–32. Springer, 2018.

[10] Lily A. Chylek, Leonard A. Harris, Chang-Shung Tung, James R. Faeder, Carlos F. Lopez, and William S. Hlavacek. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdiscip Rev Syst Biol Med*, 6(1): 1336, 2014.

[11] Lily A Chylek, Leonard A Harris, Chang-Shung Tung, James R Faeder, Carlos F Lopez, and William S Hlavacek. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 6(1):13–36, 2014.

[12] David Croft, Gavin OKelly, Guanming Wu, Robin Haw, Marc Gillespie, Lisa Matthews, Michael Caudy, Phani Garapati, Gopal Gopinath, Bijay Jassal, et al. Reactome: a database of reactions, pathways and biological processes. *Nucleic acids research*, 39(suppl_1):D691–D697, 2010.

[13] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Chris Thompson-Walsh, and Glynn Winskel. Graphs, rewriting and pathway reconstruction for rule-based models. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[14] Javier De Las Rivas and Celia Fontanillo. Protein–protein interactions essentials: key concepts to building and analyzing interactome networks. *PLoS computational biology*, 6(6):e1000807, 2010.

[15] Danos V. et al. (2007a) rule-based modelling of cellular signalling. *In: Proceedings of the Eighteenth International Conference on Concurrency Theory, CONCUR 2007*, Vol. 4703 of Lecture Notes in Computer Science. Lisbon, Portugal, Springer-Verlag Berlin Heidelberg, pp. 1741., 2007.

[16] Danos V. et al. (2007b) scalable simulation of cellular signaling networks. In *Asian Symposium on Programming Languages and Systems*, pages 139–157. Springer, 2007.

[17] Faeder J.R. et al. Rule-based modeling of biochemical systems with bionetgen. *In: Maly I.V. (ed.) Methods in Molecular Biology*, Systems Biology , Vol. 500, Springer-Verlag Berlin Heidelberg, Springer, pp. 113167, 2009.

[18] David C Fry. Targeting protein-protein interactions for drug discovery. In *Protein-Protein Interactions*, pages 93–106. Springer, 2015.

[19] Alexander Goncearenco, Minghui Li, Franco L Simonetti, Benjamin A Shoemaker, and Anna R Panchenko. Exploring protein-protein interactions as drug targets for anti-cancer therapy with in silico workflows. In *Proteomics for Drug Discovery*, pages 221–236. Springer, 2017.

[20] Woochang Hwang, Yongdeuk Hwang, Sunjae Lee, and Doheon Lee. Rule-based multi-scale simulation for drug effect pathway analysis. In *BMC medical informatics and decision making*, volume 13, page S4. BioMed Central, 2013.

[21] Woochang Hwang, Yongdeuk Hwang, Sunjae Lee, and Doheon Lee. Rule-based multi-scale simulation for drug effect pathway analysis. In *BMC medical informatics and decision making*, volume 13, page S4. BioMed Central, 2013.

[22] Eric Jain, Amos Bairoch, Severine Duvaud, Isabelle Phan, Nicole Redaschi, Baris E Suzek, Maria J Martin, Peter McGarvey, and Elisabeth Gasteiger. Infrastructure for the life sciences: design and implementation of the uniprot website. *BMC bioinformatics*, 10(1):136, 2009.

[23] Minoru Kanehisa, Susumu Goto, Miho Furumichi, Mao Tanabe, and Mika Hirakawa. Kegg for representation and analysis of molecular networks involving diseases and drugs. *Nucleic acids research*, 38(suppl_1):D355–D360, 2009.

[24] Samuel Kerrien, Bruno Aranda, Lionel Breuza, Alan Bridge, Fiona Broackes-Carter, Carol Chen, Margaret Duesbury, Marine Dumousseau, Marc Feuermann, Ursula Hinz, et al. The intact molecular interaction database in 2012. *Nucleic acids research*, 40(D1):D841–D846, 2011.

[25] Nicolas Le Novere and Thomas Simon Shimizu. Stochsim: modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575–576, 2001.

[26] Le Novere N and Endler L. Using chemical kinetics to model biochemical pathways. *Methods Mol Biol.*, 1021:147-67., 2013.

[27] Emanuele Perola, Lee Herman, and Jonathan Weiss. Development of a rule-based method for the assessment of protein druggability. *Journal of chemical information and modeling*, 52(4):1027–1038, 2012.

[28] Emanuele Perola, Lee Herman, and Jonathan Weiss. Development of a rule-based method for the assessment of protein druggability. *Journal of chemical information and modeling*, 52(4):1027–1038, 2012.

[29] Boris M Slepchenko and Leslie M Loew. Use of virtual cell in studies of cellular dynamics. In *International review of cell and molecular biology*, volume 283, pages 1–56. Elsevier, 2010.

[30] Michael W Sneddon, James R Faeder, and Thierry Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with nfsim. *Nature methods*, 8(2):177, 2011.

[31] Chris Stark, Bobby-Joe Breitkreutz, Andrew Chatr-Aryamontri, Lorrie Boucher, Rose Oughtred, Michael S Livstone, Julie Nixon, Kimberly Van Auken, Xiaodong Wang, Xiaoqi Shi, et al. The biogrid interaction database: 2011 update. *Nucleic acids research*, 39(suppl_1):D698–D704, 2010.

[32] Damian Szklarczyk, Andrea Franceschini, Michael Kuhn, Milan Simonovic, Alexander Roth, Pablo Minguez, Tobias Doerks, Manuel Stark, Jean Muller, Peer Bork, et al. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic acids research*, 39(suppl_1):D561–D568, 2010.

[33] Damian Szklarczyk and Lars Juhl Jensen. Protein-protein interaction databases. In *Protein-Protein Interactions*, pages 39–56. Springer, 2015.

[34] David Warde-Farley, Sylva L Donaldson, Ovi Comes, Khalid Zuberi, Rashad Badrawi, Pauline Chao, Max Franz, Chris Grouios, Farzana Kazi, Christian Tannus Lopes, et al. The genemania prediction server: biological network integration for gene prioritization and predicting gene function. *Nucleic acids research*, 38(suppl_2):W214–W220, 2010.

# Appendix A

# Software Version

This section includes software versions that were used to create and deploy the application onto a local machine. While deploying to a database/web server it may be helpful to have these details.

## A.1 Operating System

## A.2 MySQL Version/ Client Version

## A.3 Python Version

## A.4 MySQL Connector Version in Python scripts

## A.5 Django Version

## A.6 Data Tables Version