



Technologie-Zentrum Informatik

Technical Report 45

MicHanThi

Design and Implementation of a System
for the Prediction of Gene Functions
in Genome Annotation Projects

Diploma thesis

Christian Quast

TZI-Bericht Nr. 45
2007



Universität Bremen

TZI-Berichte

Herausgeber:
Technologie-Zentrum Informatik
Universität Bremen
Am Fallturm 1
28359 Bremen
Telefon: +49-421-218-7272
Fax: +49-421-218-7820
E-Mail: info@tzi.de
<http://www.tzi.de>

ISSN 1613-3773

Diploma thesis

MicHanThi -
Design and Implementation of a System
for the Prediction of Gene Functions
in Genome Annotation Projects

Christian Quast
January 13, 2006



Universität Bremen

Fachbereich 3



Max Planck Institut für Marine Mikrobiologie

Universität Bremen
Fachbereich 3
Postfach 33 04 40
28334 Bremen

Max Planck Institut für Marine Mikrobiologie
Mikrobielle Genomforschungsgruppe
Celsiusstrasse 1
28359 Bremen

Diplomarbeit

zur Erlangung des akademischen Grades eines Diplom-Informatikers
an der Universität Bremen vorgelegt von Christian Quast.

Erstgutachter: Prof. Dr. Otthein Herzog
Zweitgutachter: Prof. Dr. Frank Oliver Glöckner

Abstract

Introduction: The technological power of high-throughput sequencing has revolutionised our capabilities to examine the genetic complexity of organisms at the whole genome level. Since the first microbial genome was completed in 1995, the number of sequenced genomes and metagenomes has increased exponentially. To cope with this huge amount of sequence data, automated systems for functional assignment are urgently needed. The **MicHanThi** software predicts gene functions based on similarity searches using the NCBI nr (including SWISS-PROT) and InterPro databases and provides the human annotator with a starting point for thorough investigations.

Implementation: The first step when annotating a gene is to select reliable results for a functional description of the gene from the set of observations obtained by similarity searches. To ensure that only the most reliable observations are used for the annotation of a gene, each observation is rated by a reasoning system. The reliability of an observation is calculated based on the characteristics of the tool / algorithm which created it. To appropriately represent human knowledge about these tools, the reasoning is implemented using *fuzzy logic*. BLAST observations for a single gene are diverse in terms of potential functions and identical functions are often represented by different semantically equivalent descriptions. To derive a consistent gene product from this heterogeneous list of observations, they are grouped according to functional terms found in their description. Depending on the inconsistencies among the created groups, one or more annotations can be found by **MicHanThi**. Each annotation is assigned a reliability value to help the human annotator to decide which annotation can be “trusted” and which annotation needs further attention. Once the gene product is set, **MicHanThi** tries to add additional information such as a *gene name*, an *EC number*, and a list of *GO numbers*. These information is assigned based on SWISS-PROT and InterPro observations found in the list of observations supporting an annotation.

Results: The software was evaluated within the annotation jamboree of the marine bacterium ‘Gramella forsetii’ KT0803. Compared to the annotation created by the human annotator, about 60% of the annotations predicted by **MicHanThi** were syntactically identical and in addition, about 10 percent were semantically equivalent. The program performed best if clear rules exist. This is the case for *hypothetical* and *conserved hypothetical* genes, with or without *transmembrane regions* and *signal peptide predictions*. In these cases the program gave consistent results and outperformed the human annotator. Taking the results of **MicHanThi** into account, the human annotations could be improved significantly in the subsequent manual cross-checking phase.

Contents

1	Introduction	1
1.1	Motivation and Objectives	1
1.2	Outline	1
2	Application Background	3
2.1	Genome Annotation	3
2.2	Similarity Searches	5
2.2.1	Pairwise Sequence Alignment	6
2.2.2	Pattern / Profile (Motif) Searches	10
2.2.3	Fold Recognition [1, 2]	13
2.2.4	Comparative Genomics	14
2.3	Databases	15
2.3.1	Sequence Databases	15
2.3.2	Pattern / Profile Databases	18
2.4	Tools	21
3	State of the Art	23
3.1	Annotation Systems	23
3.1.1	Manual Genome Annotation	23
3.1.2	Computer Aided Annotation Systems	24
3.1.3	The GenDB database (v.2.0)	30
3.2	Automatic Annotation	33
3.2.1	Strategies for the Prediction of Gene Functions	33
3.2.2	Tools for the Prediction of Gene Functions	35
4	MicHanThi	41
4.1	The Algorithm	41
4.1.1	Preprocessing Observations	42
4.1.2	Rating Observations	46
4.1.3	Selecting Observations for the Annotation Process	52
4.1.4	Predicting the Gene Function - Annotation	54
4.1.5	Assigning additional Annotation Features	58
4.1.6	Genome Re-Annotation	59

4.2	Design and Implementation	59
4.2.1	Module IO	60
4.2.2	Module DATA	62
4.2.3	Module TOOLS	66
4.2.4	Module ANNOTATOR	67
4.2.5	The CQ Framework	68
4.3	Additional Tools implemented within the Diploma Thesis	70
4.3.1	MOBH - Mark ORFs based on best BLAST hit	70
4.3.2	UPGENEC - UPdate GENe name and EC number	71
4.3.3	GenannD	71
4.3.4	SPIMP	72
5	Results	73
5.1	The Test Run	73
5.1.1	The Test Setup	73
5.1.2	The KT0803 Annotation Jamboree	73
5.1.3	The Performance of MicHanThi	74
5.1.4	The Evaluation	74
5.1.5	The Cross-Checking	79
5.1.6	The Problem of Semantics	82
5.1.7	The Missing 19 Annotations	83
5.2	Limitations of the Prototype	84
5.2.1	The Rating of Observations	84
5.2.2	Prediction of Gene Functions	85
5.3	Conclusions	86
5.4	Perspectives	87
A	Rulesets	89
B	Presentations of this Thesis	91
C	Manual	93
C.1	Hardware Requirements	93
C.2	Software Requirements	93
C.3	Installation from Source Code	93
C.4	Usage	94
C.5	Configuration	95
D	CD-ROM	97

List of Abbreviations

E-value	Expect Value or Expectation Value
BIBIS	Bremerhaven Institute of Biological Information Systems
BLAST	Basic Local Alignment Search Tool
CDS	coding sequence
contig	Contiguous sequence
CPU	Central Processing Unit
DDBJ	DNA Database of Japan
DNA	deoxyribonucleic acid
DSR	Dissimilatory Sulfate Reductase
DUF	Domain of Unknown Function
EC number ..	Enzyme Commission number or Enzyme Classification number
EMBL	European Molecular Biology Laboratory
FSA	Finite State Automaton
GCB	German Conference on Bioinformatics
GHz	Gigahertz - One billion cycles per second
GI number ...	Unique Identifier in the GenBank database
GiB	Billion Information Bytes - 1GiB equals to 1024MiB
GO number ..	Gene Ontology number
GSA	Global pairwise Sequence Alignment
GUI	Graphical User Interface
HDD	Hard Disk Drive
HMAP	High-quality Automated and Manual Annotation of microbial Proteomes
HMM	hidden Markov models
HSP	high scoring pair
IUB	International University Bremen
JRE	Java Runtime Environment
KiB	Thousand Information Bytes - 1KiB equals to 1024 Bytes
LSA	Local pairwise Sequence Alignment
MGG	Microbial Genomics Group
MiB	Million Information Bytes - 1MiB equals to 1024KiB
MPI	Max Planck Institute
mRNA	messenger RNA

MSA	Multiple Sequence Alignment
NCBI	National Center for Biotechnology Information
NCBI nr	NCBI non-redundant protein sequence database
NCBI nt	NCBI non-redundant nucleotide sequence database
O2DBI	Object to Database Interface
ORF	Open Reading Frame
PDB	Protein Data Bank
PDF	Portable Document Format
PIR	Protein Identification Resource
PRF	Protein Research Foundation
RAM	Random Access Memory
RCSB	Research Collaboratory for Structural Bioinformatics
RefSeq	Reference Sequence
RNA	Ribonucleic acid
SGE	Sun Grid Engine
TREMBL	Translated EMBL
TTZ	Technologie-Transfer-Zentrum
TZI	Technologie Zentrum Informatik (Center for Computing Technologies)
UML	Unified Modelling Language
UniProt	Universal Protein Resource
XML	Extensible Markup Language

List of Figures

2.1	Illustration of the transcription / translation process of a gene into a protein as it is found in eukaryotes. Prokaryotes lack introns and the terms exon and gene constitute the same concept. Source: http://www.genome.gov/glossary.cfm?key=gene	4
2.2	The search space of the alignment of two sequence reduced into seeds	8
2.3	Extending seeds diagonally to either side.	9
2.4	Remaining HSPs after the evaluation process.	9
2.5	Extract from a multiple sequence alignment of five ORFs coding for ‘ <i>serine protease do-like precursor</i> ’ (degP). The <i>clustal</i> algorithm [3] was used to create the alignment.	10
2.6	An exemplary profile. The X-axis specifies the position in the sequence. The Y-axis shows the frequencies of the letters within the graph (amino acids) at a given position within the sequence. At position 7 should be either amino acid F or amino acid C. F and C do not summon up four bits (100%) because any other amino acid may occur at position 7 as well, it is “just” unlikely.	12
2.7	Gene neighbourhood of the dissimilatory sulfate reductase (DSR) operon in different sequenced organisms.	16
2.8	Example of the description of an entry in the NCBI nr database (<i>gi 16121437 ref NP_404750.1 </i>)	17
2.9	Homology relationships among a group of proteins. Proteins A and B are paralogous proteins while speciation of these proteins form an equivalog.	20
3.1	Subset of tables of the GenDB v.2.0 database model used by MicHanThi . a) Tables storing the results of a similarity search b) Tables holding information about the genome (the sequence and all predicted ORFs, tRNAs, rRNAs,...) c) Tables used for the annotation information d) Tables for different types of annotators (human / computer))	34

3.2	<i>AutoFACT</i> methodology. Sequences are classified into one of six annotation categories (purple boxes). The user decides which bit score cutoff to use (default 40) before a BLAST hit is considered significant. Figure taken from [4]	37
4.1	Embedding of MicHanThi in an annotation system	41
4.2	Four different cases of sequence alignments. The sequences depicted are scale free.	48
4.3	Modelling of BLAST attributes	53
4.4	Modelling of InterProScan attributes.	53
4.5	A) A list of example observations, B) descriptions split into atoms, C) non-redundant list of atoms.	55
4.6	Functional grouping of observations	57
4.7	The integration of MicHanThi with the GenDB System and external data sources and the possibilities of the user to interact with the system.	60
4.8	The package structure of MicHanThi	61
4.9	Classes of the IO module	63
4.10	Classes of the DATA module	66
4.11	Classes of the TOOLS module	67
4.12	Classes of the ANNOTATOR module	68
4.13	Classes of the cq-framework	70
5.1	Annotation of ORF orf7. The C-terminus of the protein is always matched by proteins involved in sugar binding. The N-terminus is always matched by proteins coding for <i>sensor histidine kinase/response regulator, hybrid</i>	87
B.1	Poster presented at the <i>German Conference on Bioinformatics 2005</i> (http://www.gcb2005.de)	92

List of Tables

2.1	Two unaligned random DNA sequences	6
2.2	Exemplary global alignment of the two sequences shown in table 2.1.	7
2.3	Exemplary local alignment of the two sequences depicted in table 2.1. The asterix character ('*') denotes to unaligned bases of the two sequences.	7
2.4	An exemplary pattern taken from: http://www.expasy.org/cgi-bin/nicedoc.pl?PDOC00013 . C is the lipid attachment site. Additional rules: (1) The sequence must start with Met. (2) The cysteine must be between positions 15 and 35 of the sequence in consideration. (3) There must be at least one Lys or one Arg in the first seven positions of the sequence.	11
5.1	Statistics from the comparison of the preliminary annotations created by the human annotator and those created by MicHanThi .	75
5.2	Number of matches between the preliminary annotations created by the human annotators (ha) and those created by MicHanThi (aa).	75
5.3	Detailed comparison of the preliminary annotations created by the human annotators and those created by the computer.	78
5.4	Statistics from the comparison of the final annotations created by the human annotator and those created by MicHanThi	79
5.5	Number of matches between the final annotations created by the human annotators (ha) and those created by MicHanThi (aa).	79
5.6	Detailed comparison of the final annotations created by the human annotators and those created by the computer.	81
5.8	Semantically comparable annotations within the first 100 ORFs, which are not reported by the statistics tool.	82
A.2	Ruleset used to rate InterPro observations	89
A.4	Ruleset used to rate BLAST observations	90
D.1	Contents and structure of the CD-ROM	97

Chapter 1

Introduction

1.1 Motivation and Objectives

The technological power of high-throughput sequencing has revolutionised our capabilities to examine the genetic complexity of organisms at the whole genome level. Since the first microbial genome of *Haemophilus influenzae* was sequenced in 1995 [5] the number of sequenced genomes and metagenomes has increased exponentially. Automated systems for the prediction of gene functions are urgently needed, to cope with this huge amount of sequence data.

The goal of this thesis is the development of a software tool (**MicHanThi**) for the prediction of gene functions to support the biologist in the interpretation of genomes. The functional description of genes (gene annotation) involves the integration of different kinds of information as well as the integration of information from different sources. Especially, the different kinds of information and the ever increasing number of publicly available genomic sequences make this task time consuming and complex. Therefore, the prototype implemented as part of this thesis should be able to provide the biologist with a basis for thorough investigations of the studied organism or metagenome fragment. It should be able to unambiguously identify those genes for which no functional description is possible because explicit rules exist for the annotation of such genes.

1.2 Outline

This thesis is divided into four major parts. It starts with an introduction of the **Application Background** (chapter 2). The term *genome annotation* and its relation to the functional description of genes (*gene annotation*) is discussed in section 2.1. Section 2.2 explains different approaches used to find similarities of a sequence of interest within bioinformatics databases. These databases are then introduced in the following section 2.3. The chapter ends with a brief introduction of two tools used to search the databases for similarities that have

be applied within this thesis.

The second part of this thesis offers an overview of the **State of the Art** in the field of *genomics* (chapter 3). It presents genome annotation systems which are used to aid the human annotator in the annotation of a genome (section 3.1). The focus of this section is on the introduction of the *GenDB* annotation system that provides a framework used by **MicHanThi**. The chapter ends with the presentation of different tools used for the prediction of gene functions (section 3.2).

Chapter 4 (**MicHanThi**) introduces the algorithm developed as part of this thesis. The implementation of the algorithm is explained in section 4.2. The chapter ends with the description of additional tools implemented as part of this thesis.

An evaluation of the implemented prototype is presented in the last chapter of this thesis (chapter 5). It introduces the method used to evaluate the prototype as well as it gives an overview of the prototype's performance. In sections 5.1.4 and 5.1.5, the annotations created by **MicHanThi** are compared to the human annotation of the organism 'Gramella forsetii' KT0803. The comparison is focused on the evaluation of the annotation of ORFs without an assigned function because clear annotations rules exists for this kind of ORFs. The problem of semantics when comparing annotations created by a human annotator and those created by a software tool are discussed in sections 5.1.6. Section 5.2 talks about limitations of **MicHanThi**. The chapter ends with conclusions that can be drawn from this work (section 5.3) and perspectives of future work (section 5.4).

Chapter 2

Application Background

2.1 Genome Annotation

The background of this thesis is *Genome Annotation*. The annotation of a genome is the analysis of an organism's genomic sequence according to certain criteria: the taxonomic identification of the organism according to some marker genes, the prediction and annotation of genes, and the reconstruction of the organism's metabolic capabilities.

Sequencing Before an organism can be annotated its genome must be sequenced. Prior to the introduction of new sequencing strategies and capillary sequencers in the early 1990's, it took a scientist days to sequence a few hundred basepairs. Once introduced, the throughput could be increased dramatically. At present, more than $6e^{10}$ basepairs are sequenced every year [6] (page 665) and the rate of sequencing is ever increasing.

The most popular sequencing approach today is *shotgun sequencing* [7]. First, the genome is randomly sheared into small (1-3Kb) DNA-pieces (fragmentation), which are then sequenced using the *Sanger method*¹ to produce *reads*. To obtain multiple overlapping reads, the fragmentation and sequencing steps are repeated. Computer programs assemble a contiguous sequence (*contig*) by searching overlapping reads.

ORF prediction Once a genome is sequenced, tools are applied to predict possible protein-encoding sequences (*open reading frames* - ORFs). This is the part of a genomic sequence (gene) that is transcribed to mRNA and later translated into protein [8]. The definition of the term gene however is ambiguous. For eukaryotes a gene includes non-coding sequence (introns) which is spliced out of the pre-mRNA once the gene is transcribed. Alternative splicing of introns allows for different proteins to be encoded by the same gene. An illustration of

¹reference: <http://www.csun.edu/hcbio027/biotechnology/lec3/sanger.html>

this definition of a gene is depicted in figure 2.1. Prokaryotic sequences usually lack introns. Therefore, the terms exon and gene refer to the same concept (\rightarrow gene equals ORF).

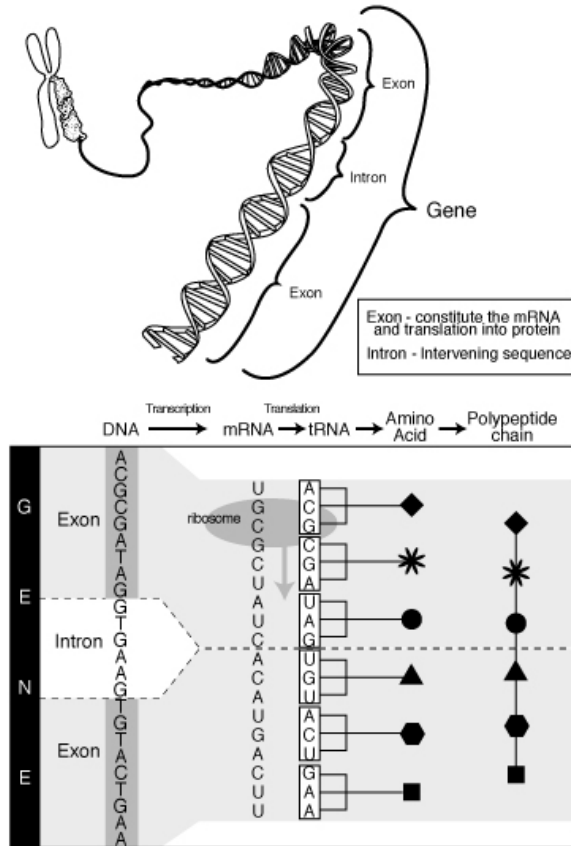


Figure 2.1: Illustration of the transcription / translation process of a gene into a protein as it is found in eukaryotes. Prokaryotes lack introns and the terms exon and gene constitute the same concept. Source: <http://www.genome.gov/glossary.cfm?key=gene>

The end of an ORFs is always identified by the same triplets of DNA bases either 'TAA', 'TAG', or 'TGA' (stop codon). If one of these triplets is encountered within the sequence the transcription of the genomic sequence is stopped. However, the triplet coding for the start of the ORF (start codon) is ambiguous. In most cases, the start codon is the triplet 'ATG' but it can be others. Furthermore, the transcription process is not always started if a start codon is encountered since 'ATG' can also code for the internal amino acid methionine. Therefore, a lot of effort is spend in the correct prediction of the start codon. In eukaryotes, prediction of protein-encoding genes is more difficult because untranslated introns have to be removed from the (pre-)mRNA using a complex

splicing mechanism. Since this process and the corresponding signals are not fully understood thus far, it is difficult to correctly predict the mature mRNA *in silico*.

But even the prediction of ORFs in bacteria is a complicated process and several different approaches / tools are currently used. Since the prediction of the start position is ambiguous, tools either predict too many ORFs or only those which are most likely real ORFs / genes (over prediction of ORFs vs. quality of the ORFs). Therefore, the quality of the predicted ORFs is rather unreliable which has to be considered when annotating the ORFs. *MORFind* is a meta ORF-finder developed by Jost Waldmann and Hanno Teeling (**Microbial Genomics Group**) to increase the quality of predicted ORFs. It combines the results of different ORF prediction tools and creates a non-redundant list of ORFs. Overlapping ORFs are considered to be errors in the ORF prediction and a sophisticated reasoning process is applied to solve discrepancies.

Gene Annotation Gene annotation is the process to associate certain information with the predicted ORFs describing their function. Among this information is: the function of the protein, a short “unique” name describing the function (*gene name*), and the classification of the ORF. The classification of an ORF can be done using different schemes. The more popular schemes are EC numbers, which classify the ORF corresponding to its metabolic pathway [9], as well as GO numbers which classify the ORF according to its molecular function, cellular component, and biological process [10]. The assignment of this set of information is the main topic of this thesis.

After the ORF prediction, all possible genes are unannotated. To derive a function for a particular ORF, its sequence is compared to already annotated genes in sequence databases (Pairwise Sequence Alignment 2.2.1). Additionally, tools can be used to assign an ORF to a certain protein family by matching its sequence to patterns or profiles describing one of the currently available protein families (Pattern / Profile Searches 2.2.2). Sometimes, fold recognition (2.2.3) is used to provide additional evidences for the function of an ORF. A recent approach to estimate the function of an ORF is *comparative genomics* (section 2.2.4), which focuses on whole genome comparisons rather than the analysis of single ORFs.

2.2 Similarity Searches

Similarity searches are the basis for the functional description of genes. Normally, a genomic sequence of interest (the query sequence) is compared to genomic sequences or a group of genomic sequences (pattern or profiles) found in various databases (see section 2.3 for details about the different databases). The different approaches which are applied to compare two or more sequences are explained in this section. The *comparative genomics* approach uses different techniques then

the other approaches introduced in this section. Nevertheless, it can be used to create a hypothesis about the function of a query sequence. Tools which utilise these approaches are introduced in section 2.4.

2.2.1 Pairwise Sequence Alignment

Pairwise sequence alignment is a scheme of writing two strings on top of each other where the characters in one position are deemed to have a common evolutionary origin (positional homology). In bioinformatics, this approach is applied to compare two DNA sequences or two protein sequences, highlighting their similarities. The sequences are arranged so that when ever possible identical bases (matches) of the sequences are placed next to each other in the alignment. If necessary, gaps (usually denoted by dashes '-') are introduced into the alignment. Gaps can be seen as deletions or insertions in the evolutionary process of a gene, whereas mismatches correspond to mutations. Table 2.2 shows an example alignment of the two random DNA sequences depicted in table 2.1. In bioinformatics,

I:	c	t	c	g	t	c	t	g	c	a	t	c	c	t	c	a	a				
II:	c	t	g	g	t	a	t	c	t	g	c	a	c	a	t	g	g	g	c	a	a

Table 2.1: Two unaligned random DNA sequences

pairwise sequence alignment is the most important approach to derive a function for an unidentified query sequence. Conserved regions, these are long stretches of matches, within a sequence alignment provide clues for the functional description of the query sequence. These conserved regions can also be used to describe the evolutionary distance between two sequences as shown in [11]. Broadly, two types of pairwise sequence alignments can be distinguished, *local sequence alignment (LSA)* and *global sequence alignments (GSA)*. An extension of the pairwise sequence alignment is the *multiple sequence alignment (MSA)* aligning more than two sequences.

Global Sequence Alignment (GSA) A global sequence alignment of two sequences is an alignment that spans along their entire length and it is most useful for aligning / finding closely related sequences. Gaps are introduced into the aligned sequences to make up for differences in length of the two sequences. A drawback of the GSA is the inability to handle evolutionary mechanism such as *domain shuffling*. The function of a protein is affected by its domain composition. The rearrangement of these domains within the evolutionary process is called *domain shuffling*. This as well as the fact that closely related sequences are also found by algorithms implementing *local sequence alignment* make the global sequence alignment a deprecated technique. The Needleman-Wunsch al-

gorithm [12] is the most known algorithms for the global pairwise alignment of two sequences.

I:	c	t	c	g	-	-	t	c	t	g	c	a	t	c	c	t	-	-	-	c	a	a
II:	c	t	g	g	t	a	t	c	t	g	c	a	-	c	a	t	g	g	g	c	a	a

Table 2.2: Exemplary global alignment of the two sequences shown in table 2.1.

Local Sequence Alignment (LSA) The alignment of subsequences of two sequences is called local sequence alignment. This approach aligns a subset of characters of sequence I with a subset of characters of sequence II. Neglecting the positions of the subsequences within their parent sequences allows for greater flexibility regarding evolutionary mechanisms such as *domain shuffling* compared to GSA. A local alignment of the two sequences taken from table 2.3 using the Smith-Waterman algorithm [13] is shown in table 2.3.

I:			*	*	*	*	t	c	t	g	c	a	*	*	*	*	*	*	*	*
II:	*	*	*	*	*	*	t	c	t	g	c	a	*	*	*	*	*	*	*	*

Table 2.3: Exemplary local alignment of the two sequences depicted in table 2.1. The asterix character (‘*’) denotes to unaligned bases of the two sequences.

BLAST - The *Basic Local Alignment Search Tool* (BLAST) [14, 15] algorithm is the most known algorithm for the local alignment of two sequences. Unlike the Smith-Waterman algorithm, it returns a number of statistically significant alignments rather than just the “best” one. Another difference between the two algorithms is that the Smith-Waterman is guaranteed to find the best local alignment between two sequences while BLAST uses a heuristic to reduce the search space (as it is depicted in figure 2.2). Using a heuristic increases the search speed at the cost of sensitivity. This means that an optimal alignment between two sequence may not be found. However, the alignment returned by the algorithm should be very close to the optimal alignment regarding its score. The heuristic used by BLAST uses three layers to refine potential alignments (*high scoring pairs* - HSPs): seeding, extension and evaluation. .

Seeding: BLAST uses the concept of *words* to reduce the search space (figure 2.2). A word is a number of characters that are considered to be ‘one’ character. For example, if a word is defined to be three letters long, then the sequence ‘BLAST’ has three words: ‘BLA’, ‘LAS’, and ‘AST’. Before BLAST starts to align two sequences, it creates a list of all words that are

common to both sequences. Some alignments may not contain any identical words, therefore BLAST use the word's *neighbourhood* to compare two sequences. The neighbourhood of a word is defined as the substitution of any character within the word that yields a score higher then T when the two words are compared via a scoring matrix. By adjusting the value of T , the neighbourhood of a word can be increased or decreased. The size of the search spaces changes accordingly. Another attribute that controls the size of the search space is the word length W . Choosing the “right” values for T and W regarding the scoring matrix used, is critical for the performance of the BLAST algorithm. Regions of the search space containing word matches are called *seeds*.

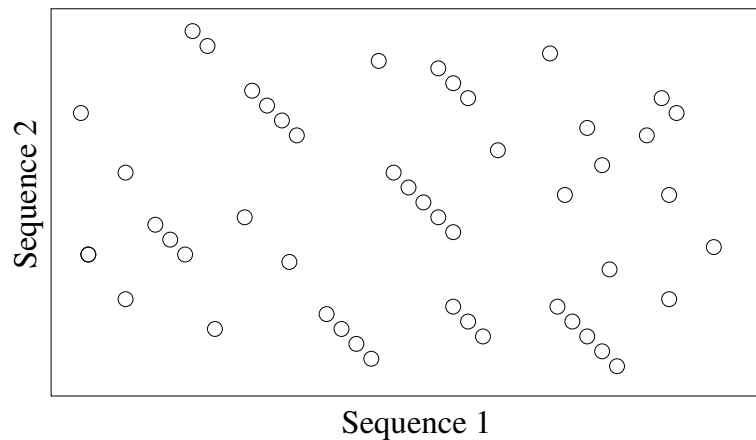


Figure 2.2: The search space of the alignment of two sequence reduced into seeds

Extension: Seeds are extended diagonally in both directions in order to find the longest local alignment starting from a certain seed (figure 2.3). The ends of the alignments are reached, if too many mismatch would occur in the alignment. Smith-Waterman “knows” when to end a local alignment because it evaluated the whole search space before looking for the “best” local alignment. However, BLAST only searches a subset of the search space and therefore needs some additional mechanism to find the end of a local alignment. BLAST uses *simulated annealing* [16] (page 157) to do this. Once the first mismatch is encountered in the alignment, the current score R of the alignment is kept. Additional words are added to the alignment, until the score of the new alignment drops beneath R minus some threshold X . The algorithm may get stuck in a local maximum if the value of X is chosen too low. If the value chosen for X is too large, then the algorithm may search for too long without finding a significantly better score. After terminating, the alignment is trimmed back to its maximum score.

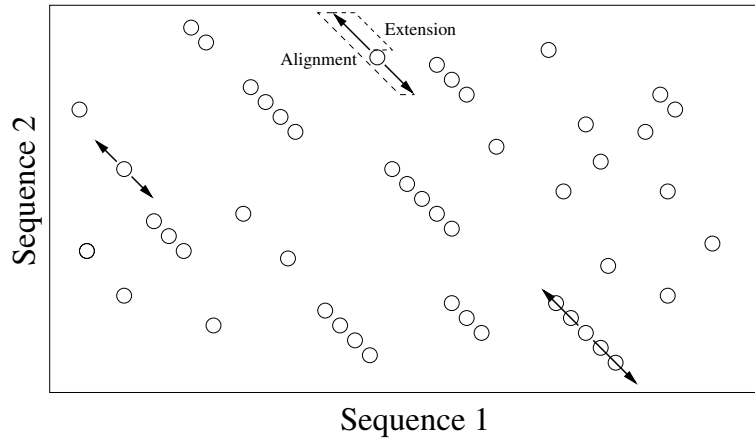


Figure 2.3: Extending seeds diagonally to either side.

Evaluation: Once the seeds are extended, the alignments are evaluated to determine if they are statistically significant. Simplified: Significant alignments (termed HSP) are those that have a score larger than a given threshold S . The goal of the evaluation is to remove as many overlapping seeds as possible and then select a *consistent* group of HSPs that stretches from the upper left corner of the search space to the lower right (figure 2.4). A group of HSPs is consistent if no word is contained in two or more HSPs and each HSP starts to the lower right of the previous HSP. Finally, the best scoring consistent group of HSPs is returned as the gapped local alignment of two sequences.

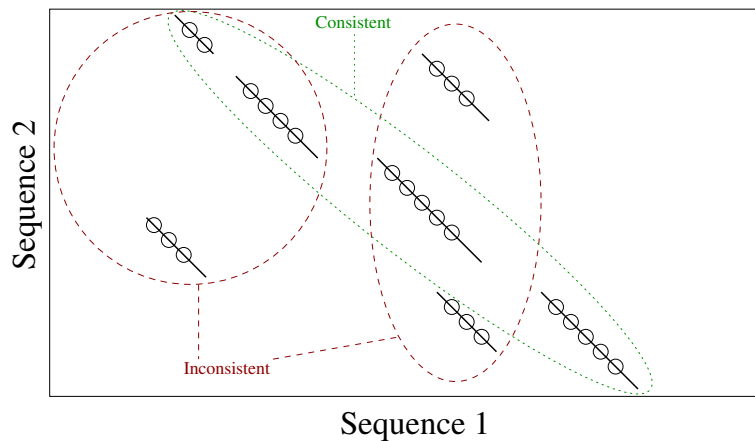


Figure 2.4: Remaining HSPs after the evaluation process.

A measure for the statistical importance of the alignment returned by the BLAST algorithm is the Expectation value (or Expect value) short E-value [17]. This is the number of alignments expected by chance E during a sequence database search of search space $m \times n$, where m denotes the length of the query sequence and n is the size of the database in characters (the length of the concatenation of sequences within the database).

Multiple Sequence Alignment (MSA) Multiple sequence alignments are used to identify conserved regions among a group of sequences. These conserved regions are often used to create profiles describing a group (or family) of proteins which are used by *Hidden Markov models* or the *PSI-BLAST* program to search for homologous sequences in databases. Furthermore MSAs are used to build phylogenetic trees. Multiple sequence alignment is computationally difficult and is classified as an NP-Hard problem.

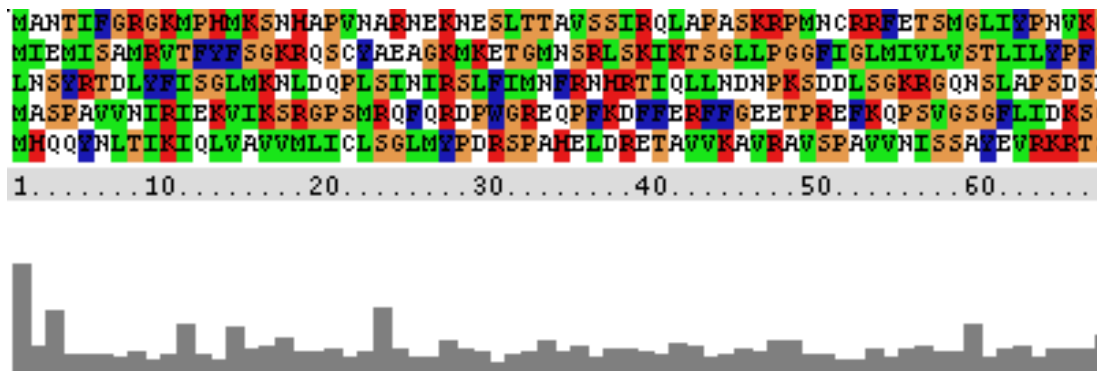


Figure 2.5: Extract from a multiple sequence alignment of five ORFs coding for ‘serine protease do-like precursor’ (degP). The *clustal* algorithm [3] was used to create the alignment.

2.2.2 Pattern / Profile (Motif) Searches

Pattern or profile searches predict the function of a query sequence by comparing the query sequence with entries in a database as do sequence similarity searches. Unlike sequence similarity searches, pattern and profile searches do not compare two sequences directly, instead they compare the query sequence to a pattern or profile describing a domain or family of proteins. This approach seems to be worth while because different domains of a protein are subject to different selective pressures [18]. This means that some parts of a protein are more conserved among a group of proteins than others. Patterns or profiles which characterise the conserved regions of a protein family can be observed by aligning multiple sequences of the same family using tools that implement a MSA algorithm.

Patterns are regular expressions describing each position of the MSA that is relevant to identify a protein family. Each position of the pattern represents one or more characters of the alphabet that are observed at the position of the MSA. Only these characters are allowed to occur at that position in a query sequence. If a character is found in the query sequence, that is not represented by the pattern for that particular position, then the query sequence is called a mismatch. This problem can be solved by allowing a number of mismatches within the pattern. Another problem of patterns is that patterns do not consider the frequency of the occurrence of a character at a particular position within the MSA. If a MSA constitutes at a specific position X of the characters ‘A’ (found in 75% of the sequences), ‘B’ (15%), and ‘C’ (10%), then a query sequence S_1 with character ‘A’ at position X should score better than a query sequence S_2 with characters ‘B’ or ‘C’ at position X that is otherwise equal to S_1 . This is not possible with patterns. An example of a pattern is shown in table 2.5.

{DERK}(6) - [LIVMFWSTAG](2) - [LIVMFYSTAGCQ] - [AGS] - C

Table 2.4: An exemplary pattern taken from: <http://www.expasy.org/cgi-bin/nicedoc.pl?PDOC00013>. C is the lipid attachment site. Additional rules: (1) The sequence must start with Met. (2) The cysteine must be between positions 15 and 35 of the sequence in consideration. (3) There must be at least one Lys or one Arg in the first seven positions of the sequence.

Profiles like Patterns describe conserved regions of a MSA. Unlike patterns, profiles specify for each position within the conserved region the probability for each character of the input alphabet by which it may occur at that particular position. Hence, profiles implicitly allow mismatches at any given position of the profile because it is “just” more likely for some characters to occur at some position of the alignment between a query sequence and a profile. Characters which do not occur in the MSA are assigned a probability close to zero. This means that it is very unlikely that one of those characters will occur. An exemplary profile is depicted in figure 2.6

Hidden Markov Models (HMM) Before being applied in sequence analysis in 1994 [19], Hidden Markov models were already being applied to problems in speech recognition [20]. Hidden Markov models are *probabilistic finite state automations (FSA)* that represent a *Markov chains*. The states of the FSA are considered to be matches, insertions, or deletions. For each state a substitution matrix exists, which describes the probability of each character of the input alphabet to be “emitted” by a certain state (*emission probability*). The transition from one state to the next is based on probabilities (*transition probability*) rather than a true / false decision made in a *deterministic FSA*. A sequence of “visited” states that describe the alignment of a query sequence to a profile is called *state*

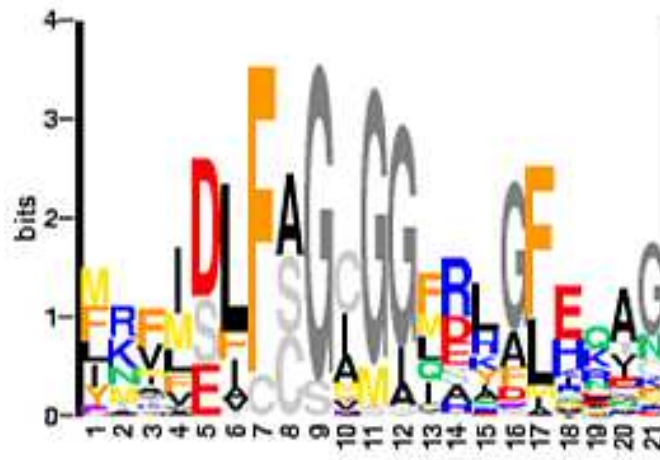


Figure 2.6: An exemplary profile. The X-axis specifies the position in the sequence. The Y-axis shows the frequencies of the letters within the graph (amino acids) at a given position within the sequence. At position 7 should be either amino acid F or amino acid C. F and C do not sum up four bits (100%) because any other amino acid may occur at position 7 as well, it is “just” unlikely.

path. An HMM is a *Markov chain*, this means that the transition from the current state to the next state according to the input sequence is solely based on the current state of the FSA. In other words, to predict the future the FSA only needs to know the present, not the past. This is also the main weakness of HMMs because they do not deal well with correlations between residues. An example is the analysis of RNA secondary structure where HMMs are usually inappropriate because RNA base pairs include long-range pairwise correlations; one position may be any residue but the base-paired partner must be complementary [21]. The field of application of HMMs in bioinformatics includes gene finding, profile searches, and regulatory site identification. In profile searches, HMMs are used to associate residues of the query sequence with homologous residues of a profile describing a certain protein family.

The probability $P(S, \pi | HMM, \theta)$ that an HMM with parameter θ generates an observed sequence S and a state path π is the product of all emission probabilities and transition probabilities used. For complex problems, an HMM generates a large number of paths. To choose the optimal path, *Bayesian probability theory* can be applied because an HMM is a *full probabilistic model*. That is, the model parameters and the sequence ‘scores’ are all probabilities. Many algorithms can be used to find the optimal state path. An efficient algorithm that is guaranteed to find the most probable state path is the *Viterbi algorithm* [20].

What is hidden about a Hidden Markov Model - The HMM generates two strings of information. The state path and the observed sequence emitted by the HMM. A Markov model is called hidden if the state path is hidden from the user. In this case, only the observed sequence is reported as a result of the analysis.

2.2.3 Fold Recognition [1, 2]

According to literature, a sequence alignment of two sequences should show at least 30% of matches to be able to unambiguously derive a function for the query sequence. Pattern or profile searches expect certain regions of a group of proteins to be highly conserved. If these two criteria are not met, then a function should not be assigned to a query sequence. The fold recognition provides a measure to assign a function to a query sequence independent of the approaches presented above. The assignment of functions derived from fold recognition is based on the assumption that the function is conserved on the 3D level better than on the sequence level [22]. Structures of proteins with at least 50% matches on the sequence level will be closely related. If the sequence similarity between two sequences drops below 20% then there will be large structural differences that are impossible to predict. However, the structure of the active site of distantly related proteins can still be very similar and a functional prediction based on this similarity is possible.

Two categories of fold recognition can be distinguished: those that validate the predicted three-dimensional (3D) structure of a protein and those that predicted the function of a query sequence. To assign a function to a gene based on fold recognition, its 3D structure has to be known. The structure can be either experimentally identified (e.g. using x-rays) or predicted *in silico*. The prediction of the 3D structure is done by first finding the closest relative in a database containing proteins with known 3D structures (e.g. PDB [23] which contains about 34,500 structures at the moment ²) based on a sequence alignment and then manipulating the 3D structure according to the differences in the sequence alignment. Methods of the first category mentioned above can be used to assign a quality measure to these predicted structures. The quality values can be considered when searching for similar 3D structures and deriving a function for the query sequence.

One important drawback to this approach is the low number of structural descriptions found in databases. Even though this number constantly increases, the number of known proteins increases even more rapidly, resulting in an ever-larger number of proteins that have no known 3D structure. Therefore, the prediction of 3D structures *in silico* is rather unreliable because the coverage of the search space is very low. A function derived from this approach should be

²A daily update of this number can be found at: <http://www.rcsb.org/pdb/Welcome.do>

considered with caution and its reliability greatly depends on the reliability of the 3D structure predicted for the query sequence.

2.2.4 Comparative Genomics

The large amount of sequenced genomes available today allows for new techniques to analyse genomic sequences. One of these new techniques is the *comparative genomics* approach. Comparative genomics focuses on the whole genome rather than individual genes within one genome. Two categories to analyse genomes can be distinguished, the *gene independent* and the *gene dependent* analysis methods. A detailed summary and application of this approach can be found in the work of Michael Richter [24].

Gene Independent Analyses: Methods which belong to the group of gene independent analysis methods (*Genome Linguistics*) focus on the structure of nucleotide sequences and the positioning of bases within a sequence. These methods are especially applied to analyse fragments of prokaryotic sequences. Gene independent analysis methods are used, among other things, for the prediction of the replication start (origin) and replication stop (terminus) positions within circular genomes. Another analysis method is the TETRA approach as described in [25]. This method uses the distribution of tetranucleotides³ within fragments of genomic sequences to assign these fragments to a certain species or to a taxonomic group. In metagenome studies, these methods can be used to assign genomic fragments to artificial organisms (*organism-bins*). This approach is implemented by Marcel Huntemann as part of his diploma thesis at the University of Bremen and the **Microbial Genomics Group** at the Max Planck Institute for Marine Microbiology.

Gene Dependent Analyses: Gene dependent analysis methods are applied to amino acid sequences. More precisely, these methods compare the ORFs predicted within one genome with all predicted ORFs of at least one other genome. To compare ORFs of different genomes, sequence similarity search tools such as BLAST are used. The results derived from these comparisons can be taken as additional evidences for the functional description of ORFs because it is possible to discriminate between orthologous and paralogous sequences. Furthermore, gene dependent analysis methods take into account:

- the percentage of a sequence coding for e.g. an ORF,
- the arrangement of ORFs within the genome,

³A tetranucleotide is a four character subsequence of nucleotides within the genomic fragment. 256 (4^4) tetranucleotides exist in a four character alphabet, ranging from *aaaa* to *tttt*

- the average length of an ORF,
- the number of ORFs, for which a similarity could be found in sequence databases,
- the number of ORFs, for which a prediction of the function is possible, and
- the number of predicted rRNAs and tRNAs.

Even though a large number of genomes have been sequenced today, significant similarities can only be found in bioinformatics databases for about 40 to 60 percent of the ORFs predicted in a prokaryotic genome. Comparative genomics can be used to help to understand the function of ORFs without similarities in databases because functional related ORFs tend to be clustered on the genome (operon) [26, 27]. If such an operon can be identified in a newly analysed genome and some of the genes mandatory for that operon are missing, then ORFs missing a functional prediction within the operon's gene neighbourhood may code for the missing genes.

Figure 2.7 shows the operon structure of the dissimilatory sulfate reductase (DSR) cluster. This cluster can be divided into two larger operons⁴ (indicated by the rho-independent transcription terminators): the operon *dsr*[ABD] and the operon yellow ORF, *dsr*[MKJ], and the two ORFs following *dsr*[MKJ]. The two organisms *Desulfitobacterium hafniense* and *Chromatium vinosum* use a different substrate, therefore this organisms do not need the *dsrD* ORF. The yellow ORF is a unique gene which only occurs in this group of organisms. The two fosmids are genomic fragments and its host organism is unknown. In *Fosmid Anke42c9*, the yellow ORF is missing. Comparing the genomes of the different organisms with comparative genomics techniques yields that the grey ORF in front of *dsrM* may be an orthologous of the yellow ORF found in the other organisms. Supporting this assumption is the fact that the sequences of the yellow ORFs within the different genomes are rather diverse. Information presented in this paragraph are based on discussions between Michael Richter and the author of this thesis. The information is directly related to the work of Michael presented in [24]

2.3 Databases

2.3.1 Sequence Databases

DDBJ [28] / EMBL [29] / GenBank [30]: The databases DDBJ, EMBL, and GenBank are the *primary databases* which collect nucleotide sequences. Whenever a paper is published that presents new sequence information, the sequences must be deposited in one of these databases. Hence, the DDBJ, EMBL,

⁴An operon is a group of genes that are contiguously transcribed.

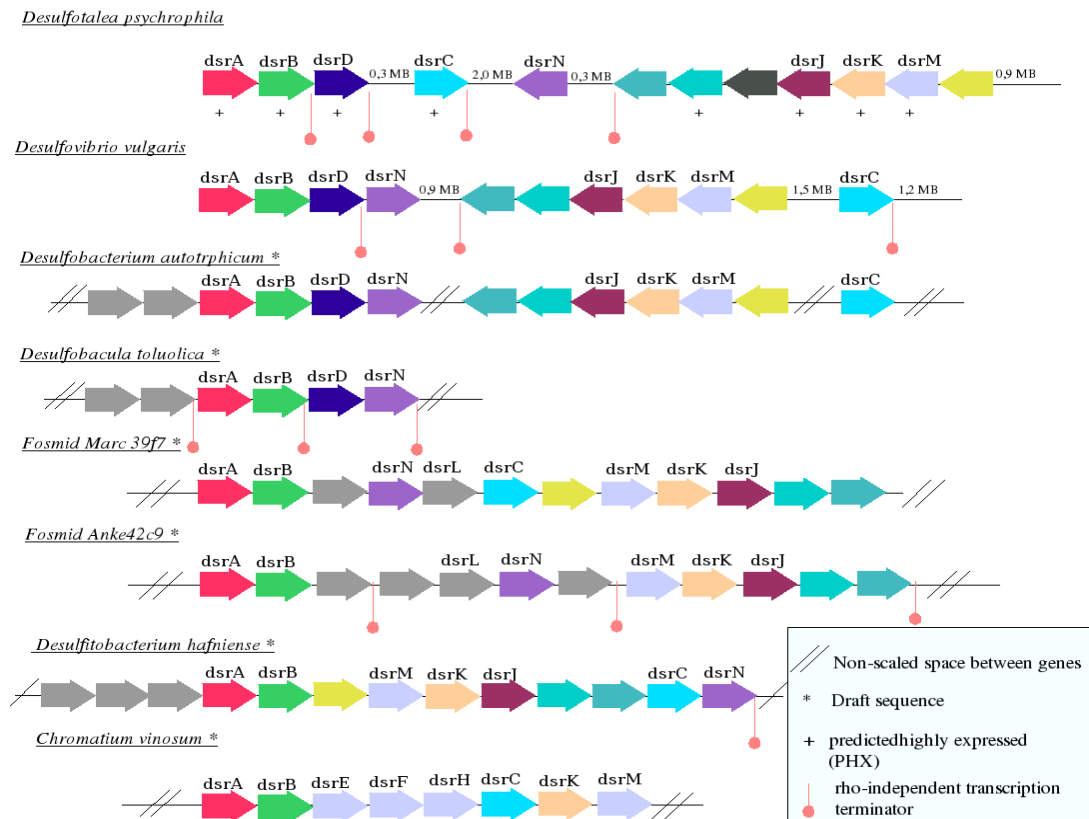


Figure 2.7: Gene neighbourhood of the dissimilatory sulfate reductase (DSR) operon in different sequenced organisms.

and GenBank databases are comprehensive collections of all publicly available sequences. The databases are not curated to be able to cope with the huge amount of available sequences at present (e.g. more than 100 billion bases in GenBank⁵) and its exponential growth. All three databases exchange newly added entries on a daily basis to ensure timeliness.

Nucleotide sequence databases are the basis for any kind of data mining. Protein sequence databases like TrEMBL keep translations of all protein-encoding sequences (CDS) found in the primary databases. Most of the protein sequence databases are manually curated and provide more quality and additional information over comprehensiveness. Pattern and profile databases are build from protein sequence databases and use their content to their description of protein families regarding accuracy and comprehensiveness.

⁵source: <http://www.ncbi.nlm.nih.gov/Genbank/index.html>

Data Integration - The three primary databases are integrative, which means that all three databases store information of the same kind. To guarantee that the information is comparable, the database maintainers agreed to use the same data definitions (*definition sharing* form of data integration). Whenever the shared definition is changed, all three databases have to change all their records accordingly. This solves the problem of the *definition copy* and the *value copy* forms of data integration, where the semantically meaning of data in the different databases may independently change over time. A drawback of the definition sharing approach is that some resources may be outdated and there is no way of telling which of the resources contains the most recent records, other than comparing the entries of all databases.

Non-Redundant Databases - The most known databases for alignment based similarity searches are the *NCBI nt* and *NCBI nr* databases. Both databases are a non-redundant collection of entries found in the different databases. Multiple entries describing the same sequence found in different databases merged into a new entry. The description of the new entry contains the description of each merged entry, separated by the merged entry's unique identifier in the GenBank database (GI number). An example of an NCBI nr entry is shown in figure 2.8. Its structure is explained in detail in section 4.1.1. The NCBI nt database integrates

```
putative membrane protein [Yersinia pestis C092] gi|45440854|ref|NP_992393.1|
putative membrane protein [Yersinia pestis biovar Medievalis str. 91001]
gi|22126919|ref|NP_670342.1| hypothetical protein y3042 [Yersinia pestis KIM]
gi|51595516|ref|YP_069707.1| putative membrane protein [Yersinia
pseudotuberculosis IP 32953] gi|21959957|gb|AAM86593.1| hypothetical [Yersinia
pestis KIM] gi|45435712|gb|AAS61270.1| putative membrane protein [Yersinia
pestis biovar Medievalis str. 91001] gi|51588798|emb|CAH20412.1| putative
membrane protein [Yersinia pseudotuberculosis IP 32953]
gi|15979204|emb|CAC89982.1| putative membrane protein [Yersinia pestis C092]
gi|25510076|pir||AC0140 probable membrane protein YP01140 [imported] -
Yersinia pestis (strain C092)
```

Figure 2.8: Example of the description of an entry in the NCBI nr database (*gi|16121437|ref|NP_404750.1|*)

nucleotide sequences while the NCBI nr database integrates protein sequences. The NCBI nr database contains sequences from

- the Protein Research Foundation (PRF) database [],
- the Protein Identification Resource (PIR) database [31],
- the RCSB Protein Data Bank (PDB) [23],
- the NCBI RefSeq database [32],
- the SWISS-PROT database [33], and

- translations of the CDS found in the three primary sequence databases DDBJ, EMBL, and GenBank.

TREMBL [34]: The TREMBL database is a supplement to the SWISS-PROT database which is not curated in order to be comprehensive. It contains translations of all CDS found in primary databases which are not already in SWISS-PROT. Additionally, it contains sequences extracted from literature and sequence that were submitted to SWISS-PROT but which have not yet been added to the SWISS-PROT database.

SWISS-PROT [33]: The SWISS-PROT database contains only annotated protein sequence. Sequence entries in the SWISS-PROT database originate from three different databases: the protein sequence database *Protein Identification Resource* (PIR) [31], translation of entries from the EMBL nucleotide sequence database, and literature. SWISS-PROT emphasises three criteria by which it tries to distinguish itself from other protein sequence databases: annotation, minimal redundancy, and integration with other databases. Each sequence entry contained in SWISS-PROT is manually curated and revised by an expert for the protein family a sequence is member of. Single entries or a group of entries (of the same protein family) are periodically updated if new information is available. The SWISS-PROT team reduces redundancy in the database by merging separate entries of the same sequence found in the source databases. SWISS-PROT entries contain cross-references to external databases which provide further information. At present, about 100 external databases are cross-referenced by SWISS-PROT⁶.

Among other databases, SWISS-PROT and TREMBL are integrated by the newly established UniProt database [35].

2.3.2 Pattern / Profile Databases

Blocks [36] Blocks are **ungapped** regions of a multiple sequence alignment which describe related proteins. Entries in the Blocks database are automatically created from patterns found in the PROSITE database. These patterns are then refined by aligning additional proteins to the MSA found in the SWISS-PROT database. Recent versions of the Blocks database use InterPro as an additional source of highly conserved regions among proteins of the same family.

Pfam [37] The Pfam database contains multiple sequence alignments and hidden Markov models derived from these alignments. Pfam is divided into two databases: Pfam-A and Pfam-B. Pfam-A is a manually curated database and contains the descriptions of more than 8000 protein families. About

⁶source: <http://www.expasy.ch/cgi-bin/lists?dbxref.txt>

75% of all proteins which can be found in public databases have at least one match in this database⁷. Pfam-B is a collection of automatically created MSAs and HMMs describing families taken from *PRODOM* that do not overlap with protein families in Pfam-A. Pfam-B is of lower quality than Pfam-A but it gives a more comprehensive coverage of known proteins (an additional 19% to the protein covered by Pfam-A). A protein may belong to more than one Pfam family.

PRINTS [38] The PRINTS database is a collection of *fingerprints* of protein families. A fingerprint is a group of motifs describing the same protein family. According to the PRINTS user manual, a motif is any conserved element of a multiple sequence alignment whose function or structure is known⁸. It is unclear whether a motif is a pattern or a profile. Since PROSITE calls patterns motifs [39] it is likely that a fingerprint is a pattern as well. Fingerprints are derived from MSAs created by manual alignment tools and only a small number of sequences are used to create the initial alignment. Fingerprints are then refined by scanning the SWISS-PROT and TREMBL databases for new members of a protein family and iteratively adding sequences to the MSA.

ProDom [40] ProDom is a database of protein families automatically created from entries in the SWISS-PROT and TREMBL databases. Proteins found in these databases are pairwise aligned, in order to find a set of homologous proteins. Then multiple sequence alignments are created based on the sets of homologous proteins. A consensus sequence is calculated and finally a ProDom ‘domain’ is derived from the consensus sequence. ProDom domains normally correspond to domains found in protein families, but domain boundaries should be treated with caution. It is unclear if the ProDom database uses patterns or profiles for the description of conserved regions in a MSA.

PROSITE [39] PROSITE is the oldest database that uses patterns to describe a family of proteins. Its development started in 1988 and it was released in 1991. PROSITE is mainly based on patterns derived from multiple sequence alignments. Patterns, which were published in literature have also been added to the database. Pattern found in the PROSITE database are designed to describe a protein family as specific as possible to reduce the number of false positives to a minimum while being sensitive enough to include most of the proteins that belong to the family. The patterns described by PROSITE are periodically reviewed by the authors.

⁷source: <http://www.sanger.ac.uk/Software/Pfam/>

⁸source: <http://umber.sbs.man.ac.uk/dbbrowser/PRINTS/printsman.html>

SMART [41] The SMART database is specialised on signal proteins which are inadequately represented by the SWISS-PROT and Pfam databases. This type of proteins is often composed of multiple domains which complicates the identification of such proteins in a genome annotation project. Most of the profiles included in the SMART database had been published previously. The multiple sequence alignments, these profiles are based on, have been refined according to certain constraints: minimisation of insertions and deletions in the conserved regions of the MSA, optimisation of amino acid property conservation, and closing of unnecessary gaps.

TIGRFAMs [42] The TIGRFAMs database contains manually curated protein families. These families are based on hidden Markov models derived from multiple sequence alignments. The database maintainers introduced the term *equivalog*. A group of equivalog proteins necessarily share the same function, unlike members of a protein family which could be paralogous proteins. Proteins which are by definition not orthologous like proteins related by lateral gene transfer can be equivalogs as well. Figure 2.9 describes a protein family that contains two equivalog ‘subfamilies’. Proteins A and B are paralogous proteins while speciation of either protein share the same function and are therefore equivalog.

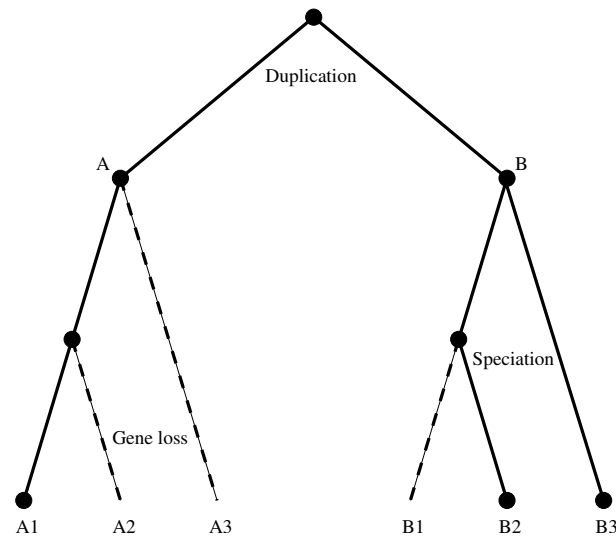


Figure 2.9: Homology relationships among a group of proteins. Proteins A and B are paralogous proteins while speciation of these proteins form an equivalog.

InterPro [43]: Like the cooperation between the maintainers of the databases DDBJ, EMBL, and GenBank, InterPro is an integrative database. Among others, the InterPro database integrates the data contained within the pattern and

profile databases mentioned in this section. The integration of data is done on the database level by keeping a local copy of each data set contained in one of the integrated databases (*value copy* form of data integration). Semantically equivalent data sets from the different sources are then integrated into a new InterPro entry to provide a common view on the different sources of information. An InterPro entry provides cross-references to all entries integrated by this entry.

2.4 Tools

Observations generated by two different tools are used by **MicHanThi** to derive a function for an ORF: *NCBI BLASTP* [14] and *InterProScan* [44]. NCBI BLASTP is used to search the NCBI nr databases for sequence similarities. It is an implementation of the BLAST algorithm developed by the NCBI. Several other implementations of the BLAST algorithm exist. Among the more widely known are the WU-BLAST implementation and an implementation optimised for PowerPC processors from IBM and Motorola featuring an *Altivec* unit. InterProScan is a meta tool which is used to scan a query sequence against protein signatures found in the InterPro database. It uses different tools based on the InterPro member database it searches. To find observations within the Pfam database, which is the only type of InterPro observations currently considered by **MicHanThi**, InterProScan uses the *hmmpfam* tool that is part of the HMMER package. This tool is an implementation of hidden Markov models used to match a sequence against a protein domain or family profile.

Chapter 3

State of the Art

3.1 Annotation Systems

This section provides an overview of commonly used annotation systems. The annotation systems are presented in chronological order, showing how such systems evolved over the years. The section begins with the very basic approach of manual genome annotation. It then explains different annotation approaches ranging from basic systems that only automate simple tasks to systems with advanced reasoning. The section ends with an in-depth introduction of the GenDB system (version 2), with special focus of underlying database schema and data model.

3.1.1 Manual Genome Annotation

The most basic approach to annotating genomes or genes is to annotate manually. The genome is represented by a list of genes stored in a spreadsheet or similar files. The structure of the spreadsheet varies from annotation project to annotation project, or it may even differ from annotator to annotator. All tools that are used to analyse the genes are run by hand or semi-automatic scripts. Often on-line tools made available by the database providers are used (see section 2.3 for more details on the different databases). Manual annotation is time-consuming and it can be very tedious to transfer data from the website to the spreadsheet. Most of the time the results of the on-line tools used for similarity searches are printed for archival and purposes of future verification. Different types of data, i.e. data from expression profiling, biochemistry, and search tools can not be cross-referenced easily, leaving the annotator with several sources to consider. This makes it more complicated for the annotator to validate his assumptions using the data from different sources. Also, it is very difficult for the annotator to find relations between a genes because every gene is looked at separately, and this approach does not provide an sufficient overview of the genome and the arrangement of the genes within.

3.1.2 Computer Aided Annotation Systems

The increasingly obvious drawbacks of manual genome annotation, and the growing number of available genomic sequences lead to the development of systems to aid the human annotator. The main objective of these systems is to automate most of the repetitive tasks, like the prediction of open reading frames (ORFs), similarity and motive searches, and the prediction of additional evidence for coding, e.g. signal peptides and trans membrane helices. Further objectives were the accessibility and cross-referencing of data, and the storage and retrieval of results. Therefore most of the systems use a database engine as their back-end. The use of a database engine makes it easy to query the database and to provide relationships between the different types of data. Overall, such systems are supposed to simplify the annotation process, make it easier to annotate in teams, and make more information accessible and cross-referenced.

A tool for the prediction of gene functions benefits from a framework that stores data, and offers an interface for the user to interact with. The major advantages for function prediction tools are the coherent data model and the automated tool pipeline. The discussion of the annotation systems presented in the following paragraphs will be focused on the reason why it was developed, the type and structure of the database used, the tool pipeline, and the front-end offered.

***GeneQuiz* [45]:** The *GeneQuiz* system was developed in 1994 by the Protein Design Group at EMBL Heidelberg¹. The development of *GeneQuiz* was driven by the experiences gained during the annotation of 171 proteins of *Saccharomyces cerevisiae* (chromosome III), and the arising need for high-throughput methods in computational genome analysis. At that time no genome had been sequenced completely. The first genome was finished in 1995[5]. The system is divided into a back-end, used to store the data, and run automated tasks such as similarity searches, and a front-end for user interactions.

The back-end is written in PERL and stores the data in a database engine called *RDB*². The database is designed to be independent of the analysis tools used. To achieve this independence the results are saved to a common directory before being parsed and written to the database. Additionally the back-end automates some of the more repetitive tasks. These automated tasks include the update of local databases used for similarity searches, the indexing of local databases for faster information retrieval, and the search for similarities in these databases. Another task the back-end performs, is the creation of database cross-references if such references are part of the downloaded database itself. The front-end is written in C++. It displays the data in tabular form, and allows

¹European Molecular Biology Laboratory Heidelberg (<http://www.embl-heidelberg.de>)

²Documentation can be found at <http://www.sander.ebi.ac.uk/RDB/RDB.html> but the link is inaccessible

the user to browse the alignments and scores of similarity searches, and the documentation available in sequence and bibliographic databases. The front-end provides a command-line interface to the database as well. This interface can be used to write additional tools for further data processing.

The *GeneQuiz* system does not provide any tools for the prediction of gene functions.

MAGPIE [46, 47]: *MAGPIE* is the abbreviation for *M*ultipurpose *A*utomated *G*enome *P*roject *I*nteraction *E*nvironment and it was developed by Gaasterland and Sensen in 1996. By the end of 1996, four complete genomes were available and it was anticipated that fifty more genomes would be completed within the next five years.

Although the dramatic increase in genome data was foreseeable at that time, the *MAGPIE* system does not use a database engine to store data. According to its developers, a database engine was not chosen because a static database format has to be defined before data can be collected, and therefore the database engine poses an overhead. Additionally, they argue that *MAGPIE* can be adapted to use new analysis tools more quickly. The system is designed around two independent daemons. The two daemons execute rules written in Prolog to preprocess the data. *MAGPIE* is the only system that uses advanced reasoning to analyse the data and that points out problems that may have occurred during the processing steps. The first daemon is the data-collection daemon. It imports new sequences placed in the input directory if necessary converts these sequences to the FASTA format, and copies each formatted sequence to its appropriate “group” directory. Gene prediction is done by taking the furthest upstream start codon for each stop codon in each frame with a minimum length of 100 amino acids³. For each predicted gene, analysis tools are run either locally or on remote servers. Remote tools are started by HTTP requests sent to the server. The data-collection daemon then waits for the results of the analysis before a new request is sent. This process would be feasible if only a few predicted genes had to be analysed, but in today’s genome projects thousands of predicted genes have to be analysed with multiple tools. *MAGPIE* is the only system known to the author to use remote tools in its analysis pipeline. The second daemon is the analysis-and-report daemon. It analyses the tool results and generates ASCII and HTML reports. The *MAGPIE* system features six different types of HTML reports. The *similarity report* reports the results of the similarity search. The *frame shift report* is based on the similarity search reports and analyses the neighbourhood of the predicted gene to detect and report frame shifts within the gene prediction. The *EcoVec report* gives information about *Escherichia coli* and cloning vector contaminations. *Repeat reports* and *RNA reports* point out repeats and RNA regions in the genome sequence respectively. The last report is the *pathway report*. This report

³for prokaryotic genome projects this is sufficient

colours an enzyme of a pathway if the enzyme was found in the genome. In addition to HTML reports, ASCII reports are generated for quick data research. To help the biologist interpret the data, *MAGPIE* offers a reasoning process based on Prolog rules. Among other things these rules help to decide whether a region is a coding region or not.

***Pedant* [48]:** *Pedant* was developed in collaboration between the MIPS⁴, and the Biomax Informatics AG⁵. The project was founded by the Federal Ministry of Education and Research (BMBF) and was finished in 2000. Its main purpose is to automate most of the tasks of a genome annotation project. Furthermore, it is a database for curating previously annotated genomes. The Biomax Informatics AG took over the development of *Pedant* and distributes it as *Pedant-PRO* which is commercially available only. The company curates the *Pedant Genome* database as well. Both the *Pedant* and *Pedant-PRO* systems can be installed locally.

The *Pedant* system is divided into three major modules, the database module to store, modify, and access data, the processing module to run analyses, and the user interface to communicate with the system. The database module uses a DBMS to store the data. The database itself is composed of two types of tables. Primary tables are used to store raw sequence data as well as raw data as it is produced by bioinformatics tools. The secondary tables store the data in structured form as it is generated by the processing module. The difference between the two forms is that the data in the secondary tables can be accessed separately while the data in the primary tables can be accessed as one record only. An example is the ORF prediction of *Orpheus*. The primary tables hold all predicted ORFs in one record while the secondary tables store each ORF as a separate record. Additionally the database model allows the user to specify custom fields for each ORF. The processing interface runs the different analysis tools automatically. It parses the results and creates cross-references between different types of the data.

Two different types of user interfaces are provided by the *Pedant* system. The web interface is used by the annotator to communicate with the system. It offers the annotator access to contigs, ORFs, and their annotations. The user can select subsets of ORFs by restricting the predicted function to certain functional categories. Furthermore, the annotator can perform text searches in annotations and BLAST searches against all sequences in the database. The DNA viewer provides a graphical view of the genome. This viewer is used to navigate through the genome. Additionally, the annotator can perform six-frame translations. For each ORF, a hyper-linked protein report is generated. This report provides a

⁴Munich Information Center for Protein Sequences a group at the Max Planck Institute for Biochemistry

⁵the Biomax Informatics AG is a spin off from the Max Planck Institute for Biochemistry

summary on general features of the protein, functional information, and structural assignments. The annotator can access further information about the ORF such as the DNA / protein sequence and the raw results of a particular bioinformatics method. The second interface is the command line interface. It is used to manage the processing module as well as accessing the data stored in the database module.

As well as running standard tools such as Orpheus and BLAST, *Pedant* features basic automatic prediction of gene functions by running similarity searches against yeast functional categories. The automatic annotation is considered to be a starting point for thorough human investigations. To facilitate this, the group designed and implemented the MIPS-FunCat classification [49] which is used supported by *Pedant* as well.

***ERGO* [50]:** *ERGO* was developed by Integrated Genomics Inc. (IG) in 2002. The focus of the *ERGO* system is the integration of data from different sources. Besides data from analysis tools, the system provides the annotator with data from biochemistry, high-throughput expression profiling, genetics, and peer-reviewed journals. *ERGO* is the first annotation system that utilises the “comparative genomics” approach in the annotation process. Like *Pedant* the *ERGO* system offers a comprehensive curated database of publicly available genomes.

ERGO is a web-only application and does not support any other kind of graphical interface. It uses the PostgreSQL DBMS as its back-end but does not provide any kind of API. Furthermore *ERGO* is an annotation system only. This means that the system does not provide any kind of automated tool pipeline for the prediction of ORFs and subsequent similarity searches. IG uses a number of proprietary scripts for predicting ORFs. The sequence similarity searches are done by running all newly predicted ORFs against the non-redundant database of ORFs already present in the *ERGO Genome* database using the *FASTA* algorithm. The DNA sequence, the predicted ORFs, and the results of the similarity searches are then imported to the *ERGO* system.

Once a genome is imported, all ORFs are annotated automatically. Again IG uses a variety of proprietary algorithms for the annotation of ORFs. The next step in annotation is the manual inspection and refinement of all ORFs. In addition to using the information present in *ERGO* the annotator has access to a variety of publicly available tools, among others *BLAST*, *Pfam*, *PROSITE*, and *COG*. Each ORF can be analysed manually by using these tools and submitting on-line queries to the NCBI, Pfam, and COG websites. The difference between the *ERGO* approach and other systems is that it does not focus on a single ORF, but rather considers the genes neighbourhood, also known as gene context (comparative genomics). The theory behind this approach is that if genes share the same neighbourhood, the functions of the genes are most likely related.

The third step in the annotation process is the metabolic reconstruction of the organism. In this step, all functional genes are mapped to metabolic pathway using the *IG-Pathdb* database.

Like the annotation systems presented above, *ERGO* features a user interface displaying the properties of a single ORF. Furthermore, it displays information about biochemistry, expression profiling, and genetics. Unique to the *ERGO* system⁶ is the visualisation of the gene neighbourhood and the putative function of the gene cluster, to which the gene belongs. Gene clusters can be inspected using the *WorkBench* tool, which provides an overview of gene clusters that are unique to a genome or shared among other genomes in the *ERGO Genome* database. Another concept of *comparative genomics* is the comparison of two or more whole genomes. To visualise this comparison, *ERGO* offers the *GenomeWalk* tool. It supports the annotator in finding group specific regions across different genomes.

The *ERGO Genome* database is a collection of annotated genomes. As of October 21th, 2005, the database contains approximately 640 completed genomes and approximately 250 unfinished draft genomes. You are required to register to access the database. *Ergo-Light*⁷ is a public version of the database containing at the moment nine genomes. The *IG-Pathdb* database is based on the *EMP* database and contains more than 5,000 pathways most of which are metabolic.

A stand-alone version of the *ERGO* annotation system can be purchase.

GenDB version 1 [51, 52]: *GenDB* is being developed by the *Center for Genome Research*⁸. It was initially published in 2003. Development continued within the framework of the EU founded project *Marine Genomics Europe*. The *GenDB* system is a genome annotation system comparable to the *Pedant* system. It features an annotation tool, a pathway inspection tool, a tool to infer GO mappings, and a tool to generate genome plots. The software was released under the terms of a proprietary license giving non-commercial users free access to the software and its source code. Since version 2.2 *GenDB* has been released under the terms of the GNU general public license (GPL).

The design of *GenDB* is closely related to the design of the *Pedant* system. It is divided into three modules, the analysis pipeline, the user interfaces, and at its core a database engine. All tools and user interfaces are build around this core using an abstraction layer called *O2DBI* - Object to Database Interface. *O2DBI* is used to define the data model of an application in an UML-like description language. It automatically creates the relational database model for supported database engines as well as the APIs written in supported programming languages. At the moment, PERL is the only supported language. The analysis pipeline uses the Sun grid engine (SGE) for the management and distribution of

⁶At the point it was released.

⁷reference: <http://www.ergo-light.com>

⁸Technische Fakultät and Department of Biology, Bielefeld University

jobs among nodes in a cluster environment. *GenDB* offers build-in support for numerous publicly available sequence analysis tools and it can be easily extended to use proprietary tools as well.

GenDB offers both a web-interface and a graphical user interface. Once a project is opened using the GUI client the annotator is presented with a genome browser giving an overview of the genome. It displays a list of all contigs the annotator can select from within a project, the sequence of the selected contig and its six reading frames, the predicted genes, and detailed information about the selected gene. The visualisation of the sequence data is dynamic. Compared to most of the other annotation systems introduced so far this offers the annotator an easy and fast method to navigate within the genome. Furthermore the annotator can switch to different views. These views include a pathway map, a genome plot, and a GO inspection tool (GOPArc). From the genome browser the user has access to further information about the selected gene or the whole genome. The fact or observation viewer shows the results of a similarity search (observations). These observations are ranked according to their reliability. The reliability is assigned by *GenDB*, e.g. reliability assignments for BLAST observations are based on the E-value. The region editor can be used to manipulate the start / stop position of a predicted gene. It can be used to join, split, or delete predicted genes. Also, search dialogs for basic information extraction tasks are provided. These dialogs allow the annotator to search for gene products, gene names, EC numbers, among other things. The web interface differs from the GUI client mostly in the techniques it uses to display the data. Where the genome browser uses scrollbars to zoom into and out of the contig and to navigate within the complete contig, the web interface displays only a clipping of the contig as static image. To view a different portion of the contig the web server has to render new images and transfer these to the client.

TheSEED [53, 54]: The *SEED* was developed by an international collaboration led by members of the Fellowship for Interpretation of Genomes (FIG) and the Argonne National Laboratory. It is used in genome annotation in a completely different way than the annotation systems introduced so far. Rather than focusing on the annotation of a single gene in a single genome, the *SEED* focuses on the annotation of groups of genes which are functionally related (subsystems). Also, the *SEED* allows the annotation of subsystems across several genomes at the same time. The software is available as open source software released under the terms of the GNU GPL.

The *SEED* is a web application based on a database engine and it offers an API written in PERL and Python to access the data directly. The *SEED* can be downloaded and installed locally. Local installations are self-contained and provide a user management system to allow several annotators work on the same or different projects. Additionally the *SEED* features a method of synchronis-

ing data from different installations using peer-to-peer techniques. This allows different institutions to use the same data as a basis for their annotations, reducing the overhead of updating and maintaining local copies of publicly available databases.

The *SEED*'s main interface is the visualisation of subsystems. It displays a subsystem based on the visualisation of metabolic pathways and the genes that are members of the selected subsystem across several genomes. It shows the annotation of each gene as well as its gene neighbourhood. This makes it easy to find differences and inconsistencies within the annotation of the “same”⁹ gene in different genomes. The annotator can update the annotation of a gene in one, all, or a subset of genomes. This kind of annotation is known as *vertical annotation* because the annotator annotates the “same” gene across several genomes which are vertically aligned rather than annotating a continuous number of genes on the same sequence strand (most commonly displayed horizontally). The system offers a simple way to spot missing genes within a subsystem and to infer a function for otherwise hypothetical genes. The *SEED* also features a database of synonyms. This database stores different descriptions, gene names and EC numbers for the *same* gene in different genomes.

The initial release of the *SEED* contains about 180,000 different proteins in 2,133 distinct functional categories. It contains about 170 subsystems and about 380 genomes.

3.1.3 The GenDB database (v.2.0)

Since **MicHanThi** does not provide any tools for either gene prediction or similarity searches it depends on an annotation system that offers these features. To integrate **MicHanThi** into an annotation system, the annotation system has to have an easy to use API in order to access the predicted genes as well as the results of the similarity searches. Of the annotation systems presented in the previous section, only *Pedant* and *GenDB* offer both a tool pipeline and an API to query the database easily. No information about an API was found for the *GeneQuiz* system. Also, the hyper-link referring to the documentation of the used database engine is broken or out of date. The *MAGPIE* system has two serious drawbacks. The first drawback is the use of a flat file-based database. While this might be suitable for read-only access, it is a performance bottleneck for updating, deleting, and modifying data in the database. The second and more serious drawback is that it depends on on-line tools to run the similarity searches. *ERGO* and the *SEED* do not provide a tool pipeline. Furthermore, the focus of the two systems is the comparisons of two or more genomes, rather than the annotation of every single gene in one genome. *Pedant* and *GenDB* are very similar in design and

⁹same in this case means that several genes in different genomes are functional equivalent, e.g. they are orthologous

implementation. The systems use MySQL as their primary database engine, but encapsulate it so they are somewhat independent. Both systems are implemented in PERL, and are separated into a front- and back-end. The major differences between these two systems are the design and functionality of the interface as well as the licensing terms. The **Microbial Genomics Group** at the MPI in Bremen has worked with both systems. Initially the *Pedant* system was used, but it was replaced by *GenDB* later because *GenDB* was available under a less restrictive license, allowing the modification of the source code and it provided the interface with the better functionality. The following paragraph will focus on the implementation of the *GenDB* system with focus on the design of the database.

As mentioned earlier *GenDB* uses O2DBI to create its database schema as well as its data model. Since O2DBI is an object-oriented modelling tool, the database schema created is somewhat object-oriented as well. O2DBI uses four different types of tables to implement object-orientation within a relational database engine. Root tables are tables storing the most generic information about a group of tables. Each of these tables are associated with a unique identifier (`_id`) and a class type (`_obj_class`). Each root table may have one or more subtables representing subclasses in a object-oriented programming language. These subtables store more specific information about the class they represent and are linked to their root table via the `_parent_id` field. This field points to the `_id` field of the root table and therefore it is the unique identifier of the subtable as well. All subtables could be parents of even more specialised subtables. These tables are linked via the `_parent_id` field, as well. Related tables share a common name. For example, the `Observation_Function_Blast` table is a subtable of `Observation_Function` which in turn is a subtable of `Observation`. The third type of table stores information contained in an array or list. These tables contain the value of an array item, the index of the item within the array, a reference to the specific record in the table to which the array belongs, and a unique identifier (`_id`). The name of the array table is the same as the name of the table it belongs to with the array name appended to it in lowercase. Each of the tables mentioned so far have an equivalent class to access it in the API. The last type of table used is the helper table that can not be accessed by the user. These tables are used to control the internal state of the database only. The `GENDB_DB_counters` table is an example of a helper table used to store the next available unique identifier for every root table and for every array table. The other helper table `GENDB_DB_unique_ids` stores unique identifier of other tables.

GenDB has ten root classes. The most important root classes for an automatic annotation tool are the `Annotation`, `Observation`, and `Region` classes. These classes, as well as their subclasses and relations, are depicted in figure 3.1 on page 34. The other seven root classes are the `Annotator`, `AssemblyInfo`, `FunCat`, `Job`, `Sequence`, `Tool`, and `TrainingSeq` classes.

Sequence: An instance of the Sequence class holds the complete genomic sequence of an annotation project. If the project has several contigs the genomic sequence of each of the contigs is stored in its own instance of the Sequence class.

Region: The Region class describes a subregion of the genomic sequence. Each contig is associated with a Region object describing the whole sequence. Gene finders and other tools analysing the genomic sequence, add further subregions describing genes, tRNAs and rRNAs, and non-coding RNAs. Each type of genomic sequence has its own subclass that contains the information unique to the type of sequence.

Observation: Observations are the results of tools analysing the (sub)regions of the genomic sequence. Two different types of Observations are derived from the main Observation class, the Observation_Region and the Observation_Function classes. Observation_Region and its subclasses store the results of the tools defining subregions while Observation_Function and its subclasses store the results of the tools analysing the subregions found. Each tool has its own subclass describing the special features of a tool.

Annotation: The information stored in the Annotation classes is an interpretation of the information within the Observation classes. Two different types of annotations are known to *GenDB*. First, annotations describing (sub)regions based on structural observations and second annotations derived from functional observations. Functional annotations are further divided into annotations describing the function of a gene and annotations describing tRNAs. Each Annotation object (regardless of its subtype) has a list of associated observations it was derived from, as well as a list of external information sources used to define the role of the (sub)region. Each functional annotation may be associated to a list of GO numbers.

Annotator: Each Annotation object is created by an annotator. The information about the annotator is stored in this group of classes. *GenDB* distinguishes between two basic types of annotators, human annotators and machine annotators. All tools supported by *GenDB* that create annotations have their own subclass of Annotator_Machine to store specific information about the tool. Also, the subclasses define whether or not the tool may update the latest_annotation_function or latest_annotation_region pointers of the Region class.

AssemblyInfo: The AssemblyInfo class contains information about the assembly method used to assemble the fragments generated by the sequencing process. An instance of this class exists only if information about the assembly process was provided by the user.

FunCat: The FunCat class contains only a unique identifier and an object class field and therefore seems to be unused at the moment.

Job: Whenever a tool is run to define or to analyse a specific (sub)region an instance of the Job class is created. The Job class controls how the tool is run by the grid engine and if the tool has any dependencies. The Job class can be used to automatically run further tools if the tool the Job was created for has finished. *GenDB* differentiates between two tool types: mandatory tools which must be run for each Region instance and optional tools. How and when optional tools are run is not documented.

Tool: The Tool classes describe the tools used by *GenDB*. Every tool must have its own subclass of either Tool.Function or Tool.Region. Special parameters for each tool can be set using this class. Some tools, i.e. InterProScan, SignalP and others, have a subclass in the data model but do not have their own subtables in the database schema. The reason for this could be that all parameters used by these tools are hard-coded and therefore no subtable is created by O2DBI. Again there is no documentation available.

TrainingSeq: Glimmer needs a set of training genes to adjust to the genomic sequence being analysed. These sequences are represented by the TrainingSeq classes. Internal sequences are subsequences from within the project while external sequences are not related to the project.

3.2 Automatic Annotation

3.2.1 Strategies for the Prediction of Gene Functions

Best matching Tool Result: The data set is analysed using one tool only. The annotation is then created from the “best” matching tool result. The definition of “best” depends on the features of the tool used. For BLAST this is normally the result with the smallest E-value. The approach is problematic because the functional assignment is only based on one observation only. For example if this observation describes an *hypothetical protein* because at the time that protein was annotated no other information was available, then the query sequence would be annotated as *hypothetical protein* as well. Another problem is that the quality of annotations differs among annotations projects. If the best observation in the list is badly annotated then incorrect information is transferred. However, this approach has the advantage of considering ‘new’ information. If a ‘new’ function was discovered in a genome annotation project it is likely that this function is under represented in a list of observations. Therefore, reasoning processes which consider the frequency of the occurrence of a functional description in a set of observations may not find the “correct” description.

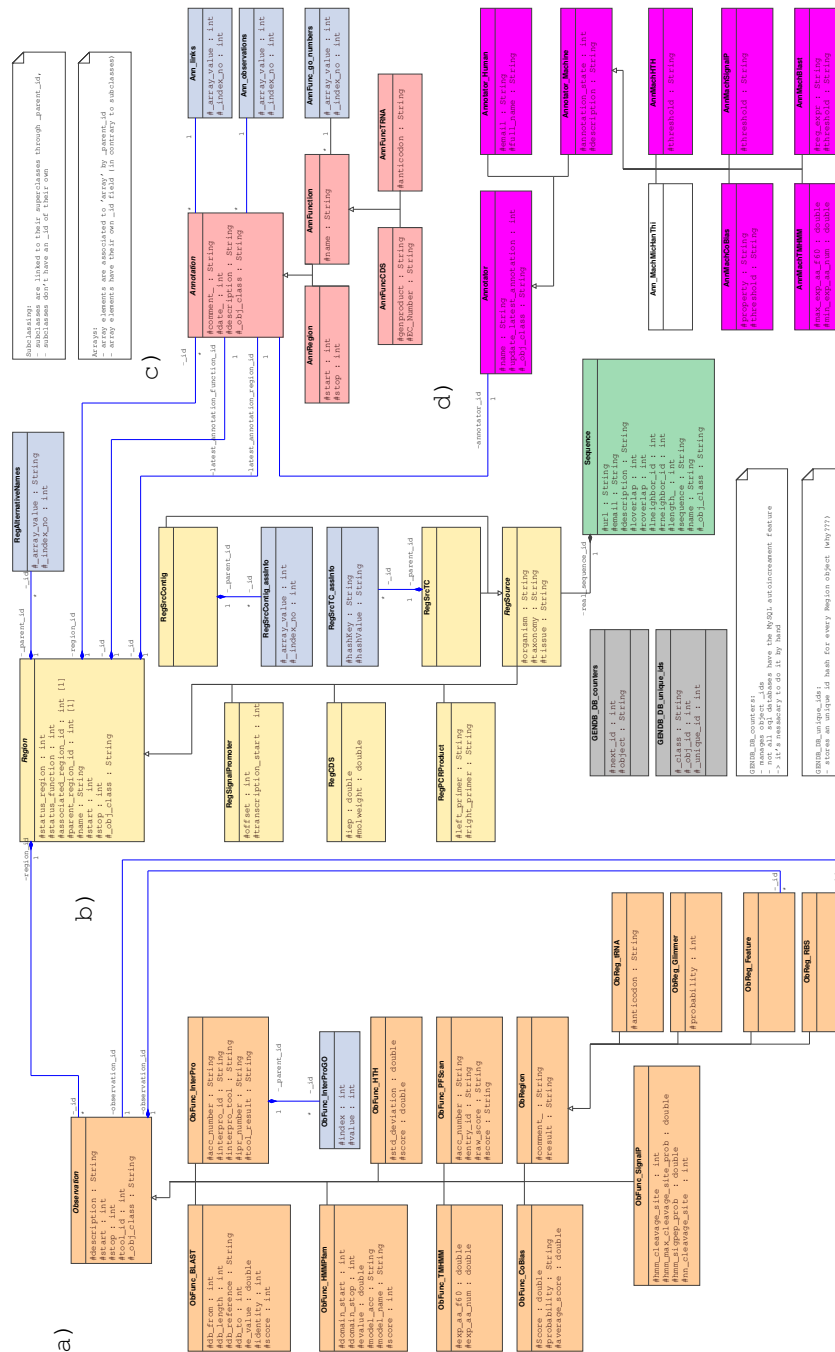


Figure 3.1: Subset of tables of the GenDB v.2.0 database model used by **MicHanThi**. a) Tables storing the results of a similarity search b) Tables holding information about the genome (the sequence and all predicted ORFs, tRNAs, rRNAs,...) c) Tables used for the annotation information d) Tables for different types of annotators (human / computer))

Ranked Best matching Tool Result: A generalisation of the **Best matching Tool Result** approach is to use different tools. Different tools are either tools that implement different algorithms like BLAST or HMMer, or tools that are run against different databases e.g. BLAST vs. NCBI nr, or BLAST vs. SWISS-PROT. The different tools are ranked according to their reliability based on the quality of the algorithm used or the quality of the database. The “best” matching results of the different tools are compared and ranked based on their reliability. The winning result is then used for the annotation of the gene. Basically, the same problems as well as advantages which apply to the *Best matching Tool Result* approach apply also to this approach.

3.2.2 Tools for the Prediction of Gene Functions

Every larger institute has its own annotation pipeline including a tool for the prediction of gene functions. Most of these tools are proprietary and no information is publicly available. Most of the tools presented were published after the design of the diploma thesis was finished or the implementation of **MicHanThi** was close to be finished.

AutoFACT[4] *AutoFACT* is the abbreviation for *An Automatic Functional Annotation and Classification Tool* and the tool was developed in cooperation by different universities in Canada. It uses multi FASTA files as input and runs BLAST against several databases for each sequence contained in the file. Before an analysis is started, the tool asks the annotator for some configuration options. Among these options are: the databases supposed to be used, the database importance, and the bit score used as a threshold for the reliability of an observation. Each sequence is assigned to one of the following annotation categories: ribosomal RNA (rRNA), [functionally annotated] proteins, unassigned protein, [domain name] containing protein, unknown EST (when using EST data), or unclassified. The assignment to one of the categories is based on a hierarchical system. First *AutoFACT* checks if the sequence is a rRNA by running BLAST against a rRNA database. The sequence is assigned to this category if it has a minimum length of 50bp and a similarity of at least 74%. If the sequence does not match any sequence in this database it is searched against the remaining databases. All BLAST observations are filtered according to the *uninformative rule* as described in [55]. This rule says that every BLAST observation, which does not describe a valid function, should be filtered based on regular expressions and lists of known words. Additionally, observations with a bit score less than the threshold (default is 40) are filtered. The tool then selects the n best observations and searches for common terms among the description lines. To transfer information from the observations to the annotation, the tool searches these in the observations of the different databases according to the importance of the database given by the user.

AutoFACT uses the Pfam or SMART databases if no common BLAST observations were found or if the observation found do not use the same terms to describe the function of the sequence. If the sequence contains a domain it annotates the sequence as *[domain name] containing protein* or *multi-domain-containing protein* in case the sequence contains more than one domain. If no similarities could be found at all, the sequence is assigned to the *unassigned protein* category. For protein coding sequences the last step in the annotation process is the classification of the potential gene according to COG, as well as the assignment of EC and GO numbers, and assignment of a locus tag. The assignment of this information is based on common terms, found in the description lines of different types of observations. EST sequences are checked against the *NCBI est_others* database and annotated as *unknown EST* if a significant similarity was found. Otherwise the sequence remains *unclassified*. The annotation process is depicted in figure 3.2 on page 37. The results are presented as HTML files, Gif images, and tab delimited text files. A log file is also generated, documenting all decision-making steps in the annotation process.

The tool is available for download at:

<http://megasun.bch.umontreal.ca/Software/AutoFACT.htm>.

BASys[56] *BASys* is a fully automated genome annotation pipeline, accessible via a web server. It is developed by the departments of Biological Sciences and Computing Science at the University of Alberta Edmonton. *BASys* allows the user to upload a genome sequence and an optional list of ORFs. If the list of genes was omitted it runs Glimmer to predict the ORFs itself. It analyses the genome and provides the user with different reports, which summarise the results of the analysis. The whole annotation process is automated and no user intervention is possible. *BASys* uses approximately thirty different tools to predict up to sixty annotation fields. Among these fields are: the function, the gene / protein name, COG function, GO function, possible paralogous and orthologous, operon structure, trans membrane regions, signal peptide, secondary structure, and 3D structure. Similarity searches are run against the UniProt, COG, PDB, and CCDB database, as well as a custom database of model organisms such as the *Caenorhabditis elegans*, *Homo sapiens*, *Saccharomyces cerevisiae*, and *Drosophila melanogaster*. Among the tools used to analyse the genomic sequence are BLAST, Pfam, PROSITE, Homodeller to generate a homology model if the similarity to a sequence in the PDB database exceeds a certain threshold, and VADAR for structural analysis. The annotation engine uses different threshold for each tool. If the similarity score between a query sequence and a database sequence exceeds the defined threshold the observation is used to fill in one or more annotation fields. For example the threshold for functional annotations has an E-value of $1e^{-10}$ or less. For the prediction of trans membrane regions a “perfect” match is needed to annotate the ORF as membrane protein. The interpretation of the term

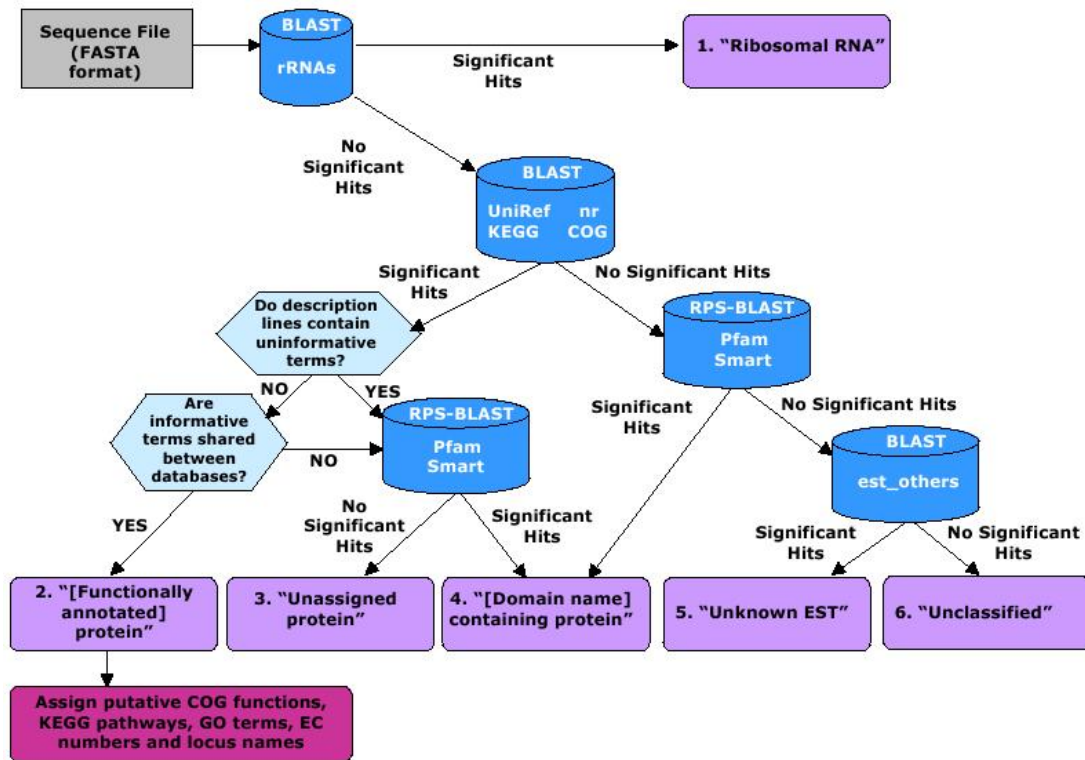


Figure 3.2: *AutoFACT* methodology. Sequences are classified into one of six annotation categories (purple boxes). The user decides which bit score cutoff to use (default 40) before a BLAST hit is considered significant. Figure taken from [4]

“perfect” is not explained within the paper. Information provided by different tools is transferred transitively. The use of the term transitively is not explained as well.

The tool is available for download at:
<http://wishart.biology.ualberta.ca/basys>.

HERON *HERON* was developed by Gopal and Gaasterland. The tool uses BLAST as its only source of information. BLAST observations are ranked not only by the features of the BLAST tool, but also according by the “meaningfulness” of the observation’s description line. A description lines is meaningful if it contains “meaningful” words. Meaningful words are words that can be found in the GO database. For each word, contained in the GO database, the observations gets a point. Additional points are “scored” for good E-values. The “best” BLAST observation is then take as the basis for the annotation.

This paragraph is based on a word document found at the website of Gopal

and an email communication with the author.

HMAP *HMAP* - High-quality Automated and Manual Annotation of microbial Proteomes - is a tool for the prediction of gene functions and it is a program for the annotation of genes. The program as well as the development of the *HMAP* tool is funded by SWISS-PROT. The *HMAP* team consist mainly of biologist annotating genes by hand, to accommodate the high quality standards of the SWISS-PROT database. Each annotator is specialised in a specific biochemical function, only annotating those genes that are involved in that particular function. Annotations created by the annotators are based on different analysis tools as well as thorough literature search. Before an annotator is allowed to enter new annotations into the SWISS-PROT database she has to go through a two year training period where all annotation are cross-checked by two experienced annotators. A computer scientist is developing the *HMAP* tool to support the human annotators. This tool uses a database of protein families called *HMAP*-families as its only source of functional evidence, to satisfy the high quality standards of the SWISS-PROT database. The *HMAP*-families are defined by the human annotators describing a group of proteins involved in the same function. Each family has a template containing most of the annotation fields, as well as the domain structure of a protein and necessary amino acids. The *HMAP*-tool uses these templates to create the annotations. It checks the template for required domains and if a domain or amino acid is missing, the annotation will be marked and the human annotator has to verify the annotation, before it is allowed to enter the database. An example would be an ORF describing a potential *histidine kinase* but is missing a histidine amino acid.

The information presented in this paragraph are the results of a meeting with the *HMAP* team at SWISS-PROT - Geneva.

Metanor The *Metanor* tool is part of the *GenDB*-2.2 distribution. It is based on the **Ranked Best matching Tool Result** approach and uses the tools, InterProScan against InterPro, BLASTP against SWISS-PROT, and BLASTP against NCBI nr. *Metanor* predicts the function of a gene based on InterPro matches first. If a gene does not have any matches in the InterPro database or the reliability of these matches is to low, BLASTP against SWISS-PROT is used. If no reliable annotation could be derived from this tool as well, BLASTP against NCBI nr is used. The gene is annotated as *hypothetical protein* if no reliable annotation could be created at all.

This paragraph is based on information provided by Alexander Goesmann¹⁰. *Metanor* is mentioned in [52] but isn't published in a paper of its own.

¹⁰Chief of Project

NCBI NCBI uses the **Best matching Tool Result** approach to automatically annotate draft genomes sequenced by the Joint Genome Institute (JGI), which are imported to the database. BLAST against the COG database is the only tool used and the information of the observation with the smallest E-value is taken to annotate the ORFs.

Chapter 4

MicHanThi

4.1 The Algorithm

The process of annotating a genome can be split into several parts. The first part of this process is the prediction of ORFs with subsequent similarity searches against public and custom databases. The second part is the prediction of functions for each ORF using the observations obtained. The *GenDB* system provides the initial observations and **MicHanThi** is used to predict gene functions. An overview of the integration of **MicHanThi** into the annotation process is shown in figure 4.1. **MicHanThi** is designed to analyse only one ORF at a time. Hence,

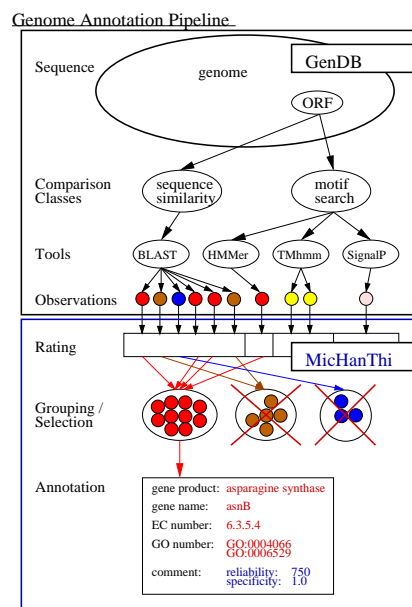


Figure 4.1: Embedding of **MicHanThi** in an annotation system

a grid engine must be used to annotate an entire genome. The **GenannD** soft-

ware was written for this purpose (4.3.3). Any other grid engine can be used, but a wrapper script to start **MicHanThi** may need to be written for that particular grid engine.

The algorithm developed as part of this thesis can be divided into three parts: (i) the preprocessing of observations, (ii) the annotation process, and (iii) the post processing of annotations. The preprocessing of observations is done in three steps. The description of each observation is cleaned up, the observation is rated based on the selected attributes of the tool which created the observation, and finally a subset of observations is selected for the annotation process. The annotation process creates annotations based on the selected observations. Different approaches are used for each type of tool. After the annotations are created, each annotation is checked, and if possible, annotations describing the same function are merged. The algorithm which was developed for this thesis is presented in pseudo code.

```

for each observation
  preprocess(observation)
  rate(observation)
  select(observation)

create annotations
assign additional features

for each created annotation
  check(annotation)
  delete(annotation)

merge annotations

```

4.1.1 Preprocessing Observations

MicHanThi uses BLAST against the NCBI nr database as a source of functional information about the ORF. As mentioned in 2.3 NCBI nr entries may contain more than one entry (sub entry) and the NCBI nr description is a concatenation of the description of the sub entries. Several processing steps are taken to “clean up” description of NCBI nr observations: the source database is identified, the multiple description is separated, and the separated descriptions are processed based on their source. The following paragraphs discuss these processing steps in detail. The grammar of a NCBI nr entry is explained using the extended Backus-Naur form (EBNF). It is broken into pieces and discussed as needed to better understand it.

```

S          := ENTRYLIST
ENTRYLIST:= ENTRY | ENTRYLIST GI ENTRY

```

```

ENTRY      := DEF | PIR | PRF | SP

ECNUMBER   := ('EC ')? D '.' DL '.' DL '.' DL |
              ('EC ')? D '.' DL '.' DL '-.' |
              ('EC ')? D '.' DL '-.-' |
              ('EC ')? D '-.-.-'

DL          := D | D DL
D           := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

FUNC        := 'functional description of the protein'
DBREF       := 'database identifier'
ORG         := 'name of the organism this protein can be found in'

```

Splitting of nr Entries: In the description of each NCBI nr entry, the descriptions of different sub entries are separated by the GI number. Therefore, the GI number is also used to split the descriptions. Each GI number is constructed from the unique identifier of the NCBI nr database entry followed by an abbreviation of the source database and the identifier of the sub entry in the source database. The term 'gi' is prefixed and the four fields are separated by the '|' character. GI numbers of the PIR and PRF databases are "broken" because the source identifier is not terminated by the '|' character. Instead, it is placed in front of the source identifier. This is handled by introducing two production rules as alternatives for the GI rule (GIOK and GIBROKEN). Newer version of the *formatdb*¹ command format the GI numbers differently. These version omit the 'gi|' D + '|' part of the identifier and write only *SDB*'*SID*'. In this case, the production rules are modified accordingly. GI numbers can be described by the following grammar. The non-terminal 'L' represents all uppercase letters of the Latin alphabet and for purposes of readability is not described by a production rule:

need to replace all quantifiers by exact quantities.

```

GI      := GIOK | GIBROKEN
GIOK    := 'gi|' D+ '|' SDB '|' SID '|'
GIBROKEN:=
            'gi|' D+ ('|pir||' | '|prf||') SREF
SID     := L L '_' D+ '.' D |
            L (L | D)+ |
            L L L D+ '.' D |
            L (L | D)+ '.' D |
            D+
SDB     := 'dbj' | 'emb' | 'gb' | 'ref' | 'sp'

```

The first sub entry is selected as the primary entry and then its description is further processed according to the source database. Other sub entries are

¹formatdb is part of the NCBI tools package and is used to format databases used by BLAST

discarded because all entries describe the same function. The description of entries in the DDBJ, EMBL, GeneBank, PRF, and RefSeq databases, share a common grammar. The function of the gene is followed by the organism name, enclosed in brackets.

DEF := FUNC ('[' ORG ' ')?

The organism name is deleted from the description because it is not needed for a functional description of the ORF, as are all non-word characters. Terms composed of two hyphenated words are split because functional terms are either spelled as one word, two words, or are hyphenated, as in the term *glycosyl transferase*. Additionally, observations found in the nr database contain a large amount of information which can not be used for the annotation of proteins. Only “informative” terms, which describe the function of a gene, are used in the annotation process. To increase the performance of the annotation process, “uninformative” terms are deleted. Uninformative terms are checked against a list of known words, as well as predefined regular expressions. Uninformative terms can be adverbs, conjunctions, comparatives, prepositions, and ORF names. For badly annotated genomes, it is common, that *hypothetical proteins* are assigned the internal ORF names of an annotation project. The filtering of uninformative terms is also done by [55].

If a SWISS-PROT or PIR database entry is found in the NCBI nr entry, then it is taken as the primary key instead of the first sub entry. This is done because the SWISS-PROT and PIR databases are of better quality, than the rest of the databases integrated by NCBI nr. Entries found in these two databases use auxiliary information to describe the function of a protein and are further processed.

Preprocessing of PIR Entries: PIR descriptions contain an additional EC number in parenthesis following the function of the protein. The EC number is parsed, stored separately, and deleted from the description. An optional database identifier could be present after the EC number. This identifier is not used and therefore deleted. An organism name appended by the ‘-’ character is deleted. The following grammar describes the syntax of a PIR description:

PIR := FUNC ECNUM DBREF? ('-' ORG)?
ECNUM := '(' ECNUMBER ')'

Preprocessing of SWISS-PROT Entries: The syntax of SWISS-PROT entries is more complex. Four types of descriptions are distinguished: single function proteins, bi- and multi-functional proteins, cleaved proteins, and proteins that form a different complex when cleaved. Single function proteins are proteins known to have only one function. The functional description of the protein can

be followed by a list of synonyms. A synonym can either use different wording or use an EC number to describe the function of the protein. The explanations of the second, third, and fourth type of entries, are taken from the SWISS-PROT user manual².

Cleaved proteins:

If a protein is known to be cleaved into multiple functional components, the description starts with the name of the precursor protein, followed by a section delimited by '[Contains: ...]'. All the individual components are listed in that section and are separated by semi-colons (;). Synonyms are allowed at the level of the precursor and for each individual component.

Bi- and multi-functional proteins:

If a protein is known to include multiple functional domains each of which is described by a different name, the description starts with the name of the overall protein, followed by a section delimited by '[Includes:]'. All the domains are listed in that section and are separated by semi-colons (;). Synonyms are allowed at the level of the protein and for each individual domain.

In rare cases, the functional domains of an enzyme are cleaved, but the catalytic activity can only be observed, when the individual chains reorganise in a complex. Such proteins are described in the DE line by a combination of both '[Includes:...]' and '[Contains:...]

The list of synonyms in the description of single function proteins is stored for later use. Synonyms present in the description are then deleted because they would influence the annotation process. For cleaved proteins, the name of the precursor protein is considered to be its functional description. Additional information present in the description is discarded. The same is done for bi- and multi-functional proteins. The name of the overall protein is kept as its description and the rest of the information is deleted.

The following grammar summarises the syntax of a SWISS-PROT description. Bi-functional proteins are a specialisation of multi-functional proteins and the description of bi-functional proteins has a different syntax then those of multi-functional proteins. This is modelled by introducing the production rule *MULTI* which distinguishes between bi- and multi-functional proteins. According to Tania Lima³, the description of bi-functional proteins should always start with 'Bifunctional protein' followed by a gene name. This is specified as a distinct rule in the grammar but is not implemented by **MicHanThi** because not all bi-functional proteins are described in the manner specified.

²http://au.expasy.org/sprot/userman.html#DE_line

³Tania Lima is a member of the HMAP annotation project at SWISS-PROT

```

SP          := CLEAVED | MULTI | SINGLEFUNC | RARE

CLEAVED     := PRECURSOR ' [Contains: ' FUNCLIST ']'
PRECURSOR   := 'name of the precursor protein' ((' ' SYNONYM '))*
FUNCLIST    := SINGLEFUNC ( ' ; ' SINGLEFUNC )+
SINGLEFUNC   := SYNONYM ( ' ( ' SYNONYM ' ) ' ) *
SYNONYM     := FUNC | ECNUMBER

MULTI       := MULTIFUNC | BIFUNC
MULTIFUNC   := SINGLEFUNC ' [Includes: ' FUNCLIST ']'

BIFUNC      := SINGLE ' [Includes: ' FUNCLIST ']'
SINGLE       := BFUNC ( ' ( ' SYNONYM ' ) ' ) *
BFUNC       := 'Bifunctional protein' GENENAME
RARE        := SINGLEFUNC
              ' [Includes: ' FUNCLIST ']'
              ' [Contains: ' FUNCLIST ']'
GENENAME    := 'the gene name of a protein'

```

4.1.2 Rating Observations

```

procedure rate(ob)
  if ob is a BLAST observation
    assign reliability based on FL(
      ob, evaluate(ob), covORF(ob), covDB(ob), ids(ob))
  else if ob is an InterPro observation describing a Pfam family
    assign reliability based on FL(ob, evaluate(ob))
  else if ob is an InterPro observation
    assign reliability(ob) <- 0
  else if ob is a SignalP observation
    if hmmSignalP probability(ob) > .75
      assign reliability(ob) <- 1000
    else
      assign reliability(ob) <- 0
    end
  else if ob is a TMHMM observation
    if ob is a trans membrane helix prediction
      assign reliability(ob) <- 1000
    else
      assign reliability(ob) <- 0
    end
  end
end
end

```

It is not uncommon for BLAST to produce several hundred observations for each ORF. To reduce the number of observations, each observation is assigned a reliability value and observations assigned a weak reliability value are deleted by the selection process. The reliability value shows whether or not an annotation, created from this observation can be trusted. Reliability values are assigned based on selected attributes of the tool that created the observation. The attributes used to rate an observation were selected based on a discussion with the human annotators of the **Microbial Genomics Group** for each tool. For BLAST observations, these attributes are: the E-value, the coverage of the query sequence by the match found in the database, the coverage of the matching sequence by the query sequence, and the relative number of identical bases in the sequence alignment. The E-value is used rather than the (Bit-) Score because the best possible (Bit-) Score is the (Bit-) Score of an alignment of a query sequence against itself. The longer the sequence the greater the score. Reasoning rules using the (Bit-) Score would have to be modified for each sequence. A disadvantage of the E-value is, that it is dependent on the size of the database. A sequence found in a smaller database will have a much better E-value because the BLAST tool is less likely to find the query sequence. Since running BLAST against database A and running BLAST against database B are considered two different tools, size differences are no longer a problem. For each tool, the importance of each attribute can be adjusted to ones needs. The coverage, as well as the relative number of identical bases, are used to “weaken” the E-value. A short coming of the E-value is that it does not accurately reflect the length of the match between two sequences. A complete alignment could have a worse E-value than a partial alignment because the partial alignment may have fewer gaps and a higher number of identical bases. Four cases of alignments can be distinguished: A) both sequences completely match, B) only subregions of the two sequences match, C) a subregion of the query sequence matches the complete sequence found in the database, and D) the complete query sequence, is matched only by a subregion of the database sequence. These four cases are depicted in figure 4.2 on page 48 In case A, a functional annotation could be derived from the database sequence because concerning coverage it is a good match. The subregion matched in case B is most likely a domain conserved among the members of a protein family. To derive a function from this type of alignment further investigations have to be made. The domain structure of the protein family must be analysed. If all domains mandatory for proteins of this family are found in the query sequence, then it can be annotated as member. A query sequence missing one or more domains can not be annotated as a protein of this family because it could either be an unknown orthologous protein belonging to the same family, or it may be a paralogous protein with a potentially different function. In this case, the query sequence should be annotated as *protein containing [domain names]*. In the third and fourth cases, matches are found in sequences of different length. The shorter of the two sequences could be a domain. If it is not a domain it could

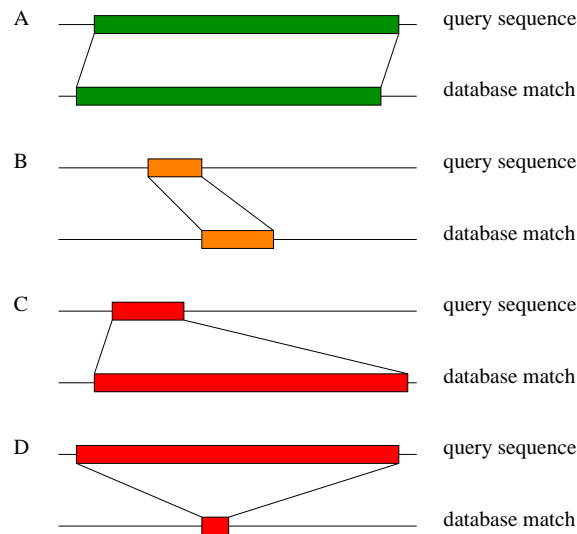


Figure 4.2: Four different cases of sequence alignments. The sequences depicted are scale free.

indicate an incomplete ORF prediction due to e.g. a frameshift which means that part of the shorter sequence is missing. The longer of the two sequences could be a fusion of two or more genes if the shorter sequence is e.g. a domain. Again, it could be an error in the gene prediction but in this case, two separate genes were predicted as one. Caution should be taken,; if a function was to be derived from this type of alignments. The relative number of identical bases is used to reflect the quality of the alignment. An alignment with a good coverage in both directions and a small number of identical bases, is not as good as an alignment with a shorter coverage but a high number of identical bases.

The second tool used to predict a gene's function is the InterProScan tool run against the InterPro database. Of the information provided by InterProScan only the E-value is used to assign a reliability value to an observation. Other information reported by the tool is neglected because it does not describe quality between the query and the matching sequence.

If no homologous could be found in the databases used, then observations created by TMHMM and SignalP are considered. These tools search for “general” features of a protein. From the TMHMM observations, only trans membrane helix predictions are considered by **MicHanThi**. These observations are assigned a reliability value of 1000 (reliable). Other observations reported by TMHMM are assigned a reliability of 0. SignalP reports the probability of the ORF being secreted. If this probability is greater than .75, then the ORF is considered to be secreted and the observation is assigned a reliability value of 1000, else it is assigned a reliability of 0. It is not necessary to use fuzzy logic to rate observations predicted by TMHMM or SignalP, because these tools make simple “true / false”

statements, either an observation is predicted or it is not.

The reasoning process for rating observations is based on *Fuzzy Logic* (FL). FL was chosen to be able to deal with vagueness of the in-put data. The following paragraph explains the concepts of FL and how it is used to represent the in-put data.

Fuzzy Logic:

Logic, according to Webster's dictionary, is the science of the normative formal principles of reasoning. In this sense, fuzzy logic is concerned with the formal principles of approximate reasoning, with precise reasoning viewed as a limiting case.

Lofti A. Zadeh [57]

The Problem: Real World Vagueness - Natural language abounds with vague and imprecise concepts, such as “Jane is old” or “Jane is of average height”. Such concepts are hard to translate to more precise language without losing semantic value. The statement “Jane is 56 years of age” does not explicitly state that she is old. The statement “Jane’s height is 1.05 standard deviations about the mean height for women of her age” is fraught with difficulties. Would a woman of 1.1 standard deviations be tall, or would she still be of average height? No one would oppose that discarding statements such as “Jane is old” from natural language, would result in the loss of valuable information. Yet this is what happens if natural language is translated into two-valued (or classic) logic. While this may not be important for most programs, it is important for “knowledge representation” in expert systems. Using classic logic for the rating of observations would mean, that an E-value of $1e^{-16}$ is good, while an E-value of $1e^{-15}$ would be uncertain⁴. Also an E-value of $1e^{-16}$ would be just as good as an E-value of $1e^{-100}$.

Basic Concepts - The main concepts that discriminates fuzzy logic from other types of logic is that it allows proportional membership. Classic logic allows a proposition to be either true or false ($x \in X$ or $x \notin X$). In fuzzy logic, the truth value of a proposition ranges over the *fuzzy subsets* in X and may be viewed as an imprecise characterisation of the set. For example, if X is the set of all E-values, then a truth value of “good” may be interpreted as the fuzzy subset of E-values less than $1e^{15}$. Membership to a fuzzy set is indicated by a value in the range $[0.0, 1.0]$, with 0.0 representing no membership at all (or false) and 1.0 representing full membership (or true). A fuzzy set is characterised by the

⁴according the annotation guidelines defined by the **Microbial Genomics Group** for the annotation of ‘*Gramella forsetii*’ KT0803

membership function $m_A(X)$ which maps each element $x \in X$ onto the real interval $[0.0, 1.0]$. This process is called *fuzzyfication*.

Further concepts of fuzzy logic are: *fuzzy predicates*, *fuzzy quantifiers*, and *linguistic variables*. A subset Y of X in classic logic can only be described by crisp predicates. If X is the set of natural numbers, then Y may be the subset of “even” natural numbers. In fuzzy logic, the set of natural numbers can be described using fuzzy predicates like “small” and “much greater” as well as crisp predicates. In addition to the two quantifiers “all” and “some” allowed in classic logic, fuzzy logic adds quantifiers like “most,” “many,” and “few”. A linguistic variable is a variable whose values are words instead of numerical values. For example, an E-value is a linguistic variable and its values are “(very) good,” “uncertain,” and “(very) bad”.

A is empty if its membership function $m_A(X)$ maps all elements of X onto the value 0.0.

$$\forall x \in X, m_A(X) = 0.0 \quad (4.1)$$

Two fuzzy sets A and B are equal if all elements of X , have the same degree of membership in A and B .

$$\forall x \in X, m_A(X) = m_B(X) \quad (4.2)$$

The complement of A is defined by the inverse membership function $m_A(X)'$

$$m_A(x)' = 1 - m_A(x). \quad (4.3)$$

A is contained in B if all elements in X have a smaller or equal degree of membership in A then in B

$$\forall x \in X, m_A(x) \leq m_B(x) \quad (4.4)$$

The union and intersection of two fuzzy sets are characterised by the minimum and maximum functions.

$$\forall x \in X \text{ MAX}(m_A(x), m_B(x)) \quad (4.5)$$

$$\forall x \in X \text{ MIN}(m_A(x), m_B(x)) \quad (4.6)$$

The Difference between Fuzzy Logic and Probability - Both fuzzy logic and probability operate over the same numeric range and at first glance both have similar values: 0.0 representing false (or non-membership) and 1.0 representing true (or membership). The probabilistic approach yields the natural-language statement, “There is an 80% chance that Jane is old,” while the fuzzy terminology corresponds to “Jane’s degree of membership within the set of old people is 0.80”. The first view supposes that Jane is either old or not and we have an 80% chance of knowing. Fuzzy logic, on the other hand, supposes that Jane is “more or less old”.

Fuzzy Reasoning - Most expert systems, fuzzy or non-fuzzy are rule-based. Fuzzy logic uses *if ... then ...* statements as rules which have the form:

$$R^i : \text{if } (X_1 \text{ is } A_1^i) \text{ and } \dots (X_n \text{ is } A_n^i) \\ \text{then } Y^i$$

R^i is a i^{th} rule in the rule set, A_j^i is a fuzzy set in X_i , Y^i is the result of a rule its conclusion. A rule as it is used by **MicHanThi** may look like:

$$R : \text{if } (\text{evaluate is good}) \text{ and} \\ (\text{covORF is complete}) \text{ and} \\ (\text{covDB is complete}) \text{ and} \\ (\text{identities is many}) \\ \text{then } (\text{Observation is good})$$

The complete list of rules can be found in appendix A. Rules in fuzzy logic are applied by calculating the confidence for each part of the proposition, which is determined by the membership of X_j in A_j^i . The result of the rule is the evaluation of the proposition. Fuzzy logic provides two simple operators for this purpose, the union or the intersection of two sets. Fuzzy logic offers to define custom operators to calculate the result of a rule as well. An example of a custom operator is the weighted sum approach as mentioned in [57]. Then the conclusion of a rule would then have the form:

$$Y^i = a_0^i + a_1^i X_1 + \dots + a_n^i X_n$$

Modelling of BLAST attributes - The E-value (evaluate), the ORF coverage (covORF), the database coverage (covDB), and the number of identical bases in an alignment, are the attributes of the BLAST tool used to rate an observation. Based on a discussion between the author and the annotators of the **Microbial Genomics Group** the number of fuzzy sets for each attribute and the membership function to map each element to its corresponding fuzzy sets have been defined.

evaluate Four fuzzy sets are distinguished for the E-value: *unreliable*, *uncertain*, *reliable*, and *very reliable*. The range of good E-values has been further discriminated because the range covers most of the E-values, starting at about $1e^{-15}$ and less. According to the annotation guidelines set up by the **Microbial Genomics Group** observations with an E-value of $1e^{-15}$ or less are reliable and can be used to annotate a ORF. Observations with an E-value in the range of $1e^{-3}$ to $1e^{-15}$ are uncertain and these observations should be considered carefully, Observations with an E-value larger then $1e^{-3}$ are unreliable and no annotation should be derived from such observations. $1e^{-3}$ is the intersection of the unreliable and uncertain fuzzy sets with a degree of membership of 0.5. The intersection of the uncertain and

reliable fuzzy sets is at $1e^{-15}$. The discriminatory power of E-values is non-linear. An E-value of $1e^{-100}$ is not five times as good as an E-value of $1e^{-20}$ but an E-value of $1e^{-15}$ is more than five times as reliable as an E-value of $1e^{-3}$. Therefore the membership function of each fuzzy set is steep close to the two thresholds mentioned before and increases more slowly the greater the difference between the E-value and the threshold.

covORF/DB For the ORF as well as the database coverage three fuzzy sets have been defined: *none*, *partial*, and *complete*. Alignments with 90% coverage of the query and database sequences or above are considered to be complete matches ($mComplete(x) \rightarrow 1.0$). The threshold is set 90% to accommodate wrongly predicted start positions of an ORF. If the alignment covers 30% of the sequences or less, then the observation should be discarded ($mNone(x) \rightarrow 1.0$). To indicate that two sequences match only in a subregion, for example the query and the database sequence contain the same domain, the fuzzy set *partial* was introduced. The membership function of this set intersects with the membership functions of the sets *none* and *complete* at about 49% ($mNone/Partial(x) \rightarrow 0.75$) and 80% ($mPartial/Complete(x) \rightarrow 0.85$) respectively.

identities An alignment should have as many identical bases as possible. Other than that no annotator of the **Microbial Genomics Group** could give any reason why the membership functions should be modelled one way or the other. Three fuzzy set were created: *none*, *some*, and *many*. According to literature, a sequence alignment of two sequences should show at least 30% of identical bases. Therefore, their membership functions are defined to favour as many identical bases as possible.

Figure 4.1.2 summarises the description of the fuzzy sets and their membership functions.

Modelling of InterProScan attributes - The only attribute of the InterProScan tool used to rate an observation is the E-value. As for the BLAST tool, three fuzzy sets are defined. The membership function of each of these sets matches closely the membership functions of the according BLAST fuzzy set. For a detailed explanation see the paragraph about the *Modelling of BLAST attributes*. Figure 4.1.2 shows the fuzzy sets and membership functions used to rate observations created by InterProScan.

4.1.3 Selecting Observations for the Annotation Process

All observations assigned a reliability of 0 are deleted. The selection of the remaining observations is different for each tool used to create them. Observation predicted by SignalP or TMHMM are always kept. Only a subset of the

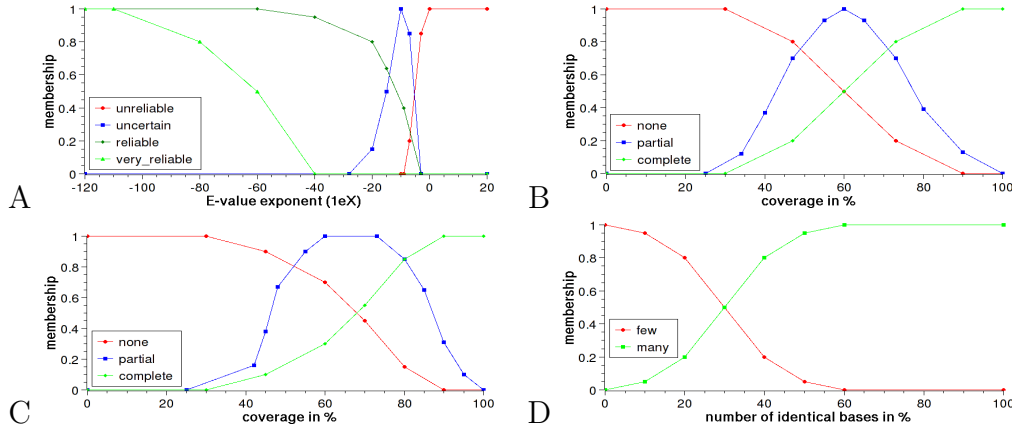


Figure 4.3: Modelling of BLAST attributes

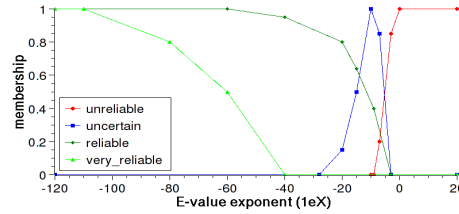


Figure 4.4: Modelling of InterProScan attributes.

BLAST observations are used to predict a function for the ORF. Those lacking a “meaningful” description line are deleted. It does not matter how reliable these observations are, they are not considered in the annotation process any further. However; they are not lost and can be used to decide whether the ORF should be annotated as *hypothetical protein* or *conserved hypothetical protein* if no functional annotation could be derived. Of the remaining BLAST observations, only the 25 most reliable are used. This rule functions in all cases except when more than 25 observations are reliable, then all observations assigned the attribute *reliable* are kept for the annotation process. This exception must be made to avoid random selection of observations and thus possible indeterministic behaviour of the annotation process. The order of the 50 best observations is random if the key used to sort these observations is the same for all observations. For example if all observations have the same reliability, then sorting the list may be ordered differently every time the sort function of the Java Array class is used and therefore the 25 observations picked for the annotation process may differ.

InterPro observations are deleted if they do not describe a Pfam family. This is motivated by the fact, that only Pfam family observations can be compared to BLAST observations.

4.1.4 Predicting the Gene Function - Annotation

BLAST:

```
function create annotations BLAST (obs) returns annotations
  common denominator(obs)
  select best supported / most reliable group(groups)
  merge based on atoms(groups)
  build annotations(groups)
  merge annotations based on function(annotations)

  return annotations
end

function common denominator (obs) returns groups of observations
  for each ob
    add to atom list(split(description(ob)))
  end
  sort alphanumerically (atom list)
  for each atom
    create group and populate(atom, obs)
  end
  n <- 1
  do
    for each group of size n
      for each atom
        create group of size n+1(group of size n, atom, obs)
        if group of size n+1 is unsupported
          delete(group of size n+1)
        end
      end
    end
    add(collection, groups of size n+1)
    n <- n+1
  while new groups of size n

  return collection of groups
end
```

BLAST observations are diverse in both functions, as well as the description of the same function, therefore a great deal of effort is exerted to derive a function based on BLAST observations. The five observations shown in figure 4.5 A describe the same function at different levels of generalisation. Observation four uses the most general description. Observations one to three differ in wording only. Observation five uses a synonym to describe the function⁵. To assign a

⁵according to the SWISS-PROT database, *Regulatory protein SIR2 homolog 2* is a synonym

function to the ORF, the most common denominator has to found.

Common Denominator - To find the largest common denominator among the functional descriptions of a group of observations, each description is broken into single words (atoms) depicted in figure 4.5 b. From that, a non-redundant list of atoms is created (figure 4.5 c), which holds each word used in at least one of the descriptions. The atoms are grouped in order to find the most specific

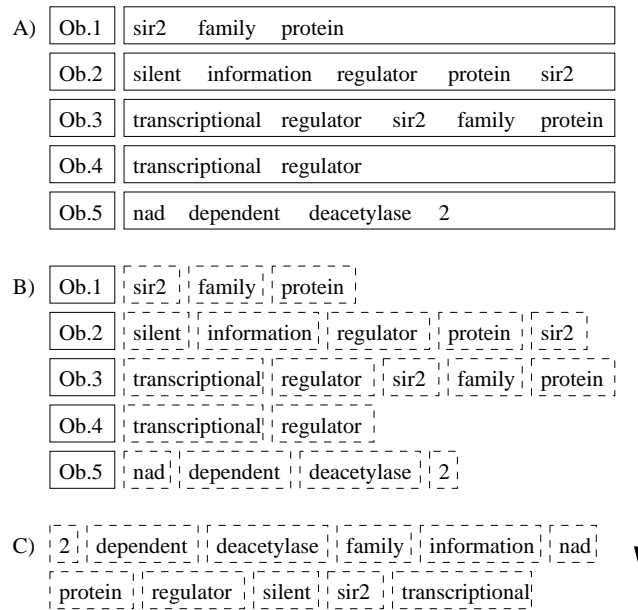


Figure 4.5: A) A list of example observations, B) descriptions split into atoms, C) non-redundant list of atoms.

functional description, common to “all” observations shown in figure 4.6. A group is a tuple of atoms and observations, whose description contains the atom. The order of a group is the number of atoms in it and the size of the group is determined by the number of observations containing the atom. For each atom, a group is created and all observations, containing the atom are added to the group as support. After the initial set of groups is created (groups of order one), it is used to create the groups of the next higher order. This is done by merging all groups of the current order with all groups from order one. Observations that do not contain all of the atoms of the newly created group are deleted because the group describes a different function than the observation. If a group is not supported by any observations, it is not describing a combination of words found in any observations and therefore it is deleted. These steps are repeated until no supported groups of order $n + 1$ can be created. Groups that are subsets of

for *NAD-dependent deacetylase 2*

another group are deleted because the goal of the common denominator is to find the most specific functional description for a list of observations and not a group that describes a single word common to “all” observations. Group A is a subset of group B if both the atoms as well as the supports of group A are a subset of the atoms and supports of group B.

select groups - Each group is checked for the number of supporting observations and their quality. If a group is to “unsupported” relative to the best supported group and the difference in the reliability of the observations supporting this group and the most reliable observations is “too large”, the group is deleted. The meaning of “unsupported” as well as the meaning of “too large” depends on the observations considered in the annotation process and therefore their interpretations are different for each ORF being annotated.

merge groups - To manage a case in which one observation combines the descriptions of two or more observations, the remaining groups are merged based on their atoms. Groups are merged if the atoms of group A are a subset of the atoms of group B. This merging is depicted in figure 4.5. In this figure, observation three contains the descriptions of both observations one and four.

build annotations - An annotation is created, for each remaining group. The annotation is based on the observation, which is best matched by the atoms of a group. An observation is matched “good” if all of the words of its description are matched by atoms. The atoms are then sorted according to the description of the observation. Once the atoms are sorted, a substring is taken from the description, ranging from the first atom to last. This substring is used to functionally describe the ORF. Each annotation is assigned a reliability value and specificity value. The reliability of an annotation is the product of the average observation reliability and the specificity of the annotation. The specificity of an annotation is the ratio of the number of informative terms used in the annotation and the number of informative terms in the description of an observation. If the specificity is 1.0 all informative terms found in the observation are included in the annotation. A value of less than 1.0 means that some terms are skipped. Annotations with a specificity value of less than 0.5 should not be trusted and a human annotator should investigate the ORF. The following example is used to clarify the specificity value:

```

annotation: glycosyl hydrolase family
description: glycosyl hydrolase family 34
informative terms (annotation): 3
informative terms (description): 4
ratio: 3/4  $\rightarrow$  specificity of 0.75
```

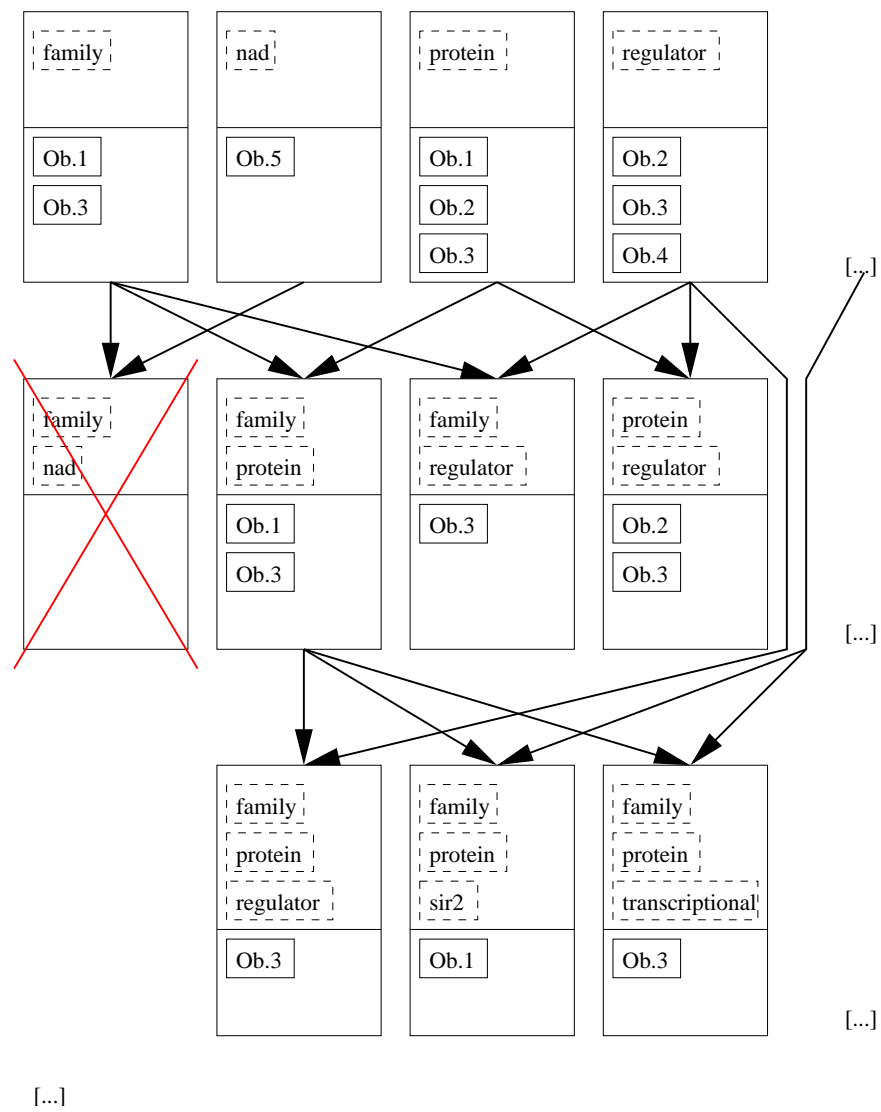



Figure 4.6: Functional grouping of observations

InterPro: Each ORF can have several InterPro observations. In rare cases only, are more than one of these observations Pfam families. If an ORF has two or more Pfam family observations, then these observations describe the same family, but different domains of that family are found in the ORF. The grouping of functionally equivalent observations is not necessary because the observations describe the same family and are based on the same entry in the InterPro database. **MicHanThi** takes the observation and uses its description for the assignment of the ORF's function. The reliability of an annotation based on an InterPro observation is the reliability of that observation. The specificity is always 1.0 because the whole description is used to describe the function of the ORF.

Merging Annotations derived from BLAST and InterProScan: Annotations based on BLAST observations and annotation based on observations created by InterProScan describe the function of a protein and the function of a protein family respectively. Both types of observations use the same informative terms to describe the function. Therefore annotations based on these tools are comparable and annotation describing the same function could be merged. Annotations could also be merged if one of the annotations describes a more general function than the other. A simple case of a generalisation of a function a (*DNA-3-methyladenine glycosylase I*) would be function b (*DNA-3-methyladenine glycosylase*). In this case, function b uses a subset of words of function a to describe the more general function. In other cases, different informative terms are used to describe the more specialised function. These cases are not handled by **MicHanThi** because it can not verify the semantic similarity of words and sentences used to describe the function of a protein.

The same approach is used to check if two annotations describe the same function, as it is used to predict a function derived from BLAST observation. First, all uninformative terms are deleted from the descriptions. Afterwards, the remaining terms are split and the two sets of informative terms are checked whether or not one of these sets is a subset of the other. If a subset relation could be established, then the two annotations are merged. The description of the specialised annotation is further used to describe the function of the protein. Additionally, if the merged function contains information that is not present in the other annotation, then the missing information is taken from the merged annotation.

4.1.5 Assigning additional Annotation Features

Further annotation features like the EC number, the gene name, and a list of GO numbers, are annotated if supporting observations are found. If an annotation is based on BLAST observations, then the supporting observations are checked for matches against the SWISS-PROT database. If a SWISS-PROT observation is found, then the EC number and gene name are taken from that observation, and

the GO numbers are also retrieved from SWISS-PROT. Since most SWISS-PROT entries associated with GO numbers are entries describing eukaryotic sequence, **MicHanThi** rarely annotates GO numbers based on BLAST observations for prokaryotic sequences.

EC and GO numbers are assigned if the annotation is based on an InterPro observation. Some of the InterPro entries contain gene names as well. Gene names are written in a free text field describing an InterPro entry. This is done, because different members of the same protein family can be associated with different gene names. In order to be able to assign a gene name to the query sequence, **MicHanThi** must know which protein within a family is matched. Since InterProScan does not provide this information, **MicHanThi** does not assign gene names based on InterPro observations.

4.1.6 Genome Re-Annotation

Over the years, more information about functional genes is being added to public databases. This information can be used to verify annotations or predict new functions for ORFs, that have been previously annotated as (*conserved*) *hypothetical protein*. Hence, it makes sense to re-annotate already annotated genomes on a regular basis. **MicHanThi** applies the same process for the re-annotation of a genome as it does for the annotation of an unannotated genome. Once a genome is published, its annotation is added to the public databases. This poses a problem for the re-annotation because a “perfect” match will always be found for each query sequence. For functional annotations, this means that the prediction of the function is biased by the previous annotation of that ORF. If an ORF can not be functionally characterised, it will always be annotated as *conserved hypothetical protein*, because a matching sequence is found in the database. To avoid this problem, **MicHanThi** can be configured to ignore a list of organisms. This list can also be used to filter badly annotated genomes.

4.2 Design and Implementation

As mentioned before, **MicHanThi** is not intended to be a stand-alone tool. It needs an annotation system to provide the data, it needs external databases to query additional annotation information, and it needs a grid engine to analyse an entire genome. The interactions between these systems are depicted in figure 4.7.

The design of **MicHanThi** was greatly influenced by the design of the **Gen-ann** annotation tool developed by the author and other members of the GEN!E project. An important design aspect of the **MicHanThi** software is adaptability. Therefore, it is necessary to abstract from the sources of information such as the annotation system as well as the analysis tools because the *state of the art* in the

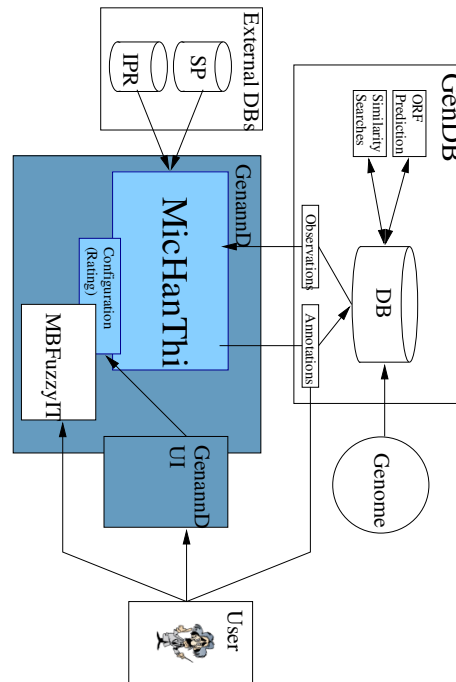


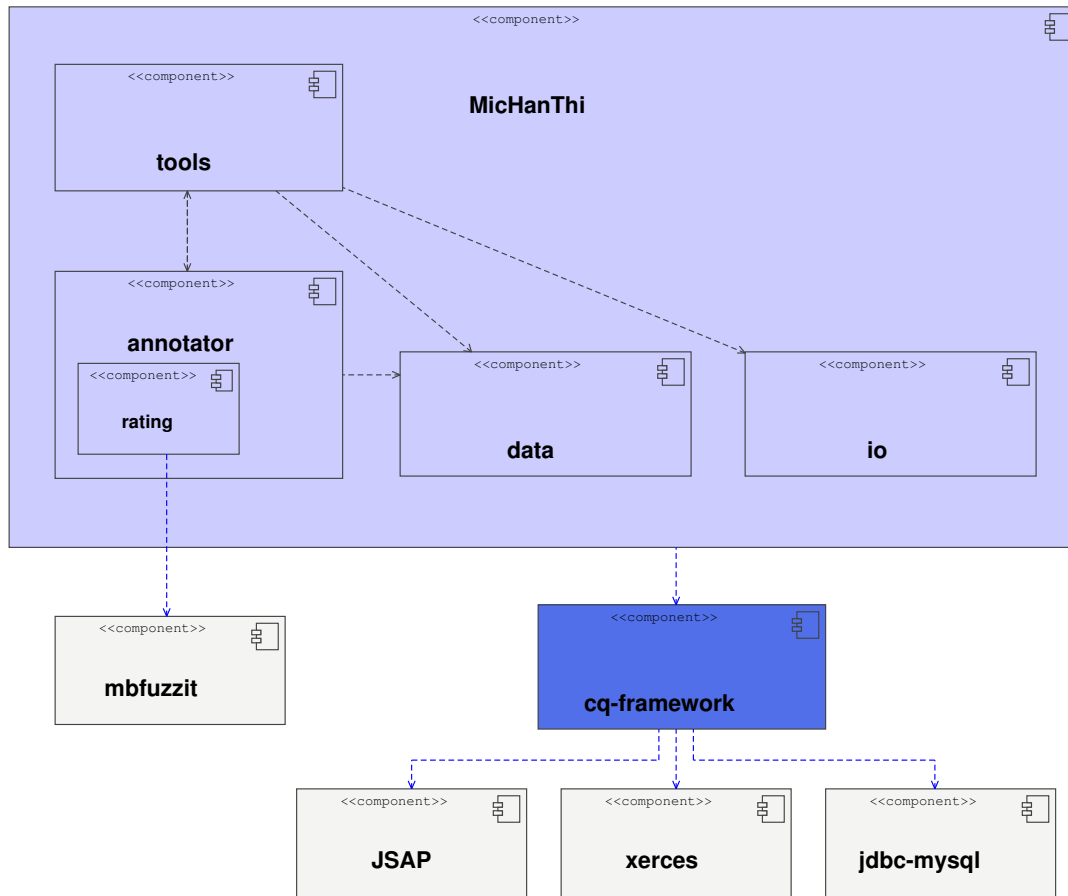
Figure 4.7: The integration of **MicHanThi** with the GenDB System and external data sources and the possibilities of the user to interact with the system.

field of bioinformatics changes rapidly.

MicHanThi is divided into four modules: (i) the **IO** module, (ii) the **DATA** module, (iii) the **TOOLS** module, and (iv) the **ANNOTATOR** module. These modules will be thoroughly explained in the sections below. After **MicHanThi** is started, the main program initialises the data sources and it retrieves information about an ORF and information about the observations describing an ORF from the data source. Once the initialisation process is finished, the **ANNOTATOR** module is called to rate the observations and to create the annotations. Last, the main program writes all annotations to the data source.

4.2.1 Module IO

The IO module offers classes to connect to the different data sources used by **MicHanThi**. Furthermore, it extends the *cqf.io.Prefs* class to add program specific options to the global configuration object provided by the main application class *cqf.Application*. As mentioned several times before, **MicHanThi** depends on an annotation system to provide and to store data. The experience gained by the **Microbial Genomics Group** during the annotation of several genomes shows that the demands regarding an annotation system change. To meet these changing requirements, the group moved from using *Pedant* to using *GenDB* ver-

Figure 4.8: The package structure of **MicHanThi**

sion 1.x and later to version 2.x of the *GenDB* system. Section 3.1.2 shows that these annotation systems offer different methods to query information about an ORF and its observations. Another factor which influenced the design of the IO module was the fact that the data model of an annotation system may change between versions. For these reasons, **MicHanThi** needs to be independent of any annotation system and its current data model.

To meet this requirement, the interface *DataSource* was introduced. This interface offers a set of methods, which can be used to query an annotation system for information about an ORF and information about the observations describing an ORF. The data returned by the query methods is then used to initialise the *DATA module*. For more information about the *DATA module* see section 4.2.2. The interface also offers methods to write annotations to the database used by an annotation system. At present, three classes implement the *DataSource* interface: *GenDBv1_SQL*, *GenDBv2_SQL*, and *GenDBv2.2_SQL*. All three implementations of this interface are used to query information from different version of

the *GenDB* annotation system (versions 1.x, 2.0.x, and 2.2.x). These classes extend the *cqf.io.SQL* class to be able to directly query the database engine used by *GenDB*. According to the developers of *GenDB*, this should not be done because the database schema may change between minor releases of the *GenDB* system, whereas the abstraction layer *O2DBI* is stable across minor releases. Therefore, the abstraction layer should be used to query the database. This is impractical because bindings exist only for the Perl programming language. The matter of missing bindings for the Java programming language was discussed at a meeting between members of the GEN!E project and the *GenDB* developers in 2003 and again, in 2005, at a meeting between members of the **Microbial Genomics Group** and the *GenDB* developers. At the first meeting, it was decided to set up a SOAP client / server architecture⁶ to exchange *GenDB* objects over a computer network. A prototype of the client was implemented within the GEN!E project. The server was to be developed by the *GenDB* developers. According to the developers, a prototype exists but the development was stopped, because internal tests yielded performance problems. The topic of Java bindings was again discussed at the second meeting, but the *GenDB* developers showed no interest in supporting the Java programming language. Therefore, **MicHanThi** queries the database directly despite the objections made by the *GenDB* developers. Other annotation systems than *GenDB* can be supported by implementing the *DataSource* interface for that particular system. Options provided by the global configuration object can be used to specify which of the supported annotation systems provides the information about the ORF to be analysed. The configuration object can also be used to specify authentication information to access the annotation system if necessary.

The *GenAnn_Connect* class was developed during the GEN!E project. It is used to query local versions of the SWISS-PROT and InterPro databases for additional information about an ORF. The class extends *cqf.io.SQL* because the local versions of these databases are stored in a MySQL server. The classes implemented in this module and their dependencies are depicted in figure 4.9

4.2.2 Module DATA

The DATA module represents the information necessary to annotate an ORF. It represents the ORF, information about the ORF (observations), information about the observations found in the SWISS-PROT or InterPro databases, and it represents the annotations of an ORF. The data provided by this module is used by the ANNOTATOR (4.2.4) and TOOLS (4.2.3) modules to create the annotations for an ORF.

One of the design goals of **MicHanThi** was to be independent of the analysis

⁶SOAP - Simple Object Access Protocol - is a protocol to exchange XML based messages over a computer network

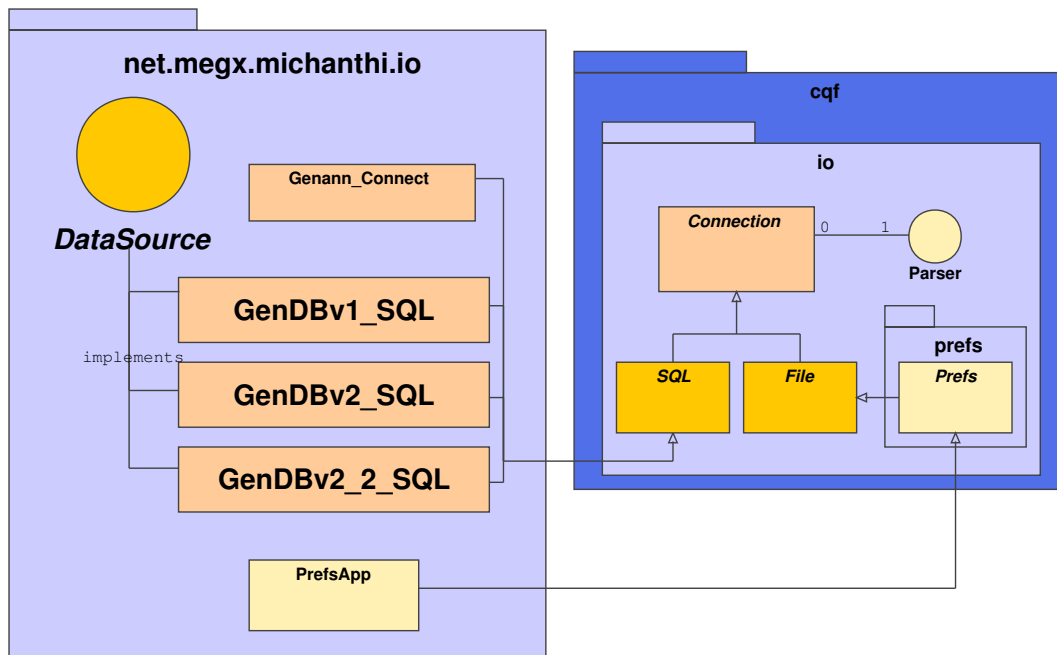


Figure 4.9: Classes of the IO module

tools used to characterise an ORF. To achieve this goal, the DATA module has to provide some kind of abstraction of the results of different tools. This abstraction is done by creating the two root classes *Observation* and *DBEntry*, which contain the most general set of attributes common to all analysis tools. Additional attributes provided by certain tools such as attributes describing the reliability of an observation are stored in subclasses of the *Observation* class. A functional description of an ORF can solely be derived using the information provided by the two classes *Observation* and *DBEntry*. A statement about the reliability of such an annotation can not be made though. To evaluate the reliability of an annotation, the information contained within the subclasses must be considered.

Initialisation of classes implemented in the DATA module is done by calling a method of the *net.megx.michanthi.io.DataSource* class. This method queries the data source for information about the class and returns a *java.util.HashMap* containing the information retrieved. The *java.util.HashMap* in turn is passed to the constructor of the class to be created. This rather complicated procedure is done to minimise the dependencies between the IO and DATA modules. It allows to change the set of attributes stored within each class, without changing the signatures of the methods involved in the initialisation process. Furthermore, the maintenance of the source code is easier. The following pseudo code shows how observations are retrieved from a data source.

```
class ORF
```

```

...
function getObservations by <tool name>
  if not retrieved obs of <tool name>
    DataSource -> getObservations by <tool name>
    for each HashMap returned
      create new observation <- HashMap
      add observation to the list of observations
    end
  end
end

  return observations of tool <tool name>
end
...
end

```

An overview of the DATA module is depicted in figure 4.10 on page 66. The paragraphs below describe the key classes of the DATA module.

ORF: *ORF* is the “main” class of the DATA module. It can be used to query all information about an ORF including observations and annotations. After an instance of the *ORF* class is created, it does not contain any information besides the ORFs start and stop positions within the genomic sequence, the name of the ORF, the unique identifier of the ORF within the data source, the name of the annotation project, and its reading frame. To keep a low memory footprint, observations and database entries are retrieved from the data source the first time they are used. All observations about an ORF are stored in the same list. The *ORF* class offers methods to query one observation based on its unique identifier in the data source, a set of observation created by the same tool, and all observations. To get faster access to observations of a specific tool, caches for each tool are created which keep a pointer to all observations of the same tool. The memory used to build the caches can be neglected compared to the speed gained to access observations of tool *X*.

Observation: The *Observation* class is the root class for all observations. It contains information that should be common to most of the analysis tools. These information is:

- the start and stop positions of the ORFs subregion described by the observation (from, to),
- a short text describing the findings (description_raw),
- a processed version of the description (description - see section 4.1.1 for more details on the processing of descriptions),

- the source of the observation if it was created by comparing an ORF to entries in a database (source),
- the unique identifier of the entry within the database queried (sourceRef),
- the database entry itself (*DBEntry*),
- a list of semantically equivalent descriptions (synonyms - this list exist only if the observation is based on an entry of the SWISS-PROT database),
- a reliability value assigned by the ANNOTATOR module (see section 4.1.2 for more details on the rating of observations), and
- the name of analysis tool which created the observation.

This class must be specialised if additional information of an analysis tool is needed to derive a function of an ORF. Subclasses implemented as part of this thesis are: *ObBlast*, *ObInterPro*, *ObSignalP*, and *ObTMHMM*. The three classes mentioned last hold only information specific to the particular tool. Class *ObBlast* additionally implements methods to process the description of BLAST observations as described in section 4.1.1.

***DBEntry*:** Database entries are represented by the *DBEntry* class and its subclasses *SwissProtEntry* and *InterProEntry*. Every database entry contains a text describing the entry as well as the unique identifier of the entry within the database. Both subclasses contain an EC number as well as a list of GO numbers. These are not implemented in the root class because not all databases necessarily provide this data. Entries of the SWISS-PROT database additionally contain the gene name of a protein. InterPro entries do not provide a gene name but they contain the type of the entry. The type attribute is checked by the *tools.ToolInterPro* class to determine if the observation should be deleted. Information provided by the *DBEntry* class and its subclasses is used by the annotation tools to assigned other annotation attributes than the gene function (see section 4.1.5).

***Annotation*:** A functional description of an ORF is represented by the *Annotation* class. It contains all information of a functional description that should be assigned to the ORF. Default values are assigned to each attribute, in the case that observations of a specific tool do not provide this information. The *Annotation* class provides methods to create “hypothetical” annotations. These methods can be used to describe ORFs for which no functional prediction is possible.

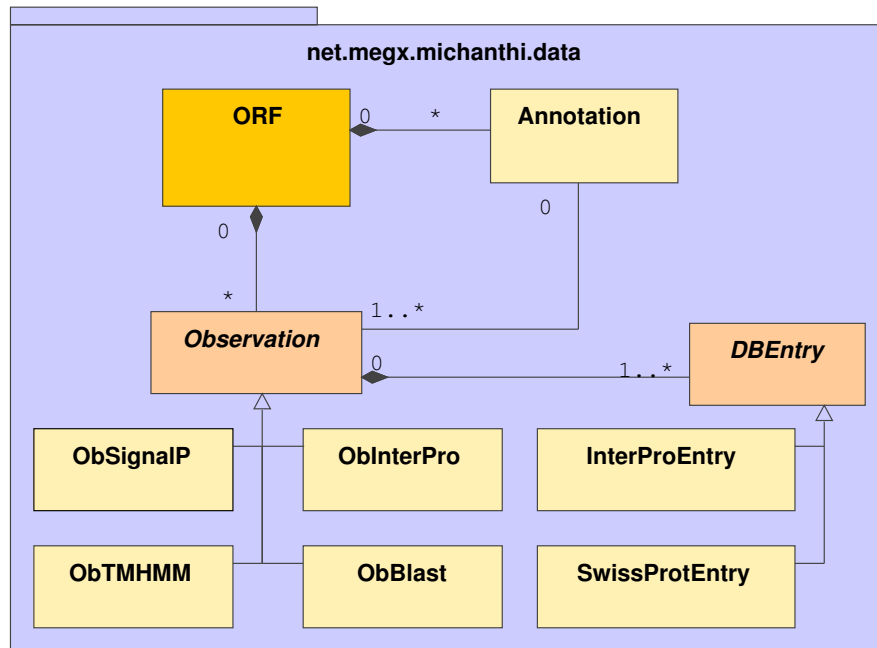


Figure 4.10: Classes of the DATA module

4.2.3 Module TOOLS

As mentioned before, a large number of different tools can be used to analyse genomic sequences. The work of the GENIE project showed, that the integration of information from different analysis tools is at best difficult if not impossible because the information provided by these tools may describe different things. The conclusion which can be drawn from this perception is that the observations of each tool need to be handled differently. Therefore, annotations derived from different types of observations are created by different implementation of the root class *Tool*.

The ANNOTATOR module expects subclasses of the *Tool* class to implement two methods. The method *rateObservations: Observations → rated Observations* must be implemented to rate observations and the method *createAnnotations: Observations → Annotations* must be implemented to create annotations. The *createAnnotations* method receives a list of all observations. From this list, the implementing class needs to choose a subset of observations as the basis of a functional description of an ORF.

The implementations of the tools InterPro, SignalP and TMHMM return a single annotation. An annotation based on InterPro is either an “hypothetical” annotation or an annotation which classifies an ORF as a member of a certain protein family. The SignalP and TMHMM tools always return “hypothetical”

annotations. If the observations for either tool provide enough evidence that an ORF is secreted or it contains a transmembrane region, then the annotation of the ORF is marked accordingly. It is more complex to derive a function from a set of observations based on the BLAST tool. These observations need to be grouped to find the most common annotation. This grouping process is complex in it self and therefore it is encapsulated in the class *Clusterer*. Groups of observations created by the *Clusterer* are described by the *Cluster* class. This class stores the informative terms that define a cluster and the observations that support a cluster. It provides methods to query the state of the group as well as methods to check if the group is supported by some type of observations. The different strategies applied to create a functional description of an ORF are explained in section 4.1.4

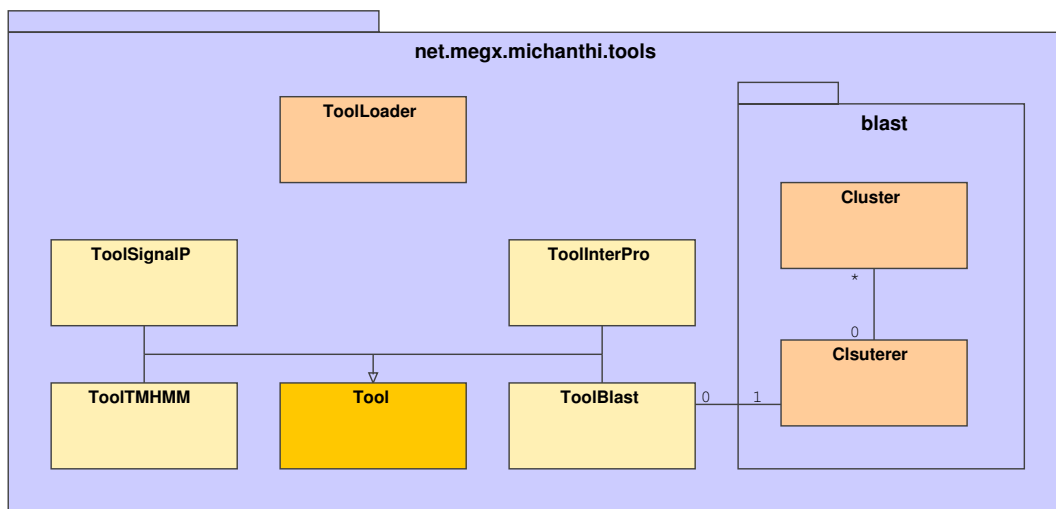


Figure 4.11: Classes of the TOOLS module

4.2.4 Module ANNOTATOR

The ANNOTATOR module is a rather simple module. It contains only two classes, *Annotator* and *Rater*. For each type of observation, the *Rater* class calls the method *rateObservations: Observations → rated Observations*. After each observation is rated, a subset of observations is selected by the *Rater* for the annotation process. The rating and selection of observations is explained in sections 4.1.2 and 4.1.3. From the set of selected observations, a functional description of an ORF is created. The annotation process is controlled by the *Annotator* class. It calls the method *createAnnotations: Observations → Annotations* of each analysis tool, which returns a set of annotations based on that

particular tool. The annotations created by the different tools are then merged as described in section 4.1.4 and returned to the main program.

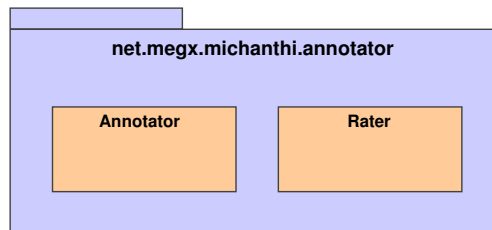


Figure 4.12: Classes of the ANNOTATOR module

4.2.5 The CQ Framework

MicHanThi is based on the *cq-framework*⁷. Among other things, this framework provides basic features like the logging of output to (file-)streams and a global configuration object. It also provides more complex modules such as the plug-in and the IO modules.

Logging: The logging of output is implemented in a simple class (*Logfile*) which provides methods for different levels of output. The user can select from one of the following output levels: ‘no messages at all’, ‘error messages’, ‘warning messages’, ‘default message’, ‘verbose messages’, and ‘debug messages’. The output levels are sorted by verbosity, that is, each level prints the messages of the previous level(s) as well. For example, the level ‘warning messages’ prints also messages that are marked as ‘error messages’ and the level ‘debug messages’ prints all messages.

Configuration: The configuration module is based on the *Java-based Simple Argument Parser* (JSAP) library⁸. This library allows to merge a set of default options with those provided in a configuration file and those provided on the command line. Configuration options specified on the command line overwrite those read from a configuration file (user or system level), which in turn overwrite hard coded default values. Configuration files can either be written in “native” JSAP format or XML. JSAP provides a parser that handles files written in its “native” format. To be able to use configuration files written in a different format, it is required to implement the *JSAP.DefaultSource* interface. To meet this requirement, the *cq-framework*

⁷The *cq-framework* is a collection of classes written by the author prior to this thesis

⁸<http://www.martiansoftware.com/jsap/>

implements the *Prefs* class which implements the *JSAP.DefaultSource* interface. The *Prefs* class includes a parser based on the Xerces2 library⁹ to support configuration files written in XML. A different file format can be used if the *cq.io.Parser* interface is implemented. An instance of the parser must then be passed to the *Prefs* class using the *set_parser: cqf.io.Parser* \rightarrow *void* method.

Plug-ins: The plug-in module can be used to dynamically load extensions of third parties, which extend the functionality of the main program. Although, **MicHanThi** does not facilitate the loading of plug-ins at present, it may prove useful in future version of the software. For example, the tools, utilised by **MicHanThi** to derive a functional annotation, could be implemented as plug-ins. The plug-in would specify which kind of observations should be retrieve from the data source, then it would rate the observations, and last, it would create annotations annotations based on these observations.

IO: The IO module offers basic data access classes such as a *File* class and a *SQL* class. The *File* class encapsulates the *java.io.File* class and it provides methods to read the content of a file and to create a copy of a file. For example, the *Prefs* class is derived from *File* and it uses its copy method to copy configuration files to the users home directory.

The *SQL* class abstracts of the SQL and the *Java Database Connection* (JDBC) driver. It loads and unloads the JDBC driver used to connect to a particular database engine such as MySQL or PostgreSQL, it creates and closes connections to a database, and it offers methods to query and to modify a database. The most beneficial feature provided by the *SQL* class is the abstraction of the SQL. For example if new data should be added to a table, then the user would have to write a SQL statement much like "INSERT [INTO] tablename [(col.name,...)] VALUES (expr|DEFAULT,...)". Adding data to a table using the *SQL* class is done by calling a method that receives the table name and a list of field and value tuples to be added to the table (*insert: java.io.String* \times *java.io.HashMap* \rightarrow *int*). Based on the parameters provide, the methods then creates the appropriate SQL statement for the user.

The *File* and *SQL* classes are derived from the *connection* class. This class provides a common interface to connect to different types of sources from which data can be obtained and defines a set of method that have to implemented by subclasses.

The *configuration module* as well as the *Logfile* class are also part of the IO module.

⁹Xerces2 is a DOM level 3 compliant XML parser developed by the apache foundation.

In addition to the modules described above, the cq-framework provides a collection of miscellaneous helper classes. Of these classes, **MicHanThi** uses only the *cfg.misc.Version* class, which represents a version string. This class can be used to compare two versions to check if some requirements are met. Figure 4.13 gives an overview of the classes provided by the cq-framework.

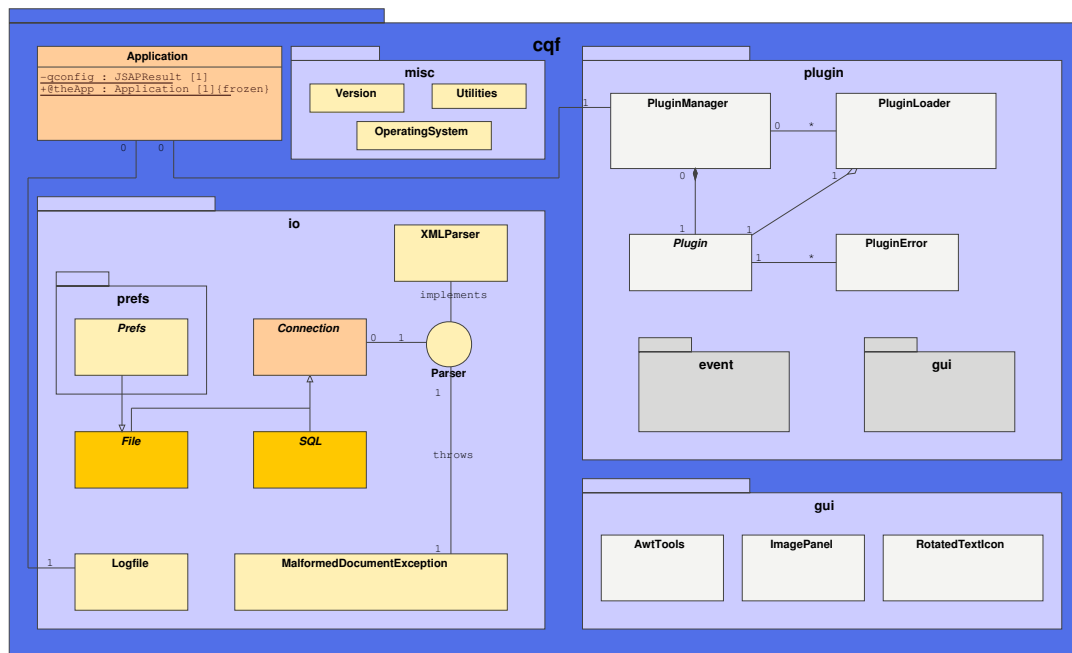


Figure 4.13: Classes of the cq-framework

4.3 Additional Tools implemented within the Diploma Thesis

After the main work implementing **MicHanThi** was completed, some additional features were requested by the human annotators. These features, although closely related to the annotation of a genome, are not related to the prediction of gene functions. The author decided to implement these features as additional tools using the same framework as the annotation tool.

4.3.1 MOBH - Mark ORFs based on best BLAST hit

This script sets the *status_region* attribute of class *Region*. Originally, *GenDB* used this attribute to show the reliability of the ORF prediction. An ORF can either be putative, final, or at a stage in between. MOBH marks each ORF based

on the reliability of the best BLAST observation found, which can be seen as the probability of the Region existing as a real gene. MOBH uses as many different status values as *GenDB* does, but different terms are used at the request of the annotator.

4.3.2 UPGENEC - UPdate GENe name and EC number

Another special feature requested was the ability to update of gene names, EC numbers and GO numbers for already annotated ORFs. The gene function can not have been changed because it may have been validated by a human annotator. Therefore old annotations are updated with missing information and new annotations are only created if a ORF does not have any annotation associated to it. For each ORF, a list of BLAST against SWISS-PROT and InterPro observations is retrieved from the database. This list is then checked for observations matching the function of the ORF. To check if an observation matches the predicted function of an ORF, **MicHanThi** uses the same approach as it does to merge two annotations during the annotation process. If an observation matches, it is checked for new information. The information is then transferred to the annotation and is skipped for the rest of the matching observations. For example, if a new gene name is found, the gene names of subsequent matching observations will be ignored. If all annotation fields are updated or there are no more observations in the list the process is ended. The user has two options for changing the behaviour of UPGENEC. The user can decide if substring matches of the function of two annotations are allowed and if information already present in the annotation should be overridden with information found in the observation. For more detailed information about substring matches see 4.1.4.

4.3.3 GenannD

GenannD is part of the Genann annotation suit developed by the GEN!E project. It was written by the author of this thesis and is used as a grid engine to distribute jobs among cluster nodes. **MicHanThi** creates annotations for one ORF only. To annotate all ORFs of a genome, **MicHanThi** must be started for each ORF separately. This is done by the GenannD tool. A job “containing” all ORFs that have to be annotated is created. Each node of the cluster is assigned one ORF at a time, until all ORFs are annotated. The cluster node then starts the prediction tool which annotates the ORF, writes the results to the *GenDB* database, and gives feedback about the reliability of the annotation. The cluster node reports this status back to the master node, which keeps a statistic about all annotated ORFs. When a job is created several options can be set. Among these options are: access information about the *GenDB* database, the *GenDB* project to be used, and the ORFs of the project to be annotated. The user can define any MySQL statement, valid to query the *GenDB* database, in order to restrict

the number of ORFs. For example, only ORFs that have not been annotated yet, can be scheduled for annotation, or all ORFs that have been annotated as *hypothetical protein* can be re-annotated. As part of this thesis, the GenannD tool was extended to work with different versions of *GenDB* and use **MicHanThi** as the annotation tool instead of **genann**.

4.3.4 SPIMP

SPIMP was written by Thomas Soller as part of the GEN!E project. It creates a local SWISS-PROT MySQL database from a SWISS-PROT XML dump. Local versions of the SWISS-PROT and InterPro databases are necessary because **MicHanThi** uses these databases to get access to the additional annotation features. For further details see the final report of the GEN!E project.

Chapter 5

Results

5.1 The Test Run

5.1.1 The Test Setup

The hardware platform used to run the annotation software is a cluster consisting of: one dedicated database server running the MySQL software, one dedicated file server (NFS), one head node / web server controlling the grid engine (SGE) and serving the *GenDB* web front-end, and eleven nodes computing the observations. Each compute node is a dual Intel Xeon based IBM x335 server clocked at 2.80GHz (each CPU) and equipped with a total of 4GiB RAM. The local hard disk drives (HDDs) are used to increase the speed of similarity searches by storing a local copy of the bioinformatics databases used.

The analysis of the organism ‘*Gramella forsetii*’ KT0803 started by running MORFind to predict ORFs. Once the ORFs were predicted, the genome was imported into the *GenDB* annotation system and the observations for each ORF were computed. The ORF prediction and the similarity searches for the 3593 ORFs took approximately 24 hours to compute. The resulting data set has a size of approximately 330MiB. Thereof, 1,367,992 observations totalling approximately 320MiB. These observations have to be evaluated by the annotator in order to predict a biological function of the ORFs.

5.1.2 The KT0803 Annotation Jamboree

MicHanThi was evaluated within the context of the annotation jamboree of the marine bacterium ‘*Gramella forsetii*’ KT0803. The aim of this annotation jamboree was to explore the organism’s genome and to get a first insight into the features of the organism and its role in the environment. Eleven PhD and diploma / master students of the MPI Bremen and the International University Bremen (IUB) were trained by members of the **Microbial Genomics Group** to perform this task. The students and members of the **Microbial Genomics**

Group then annotated the organism in about nine weeks. Experienced annotators had to invest about three additional weeks to cross-check the annotations of all ORFs. Annotations were supplemented with missing information and if necessary, new annotations were created. Finally, Margarete Bauer (senior member of the **Microbial Genomics Group**) spent several months studying the organism's metabolism and its role in the environment. Primarily, she focused on the identification and understanding of key features of the organism such as a potential phytochrome (light sensor) and carbohydrate degradation. All together, it took sixteen annotators about three months to achieve a basic understanding of the organism and another three months were invested to find annotations highlights. The thorough annotation of the organism 'Gramella forsetii' KT0803 serves as the basis of the evaluation of the **MicHanThi** software implemented as part of this thesis.

5.1.3 The Performance of MicHanThi

MicHanThi was executed after the ORF prediction and the similarity searches were finished. Running eleven instances of **MicHanThi** on the above hardware setup (section 5.1.1), **MicHanThi** was able to annotate the 3593 ORFs of the organism 'Gramella forsetii' KT0803 in about 15 to 20 minutes. It created a total of 5404 annotations, which is an average of 1.50 annotations per ORF. 1289 of these annotations describe ORFs for which no functional prediction was possible (*[conserved] hypothetical proteins [, membrane or secreted]*).

5.1.4 The Evaluation

A simple tool (*create_statistics*) was written to compare the annotations created by the human annotators (human annotated) and those created by **MicHanThi** (automatically annotated). It compares two lists of annotations contained in separate text files. Each ORF found in the first file (annotated by a human annotator) is compared to all annotations for the same ORF found in the second file (annotations created by the program). The text files must have the following syntax and annotations must be sorted by ORF ID:

```
ORF ID \ t ORF Name \ t Gene Product \ n
```

create_statistics reports statistics from the annotation project such as: the number of human annotated ORFs, the number automatically annotated ORFs, the average number of annotation created per ORF, and the number of ORFs not annotated by the computer (total number and percentage). A comparison between the preliminary annotations created by the human annotators and those create by the computer are shown in table 5.1.

Next, the tool reports information about the number of matches found between the annotations created by the human annotator and the computer: the

# human annotations:	3574	
# automatic annotations:	5404	[1.51]
# no auto annotations:	19	[0.5%]

Table 5.1: Statistics from the comparison of the preliminary annotations created by the human annotator and those created by **MicHanThi**.

number of annotations using exactly the same words (in the same / different order), the number of annotations in which the gene product described by the human annotator is “more general” than the description created by **MicHanThi** (subset matches - ha), and the number of annotations in which the computer is “more general” than the human (subset matches - aa). For each class of matches, the tool reports additionally the number of ORF that have exactly two annotations, one created by a human annotator (marked as latest annotation in the *GenDB* annotation software) and one created by **MicHanThi**. Brackets are used to indicate the relative number of matches between the annotation of all ORFs. An overview of the preliminary number of matches is given in table 5.2. The numbers of the first, third, and fifth row make up the total number of matches between the human and automatically annotated ORFs. Adding these numbers yields a total number of 1918 matches which is about 54% of all annotations.

# exact matches/same words:	1350	38%	/ 16
# exact matches (one):	970	27%	
# subset matches - ha:	328	9%	
# subset matches (one) - ha:	100	3%	
# subset matches - aa:	240	7%	
# subset matches (one) - aa:	76	2%	

Table 5.2: Number of matches between the preliminary annotations created by the human annotators (ha) and those created by **MicHanThi** (aa).

The last set of statistics reported by the tool are details about the class ORF which can not be described functionally. This class of ORFs was investigated in detail because clear rules exist for the cases in which an ORF should be annotated as *hypothetical protein* or *conserved hypothetical protein*:

hypothetical protein: An ORF is described to be a *hypothetical protein* if no matches could be found in any of the sequence databases or if such matches exist but they are unreliable ($E - value \geq 1e^{-3}$).

conserved hypothetical protein: The attribute *conserved* is assigned to an ORF if at least one reliable match could be found in one of the sequence

databases. A match is reliable if $E - value < 1e^{-3}$ and ORF coverage $\geq 25\%$ and DB coverage $\geq 25\%$.

transmembrane prediction: ORFs that have at least two reliable transmembrane helix predictions are assigned the attribute *membrane*.

signal peptide prediction: If no more than one transmembrane helix was predicted for an ORF and a reliable signal peptide prediction exists, then the ORF is annotated as secreted. A signal peptide prediction is considered to be reliable if its probability as reported by the HMM is ≥ 0.75 and its cleavage site probability is > 0.5 .

transmembrane and signal peptide predictions: If exactly one reliable transmembrane helix prediction exists for an ORF and the predicted signal peptide prediction is uncertain because its HMM cleavage site probability is ≤ 0.5 , then the ORF is annotated as *membrane or secreted*.

The first column of table 5.3 lists the classes to which an ORF without functional description can be assigned. Absolute and relative numbers of ORFs assigned to each class are shown in columns two and three (human annotator / software). ORFs that have been assigned to the same class by both the human annotator and the software (annotation matches) are represented by the first number in the last column. The last two numbers in the last column represent the precision and recall measures.

The table is divided into three parts: the total number of ORFs without a functional description (*hypotheticals* and *conserved hypotheticals*), statistics of ORFs without any reliable similarities found in sequence databases (*hypotheticals*), and statistics of ORFs with at least one reliable match but no function could be derived from these matches (*conserved hypotheticals*). ORFs annotated as *conserved hypothetical protein* may contain one or more domains.

Annotations created by the human annotator are considered to be very reliable. If it is assumed that the objective of **MicHanThi** is to find all ORFs that can not be functionally described, then the tool should find all ORFs that are represented by the numbers in the second column. Therefore, the first number in the last column describes the number of ORFs correctly identified by **MicHanThi** (*true positives*). From this the number of *false positives* and *false negatives* can be calculated. *False positives* are those annotations created by **MicHanThi** that could not be verified by a human annotator: annotations created by **MicHanThi** minus number of matches. ORFs assigned to one of these classes not found by **MicHanThi** are the *false negatives*: the number of matches subtracted by the number of ORFs assigned to a class by the human annotator. The precision and recall values are then calculated as:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \quad (5.1)$$

$$\leftrightarrow precision = \frac{truepositives}{\# \text{ MicHanThi } annotations} \quad (5.2)$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad (5.3)$$

$$\leftrightarrow recall = \frac{truepositives}{\# human\ annotations} \quad (5.4)$$

If the assumption that annotations created by a human annotator are very reliable is true, then **MicHanThi** performed poorly. Over-all, only about 50% of the ORFs that should have been assigned to one of these classes could be identified by **MicHanThi**. Most of these matches occur in the class *hypothetical protein*. If additional information is present such as transmembrane helix or signal peptide predictions, then the number of matches decreases drastically. The number of matches decreases further if similarities could be found in sequence databases. If a conclusion has to be drawn based on these numbers it could only be that matches between the annotations of human and automatically annotated ORFs occur, at best, randomly.

	Human [% of all ORFs]	MicHanThi	matches [precision, recall]
ORFs without function	1566 [44]	1284 [36]	725 [0.56, 0.46]
hypothetical proteins			
hypothetical protein	578 [16]	514 [14]	425 [0.83, 0.74]
transmembrane prediction	95 [03]	120 [03]	50 [0.42, 0.53]
signal peptide prediction	107 [03]	206 [06]	63 [0.31, 0.59]
transmembrane and signal peptide predictions	0 [00]	10 [00]	0 [0, —]
conserved hypothetical proteins			
conserved protein	443 [12]	188 [05]	131 [0.70, 0.30]
transmembrane prediction	64 [02]	68 [02]	22 [0.32, 0.34]
signal peptide prediction	84 [02]	104 [03]	26 [0.25, 0.31]
transmembrane and signal peptide predictions	8 [00]	4 [00]	0 [0, 0]
conserved hypothetical proteins containing some domain			
conserved protein	164 [05]	44 [01]	6 [0.14, 0.04]
transmembrane prediction	8 [00]	17 [01]	2 [0.12, 0.25]
signal peptide prediction	14 [00]	8 [00]	0 [0, 0]
transmembrane and signal peptide predictions	1 [00]	1 [00]	0 [0, 0]

Table 5.3: Detailed comparison of the preliminary annotations created by the human annotators and those created by the computer.

5.1.5 The Cross-Checking

The second part of the annotation jamboree was the evaluation of all annotations by experienced annotators. Subsequent to the cross-checking of annotations, the annotations created by the human annotators (now final) were once more compared to the same set of annotations created by **MicHanThi**. The results of these comparisons are shown in tables 5.4, 5.5 and 5.6.

# human annotations:	3593	
# automatic annotations:	5404	[1.51]
# no auto annotations:	19	[0.5%]

Table 5.4: Statistics from the comparison of the final annotations created by the human annotator and those created by **MicHanThi**.

The only difference between table 5.4 and table 5.1 is that all ORFs found in ‘Gramella forsetii’ KT0803 have now been curated by human annotators. Table 5.5 shows an over-all increase in the matches of the annotations of all ORFs from about 8% to a total of about 62%. However, the most important results of this new comparison are shown in table 5.6.

# exact matches/same words:	1633	45%	/ 19
# exact matches (one):	1235	34%	
# subset matches - ha:	309	09%	
# subset matches (one) - ha:	89	02%	
# subset matches - aa:	274	08%	
# subset matches (one) - aa:	97	03%	

Table 5.5: Number of matches between the final annotations created by the human annotators (ha) and those created by **MicHanThi** (aa).

The number of matches in all “ORF classes” increased. This is the case especially for ORFs without any similarities found in sequence databases (*hypothetical proteins*). **MicHanThi** is capable of finding almost all ORFs that have been classified by the human annotator as members of one of these subclasses of annotations. On the other hand, more than three quarters of the annotations assigned to one of these classes by the software could be verified by annotations created by a human annotator. The increase in the number of matches within the other classes is in some cases as much as it is in the group of *hypothetical proteins*. However, the over-all percentage of matches is lower - in general < 50% - but ORFs predicted by **MicHanThi** to be conserved are normally verified by a human annotator. Most significantly, this is true for the group of *conserved hypothetical proteins* that contain domains found in an ORF. The software assigns

only very few ORFs to these subclasses, whereas the human annotators assigns many more ORFs to these subclasses. Hence the small number of matches in these subclasses compared to the other classes. The small number of predicted ORFs can be explained by the strategy used by **MicHanThi** to describe domains found in an ORF. The software only uses a domain to describe the ORF if this domain is a *domain of unknown function* (DUF). These type of domains can be found in the InterPro database and describes conserved regions of a protein for which a function is not known. The low number of matches between annotations created by a human annotator and those created by **MicHanThi** in the group of *conserved hypothetical proteins* can be explained by the over prediction of ORFs with functional assignment by the software. It was decided that **MicHanThi** should tend to the over prediction of function for genes for two reasons. First, any faint similarity to one or more sequences in bioinformatics databases should be reported. Secondly, the automatic evaluation of functional predictions of genes is, at best, difficult.

Another number that stands out in the second table is the total number of ORFs to which no function could be assigned by **MicHanThi**. Even though the *create_statistics* tool used exactly the same set of annotation created by **MicHanThi** as it did before, five more ORFs were found by the tool the second time it was applied. This can be explained by the fact that the tool only evaluates annotations created by **MicHanThi** for those ORFs which have also been annotated by a human annotator. The five ORFs which occur in the second set of statistics previously lacked an annotation created by a human annotator. Therefore, they were ignored by the *create_statistics* tool, in the first comparison.

	Human [% of all ORFs]	MicHanThi	matches [precision, recall]
ORFs without function	1613 [45]	1289 [36]	1024 [0.79, 0.63]
hypothetical proteins			
hypothetical protein	463 [13]	516 [14]	454 [0.88, 0.98]
transmembrane prediction	99 [03]	120 [03]	94 [0.78, 0.95]
signal peptide prediction	171 [05]	207 [06]	158 [0.76, 0.92]
transmembrane and signal peptide predictions	10 [00]	10 [00]	6 [0.60, 0.60]
conserved hypothetical proteins			
conserved protein	341 [09]	188 [05]	142 [0.76, 0.42]
transmembrane prediction	107 [03]	68 [02]	45 [0.66, 0.42]
signal peptide prediction	167 [05]	105 [03]	78 [0.74, 0.47]
trans membrane and signal peptide predictions	12 [00]	4 [00]	3 [0.75, 0.25]
conserved hypothetical proteins containing some domain			
conserved protein	160 [04]	44 [01]	29 [0.66, 0.18]
transmembrane prediction	47 [01]	18 [00]	13 [0.72, 0.28]
signal peptide prediction	35 [01]	8 [00]	3 [0.38, 0.09]
transmembrane and signal peptide predictions	1 [00]	1 [00]	0 [0, 0]

Table 5.6: Detailed comparison of the final annotations created by the human annotators and those created by the computer.

5.1.6 The Problem of Semantics

A fully automatic evaluation of annotations of functionally described ORFs is difficult. Automatic tools or scripts can only be used to find annotations that use the same or a subset of words to describe the function of an ORF. An additional point hampering the automatic evaluation is that different human annotators tend to use different words for the same functional description of a protein. Therefore the list of observations as well as the annotations created by a human annotator are rather diverse. Table 5.8 is an excerpt of annotations from the first 100 ORFs comparing the annotations of human annotated ORFs and those that have been automatically annotated, illustrating the problem. The different descriptions used for the annotation of each ORF describe the same function. About 10% of all annotations fall into this category. This number should be considered when finally evaluating the performance of **MicHanThi**.

ORF	Human Annotator	MicHanThi
1: orf9	- permease, major facilitator superfamily	- Permeases of the major facilitator superfamily - [...]
2: orf10	- glycosyl hydrolase, family 32	- Glycoside hydrolase, family 32 - [...]
3: orf21	- permease, major facilitator superfamily	- Permeases of the major facilitator superfamily
4: orf25	- heavy metal-(Cd/Co/Hg/Pb/Zn)-translocating P-type ATPase	- cadmium-translocating P-type ATPase - Cation transport ATPase - cation-transporting ATPase, P-type, putative zinc-transporting ATPase - heavy metal transporting ATPase
5: orf30	- heavy metal cation efflux protein containing OEP domain, CzcA family	- Cation efflux system protein czcA - heavy metal efflux pump, CzcA family - [...]
6: orf33	- glycosyl hydrolase, family 53 - likely arabinogalactan 1,4-beta-galactosidase	- Arabinogalactan endo-1,4-beta-galactosidase - [...]
7: orf40	- short-chain dehydrogenase/reductase family protein	- oxidoreductase, short-chain dehydrogenase/reductase family - [...]
8: orf62	- protein involved in phosphonate metabolism	- PhnA protein
9: orf69	- arylformamidase	- N-formylkynurenine (aryl-) formamidase
10: orf70	- Holliday junction nuclease RuvC	- Crossover junction endodeoxyribonuclease RuvC - Holliday junction resolvase, endonuclease subunit

Table 5.8: Semantically comparable annotations within the first 100 ORFs, which are not reported by the statistics tool.

In Examples 1 and 3, the difference is that the term *permease* is used in singular form by the human annotator while it is found in plural form in the list of observations. Hence, **MicHanThi** used the plural form to annotate the ORF. The annotation in example 2 use slightly different spellings of the term describing the function (*glycosyl* vs. *glycoside*). Examples 6 to 9 differ in the specialisation

of the function assigned to the ORF. Normally, the statistics tool would classify these cases as a subset match between human and automatically annotated ORFs. In these cases the tool failed to do so because in both annotations, words are used that are not used in the other annotation. In all four cases, the human annotator is more careful when assigning the function. In example 6, the annotator uses a more general description of the same function than the computer does and adds the more specific description as a supplement. The annotation created by the computer for ORF orf40 is *oxidoreductase* with the addition that this type of reductase belongs to the *short-chain dehydrogenase/reductase family*. The human annotator was uncertain whether or not the ORF is an oxidoreductase and described it as a protein belonging to this family. Example 8 is a rather interesting case because the protein name ‘PhnA’ is ambiguous, describing two “slightly” different functions within the phosphonate metabolism. The human annotator decided to describe the ORF as a protein involved in that metabolism. Examples 4, 5, and 10 are more difficult. Nevertheless all annotations created for the same ORF describe the same function.

5.1.7 The Missing 19 Annotations

19 ORFs were not annotated by **MicHanThi**. This number changes each time the program is used to annotate all ORFs of a genome annotation project. Why **MicHanThi** is not able to annotate all ORFs can either be found in the MySQL server software, in the JDBC-driver used to create a connection to the MySQL server, or in the design of the *GenDB* database. Each time **MicHanThi** writes an annotation to the *GenDB* database it locks all tables of the project database for both read and write access. Under some circumstances this locking fails and two instances of **MicHanThi** try to write to the database at the same time. By definition of the MySQL server software this should not be a problem. However, the design of the *GenDB* database relies on a single unique identifier (primary key) for each table. If two instances of **MicHanThi** try to write the annotations they created to the database at exactly the same time, then one of the instances reports an exception of the primary key already being used. A look into **MicHanThi** log files reveals, however, that annotations have been created. If only one instance of **MicHanThi** is running and therefore the database is not access concurrently, then annotations of all ORFs are written to the database. If multiple instances of the annotation tool are used, approximately 0.5% to 1% of the ORF will not be annotated because the problem described above. The small number of ORFs not annotated as well as the fact, that this problem is considered to be outside of the implementation of **MicHanThi** makes it a low priority.

5.2 Limitations of the Prototype

A feature that was not implemented as part of this thesis is the interpretation of observations describing a domain match within the InterPro database. Each protein is composed of a combination of domains which make up its function. If one of the domains is substituted by another domain, then the over-all function of the ORF may or may not change. To adequately handle this kind of information every protein found in a sequence database would have to be broken down into the domains that build the protein. Additionally, the domains that make up the ORF would have to be identified. The analysis of the domain composition of a protein would only be possible if a comprehensive database of protein domain relationships existed. Since this is not currently the case, InterPro domains are ignored.

5.2.1 The Rating of Observations

In section 4.1.2, different attributes of the BLAST family of tools which are used to evaluate observations are listed: the E-value, the ORF coverage, the DB coverage, and the relative number of identities. The rule basis used by the prototype to rate BLAST observations does not use the relative number of identities. This number depends on the number of already sequenced organisms that are closely related to the studied organism (in the sense of taxonomy). The more organisms are sequenced, the more likely it is to find sequences in databases with a large number of matching bases. The membership functions defined for this set would have to be adapted for each analysed organism. Another reason why this number is not used is that the rule basis became unmanageable. At present, 36 rules are used to rate a BLAST observation. If the relative number of identities is included in the rating, then 108 rules would have to be maintained. However, the increase in the expressiveness of the rule basis does not increase proportional to its complexity.

Fuzzy logic is well suited to represent knowledge about a discreet universe whose elements can be mapped onto a small range between its upper and lower bounds. This is certainly the case for the two coverages as well as the relative number of identities but not for the E-value. First of all, the concept represented by an E-value is not discreet and secondly, only a very small subset of E-values is of interest. It makes almost no difference if an observations has an E-value of $1e^{-100}$ or $1e^{-120}$. Both E-values are considered to be equally good. An observation with an E-value of $1e^{-2}$ should not be considered for the prediction of a gene function while an observation with an E-value of $1e^{-20}$ should be considered. An E-value of $1e^{-100}$ is not five times as reliable as an E-value of $1e^{-20}$ but this E-value is more than five times more reliable than an E-value of $1e^{-4}$. Representing all these constraints within a set of membership functions is tedious and error-prone.

5.2.2 Prediction of Gene Functions

The functional description of ORFs seems to be more or less usable but a thorough evaluation of this class of annotations is missing. In most cases, **MicHanThi** predicts a function that is comparable to the annotation created by a human annotator even though different words may be used. In some cases, the software utterly fails and it assigns “functions” such as ‘orf234’, ‘A’, or an incomplete description. The assignment of the function ‘orf234’ can be traced back to badly annotated genomes in which *hypothetical proteins* or *conserved hypothetical proteins* are often assigned internal ORF names. The description of observations containing only an ORF name should be filtered during the observation preprocessing (filtering of “uninformative terms”). The filtering of ORF names is implemented as a list of regular expressions (black list) that are matched against the description of an observation. If some terms used in the descriptions match one of the regular expressions, then the term is deleted from the processed description. Apparently, this list can never be complete. A better approach to filter “uninformative terms” may be the application of a white list based on a controlled vocabulary such as the *Gene Ontology*. Terms not found in that list would then be deleted.

The assignment of single terms such as ‘A’ as well as the assignment of incomplete descriptions are both caused by the same reason. The grouping of observations is based on single terms found in an observation’s description. The order of the appearance of a term within the description is also unimportant. This approach was chosen for two reasons: subset relations among the observations can be established which can be used to create a more general functional annotation and observations are placed in the same group even though the order of terms is different. An example of the latter would be the description ‘A B C D’ which is equivalent to ‘C D, A B’. This case is not unlikely among a lists of observations. The drawback of this approach is that if a single term or a “meaningless” combination of terms is over represented within the list of observations, then it may be used as the functional description of a gene.

5.3 Conclusions

An ever increasing number of available genomic sequence information makes the interpretation of genomes and especially metagenomes more and more difficult. Maintaining a comprehensive view on the data, databases, and tools used for the analysis of genomic sequences is a nearly impossible task. The automation of initially simple tasks such as the prediction of ORFs and similarity searches is the main focus of the annotation systems presented in chapter 3. As the field of *genomics* matures the questions considered increase in complexity. A newly investigated topic is the automatic prediction of gene functions. At present, several groups are working on this topic but no adequate solution is publicly available.

The work described in this thesis offers a reliable foundation for further studies of organisms and it is applied to all annotation project initialised by the **Microbial Genomics Group**. Compared to the annotations created by the human annotator, about 60% of the annotations predicted by **MicHanThi** were syntactically identical and in addition, about 10 percent were semantically equivalent as described in section 5.1.6. The class of ORFs without any similarities found in sequence databases could be reliably identified by **MicHanThi**. More than 90% of the ORFs without a functional assignment annotated by a human annotator were also found by **MicHanThi**. About 20% of all ORFs without functional assignment have been inaccurately annotated by **MicHanThi** in regards to the annotation for the same ORF by a human annotator. Compared to a group of human annotators this is a rather impressive achievement. The number of wrongly predicted ORFs among human annotators is much larger than that of **MicHanThi** (tables 5.2 and 5.5). Most annotations of ORFs without a functional assignment created by **MicHanThi** which could not be verified by annotations created by a human annotator are for ORFs whose observations questionable (E-value of about $1e^{-10}$ to about $1e^{-3}$ and coverages of about 25%). Annotations based on such observations greatly depends on the expertise of the human annotator. Even experienced annotators are indecisive as how to annotate these ORFs. The advantage of **MicHanThi** is that it always applies the same rules to decide if the attribute *conserved* should be assigned to the annotation of the ORF. A limitation of the **MicHanThi** prototype is its over prediction of ORFs with functional assignments. Nevertheless, incorrectly predicted functional annotations pose no significant problem because ORFs with functional assignments are further studied by human annotators. However, a thorough evaluation of the class of ORFs with a functional assignment is missing.

5.4 Perspectives

Comparative genomics: The most promising approach to increasing the reliability of functional assignments is the *comparative genomics* approach. The importance of this approach could be shown in [24, 54]). **MicHanThi** could benefit from this kind of information by considering the surrounding ORFs when it evaluates the annotations created for a certain ORF. This could reduce the average number of annotations created by **MicHanThi**, and, more importantly, increase the reliability of predicted annotations.

Metabolic Pathways: Metabolic pathways could be used to identify “missing” genes as shown by [27]. If most of the genes of a pathway have been found in an organism, then it is very likely that the remaining genes are present in the genome as well. **MicHanThi** could then use the list of “missing” genes and therefore missing functionally of a pathway to particularly look for hints of the missing function within all similarities found for an organism in bioinformatics databases.

ORF orf7: The annotation of orf7 created by the human annotator deserves special attention. Most of the descriptions found in the list of observations describe the potential function of this protein as *two-component system sensor histidine kinase/response regulator, hybrid* (*‘one component system’*). **MicHanThi** could not find any obvious rejections of this description and therefore used it as the functional prediction for the ORF. However, the human annotator decided to use a slightly different functional description and annotated the ORF as *sensor histidine kinase/response regulator hybrid, sugar binding*. This decision is based on the fact that the N-terminus of the protein is always matched by a *sensor histidine kinase/response regulator, hybrid* protein whereas the C-terminus is matched by proteins coding for some type of sugar binding. It would be the ultimate aim of **MicHanThi** to find these kinds of information. Even if **MicHanThi** is not able to combine the two functional descriptions, then it should at least be able to indicate to the human annotator that there could be additional information.

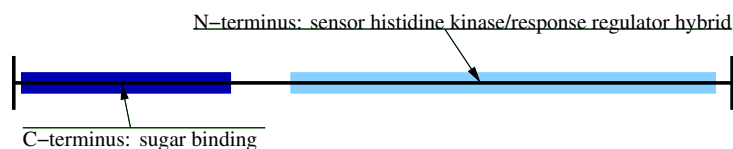


Figure 5.1: Annotation of ORF orf7. The C-terminus of the protein is always matched by proteins involved in sugar binding. The N-terminus is always matched by proteins coding for *sensor histidine kinase/response regulator, hybrid*

Appendix A

Rulesets

Rule	E-value	→ Reliability Observation
1	"unreliable"	→ "bad"
2	"uncertain"	→ "average"
3	"reliable"	→ "good"
4	"very_reliable"	→ "very_good"

Table A.2: Ruleset used to rate InterPro observations

Rule	E-value	\wedge Coverage ORF	\wedge Coverage DB	\longrightarrow Reliability Observation
1	"unreliable"	"none"	"none"	\longrightarrow "bad"
2	"unreliable"	"none"	"partial"	\longrightarrow "bad"
3	"unreliable"	"none"	"complete"	\longrightarrow "bad"
4	"unreliable"	"partial"	"none"	\longrightarrow "bad"
5	"unreliable"	"partial"	"partial"	\longrightarrow "bad"
6	"unreliable"	"partial"	"complete"	\longrightarrow "bad"
7	"unreliable"	"complete"	"none"	\longrightarrow "bad"
8	"unreliable"	"complete"	"partial"	\longrightarrow "bad"
9	"unreliable"	"complete"	"complete"	\longrightarrow "bad"
10	"uncertain"	"none"	"none"	\longrightarrow "bad"
11	"uncertain"	"none"	"partial"	\longrightarrow "bad"
12	"uncertain"	"none"	"complete"	\longrightarrow "bad"
13	"uncertain"	"partial"	"none"	\longrightarrow "bad"
14	"uncertain"	"partial"	"partial"	\longrightarrow "average"
15	"uncertain"	"partial"	"complete"	\longrightarrow "average"
16	"uncertain"	"complete"	"none"	\longrightarrow "bad"
17	"uncertain"	"complete"	"partial"	\longrightarrow "average"
18	"uncertain"	"complete"	"complete"	\longrightarrow "average"
19	"reliable"	"none"	"none"	\longrightarrow "average"
20	"reliable"	"none"	"partial"	\longrightarrow "average"
21	"reliable"	"none"	"complete"	\longrightarrow "average"
22	"reliable"	"partial"	"none"	\longrightarrow "average"
23	"reliable"	"partial"	"partial"	\longrightarrow "average"
24	"reliable"	"partial"	"complete"	\longrightarrow "good"
25	"reliable"	"complete"	"none"	\longrightarrow "average"
26	"reliable"	"complete"	"partial"	\longrightarrow "average"
27	"reliable"	"complete"	"complete"	\longrightarrow "good"
28	"very_reliable"	"none"	"none"	\longrightarrow "average"
29	"very_reliable"	"none"	"partial"	\longrightarrow "average"
30	"very_reliable"	"none"	"complete"	\longrightarrow "average"
31	"very_reliable"	"partial"	"none"	\longrightarrow "average"
32	"very_reliable"	"partial"	"partial"	\longrightarrow "good"
33	"very_reliable"	"partial"	"complete"	\longrightarrow "very_good"
34	"very_reliable"	"complete"	"none"	\longrightarrow "average"
35	"very_reliable"	"complete"	"partial"	\longrightarrow "good"
36	"very_reliable"	"complete"	"complete"	\longrightarrow "very_good"

Table A.4: Ruleset used to rate BLAST observations

Appendix B

Presentations of this Thesis

The diploma thesis was presented at different conferences, meetings, and workshops:

MPI - Breakfast Talks On two occasions, this thesis was presented in the internal group seminar of the **Molecular Ecology Group** at the MPI Bremen. The first talk focused on the representation of knowledge and the rating of observation using fuzzy logic. The second talk explained the annotation process and gave an overview of preliminary results.

International University Bremen (IUB) A 15 minutes talk was given at the Bioinformatics Summer School, organised by the IUB in July, 2004. The talk presented the overall ideas of **MicHanThi**.

Meeting at SWISS-PROT Geneva 2005 A one hour presentation was given at a meeting with members of the HMAP team in September, 2005.

German Conference on Bioinformatics 2005 (GCB) A poster describing the work of the diploma thesis was presented at the GCB in October, 2005.

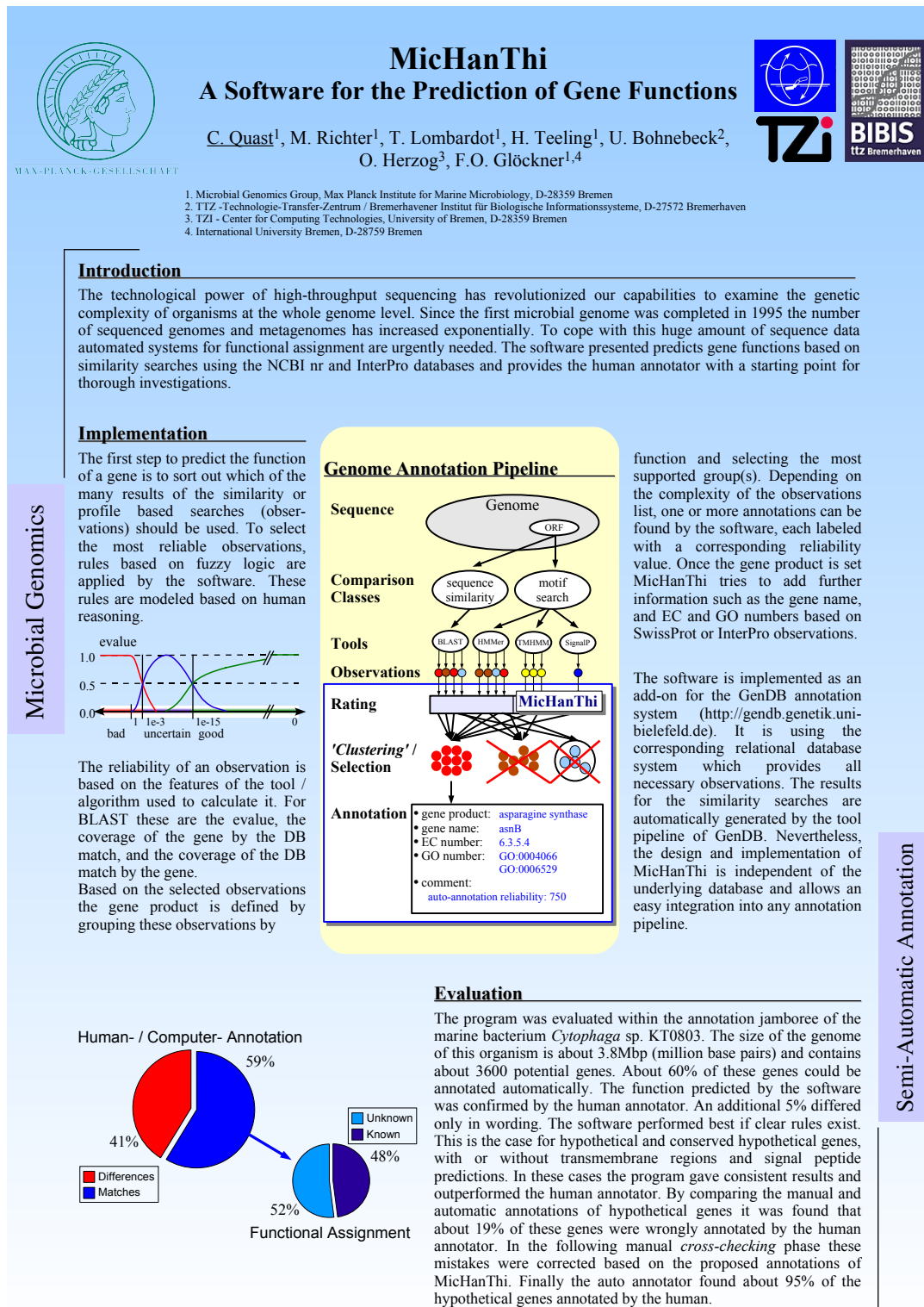


Figure B.1: Poster presented at the *German Conference on Bioinformatics 2005* (<http://www.gcb2005.de>)

Appendix C

Manual

C.1 Hardware Requirements

MicHanThi runs on any hardware platform for which an Java Runtime Environment (JRE) exists.

C.2 Software Requirements

Dependencies

=====

MicHanThi requires a recent version ($\geq 1.5.0$) of the Java Runtime Environment (JRE) and it depends on the following external libraries:

JDBC-mysql: A recent version ($\geq 3.1.7$) of the mysql databases connector if you use a MySQL project database.

<http://www.mysql.com/products/connector/j/>

JSAP: A recent version ($\geq 2.0a$) of the Java Simple Argument Parser (JSAP) used to parse options provided on the command line.

<http://www.martiansoftware.com/jsap/>

Xerces2: A recent version (2.6.0) of the Xerces XML parser used to read the configuration files used by MicHanThi.

<http://xerces.apache.org/xerces2-j/>

mbfuzzyit: The version of the mfuzzit fuzzy logic library supplied with this distribution.

Sufficient versions of these libraries are supplied in the lib directory. These libraries are used by default to build and run MicHanThi.

C.3 Installation from Source Code

Installation

=====

In order to build MicHanThi from source code you must have a recent

(>= 1.5.0) of the Java Development Kit (JDK) installed.

<http://java.sun.com/>

Also, you need an installation of the ant build tool if you want to build to software from source code.

<http://ant.apache.org/>

1. uncompress the distribution

```
% zcat michanthi-x.y.z.tar.gz | tar xvf -
```

or

```
% tar zxvf michanthi-x.y.z.tar.gz
```

or

```
% gunzip -c michanthi-x.y.z.tar.gz | tar zvf -
```

This should create a subdirectory michanthi-x.y.z.

```
% cd michanthi-x.y.z
```

2. compile MicHanThi

Read the NOTES-section found in the end of this file.

```
% ant
```

3. installation

The software is self contained and can run from the source code directory.

C.4 Usage

Usage

=====

michanthi:

Annotation of an ORF, e.g. :

```
./michanthi --project cyto_kt0803 --orf orf7
```

Re-Annotation of an orf ignoring matches to sequences of the same organism found in the databases, e.g.:

```
./michanthi --project blasto_pirellula_marina --orf orf7 --soo
```

"Rhodopirellula baltica SH 1"

Deleting Annotation created by MicHanThi, e.g.:

```
./michanthi --project cyto_kt0803 --orf dummy --del-autoannotations
```

mobh:

Mark ORFs based on best blast hit, e.g. :

```
./mobh --ds-host mgdbs --ds-user chriss --ds-passwd '*****' --project cyto_kt0803
```

```

sp-importer:
Import of a SWISS-PROT xml dump into a local MySQL server
./sp-importer --genann-host mgdbs --genann-user chriss --genann-passwd
'*****' --sp-dump swiss_prot_dump.xml

upgenec:
Update additional annotation information of an already annotated ORF.
./upgenec --ds-host mgdbs --ds-user chriss --ds-passwd '*****'
--genann-host mgdbs --genann-user chriss --genann-passwd '*****'
--project cyto_kt0803 --orf orf7

Submit 'update annotation jobs' to the SGE grid engine
./upgenec.submit --ds-host mgdbs --ds-user chriss --ds-passwd '*****'
--genann-host mgdbs --genann-user chriss --genann-passwd '*****'
--project cyto_kt0803 --orf orf7

```

C.5 Configuration

Configuration

=====

By default, MicHanThi create an application directory in the users home directory (.michanthi). This directory contains a configuration file use to setup the software as well as the default directory where log files are created. All option specified in the configuration file may be overwritten on the command line.

Configuration Options:

loglevel	how much information is printed onscreen and to the logfile NONE: No messages at all. ERROR: Error messages only. WARNING: Error and warning messages. INFO: Default output level. DEBUG: Prints anything
data_source	source of information about the genome annotation project type: Type of the database to connect to. gendb-mysql is the only supported type atm. version: 2.0 2.2 according to the version of the GenDB database. host: name or IP of the server port: port used to connect to the server user: username to use when connecting to the server passwd: password (in plain text) to use when connecting to the server
server	source of additional information such as the SWISS-PROT and InterPro entries name: unique name identifying this entry. version: not used atm. host: name or IP of the server port: port used to connect to the server user: username to use when connecting to the server passwd: password (in plain text) to use when connecting to the server
tmppath	pathname of the directory where to create temporary files. If the pathname starts with '/' then the path interpreted as absolut otherwise it is considered to be relative to .michanthi

The `upgenec.submit` tool creates its own configuration in the users home directory (`.upgenecrc`).

```
DS_HOST, DS_USER, DS_PASS, DS_TYPE, DS_VERS == MicHanThi data_source
GA_HOST, GA_USER, GA_PASS == MicHanThi server
```


Appendix D

CD-ROM

Table D.1 provides an overview of the contents and structure of the CD-ROM. All documents other than the source codes are provided as PDF documents.

Directory	Contents
data	This directory contains a dump of the ‘Gramella forsetii’ KT0803 database in the MySQL format as well as the list of all annotations created by MicHanThi and the final list of annotations created by the human annotators. These lists were used to create the statistics presented in chapter 5.
doc	Documentation about this thesis: the UML class diagram showing dependencies and interactions of the different modules, an UML class diagram explaining the <i>GenDB</i> database schema, and an UML class diagram describing the fuzzy logic library.
papers → subject → thesis	Documents that discuss this thesis. Short abstract of this thesis. This thesis.
posters → gcb2005	Posters about this work presented at conferences Presentation of this thesis at the GCB 2005.
sources → michanthi	Programs / scripts / tools developed as part of this thesis. Source code of the prototype developed as part of this thesis. Detailed information about this directory can be found in appendix C
talks	Presentations of the diploma thesis at different conferences and meetings. See appendix B for a detailed list of conferences and meetings attended.

Table D.1: Contents and structure of the CD-ROM

Bibliography

- [1] C. J. J. Francoijs, J. P. G. Klomp, and R. M. A. Knegtel, "Sequence annotation of nuclear receptor ligand-binding domains by automated homology modeling," *Protein Engineering*, vol. 13, no. 6, pp. 391–394, 2000.
- [2] M. Williams, H. Shirai, J. Shi, H. Nagendra, J. Mueller, K. Mizuguchi, R. Miguel, S. Lovell, C. Innis, C. Deane, L. Chen, N. Campillo, D. Burke, T. Blundell, and P. de Bakker, "Sequence-structure homology recognition by iterative alignment refinement and comparative modeling," *Proteins: Structure, Function, and Genetics*, vol. 45, no. S5, pp. 92–97, 2001.
- [3] D. G. Higgins and S. P. M., "CLUSTAL: a package for performing multiple sequence alignment on a microcomputer.," *Gene*, vol. 73, pp. 237–244, December 1988.
- [4] L. B. Koski, M. W. Gray, B. F. Lang, and G. Burger, "AutoFACT: An Automatic Functional Annotation and Classification Tool," *BMC Bioinformatics*, vol. 6, pp. 1–11, 2005.
- [5] R. Fleischmann, M. Adams, O. White, R. Clayton, E. Kirkness, A. Kerlavage, C. Bult, J. Tomb, B. Dougherty, and J. Merrick, "Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd.," *Science*, vol. 269, pp. 496–512, 1995.
- [6] J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine, and R. Losick, *Molecular Biology of the Gene*. Benjamin Cummings, fifth ed., 2003.
- [7] J. Messing, R. Crea, and P. H. Seeburg, "A system for shotgun DNA sequencing.," *Nucl. Acids Res.*, vol. 9, pp. 309–321, January 1981.
- [8] D. W. Mount, *Bioinformatics Sequence and Genome Analysis*. CSHL Press, second ed., 2004.
- [9] M. L. Green and P. D. Karp, "Genome annotation errors in pathway databases due to semantic ambiguity in partial EC numbers," *Nucl. Acids Res.*, vol. 33, no. 13, pp. 4035–4039, 2005.

- [10] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock, "Gene Ontology: tool for the unification of biology," *Nat Genet*, vol. 25, pp. 25–29, May 2000.
- [11] K. Hanekamp, "Entwicklung eines Systems zur phylogenetischen Analyse von Sequenzvergleichen im Rahmen von Genomannotationsprojekten," diploma thesis, University Bremen, 2005.
- [12] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443–453, March 1970.
- [13] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences.," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.
- [14] S. F. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman, "Basic Local Alignment Search Tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, Oct. 1990.
- [15] I. Korf, M. Yandell, and B. Joseph, *BLAST*. O'REILLY, first ed., 2003.
- [16] K. Weicker, *Evolution"are Alogrithmen*. Teubner, first ed., 2002.
- [17] S. Karlin and S. Altschul, "Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes," *PNAS*, vol. 87, no. 6, pp. 2264–2268, 1990.
- [18] S. R. Eddy, "Profile hidden Markov models.," *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.
- [19] Y. Fujiwara, M. Asogawa, and A. Konagaya, "Stochastic Motif Extraction Using Hidden Markov Model.," in *ISMB* (R. B. Altman, D. L. Brutlag, P. D. Karp, R. H. Lathrop, and D. B. Searls, eds.), pp. 121–129, AAAI, 1994.
- [20] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *IEEE*, vol. 77, pp. 257–286, 1989.
- [21] S. R. Eddy, "What is a hidden Markov model?," *Nature Biotechnology*, vol. 22, pp. 1315–1316, October 2004.
- [22] C. Chothia and A. M. Lesk, "The relation between the divergence of sequence and structure in proteins," *EMBO J*, vol. 5, pp. 823–826, 1986.
- [23] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The Protein Data Bank," *Nucl. Acids Res.*, vol. 28, no. 1, pp. 235–242, 2000.

- [24] M. Richter, "Bioinformatischer Genomvergleich der sulfatreduzierenden Bakterien *Desulfotalea psychrophila* und *Desulfovibrio vulgaris*," diploma thesis, University Bremen, 2004.
- [25] H. Teeling, A. Meyerdierks, M. Bauer, R. Amann, and F. O. Glöckner, "Application of tetranucleotide frequencies for the assignment of genomic fragments," *Environmental Microbiology*, vol. 6, no. 9, pp. 938–947, 2004.
- [26] M. Huynen, B. Snel, W. Lathe, and P. Bork, "Exploitation of gene context," *Curr Opin Struct Biol.*, vol. 10, pp. 366–370, June 2000.
- [27] A. Ostermann and R. Overbeek, "Missing genes in metabolic pathways: a comparative genomics approach," *Current Opinion in Chemical Biology*, vol. 7, pp. 238–251, April 2003.
- [28] K. Okubo, H. Sugawara, T. Gojobori, and Y. Tateno, "DDBJ in preparation for overview of research activities behind data submissions," *Nucl. Acids Res.*, vol. 34, pp. D6–9, 2006.
- [29] G. H. Hamm and G. N. Cameron, "The EMBL data library," *Nucl. Acids Res.*, vol. 14, pp. 5–9, January 1986.
- [30] C. Burks, J. W. Fickett, W. B. Goad, M. Kanehisa, F. I. Lewitter, R. W. P., C. D. Swindell, T. C. S., and B. H. S., "The GenBank nucleic acid sequence database.," *Comp Appl Biosci*, vol. 1, pp. 225–233, December 1985.
- [31] W. C. Barker, L. T. Hunt, D. G. George, L. S. Yeh, H. R. Chen, M. C. Blomquist, E. I. Seibel-Ross, A. Elzanowski, B. J. K., and F. D. A. et al., "Protein sequence database of the protein identification resource (PIR).," *Protein Seq Data Anal.*, vol. 1, no. 1, pp. 43–49, 1987.
- [32] K. D. Pruitt, K. S. Katz, H. Sicotte, and D. R. Maglott, "Introducing RefSeq and LocusLink: curated human genome resources at the NCBI," *Trends in Genetics*, vol. 16, pp. 44–47, January 2000.
- [33] A. Bairoch and B. Boeckmann, "The SWISS-PROT protein sequence data bank," *Nucl. Acids Res.*, vol. 19, pp. 2247–2249, 1991.
- [34] A. Bairoch and R. Apweiler, "The SWISS-PROT protein sequence data bank and its new supplement TREMBL," *Nucl. Acids Res.*, vol. 24, no. 1, pp. 21–25, 1996.
- [35] R. Apweiler, A. Bairoch, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh, "UniProt: the Universal Protein knowledgebase," *Nucl. Acids Res.*, vol. 32, no. 90001, pp. D115–119, 2004.

- [36] S. Henikoff and J. G. Henikoff, "Automated assembly of protein blocks for database searching.," *Nucl. Acids Res.*, vol. 19, no. 23, pp. 6565–6572, 1991.
- [37] E. L. Sonnhammer, S. R. Eddy, and R. Durbin, "Pfam: a comprehensive database of protein domain families based on seed alignments.," *Proteins: Structure, Function, and Genetics*, vol. 28, pp. 405–420, December 1998.
- [38] T. K. Attwood and M. E. Beck, "PRINTS - A protein motif fingerprint database.," *Protein Engineering*, vol. 7, no. 7, pp. 841–848, 1994.
- [39] A. Bairoch, "PROSITE: a dictionary of sites and patterns in proteins," *Nucl. Acids Res.*, vol. 19, pp. 2241–2245, 1991.
- [40] F. Corpet, J. Gouzy, and D. Kahn, "The ProDom database of protein domain families," *Nucl. Acids Res.*, vol. 26, no. 1, pp. 323–326, 1998.
- [41] J. Schultz, F. Milpetz, P. Bork, and C. P. Ponting, "SMART, a simple modular architecture research tool: Identification of signaling domains," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 95, no. 11, pp. 5857–5864, 1998.
- [42] D. H. Haft, B. J. Loftus, D. L. Richardson, F. Yang, J. A. Eisen, I. T. Paulsen, and O. White, "TIGRFAMs: a protein family resource for the functional identification of proteins," *Nucl. Acids Res.*, vol. 29, no. 1, pp. 41–43, 2001.
- [43] R. Apweiler, T. K. Attwood, A. Bairoch, A. Bateman, E. Birney, M. Biswas, P. Bucher, L. Cerutti, F. Corpet, M. D. R. Croning, R. Durbin, L. Falquet, W. Fleischmann, J. Gouzy, H. Hermjakob, N. Hulo, I. Jonassen, D. Kahn, A. Kanapin, Y. Karavidopoulou, R. Lopez, B. Marx, N. J. Mulder, T. M. Oinn, M. Pagni, F. Servant, C. J. A. Sigrist, and E. M. Zdobnov, "The InterPro database, an integrated documentation resource for protein families, domains and functional sites," *Nucl. Acids Res.*, vol. 29, no. 1, pp. 37–40, 2001.
- [44] E. M. Zdobnov and R. Apweiler, "InterProScan - an integration platform for the signature-recognition methods in InterPro," *Bioinformatics*, vol. 17, no. 9, pp. 847–848, 2001.
- [45] M. Scharf, R. Schneider, G. Casari, P. Bork, A. Valencia, C. Ouzounis, and C. Sander, "GeneQuiz: a workbench for sequence analysis.," *Proceeding International Conference Intelligent System Molecular Biology*, vol. 2, pp. 348–353, 1994.
- [46] T. Gaasterland and C. W. Sensen, "MAGPIE: automated genome interpretation," *Trends in Genetics*, vol. 12, pp. 78–80, 1996.

- [47] T. Gaasterland and C. W. Sensen, "Fully automated genome analysis that reflects user needs and preferences. A detailed introduction to the MAGPIE system architecture.," *Biochimie*, vol. 78, pp. 302–310, 1996.
- [48] D. Frishman, K. Albermann, J. Hani, K. Heumann, A. Metanowski, A. Zollner, and H. Mewes, "Functional and structural genomics using PEDANT.," *Bioinformatics*, vol. 17, no. 1, pp. 44–57, 2001.
- [49] H. W. Mewes, K. Albermann, K. Heumann, S. Liebl, and F. Pfeiffer, "MIPS: a database for protein sequences, homology data and yeast genome information.," *Nucl. Acids Res.*, vol. 25, no. 1, pp. 28–30, 1996.
- [50] R. Overbeek, N. Larsen, T. Walunas, M. D'Souza, G. Pusch, J. Eugene Selkov, K. Liolios, V. Joukov, D. Kaznadzey, I. Anderson, A. Bhattacharyya, H. Burd, W. Gardner, P. Hanke, V. Kapatral, N. Mikhailova, O. Vasieva, A. Osterman, V. Vonstein, M. Fonstein, N. Ivanova, and N. Kyrpides, "The ERGO genome analysis and discovery system," *Nucl. Acids Res.*, vol. 31, no. 1, pp. 164–171, 2003.
- [51] F. Meyer, A. Goesmann, A. McHardy, D. Bartels, T. Bekel, J. Clausen, J. Kalinowski, B. Linke, O. Rupp, R. Giegerich, and A. PÄEhler, "GenDB-an open source genome annotation system for prokaryote genomes," *Nucl. Acids Res.*, vol. 31, no. 8, pp. 2187–2195, 2003.
- [52] A. Goesmann, B. Linke, D. Bartels, M. Dondrup, L. Krause, H. Neuweger, S. Oehm, T. Paczian, A. Wilke, and F. Meyer, "BRIGEP-the BRIDGE-based genome-transcriptome-proteome browser," *Nucl. Acids Res.*, vol. 33, pp. 710–716, 2005.
- [53] R. Overbeek, T. Disz, and R. Stevens, "The SEED: a peer-to-peer environment for genome annotation," *Communications of the ACM*, vol. 47, no. 11, pp. 46–51, 2004.
- [54] R. Overbeek, T. Begley, R. M. Butler, J. V. Choudhuri, H.-Y. Chuang, M. Cohoon, V. de CrÄ©cy-Lagard, N. Diaz, T. Disz, R. Edwards, M. Fonstein, E. D. Frank, S. Gerdes, E. M. Glass, A. Goesmann, A. Hanson, D. Iwata-Reuyl, R. Jensen, N. Jamshidi, L. Krause, M. Kubal, N. Larsen, B. Linke, A. C. McHardy, F. Meyer, H. Neuweger, G. Olsen, R. Olson, A. Osterman, V. Portnoy, G. D. Pusch, D. A. Rodionov, C. RÄEckert, J. Steiner, R. Stevens, I. Thiele, O. Vassieva, Y. Ye, O. Zagnitko, and V. Vonstein, "The Subsystems Approach to Genome Annotation and its Use in the Project to Annotate 1000 Genomes," *Nucl. Acids Res.*, vol. 33, pp. 5691–5702, 2005.
- [55] M. A. Andrade, N. P. Brown, C. Leroy, S. Hoersch, A. de Daruvar, C. Reich, A. Franchini, J. Tamames, A. Valencia, C. Ouzounis, and C. Sander, "Auto-

- mated genome sequence analysis and annotation,” *Bioinformatics*, vol. 15, no. 5, pp. 391–412, 1999.
- [56] G. H. van Domselaar, P. Stothard, S. Shrivastava, J. A. Cruz, A. Guo, X. Dong, P. Lu, D. Szafron, R. Greiner, and D. S. Wishart, “BASys: a web server for automated bacterial genome annotation.,” *Nucl. Acids Res.*, vol. 33, pp. 455–459, 2005.
- [57] L. A. Zadeh, “Fuzzy Logic,” *IEEE*, vol. 88, pp. 83–93, 1988.

Acknowledgements

First of all, I would like to thank my whole family for supporting me and providing me with shelter. I know, times have not always been easy, but apparently I found my way . . . Many thanks to Mrs. Silva. I know I would not be writing this thesis if it wasn't for here.

Thanks to Otthein Herzog and Frank Oliver Glöckner for giving me the chance to write this thesis. In particular, I would like to thank Frank Oliver for giving me the opportunity to use the resources of his group and for buying me all this nice hardware that I could use to do all these nifty things.

I would like to thank Frank Oliver and Uta Bohnebeck for fruitful discussions about all quaint ideas I had regarding **MicHanThi**. I would to thank Renzo Kottmann for all those discussions that seemed to never end. Furthermore, I would like to thank **Michael** Richter, **Hanno** Teeling, and **Thierry** Lombardot for answering all my stupid questions about biology and helping me to tweak the membership function for the different fuzzy sets as well as helping me to fine-tune **MicHanThi**. To make up for all their troubles I decided to name the program based on their first names.

I would like to thank all the participants of the '*Gramella forsetii*' *KT0803 Annotation Jamboree* who provided me with a huge amount of data that I could use to evaluate and enhance **MicHanThi**. Especially, I have to thank Margarete Bauer because she had the luck of annotating the first 250 ORFs of the organism which happened to be my test set. I think I questioned her about half the annotation she made and why she did what she did.

I would like to thank all the members of the **Microbial Genomics Group** and **Molecular Ecology Group** not particularly mentioned here for just being around.

I would like to thank Kristian Hahnekamp, Thomas Soller and the other members of the Genann group of the GEN!E Project who provided me with a solid basis for this thesis. Special thanks to Kristian for having the initial thought of grouping observations.

Finally, many many many thanks to Rachel Montague for all the work she spent in proof-reading this thesis.

Selbstständigkeitserklärung

gemäß §11 Abs. 7 DPO 1993

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmitteln benutzt habe.

Bremen, October 4, 2007

Christian Quast