Fachhochschule Bingen
Fachbereich 1 - Life Sciences and Engineering
Dr. Heinrich Wippermann

Webbasierte bijektive Datenkonvertierung der *Seefeld-Konvention/PHI-BLAST-Nomenklatur* zur Betrachtung von Protein-Protein-Wechselwirkungen

Studienarbeit

von

Thomas Schmidt
Matrikel-Nummer: 190571

Richard-Wagner-Straße 2
68165 Mannheim

Aufgabensteller:

Prof. Dr. rer. nat. Daniel Hoffmann
Zentrum für medizinische Biotechnologie
Universität Duisburg-Essen

Datum: 23. März 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcode-Verzeichnis

Aufgabenstellung

Ursprüngliche Aufgabenstellung

Entwickeln Sie ein Perl-Skript zum Übersetzen regulärer Ausdrücke: Ausdrücke in der sogenannten „*Seefeld convention*“ (Aasland et al. 2002 FEBS Lett 513:141-144[1]) sollen übersetzt werden in reguläre Ausdrücke, wie sie von *PHI-BLAST* verwendet werden und umgekehrt, d.h. von *PHI-BLAST* nach *Seefeld*.

Beachten Sie, dass nicht alle Elemente der *Seefeld-Konvention* in *PHI-BLAST* ausgedrückt werden können! Geben Sie in solchen Fällen eine geeignete Fehlermeldung aus.

Der Nutzer sollte die Wahl haben, ob er die *ASCII-Version* der *Seefeld-Konvention* nutzt, oder ob er die Symbole ein- oder ausgeben will, wie sie in LaTeXzur Verfügung stehen (z.B. `\Phi` oder `\Omega`)[2].

Geänderte Aufgabenstellung

Im Rahmen dieser Studienarbeit wurde die ursprüngliche Aufgabenstellung geändert.

Recherchen ergaben, dass die *Seefeld-Konvention* in mehreren wissenschaftlichen Publikationen genutzt wird, teilweise mit eigenen Ergänzungen der jeweiligen Autoren.

[1] Literatur: [1] (*Normalization of nomenclature for peptide motifs as ligands of modular protein domains*)

[2] Original-Aufgabenstellung von Prof. Dr. D. Hoffmann

Die angestrebte Implementierung der oben genannten Aufgabenstellung könnte somit als frei zugängliche Bioinformatik-Anwendung von Nutzen sein.

Ziel dieser Arbeit ist eine webbasierte, (hochverfügbare) Anwendung zur Konvertierung von regulären Ausdrücken nach der Seefeld-Konvention in die PROSITE-Nomenklatur und umgekehrt.

Kapitel 1

Grundlagen

1.1 Biologische Grundlagen

1.1.1 Aminosäuren

Aminosäuren sind kleine, organische Moleküle, die aus mindestens einer *Carboxylgruppe* (*-COOH*) und einer *Aminogruppe* (*-NH_2*) gebunden an ein zentrales Kohlenstoffatom bestehen. Zusätzlich zu diesen beiden Gruppen ist am zentralen Kohlenstoffatom ein variabler Rest *R* assoziiert, welcher für jede Aminosäure spezifisch ist (siehe Abbildung 1.1).

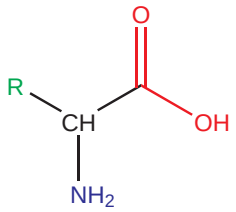


Abbildung 1.1: Die Abbildung zeigt die allgemeine Strukturformel von (α-)Aminosäuren. Rot: *Carboxylgruppe* (*-COOH*), blau: *Aminogruppe* (*-NH_2*), grün: variabler Rest.

Im einfachsten Fall – der Aminosäure *Glycin* – ist der variable Rest ein Wasserstoffatom *H*. Bei allen anderen Aminosäuren ist er ein von Wasserstoff unterschiedliches

Atom, was bedeutet, dass vier verschiedene Substituenten am zentralen Kohlenstoff assoziiert sind. Dies ist der Grund weshalb das zentrale Kohlenstoffatom bei den meisten Aminosäuren ein chirales Zentrum darstellt, also die Ebene linear polarisierten Lichts um einen bestimmten Winkel dreht, entweder nach links oder nach rechts.

Bekannt sind mittlerweile mehr als 270 Aminosäuren, die eine biologische Bedeutung aufweisen. Im Rahmen dieser Studienarbeit wird jedoch nur auf die sogenannten proteinogenen Aminosäuren eingegangen, von denen bisher 23 bekannt sind (siehe Abbildung 1.2). Proteinogene Aminosäuren sind die Bausteine der Peptide und Proteine von Lebewesen.

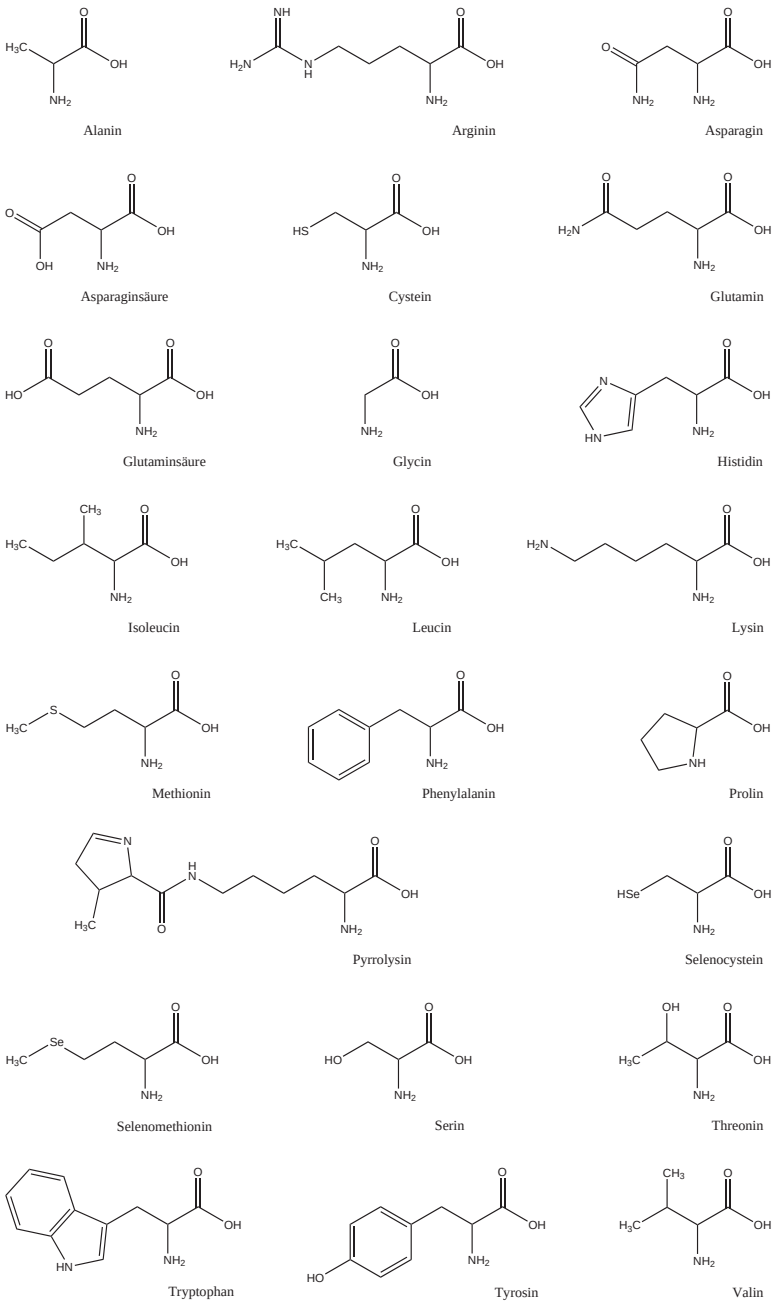


Abbildung 1.2: Die Abbildung zeigt die Strukturformeln der 23 proteinogenen Aminosäuren.

1.1.2 Peptide und Proteine

Peptide sind organische Verbindungen mehrerer, über die sog. *Peptidbindung*, miteinander verknüpfter Aminosäuren. Entscheidend dabei ist die Reihenfolge der Aminosäuren, die sog. *Aminosäuresequenz*.

Die Funktionen von *Peptiden* sind vielfältig. Oftmals wirken sie als Hormon, entzündungshemmend oder -fördernd, sowie antibiotisch oder antiviral.

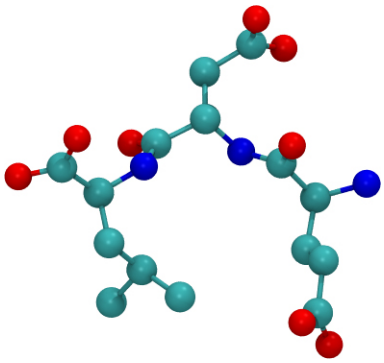


Abbildung 1.3: Die Abbildung zeigt ein Beispiel für ein Peptid, bestehend aus drei Aminosäuren (*Tripeptid*): Leucin (Leu), Asparaginsäure (Asp) und Glutaminsäure (Glu). Das Modell stammt aus der *PDB-Datei* `1A30` und wurde mit *VMD* visualisiert.

Peptide lassen sich – je nach ihrer Länge klassifizieren. So spricht man bei einer Sequenzlänge von zwei, bzw. drei Aminosäuren von einem *Di-*, bzw. *Tripeptid* (siehe Abbildung 1.3). Bis zu zehn Aminosäuren klassifiziert man als *Oligopeptid*. Ab einer Sequenzlänge von ca. 100 Aminosäuren spricht man von einem *Protein*.

Proteine gehören zu den Grundbausteinen aller Zellen. Ihre Funktionen sind vielfältig. Beispielsweise der Transport und die Umformung von Stoffen, sowie die Erkennung von Signalstoffen wird in lebenden Zellen von *Proteinen* abgewickelt.

1.1.3 Protein-Ligand-Wechselwirkungen

Protein-Ligand-Wechselwirkungen beschreiben die Interaktion zwischen einem Protein und einem, an das Protein bindenden Stoff (Ligand). Protein-Liganden können Atome, Moleküle oder Ionen sein.

In dieser Arbeit sind lediglich Peptide, also Moleküle bestehend aus Aminosäuren, von Bedeutung. Diese binden an bestimmter Stelle an das Protein und lösen eine Reaktion aus, oder blockieren lediglich die Bindungsstelle, sodass andere Peptide nicht mehr an das Protein binden können.

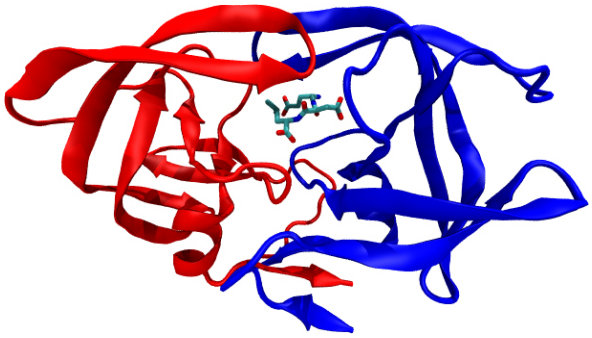


Abbildung 1.4: Die Abbildung zeigt die Bindung des *Peptids* aus Abbildung 1.3 an eine *HIV-Protease*, einem Protein, dass die Reproduktion von *HIV* in lebenden Zellen ermöglicht. Im *Protein* sind α-*Helix*- und β-*Faltblattstrukturen* zu erkennen. Die beiden Ketten des *Proteins* sind zur besseren Veranschaulichung farblich separiert (rot und blau). Zwischen den *Proteinketten*, in der *Bindungstasche*, ist der *Peptid-Ligand* gebunden. Das Modell stammt aus der *PDB-Datei* `1A30` und wurde mit *VMD* visualisiert.

Für eine Bindung an ein *Protein* muss ein *peptidischer Ligand* eine bestimmte Aminosäuresequenz haben, die seine Eigenschaften definiert, wie zum Beispiel Konformation oder Ladung. Aufgrund dieser Eigenschaften kann das *Peptid* spezifisch an bestimmte Stellen im *Protein* binden. Ersetzt man jedoch eine der Aminosäuren in der Kette ist die Wahrscheinlichkeit sehr hoch, dass der *Ligand* seine Eigenschaften zur spezifischen Bindung an ein bestimmtes *Protein* verliert.

1.2 PHI-BLAST/PROSITE-Nomenklatur

Reguläre Ausdrücke zur *PHI-BLAST-Suche* entsprechen der *PROSITE-Nomenklatur*.

1.2.1 Sprachelemente

Aminosäuren

In der *PHI-BLAST-Nomenklatur* sind alle nach *IUPAC/IUBMB* festgelegten Aminosäurebezeichnungen ([3]) erlaubt, in der *Einbuchstaben-Codierung* (siehe nachfolgende Tabelle).

1-Buchstabencode	*3-Buchstabencode*	*Bezeichnung (Aminosäure)*
A	Ala	*Alanin*
B	Asx	*Aspartatsäure oder Asparagin*
C	Cys	*Cystein*
D	Asp	*Aspartat*
E	Glu	*Glutamat*
F	Phe	*Phenylalanin*
G	Gly	*Glycin*
H	His	*Histidin*
I	Ile	*Isoleucin*
K	Lys	*Lysin*
L	Leu	*Leucin*
M	Met	*Methionin*
N	Asn	*Asparagin*
P	Pro	*Prolin*
Q	Gln	*Glutamin*
R	Arg	*Arginin*
S	Ser	*Serin*
Fortsetzung: siehe nächste Seite		

1-Buchstabencode	*3-Buchstabencode*	*Bezeichnung (Aminosäure)*
`T`	`Thr`	*Threonin*
`V`	`Val`	*Valin*
`W`	`Trp`	*Tryptophan*
`X`	`Xaa`	*variable Aminosäure*
`Y`	`Tyr`	*Tyrosin*
`Z`	`Glx`	*Glutaminsäure oder Glutamin* (oder verwandte Substanzen)
`U`	`Sec`	*Selenocystein*

Tabelle 1.1: Codierung von Aminosäuren nach *IUPAC/IUBMB*.

META-Zeichen

Die angegebene Syntax stammt aus der *ExPASy PROSITE Manual* ([2]).

`[...]` bedeutet, an dieser Stelle der Sequenz steht eines der zwischen den Klammern definierten Zeichen (Aminosäuren);
Beispiel: `[LFYT]` bedeutet entweder `L`, `F`, `Y` oder `T`.

`...` bedeutet, an dieser Stelle der Sequenz steht eines der zwischen den Klammern definierten Zeichen (Aminosäuren);
Beispiel: `LFYT` bedeutet weder `L`, `F`, `Y` oder `T` an der aktuellen Position.

`-` wird als Trennzeichen genutzt;
Beispiel: `K-A-G-[LFYT]-W` bedeutet Aminosäure `K`, dann `A`, dann `G`, dann entweder `L`, `F`, `Y` oder `T`, dann `W`.

`x` steht für eine beliebige Aminosäure;
Beispiel: `K-A-x-[LFYT]-W` bedeutet Aminosäure `K`, dann `A`, dann eine beliebige Aminosäure, dann entweder `L`, `F`, `Y` oder `T`, dann `W`.

`x(`n`)` heißt, dass an den nächsten n Stellen, beliebige Aminosäuren stehen;
Beispiel: `K-A-x(4)-[LFYT]-W` bedeutet: Aminosäure `K`, dann `A`, dann vier beliebige Aminosäuren, dann entweder `L`, `F`, `Y` oder `T`, dann `W`.

`x(`m`,`n`)` heißt, dass an den nächsten m bis n Stellen, beliebige Aminosäuren stehen (Es gilt: $m < n$);
Beispiel: `K-A-x(2,4)-[LFYT]-W` bedeutet Aminosäure `K`, dann `A`, dann entweder zwei, drei oder vier beliebige Aminosäuren, dann entweder `L`, `F`, `Y` oder `T`, dann `W`.

Beispiel

```
1 [LIVMF]-G-E-x-[GAS]-[LIVM]-x(5,11)-R-[STAQ]-A-x-[LIVMA]-x-[STACV]
```

Erklärung

- Eine der fünf angegebenen Aminosäuren (entweder `L`, `I`, `V`, `M` oder `F`)
- gefolgt von `G`
- gefolgt von `E`
- gefolgt von einer beliebigen Aminosäure
- gefolgt von entweder `G`, `A` oder `S`
- gefolgt von entweder `L`, `I`, `V` oder `M`
- gefolgt von einer variablen Sequenz bestehend aus fünf bis elf Aminosäuren
- gefolgt von `R`
- gefolgt entweder `S`, `T`, `A` oder `Q`
- gefolgt `A`
- gefolgt von einer beliebigen Aminosäure

- gefolgt von entweder L, I, V, M oder A
- gefolgt von einer beliebigen Aminosäure
- gefolgt entweder S, T, A, C oder V

1.3 Seefeld-Konvention

Die *Seefeld-Konvention* ist eine Nomenklatur zur Beschreibung von Wechselwirkungen zwischen kurzen, funktionellen *Protein-Domänen* und *Peptid-Liganden* ([1]). Dabei werden speziell die Aminosäureabschnitte betrachtet, welche die Bindungsoberfläche, des Proteins zum Ligand bilden.

1.3.1 Sprachelemente

Die folgende Tabelle zeigt alle Sprachelement der *Seefeld-Konvention.*

Regelung	*Betreffende Aminosäure*	*Symbol*	*ASCII*
Unbekannte/variable Aminosäure		x	x
Aminosäure-Kette am N-Terminus des Ligand-Kerns		fn	fn
Aminosäure-Kette am C-Terminus des Ligand-Kerns		fc	fc
Hydrophobe Aminosäure	V, I, L, F, W, Y, M	Φ	%
Aromatische Aminosäure	F, W, Y	Ω	@
Hydrophile Aminosäure	N, Q, S, T (*ungeladen*)	ζ	&
positive Ladung	K, R, H (*positiv geladen*)	[+]	[+]
negative Ladung	E, D (*negativ geladen*)	[-]	[-]
Aminosäuren mit längerem aliphatischem Rest	V, I, L, M	Ψ	#
Fortsetzung: siehe nächste Seite			

Regelung	*Betreffende Aminosäure*	*Symbol*	*ASCII*
Kurzkettige Aminosäuren	`P`, `G`, `A`, `S`	π	$\sim$
Phosphorylierte Aminosäuren (Tyr)		*po*Y	`[Y:po]`
Sulfatisierte Aminosäuren (Tyr)		*su*Y	`[Y:su]`
O-glycosilierte Aminosäuren (Ser)		*gl*S	`[S:gl]`
N-glykosilierte Aminosäuren (Asp)		*gl*N	`[N:gl]`

Methylierte Aminosäuren (Arg)
symmetrisch methyliert
asymmetrisch methyliert
		*me*R
		*sme*R
		*ame*R
			`[R:me]`
			`[R:sme]`
			`[R:ame]`

Acetylierte Aminosäuren (Lys)		*ac*K	`[K:ac]`
Hydroxyliertes Prolin		*hy*P	`[P:hy]`
Ausgeschlossene Aminosäuren (Pro)		^P	`(^P)`
Aminosäuren des Liganden, die vollständig in die Oberfläche des gebundenen Proteins eingelagert sind		*po*Y	`{Y:po}`
ϵ-*Determinante*		β4-5 α2-2	`(beta4)5` `(alpha2)2`

Tabelle 1.2: Sprachelemente der *Seefeld-Konvention* (*Quelle: Aasland et al. 2002*).

Mehrere verschiedene Aminosäuren, die an einer bestimmten Position in Frage kommen, werden über die Syntax `(Q/A/P)` definiert, also im gewählten Beispiel entweder der Aminosäure Glutamin, Alanin oder Prolin. Ebenso können auch nicht erlaubte Aminosäuren für eine Position definiert werden. Beispielsweise `^(Q/A/P)` meint, dass Glutamin, Alanin und Prolin an der angegebenen Position verboten sind.

Die wichtigste Aminosäure des Peptid-Liganden bekommt die Stellung „0“ (*Null*), alle folgenden werden aufsteigend nummeriert. Falls Aminosäure „0“ nicht am aminoterminalen Ende der Sequenz sitzt, sondern mittig, werden ab dieser Position die Aminosäuren in carboxyterminaler Richtung aufsteigend nummeriert, und die Aminosäuren in aminoterminaler Richtung absteigend, also mit einem vorangeschriebenen „−“ (*Minus*), aufgezählt. Gibt es mehrere, (gleich-)bedeutsame Aminosäuren, wird diejenige, welche näher am aminoterminalen Ende sitzt, mit Null nummeriert.

Ist der betrachtete, bindende Peptidabschnitt von anderen Aminosäuren umgeben, wird dies durch fc (*flanking carboxy-terminus*), bzw. fn (*flanking amino-terminus*) dargestellt.

Die ϵ-*Determinante* steht für die Aminosäuren, die bestimmen, welcher Ligand bevorzugt gebunden werden soll.

1.3.2 Modifizierte Seefeld-Konvention

In einigen Publikationen wurde eine Modifikation der *Seefeld-Konvention* vorgenommen ([5]). So werden konservierte Aminosäuren durch Großbuchstaben dargestellt, während variable Bereiche in Kleinbuchstaben geschrieben werden. Durch dieses System wird jedoch die Nutzung einiger Spracheigenschaften der *Seefeld-Nomenklatur* eingeschränkt, da nicht mehr eindeutig unterschieden werden kann, zwischen Aminosäuren mit angehängter Gruppe (z.B. bei einer Phosphorylierung) und Bereich mit variablen (wenig konservierten) Aminosäuren.

Das *Seefeld-Muster* „`AQrhyPT`“ verdeutlicht das Problem der Zweideutigkeit. So bleibt unklar, ob es sich bei dem Ausdruck „*hyP*“ um ein hydroxyliertes Prolin (*hy*P), oder um die Sequenz „*His-Tyr-Pro*“ (als variabler Sequenzbereich) handelt. Aufgrund dieser Zweideutigkeit wird von der ursprünglichen *Seefeld-Konvention* und nicht von einer modifizierten Variante ausgegangen.

1.4 Kompatibilität

Da die *Seefeld-Nomenklatur* speziell auf *Protein/Ligand-Wechselwirkungen* ausgelegt ist, enthält sie zahlreiche Optionen, welche die *PHI-BLAST-Nomenklatur*, die für alle Arten von Proteinen gedacht ist, nicht bietet. So können beim Übersetzen (von *Seefeld* nach *PHI-BLAST/PROSITE*) Informationen verloren gehen, wie z.B. die *Phosphorylierung* einer Aminosäure.

Die folgende Tabelle soll die Kompatibilität, und gegebenenfalls den Informationsverlust beim Übersetzen, verdeutlichen.

Eigenschaft	*PHI-BLAST → Seefeld*		*Seefeld → PHI-BLAST*	
	kann übersetzt werden	Beispiel	kann übersetzt werden	Beispiel
unbekannte/variable Aminosäuren	ja	X → x x → x	ja	x → x
unbekannte/variable Aminosäuren mit definierter Sequenzlänge	ja	x(3) → xxx	ja	xxx → x(3)
unbekannte/variable Aminosäuren mit variabler Sequenzlänge	Informationsverlust	x(2,4) → xx x(2,4) → xxx x(2,4) → xxxx	ja	xx → x(2) xxx → x(3) xxxx → x(4)
bekannte Aminosäuresequenz	ja	Q-A-R → QAR	ja	QAR → Q-A-R
Aminosäure 1 oder Aminosäure 2 ...	ja	[QAR] → (Q/A/R)	ja	(Q/A/R) → [QAR]
Aminosäureklasse	ja	[APGS] → π	Änderung der Reihenfolge	π → [AGPS]
End-/Mittelständigkeit der Domäne im Protein	nicht möglich		Informationsverlust	fnQARfc → Q-A-R
Fortsetzung: siehe nächste Seite				

Eigenschaft	*PHI-BLAST → Seefeld*		*Seefeld → PHI-BLAST*	
	kann übersetzt werden	Beispiel	kann übersetzt werden	Beispiel
funktionelle Gruppen	nicht möglich		Informationsverlust	QA*hy*PR → Q-A-P-R
Ausschließen von Aminosäuren („alle Aminosäuren außer ...“)	ja	{NPQY} → ^(N/P/Q/Y)	ja	^(N/P/Q/Y) → {NPQY}
Einlagerung von Ligand-Bereichen in gebundenes Protein	nicht möglich		Informationsverlust	*ac*K → K
ϵ-Determinante	nicht möglich		nicht möglich	

Tabelle 1.3: Kompatibilität der Sprachelemente zwischen der *PHI-BLAST-Nomenklatur* und der *Seefeld-Konvention*.

Bei der Übersetzung liegt – neben dem häufig auftretenden Informationsverlust – bei variablen Aminosäuren mit variabler Sequenzlänge, eine relativ große Fehlerwahrscheinlichkeit vor. So beschreibt der *PHI-BLAST-Ausdruck* „x(2,4)“ nicht wieviele variable Aminosäuren tatsächlich in der Eingabesequenz vorliegen. Der Vollständigkeit wegen müsste man sich als Ergebnis alle möglichen Sequenzen (hier: drei verschiedene: „xx“, „xxx“, „xxxx“) ausgeben lassen. Vermutlich wurde bei der Entwicklung der *Seefeld-Nomenklatur* die Syntax für Sequenzabschnitte mit variabler Länge absichtlich ausgelassen, da diese bei Bindungsdomänen von Peptid-Liganden wenig Sinn machen. Aus biochemischer Sicht würde ein variierend langer Bereich die Bindungseigenschaften des Protein-Liganden zu stark verändern, sodass von vornherein ein relativ spezifisches Sequenz-Muster gegeben sein muss. Da die Seefeld-Konvention speziell für *Protein-Ligand-Bindungsstellen* entwickelt wurde, ist es sinnvoll, mit dem zu schreibenden Konverter nur Bindungsdomänen in die *PHI-BLAST-Nomenklatur* übersetzen zu lassen.

Kapitel 2

SeePHI

2.1 Anforderungen

Ziel dieser Arbeit ist die Entwicklung und Implementierung einer webbasierten Anwendung für die wissenschaftliche Nutzung. Dies setzt eine effiziente Umsetzung des Programms voraus, sowie eine hohe Benutzerfreundlichkeit und Toleranz gegenüber Fehleingaben. Ebenso wichtig ist eine hohe Verfügbarkeit des Webservers, welcher die Anwendung bereitstellt.

2.2 Vorüberlegungen

2.2.1 Programmiersprache

Da die Aufgabenstellung eine webbasierte Anwendung fordert, wird zur Implementierung *PHP* gewählt. *PHP* (*PHP Hypertext Preprocessor*) ist eine Scriptsprache die speziell dazu entwickelt wurde, um zum Anfragezeitpunkt generierte Inhalte im Web darstellen zu können. Durch diese Spezialisierung auf den Bereich der Webanwendungen, sowie zahlreichen Funktionen, welche fester Sprachbestandteil (kompiliert) sind, überzeugt *PHP* als zu verwendende Scriptsprache. Durch die Tatsache, dass *PHP*

als Scriptsprache von einem Interpreter ausgeführt wird, ist eine Portierung auf andere Betriebssystem- und Rechnerarchitekturen möglich. Dort muss lediglich *PHP* – idealerweise in der gleichen Version – installiert sein.

Auch die in der Aufgabenstellung beschriebenen *regulären Ausdrücke* werden von *PHP* unterstützt. Durch zahlreiche Erfahrungen mit *Perl* werden in dieser Arbeit *reguläre Ausdrücke* nach der *Perl-Notation* verwendet. Ebenfalls verfügbar sind *reguläre Ausdrücke* im *POSIX-Stil*, welche jedoch keine Vorteile in Bezug auf Leistungsfähigkeit und Geschwindigkeit bieten.

2.2.2 Programmiertechnik

Die Einfachheit von *PHP* verleitet zu unsauberer Programmierung, was die spätere Wartung, eventuelle Ergänzungen und die Fehlerlokalisation/-behebung erschwert. Um diesen Problemen vorzubeugen wird ein objektorientierter Ansatz angestrebt.

2.2.3 Server

Um eine hohe Verfügbarkeit zu gewährleisten wird Speicherplatz (*Webspace*) auf einem privaten Server genutzt. Auf dem Webserver muss *PHP5* und eine *MySQL-Datenbank* für diverse angedachte Programmfunktionen verfügbar sein. Die implementierte Anwendung soll später unter einer *Subdomain* erreichbar sein.

2.2.4 Vorgehensweise und Umsetzung

Zunächst wird ein Programmentwurf durchgeführt, welcher die verschiedenen Funktionen des Programms in Worte gefasst enthält. Diese Programmfunktionen werden anschließend in Form von Abläufen dargestellt. Später wird überlegt, wie die Anforderungen an das Programm möglichst optimal umgesetzt werden können. Bei Klarheit über den Ablauf werden die Einzelaufgaben in modulare Pakete aufgeteilt. Ein

grafisches Benutzerinterface wird – mit Hinblick auf seine Umsetzung in *HTML* – entworfen.

Im Anschluss daran beginnt die Implementierung, bei der zunächst eine sinnvolle Verzeichnisstruktur der Webanwendung erarbeitet wird. Die erdachten modularen Arbeitsschritte werden für den objektorientierten Ansatz zu Klassen abstrahiert. Die implementierten Programmelemente werden in einem *HTML-Grundgerüst* eingebettet und auf den Server kopiert. In einem ersten Test wird das Programm auf Fehler überprüft, welche direkt behoben werden. Schließlich wird das *HTML-Grundgerüst* durch die gewünschte *HTML-Benutzeroberfläche* ersetzt.

Nach der Implementierung prüft man die korrekte Ausführung und Darstellung der Anwendung anhand von verschiedenen, relevanten Anwendungsbeispielen und korrigiert eventuell auftretende Fehler (*alpha*-Tests).

2.3 Entwurf

2.3.1 Spezifikation der Programmfunktionen

- Benutzer sollen die Möglichkeit haben per *HTML-Formular* ein beliebiges Muster einer Aminosäuresequenz nach der *Seefeld-Konvention* oder der *PHI-BLAST-Nomenklatur* einzugeben, welches in die jeweils andere Regelung übersetzt werden soll.
- Ist eine eindeutige Übersetzung nicht möglich, soll der Benutzer den kritischen Abschnitt über eine Zwischenabfrage manuell definieren können.
- Benutzer sollen mit resultierenden *PHI-BLAST-Ausdrücken* eine *PHI-BLAST-Suche* am *NCBI* durchführen können.
- Benutzer sollen die Möglichkeit einer Online-Hilfe haben.

- Benutzer sollen die Möglichkeit haben sich registrieren und einloggen zu können, zur Nutzung von weitergehenden Programmfunktionen.
- Die Registrierungsprozedur soll in zwei Schritten erfolgen, und den Benutzer per E-Mail authentifizieren, um Missbrauch zu verhindern.
- Registrierte Benutzer sollen über eine erweiterte Programmausgabe zwischen verschiedenen Formaten wählen können und diese als Dateien herunterladen.
- Die Benutzeroberfläche (*HTML*) soll in englischer Sprache sein.
- Die eigentlichen Übersetzungsroutinen sollen in ein speziell entwickeltes *PHP-Basisprogramm* eingebettet werden.

2.3.2 Ablaufdiagramme der Programmfunktionen

Nachfolgend werden die Ablaufdiagramme des *PHP-Basisprogramms*, sowie der Übersetzungsfunktionen dargestellt. In den Diagrammen beschreiben die farbig unterlegten Flächen bestimmte Kategorien im Programmablauf:

gelb Benutzereingaben

grün Programmausgaben

grau programminterne Verarbeitungsschritte

rot Verzweigungen

blau Übersetzungsschritt

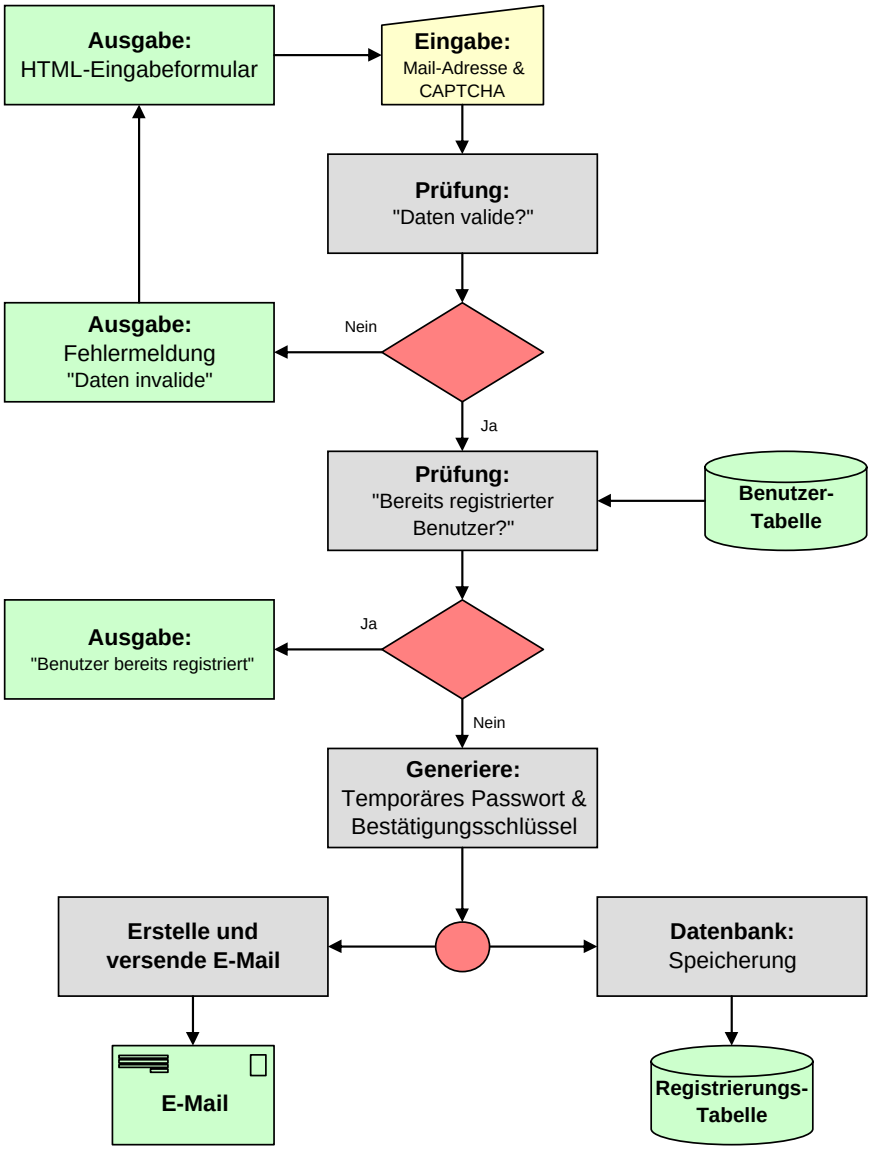


Abbildung 2.1: **Registrierungsschritt 1:** Die Abbildung zeigt das Ablaufdiagramm des Registrierungsvorgangs (*Initialisierungsschritt*).

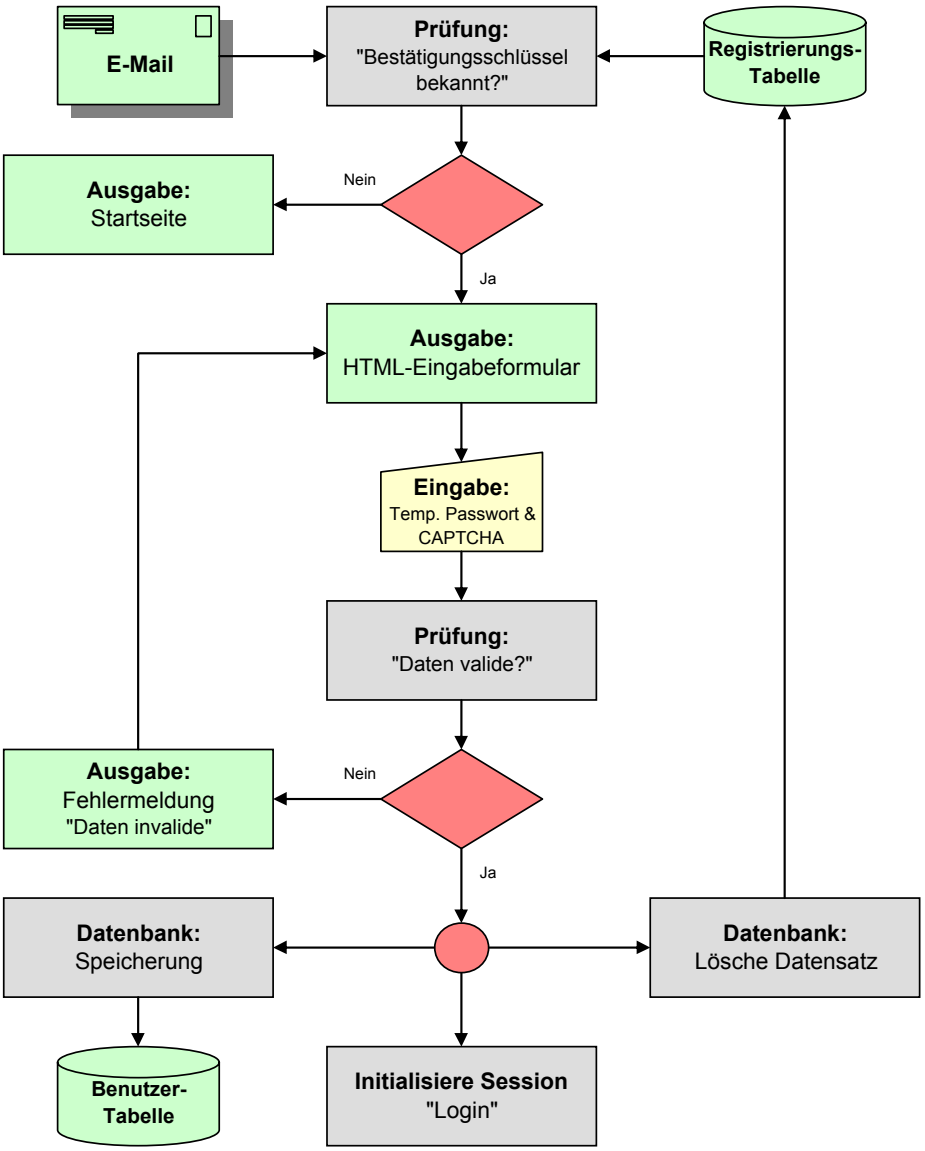


Abbildung 2.2: **Registrierungsschritt 2:** Die Abbildung zeigt das Ablaufdiagramm des Registrierungsvorgangs (*Bestätigungsschritt*).

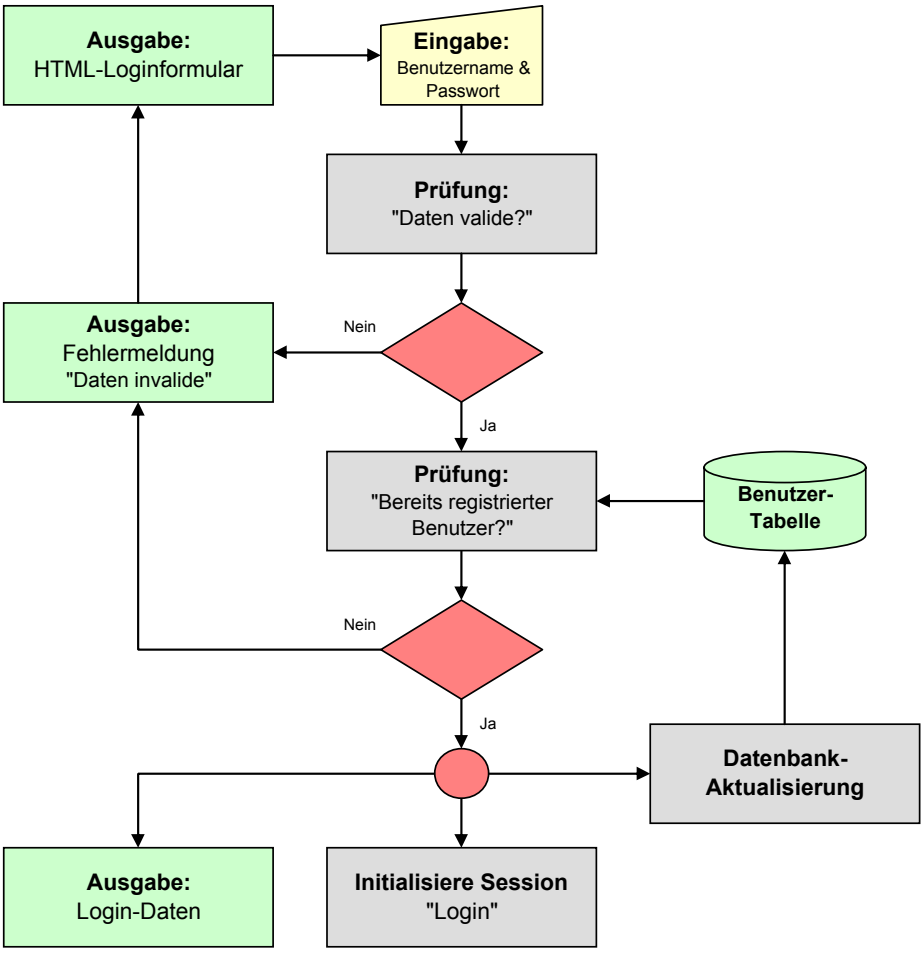


Abbildung 2.3: Die Abbildung zeigt das Ablaufdiagramm des *Login-Vorgangs*.

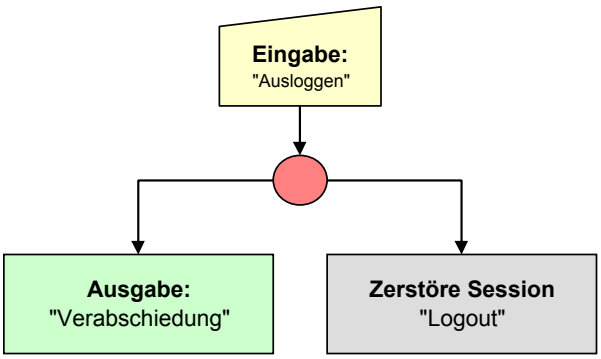


Abbildung 2.4: Die Abbildung zeigt das Ablaufdiagramm des *Logout-Vorgangs*.

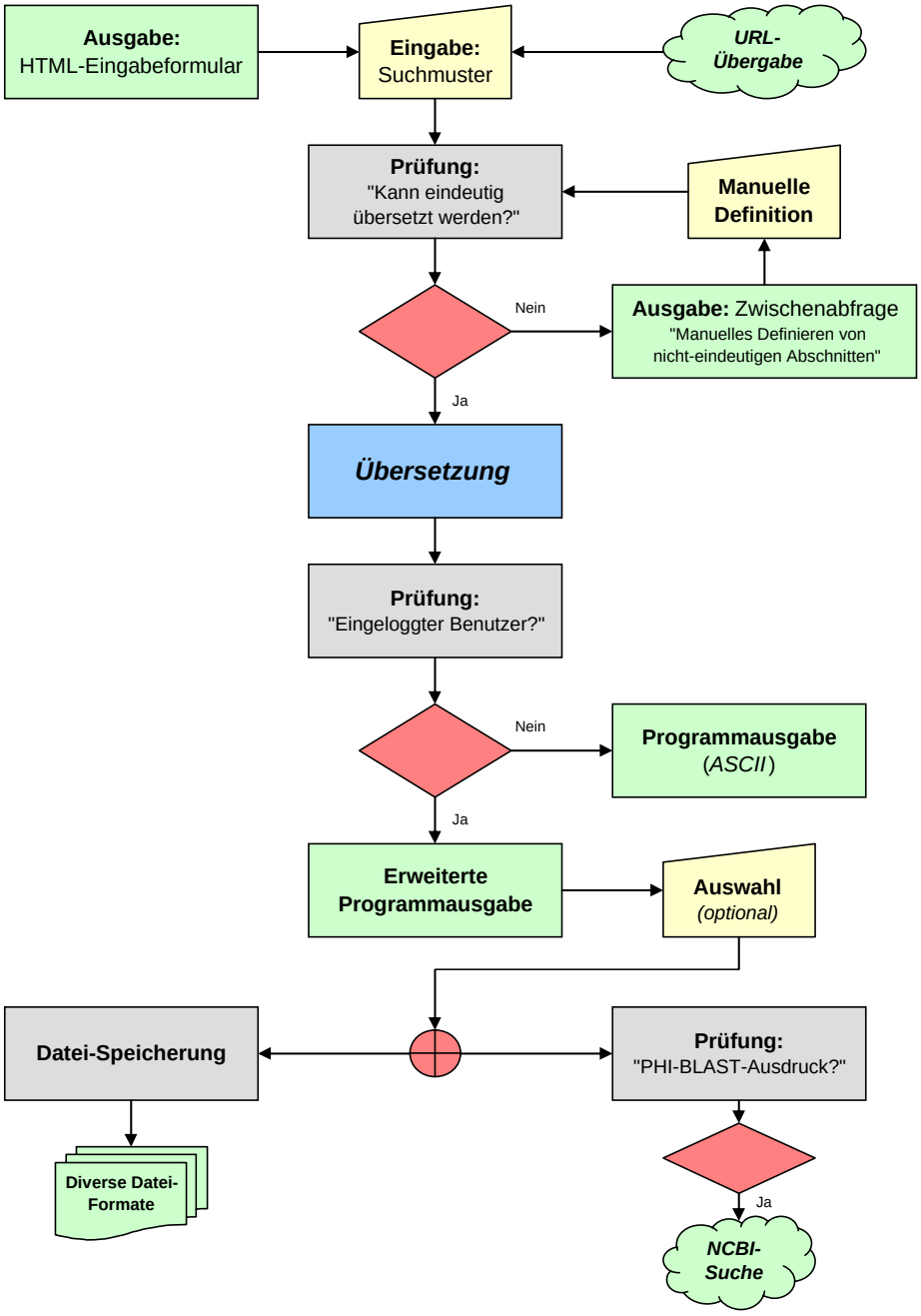


Abbildung 2.5: Die Abbildung zeigt das Ablaufdiagramm der Übersetzungsfunktion.

2.3.3 Datenbankentwurf

Die nachfolgenden Punkte erklären weshalb die Integration einer *MySQL-Datenbank* in die Basis-Anwendung sinnvoll ist:

Registrierung Zur Automatisierung der Registrierungsprozedur werden E-Mail-Adresse, Passwort und Bestätigungsschlüssel in einer Datenbanktabelle zwischengespeichert.

Benutzerverwaltung Die Zugangsdaten aller registrierten Benutzer werden in einer separaten Tabelle gespeichert. Dabei wird das Benutzerpasswort irreversibel nach dem *MD5-Algorithmus*[1] verschlüsselt und so gespeichert.

Metadaten Die *HTML-Ausgabe* enthält in ihrem *HTML-Head* zahlreiche *META-Angaben*, die aus einer Datenbanktabelle gelesen werden sollen.

Statistik Für Statistikzwecke wird jeder vom Server ausgeführte Schritt aller Benutzer als *Clicklog* gespeichert.

[1] *Message-Digest Algorithm 5*; Kryptographische *Hash-Funktion* zur Erzeugung von 128 Bit langen *MD5-Hashes*; Normale Notation als 32-stellige Hexadezimalzahl.

Entity-Relationship-Diagramm

Basierend auf den Anforderungen an die Datenbank wird ein *Entity-Relationship-Diagramm* (*ER-Diagramm*) entwickelt. Die letztliche Umsetzung in *SQL-Code* wird in Anhang C (Punkt C.2) beschrieben.

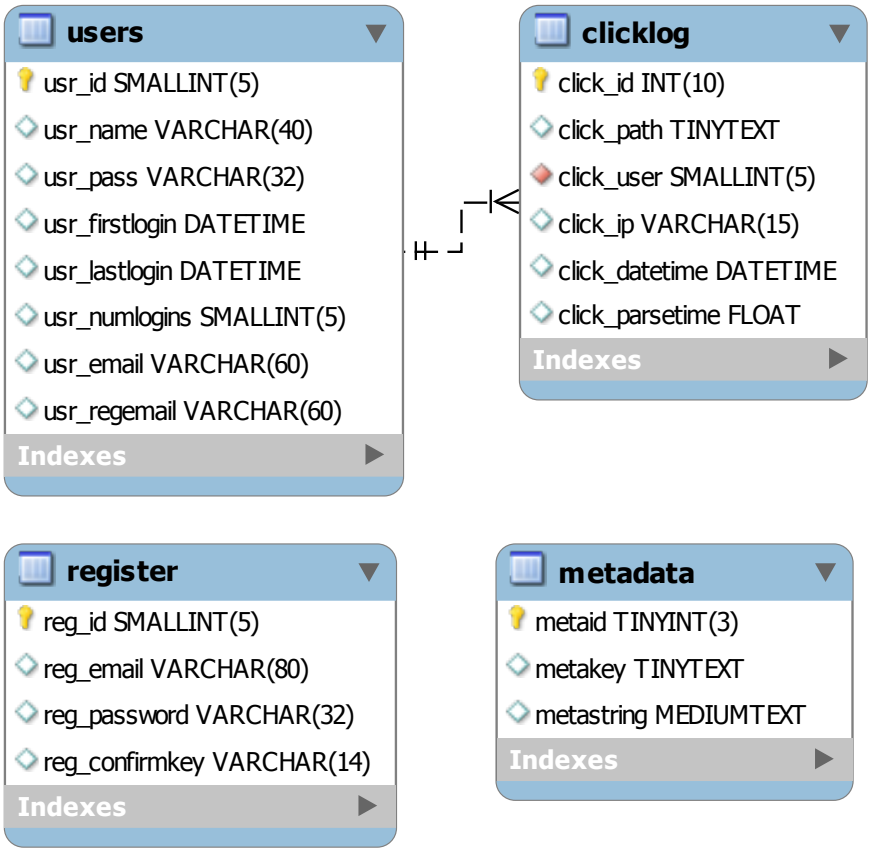


Abbildung 2.6: Die Abbildung zeigt das *Entity-Relationship-Diagramm* zum Datenbankentwurf. Zu sehen sind fünf Datenbanktabellen mit angegebenen Spaltennamen und deren zugehörigen Datentypen. Erstellt mit *MySQL-Workbench 5.0.29 OSS*.

2.3.4 Schema der HTML-Benutzeroberfläche

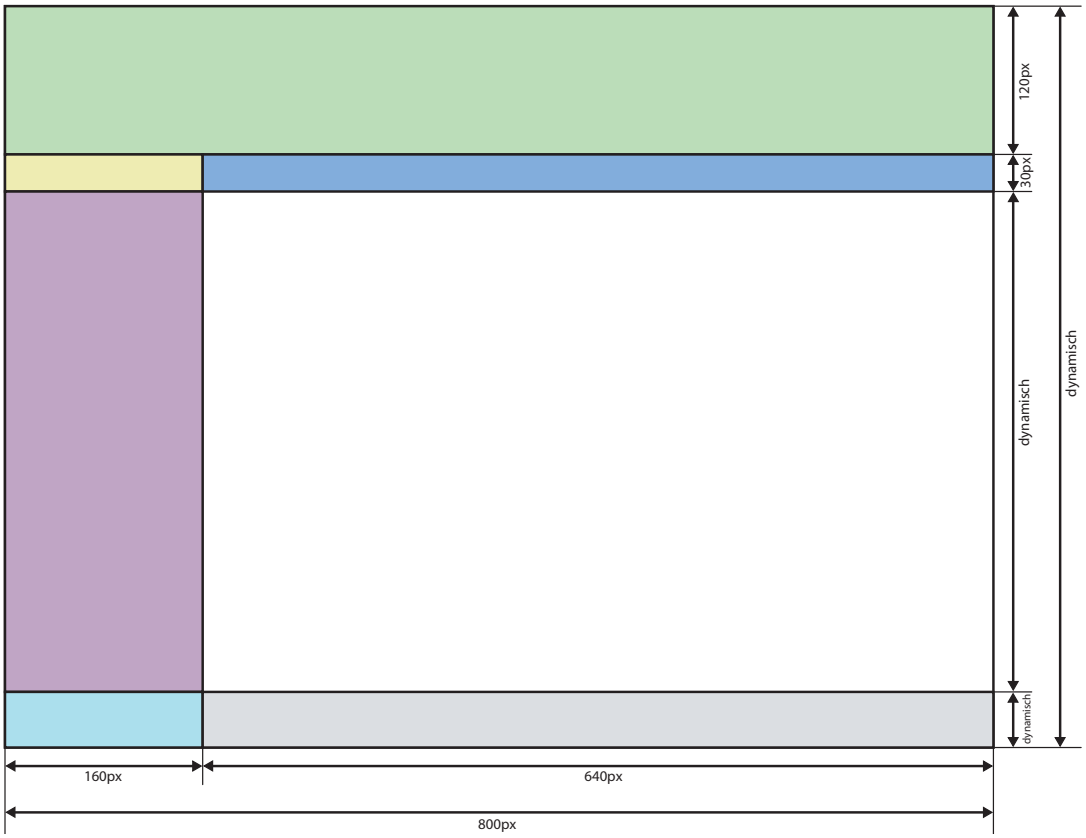


Abbildung 2.7: Die Abbildung zeigt schematisch die *HTML-Benutzeroberfläche*. Die farbig unterschiedlichen Flächen markieren bestimmte Elemente der Website, welche später mit Inhalt oder Bildern, bzw. Hintergrundbildern gefüllt werden.

2.4 Implementierung

2.4.1 Spezifikation zur Implementierung

- Der Quellcode soll weitestgehend von der *HTML-Ausgabe* getrennt sein.
- Die Programmierung erfolgt weitestgehend in objektorientiertem *PHP*.
- Nach Möglichkeit werden die Standard-Funktionen von *PHP* genutzt.

2.4.2 Zeitplan

Die Implementierung der Anwendung erfolgt gemäß den folgenden Punkten:

1. Implementierung des Web-Grundgerüsts (*Sitemap*, Verzeichnisstruktur, Klassen)
2. Implementierung der Übersetzungsklassen
3. Einbindung der *PHI-BLAST-Schnittstelle* des *NCBI*

2.4.3 Web-Grundgerüst

Zur Implementierung der Webanwendung wird ein Grundgerüst entwickelt, dass alle Basisfunktionen der Website bereitstellt. Dazu gehören:

- Einheitliche Benutzeroberfläche
- Dynamische Navigation
- Benutzer-Registrierung
- *Login/Logout-Handling*

In dieses autonom funktionierende Grundgerüst wird später die Übersetzungsfunktion eingebettet.

Sitemap

Ausgehend vom Web-Basisverzeichnis (`./`) werden nachfolgend die bereitzustellenden Webseiten genannt.

- ▷ **`./index.php`** *Startseite*
  Titel: „*Home*“
- ▷ **`./help/index.php`** *Hilfeseite*
  Titel: „*Help*“
- ▷ **`./register.php`** Schnittstelle zur Registrierungsfunktion
  Titel: „*Registration*“
- ▷ **`./translate/index.php`** Schnittstelle zur Übersetzungsfunktion
  Titel: „*Translate*“

Verzeichnisstruktur

Nachfolgend sind die Verzeichnisse mit ihren jeweiligen Funktionen, ausgehend vom Web-Basisverzeichnis, aufgelistet.

```
/
├── classes ..... PHP-Klassen
├── general ..... Allgemeine Dateien (Kontakt/Impressum)
├── help ..... Programmhilfe
├── images ..... Bilder
├── include ..... Einzubindende PHP-Scripts
├── translate ..... Übersetzungsfunktionalität
│   └── odata ..... Dateiausgabeverzeichnis
│       └── <userid> ..... Benutzerverzeichnis
└── styles ..... Dateien zur Darstellung von SeePHI
    └── default ..... SeePHI-Standard-Layout
```

Klassen

Nachfolgend sind die entwickelten Klassen des *HTML-Grundgerüsts* dargestellt. Auf die Übersetzungsklassen wird unter Punkt 2.4.4 eingegangen.

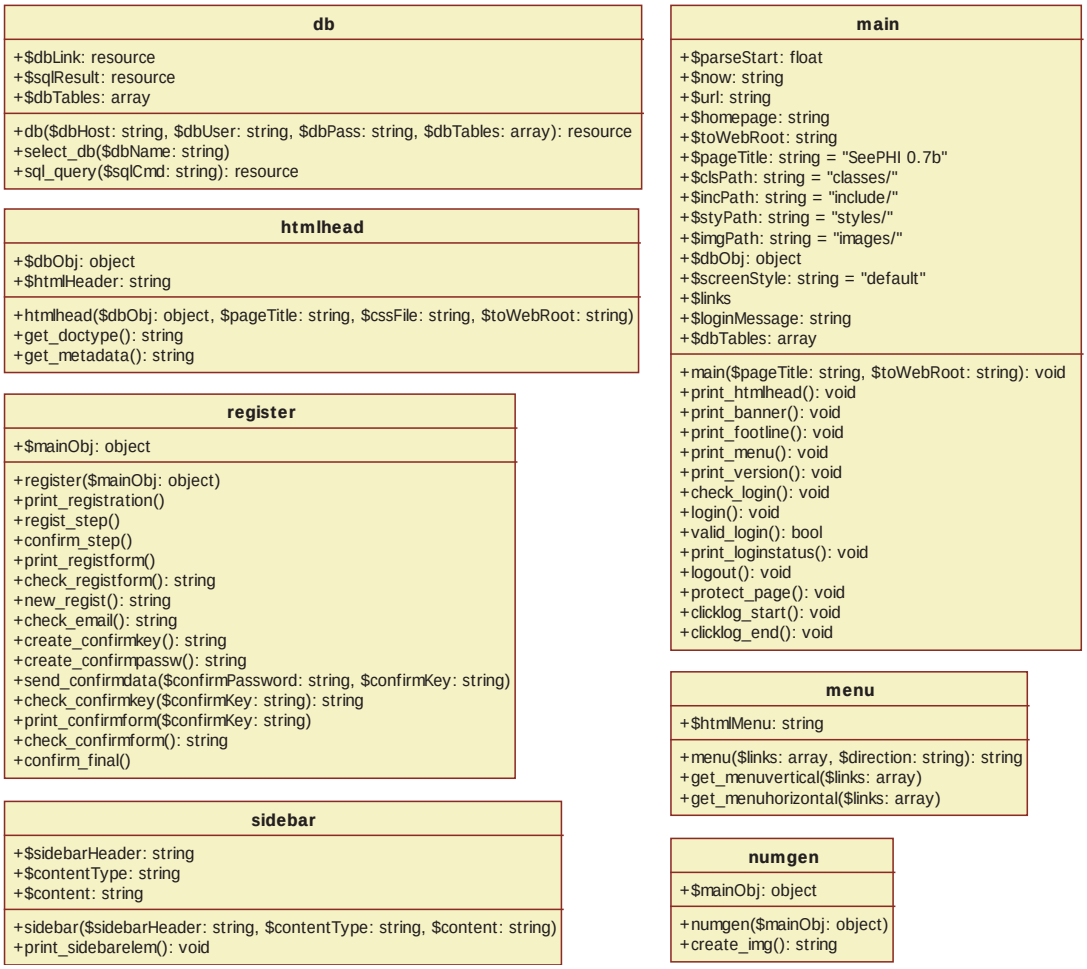


Abbildung 2.8: Die Abbildung zeigt die *UML-Diagramme* der entwickelten Klassen des *Web-Grundgerüsts*. Erstellt mit *StarUML 5.0.2.1570*.

Basis-Benutzeroberfläche

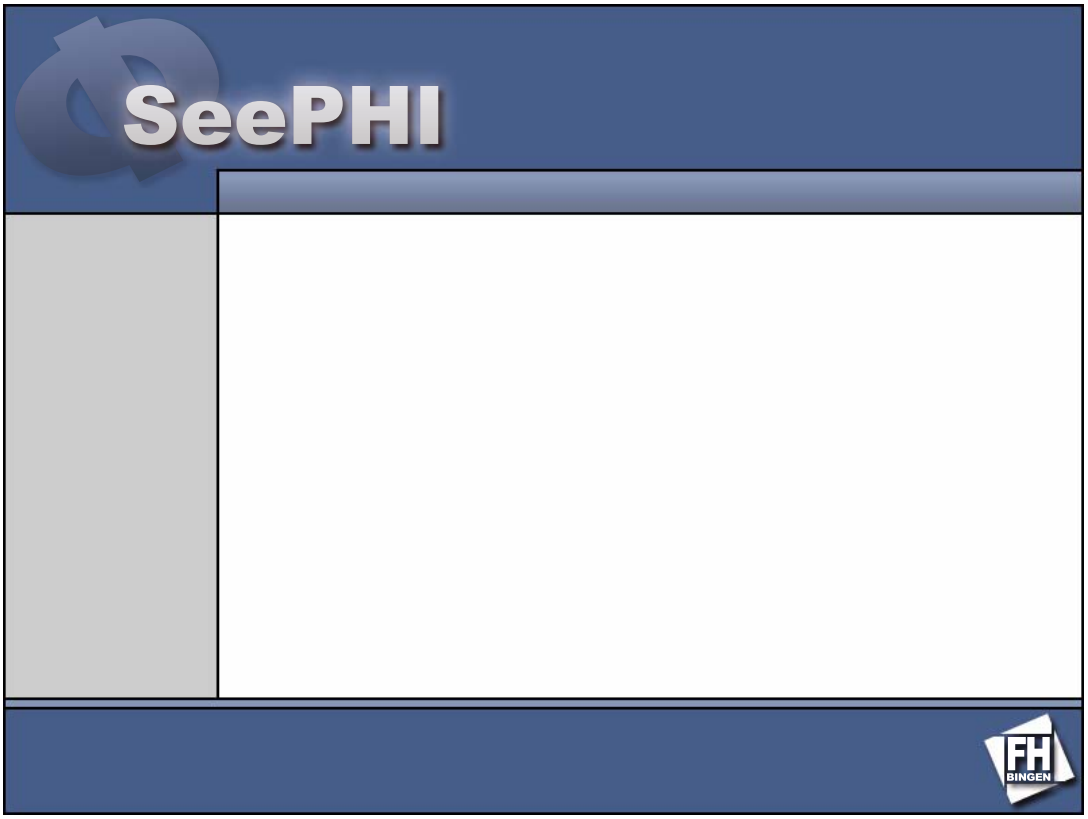


Abbildung 2.9: Basis-Benutzeroberfläche der Website. An diesem Design orientieren sich alle anzuzeigenden Seiten.

2.4.4 Übersetzung

Die für die Übersetzung relevanten Eigenschaften beider Notationen werden gemäß Tabelle 1.3 (*Kompatibilität*) implementiert. Aufgrund des objektorientierten Ansatzes kann die Übersetzungsroutine problemlos in das bestehende Web-Grundgerüst integriert werden.
Zur Übersetzung wird eine eigene Klasse entwickelt, welche dynamisch, je nach Übersetzungsweg, das entsprechende Übersetzungsmodul (*PROSITE zu Seefeld* (`p2s`) oder *Seefeld zu PROSITE* (`s2p`)) lädt und darin existierende Funktionen ausführt.

Übersetzungsklasse

Die hier gezeigte Klasse wird in die Programmstruktur des *HTML-Grundgerüsts* eingebettet.

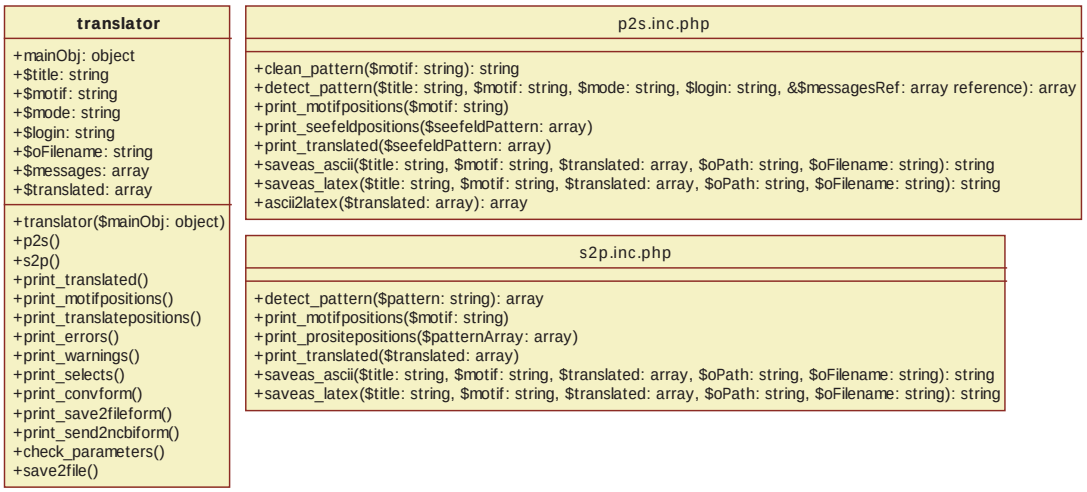


Abbildung 2.10: Die Abbildung zeigt die *UML-Diagramme* der entwickelten Übersetzungsklasse. Aus der Hauptklasse „*translator*“ (`translator.class.php`) wird dynamisch eines der beiden Übersetzungsmodule (**p2s.inc.php** oder **s2p.inc.php**) geladen und dessen Funktionen ausgeführt. Erstellt mit *StarUML 5.0.2.1570*.

2.4.5 PHI-BLAST-Schnittstelle

Die Anwendung soll – im Falle der Übersetzung in einen *PHI-BLAST-Aus-druck* – an das *PHI-BLAST-Suchformular* am *NCBI* weitergeleitet werden. Dort kann der Benutzer – neben dem vorgegebenen, übersetzten Ausdruck – eine Suchsequenz eingeben nach der die *BLAST-Suche* erfolgen soll.
Das *NCBI* bietet für automatisierte *BLAST-Anfragen* eine *URL-API* über die folgende Seite an:

```
1 http://www.ncbi.nlm.nih.gov/blast/Blast.cgi
```

Zum erreichen des teilweise ausgefüllten Sucheingabeformulars am *NCBI* müssen folgende Parameter per *URL* übergeben werden:

`CMD=Web` Der durchzuführende Befehl soll die *NCBI-Webschnittstelle* zurückgeben.

`PAGE=Proteins` Typ der Website (trifft zu im Falle von `CMD=Web`).

`PHI_PATTERN=<phi_pattern>` Der *PHI-BLAST-Ausdruck*.

`PROGRAM=blastp` Gibt das auszuführende *BLAST-Suchprogramm* an (*blastp* = *Protein BLAST*).

`BLAST_PROGRAMS=phiBlast` Ermöglicht die Vorauswahl von *PHI-BLAST-spezifischen* Attributen, sowie die Eintragung von Werten in das Formular (nicht in der offiziellen Dokumentation der *NCBI-URL-API* enthalten).

Es ergibt sich die folgende *URL*:

```
1 http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Web&PAGE=Proteins&\
2 PHI_PATTERN=A-Z-H-P-x(2,4)-G&PROGRAM=blastp&BLAST_PROGRAMS=phiBlast
```

2.4.6 Programmausgabe

Abbildung 2.11 zeigt die Programmausgabe von *SeePHI* für das *PROSITE-Muster* „A-Z-H-P-x(2,4)“. Zu sehen ist das Eingabemuster („*Detected motif*“), sowie die Übersetzung an jeder Position („*Detected Seefeld pattern*“) der Sequenz mit dem Ergebnismuster nach der *Seefeld-Konvention* („*Result Seefeld pattern*“).

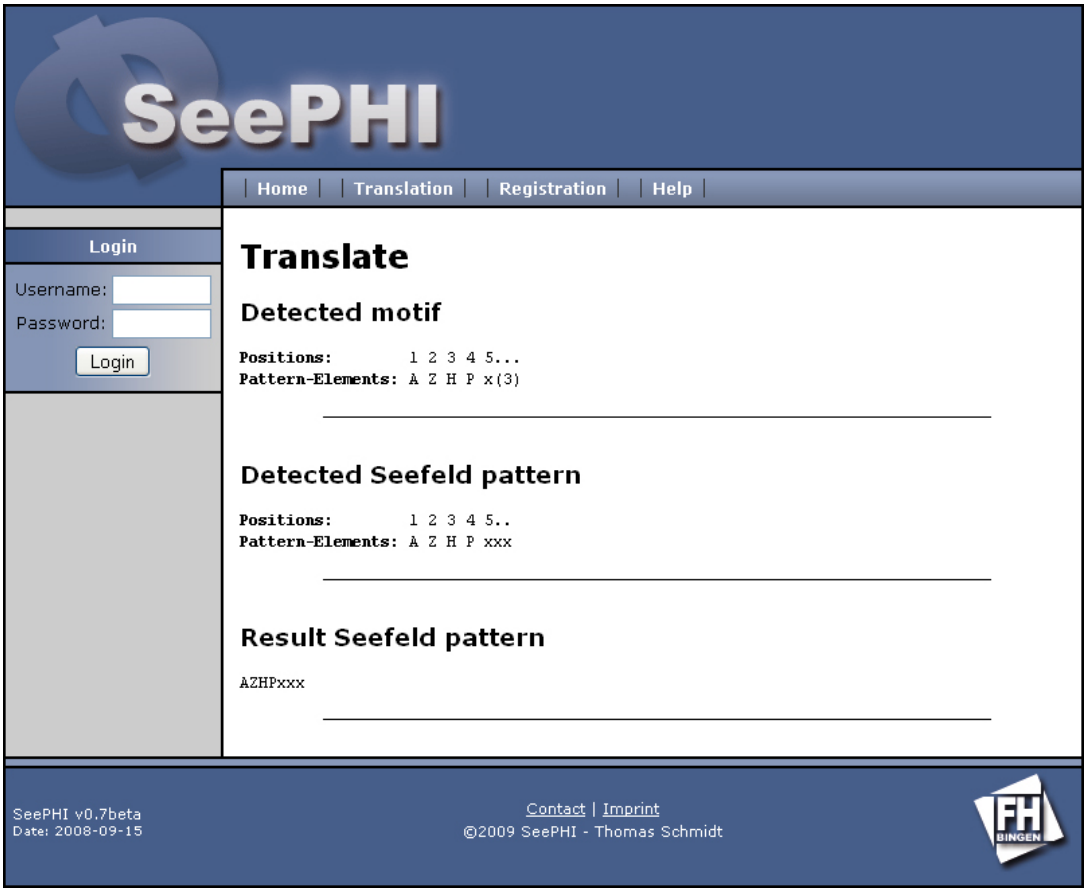


Abbildung 2.11: Die Abbildung zeigt die Programmausgabe von *SeePHI*.

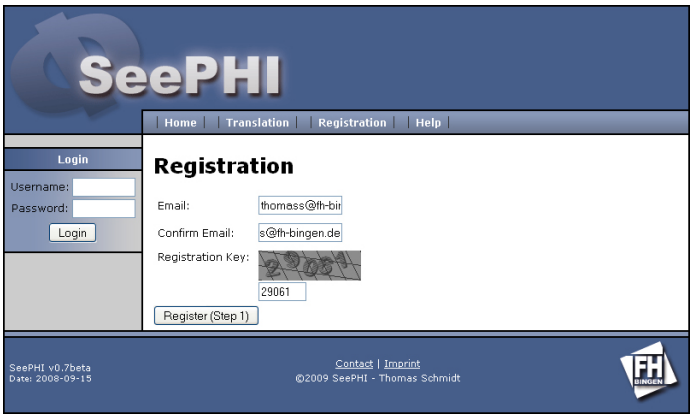
Kapitel 3

Anwendungsbeispiele

Die Anwendungsbeispiele zeigen die verschiedenen Funktionen der Webanwendung in der Praxis. Zudem werden anhand von verschiedenen Sequenzmustern die Übersetzungsfunktionen getestet.

3.1 Registrierung

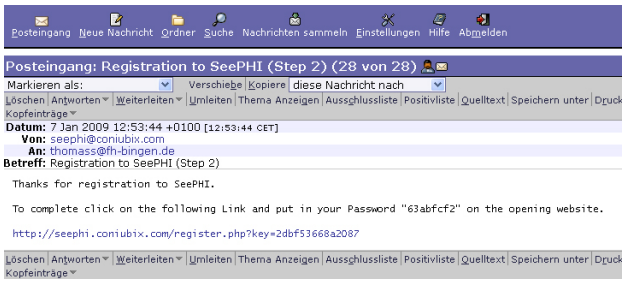
Als Beispiel soll ein Benutzer mit der E-Mail-Adresse „*thomass@fh-bingen.de*“ angelegt werden. Hierzu wird zunächst zweimal die E-Mail-Adresse in das Registrierungsformular eingegeben (siehe Abbildung 3.1a). Bevor das Formular abgeschickt wird, muss zusätzlich die Zeichenfolge aus dem *CAPTCHA* korrekt angegeben werden.
Ist dies geschehen, erhält der soeben registrierte Benutzer eine E-Mail von der Adresse „*seephi@coniubix.com*“ mit einem Link und einem temporären Passwort (siehe 3.1c). Nach Klick auf den Link muss in einem zweiten Formular das temporäre Passwort zur Bestätigung eingegeben werden, sowie zweimal das vom Benutzer gewünschte Passwort (Abbildung 3.1d). Nach dem Abschicken des Formulars wird dieses irreversibel verschlüsselt in der integrierten Datenbank gespeichert. Der neu registrierte Benutzer ist direkt eingeloggt.



(a) *Schritt 1*

(b) *Schritt 2*

Abbildung 3.1: Die Abbildungen zeigen Screenshots der ersten beiden Schritte des Registrierungsvorgangs der Webanwendung.



(c) *Schritt 3*

(d) *Schritt 4*

Abbildung 3.1: Die Abbildungen zeigen Screenshots der Schritte 3 und 4 des Registrierungsvorgangs.

(e) *Schritt 5*

(f) *Schritt 6*

Abbildung 3.1: Die Abbildungen zeigen Screenshots der letzten beiden Schritte des Registrierungsvorgangs.

3.2 Login/Logout

Bereits registrierte Benutzer können sich bei *SeePHI* ein- und ausloggen, um weiterführende Programmfunktionen nutzen zu können.
Eingeloggte Benutzer, die sich ausloggen möchten müssen hierzu lediglich auf die Schaltfläche „*Logout*“ im Menü klicken. Abbildung 3.2 zeigt die Verabschiedung auf der *SeePHI-Website* nach erfolgtem Logout.
Für das Einloggen muss im Login-Formular die E-Mail-Adresse mit dem passenden Passwort des Benutzers eingegeben werden (siehe Abbildung 3.3a). Abbildung 3.3b zeigt den Begrüßungstext auf der *SeePHI-Website*.

Abbildung 3.2: Die Abbildung zeigt als Screenshot den Logout-Vorgang der Webanwendung.

(a) *Login Schritt 1*

(b) *Login Schritt 2*

Abbildung 3.3: Die Abbildungen zeigen Screenshots des Login-Vorgangs der Webanwendung.

3.3 Seefeld zu PROSITE/PHI-BLAST

Die dargestellten Beispiele entsprechen denen der Literaturstelle zur Definition der *Seefeld-Konvention*. Getestet werden die Sequenzen `fnAG[Y:po](G/A/P/S)HFfc`, `fnAG[Y:po]~HFfc`, `fnPALPPA[L:me]` und `(beta2)6=@`.
Die angegebenen Zeitmessungen, die der *PHP-Interpreter* zum Verarbeiten der Seite benötigt, stammen aus der *Clicklog-Tabelle* der Datenbank.

3.3.1 Anwendungsbeispiel 1

Analog zu diesem Beispiel sollen sieben weitere Anwendungen, mit anderen Testsequenzen, die Funktionsweise des Programms veranschaulichen. Die nachfolgenden Beispiele werden weniger ausführlich erklärt als dieses, sondern beschränken sich lediglich auf die Programmein- und Ausgabe mit Screenshots und *PHP-Interpretationszeiten*.

Seefeld-Muster	`fnAG[Y:po](G/A/P/S)HFfc`
Übersetzt	`A-G-Y-[GAPS]-H-F`

1a. Anonyme Benutzer

Nicht-registrierte Benutzer können *SeePHI* zur Übersetzung von *Peptid-Ligand-Sequenzmustern* nutzen.
Die benötigte Zeit zur Interpretation der Seite ist in Tabelle 3.1 aufgeführt. Die Abbildungen 3.4a und 3.4b zeigen die Programmein- und Ausgabe.

Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.008603 Sekunden

Tabelle 3.1: Interpretationszeit (Übersetzung)

1b. Eingeloggte Benutzer

Registrierte Benutzer können nach dem Einloggen weiterführende Programmfunktionen, zum Beispiel das Speichern der übersetzten Sequenz in verschiedenen Formaten, ausführen. Dies erfolgt in zwei Schritten (siehe Tabelle 3.2, sowie Abbildungen 3.5).
Im gezeigten Beispiel möchte der Benutzer sowohl eine ASCII-, als auch eine LaTeX-Datei der Übersetzungsergebnisse erzeugen (siehe Abbildung 3.5b). Zudem möchte

er, die erhaltene *PROSITE-Sequenz* an die *NCBI-PHI-BLAST-Suche* schicken (Abbildungen 3.5c und 3.5d)..

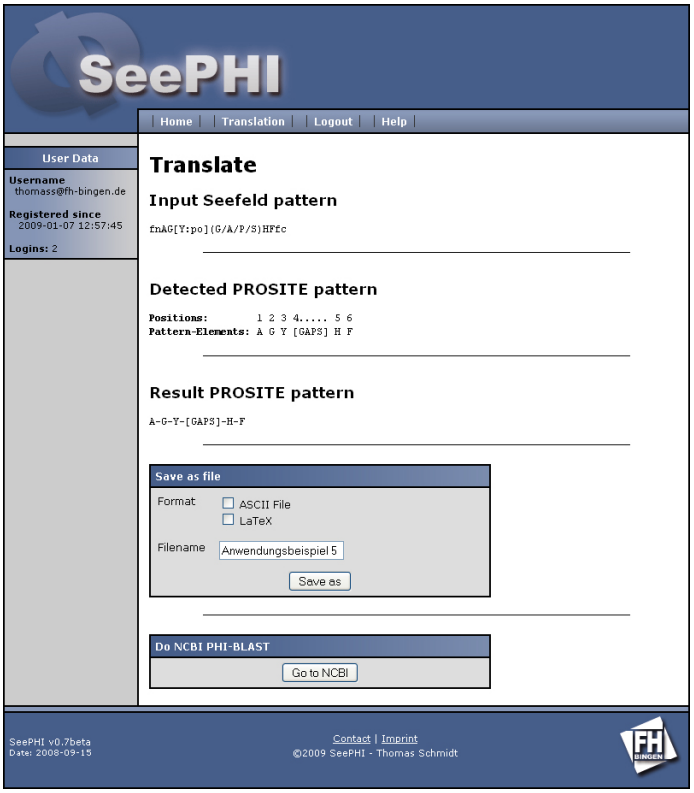
Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.009572 Sekunden
Schritt 2: Dateiexport	0.02493 Sekunden

Tabelle 3.2: Interpretationszeiten (Übersetzung & Dateiexport)

(a) *Programmeingabe*

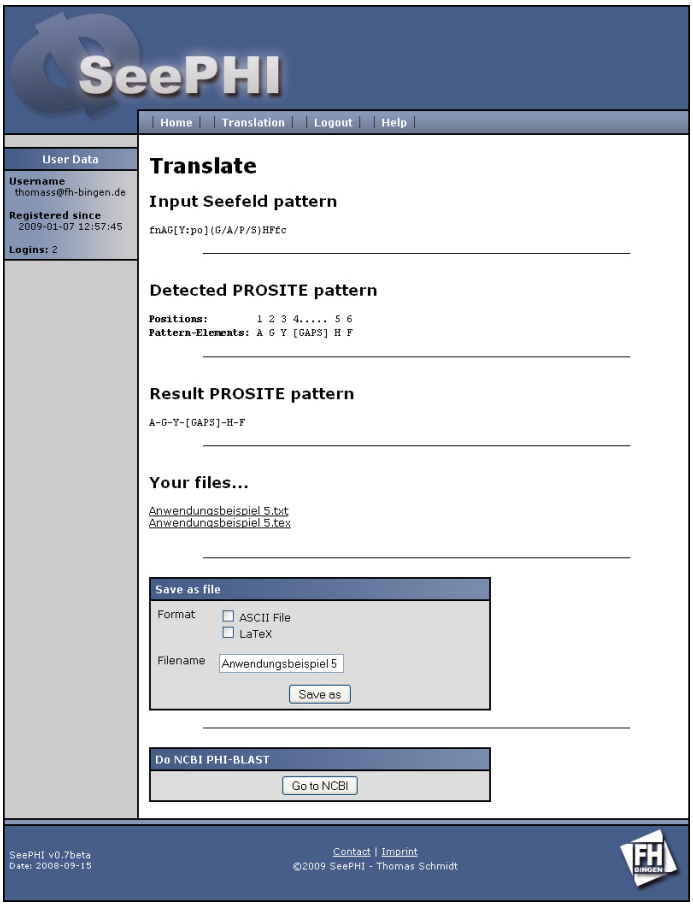
(b) *Programmausgabe*

Abbildung 3.4: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 1a.



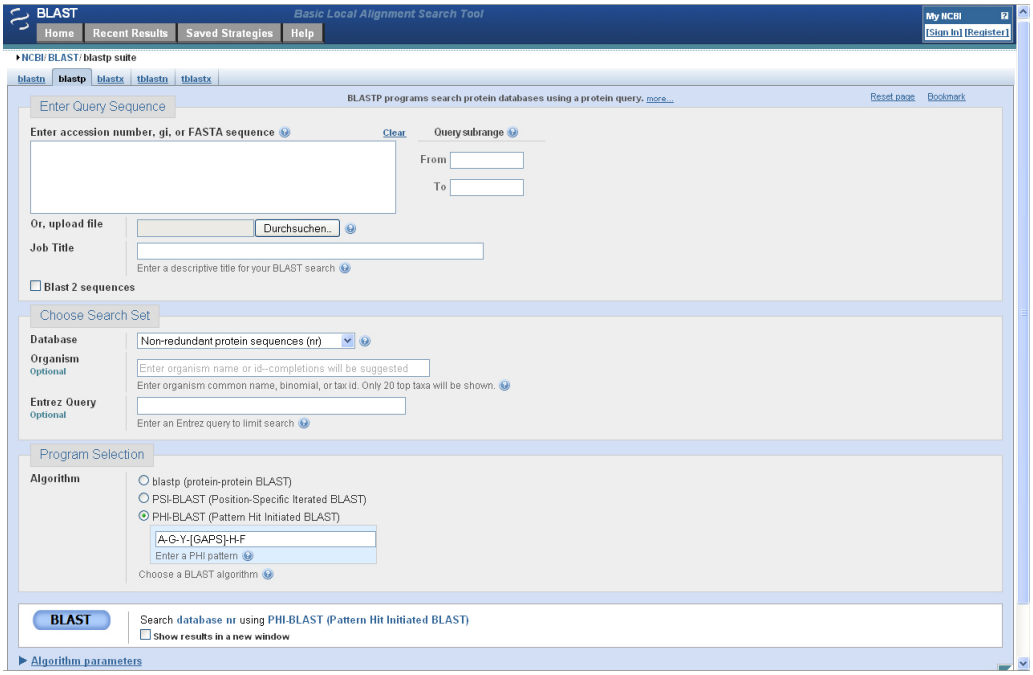
(a) *Programmausgabe nach dem Übersetzungsschritt*

Abbildung 3.5: Die Abbildung zeigt als Screenshot die Webanwendung zu Anwendungsbeispiel 1b. Für Abbildung 3.5b werden die beiden *HTML-Checkboxen* ausgewählt und das Formular per Schaltfläche „*Save as*“ abgeschickt.

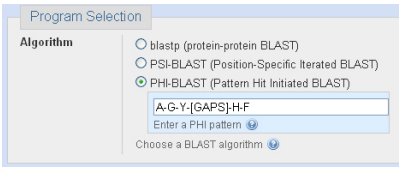


(b) *Programmausgabe nach dem Dateiexport*

Abbildung 3.5: Die Abbildung zeigt als Screenshot die Webanwendung zu Anwendungsbeispiel 1b. Für Abbildung 3.5c wird das Formular per Schaltfläche „*Go to NCBI*“ abgeschickt.



(c) *Weiterleitung zum NCBI-PHI-BLAST-Suchformular*



(d) *Vorausgefülltes Suchformular*

Abbildung 3.5: Die Abbildung zeigt als Screenshot die Webanwendung zu Anwendungsbeispiel 1b.

3.3.2 Anwendungsbeispiel 2

Seefeld-Muster	`fnAG[Y:po]∼HFfc`
Übersetzt	`A-G-Y-[AGPS]-H-F`

2a. Anonyme Benutzer

Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.008815 Sekunden

Tabelle 3.3: Interpretationszeit (Übersetzung)

2b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.020875 Sekunden
Schritt 2: Dateiexport	0.011172 Sekunden

Tabelle 3.4: Interpretationszeiten (Übersetzung & Dateiexport)

(a) *Programmeingabe*

(b) *Programmausgabe*

Abbildung 3.6: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 2a.

3.3.3 Anwendungsbeispiel 3

Seefeld-Muster	`fnPALPPA[L:me]`
Übersetzt	`P-A-L-P(2)-A-L`

3a. Anonyme Benutzer

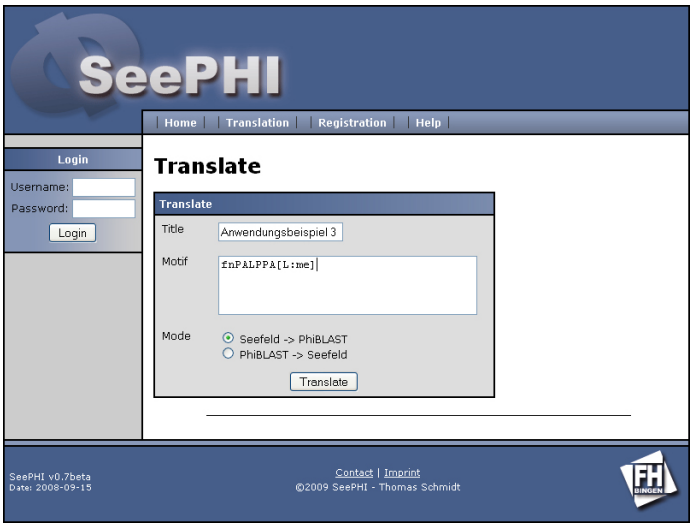
Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.021164 Sekunden

Tabelle 3.5: Interpretationszeit (Übersetzung)

3b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.009729 Sekunden
Schritt 2: Dateiexport	0.011636 Sekunden

Tabelle 3.6: Interpretationszeiten (Übersetzung & Dateiexport)



(a) *Programmeingabe*

(b) *Programmausgabe*

Abbildung 3.7: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 3a.

3.3.4 Anwendungsbeispiel 4

Seefeld-Muster	`(beta2)6=@`
Übersetzt	`[FWY]`

4a. Anonyme Benutzer

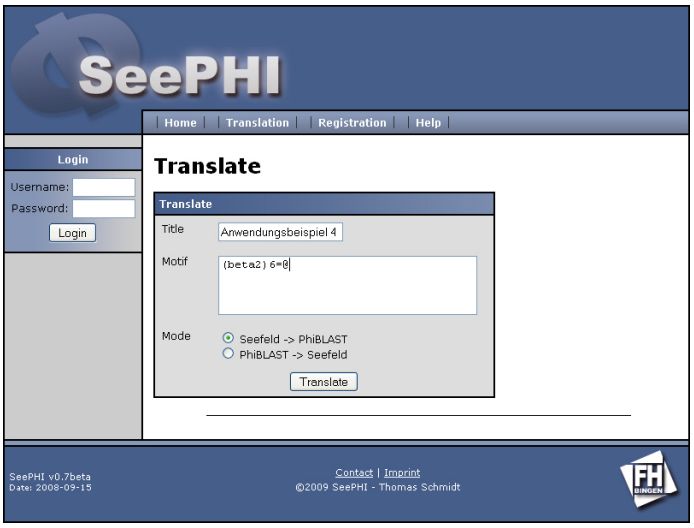
Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.008577 Sekunden

Tabelle 3.7: Interpretationszeit (Übersetzung)

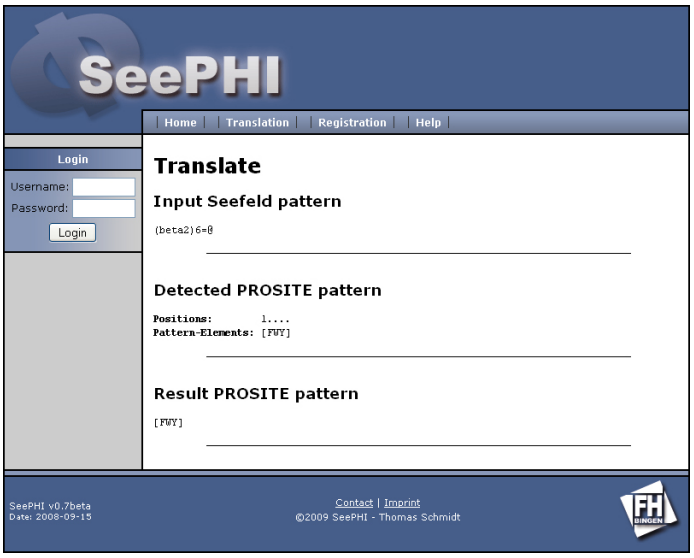
4b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.019179 Sekunden
Schritt 2: Dateiexport	0.02241 Sekunden

Tabelle 3.8: Interpretationszeiten (Übersetzung & Dateiexport)



(a) *Programmeingabe*



(b) *Programmausgabe*

Abbildung 3.8: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 4a.

3.4 PROSITE/PHI-BLAST zu Seefeld

3.4.1 Anwendungsbeispiel 5

PROSITE/PHI-BLAST-Muster	`[CS][CS]-x(0,2)-G-x(1)-C-x(2,3)-S-x(3)-L`
Übersetzt	`(C/S)(C/S)GxCxxxSxxxL`

5a. Anonyme Benutzer

Vorgang	**Interpretationszeit (*PHP*)**
Schritt 1: Request	0.009558 Sekunden
Schritt 2: Request	0.009616 Sekunden
Schritt 3: Request	0.009563 Sekunden
Schritt 4: Übersetzung	0.009055 Sekunden

Tabelle 3.9: Interpretationszeiten (Requests & Übersetzung)

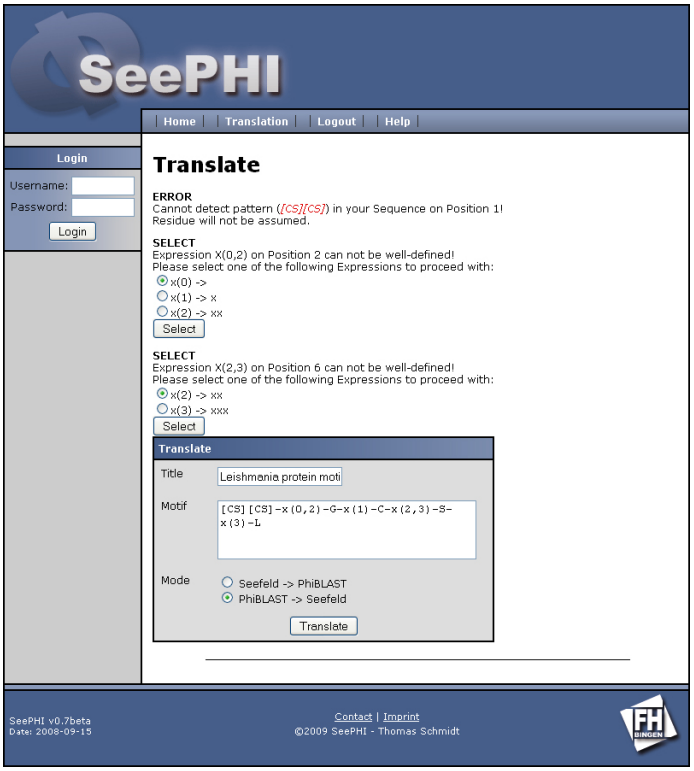
5b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Request	0.009556 Sekunden
Schritt 2: Request	0.009736 Sekunden
Schritt 3: Request	0.00968 Sekunden
Schritt 4: Übersetzung	0.009608 Sekunden
Schritt 5: Dateiexport	0.011597 Sekunden

Tabelle 3.10: Interpretationszeiten (Requests, Übersetzung & Dateiexport)

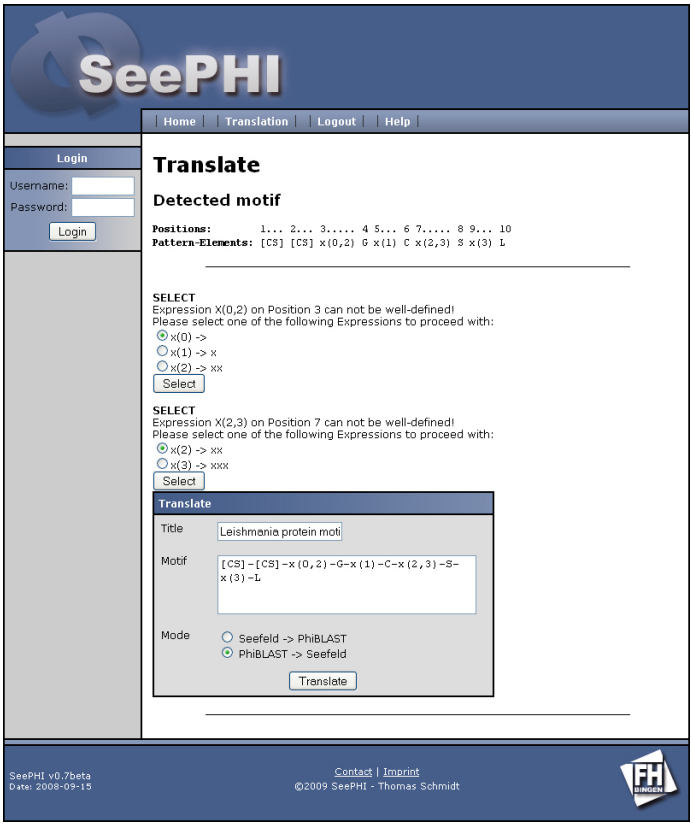
(a) *Programmeingabe*

Abbildung 3.9: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 5a.



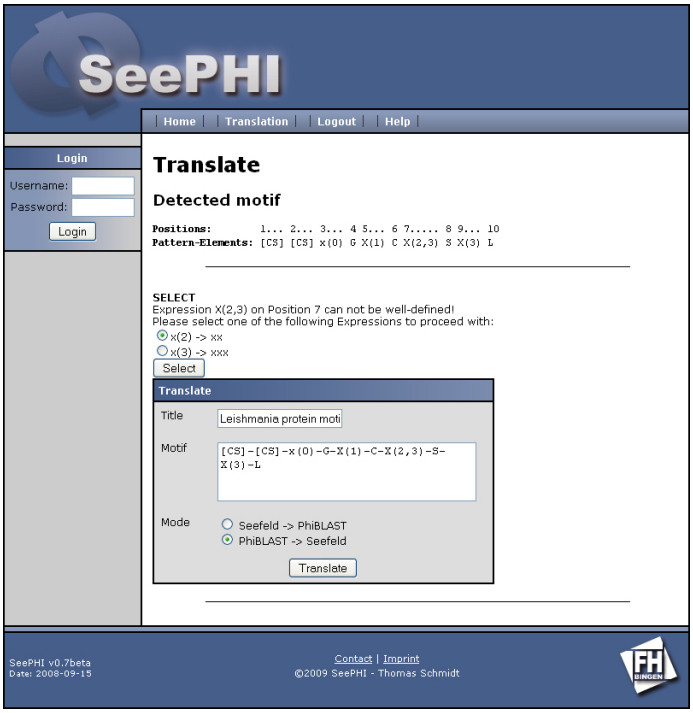
(b) *Request 1*

Abbildung 3.9: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 5a.



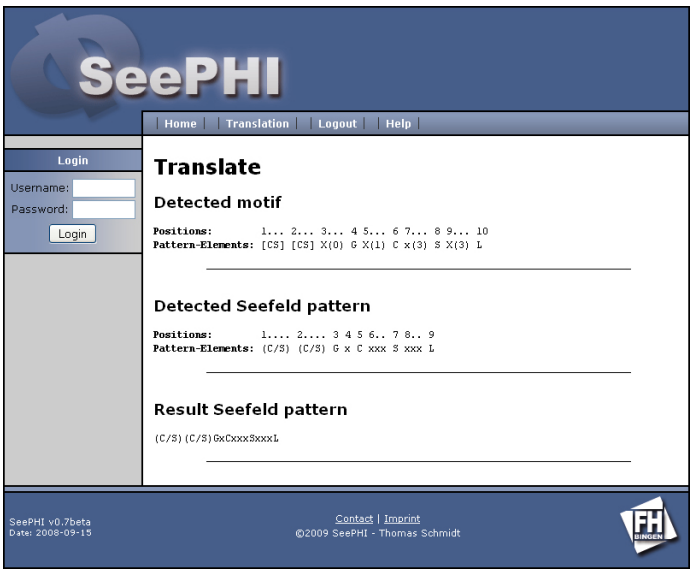
(c) *Request 2*

Abbildung 3.9: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 5a.



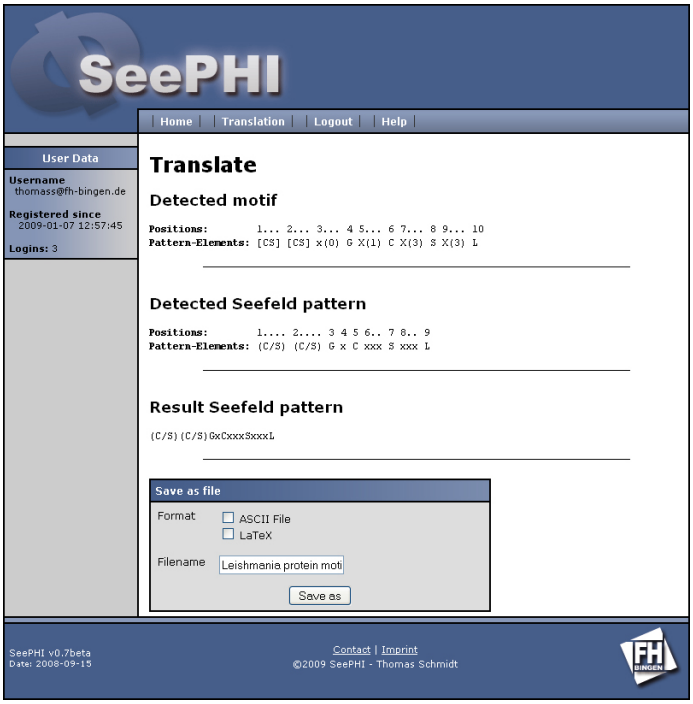
(d) *Request 3*

Abbildung 3.9: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 5a.



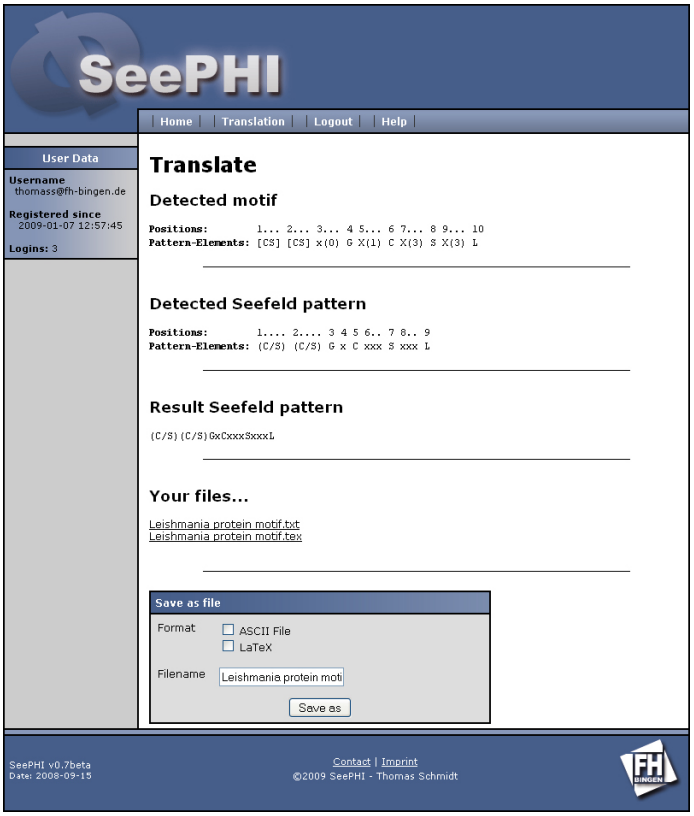
(e) *Programmausgabe*

Abbildung 3.9: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 5a.



(a) *Programmausgabe nach dem Übersetzungsschritt*

Abbildung 3.10: Die Abbildung zeigt als Screenshot die Webanwendung zu Anwendungsbeispiel 5b.



(b) *Programmausgabe nach dem Dateiexport*

Abbildung 3.10: Die Abbildung zeigt als Screenshot die Webanwendung zu Anwendungsbeispiel 5b.

3.4.2 Anwendungsbeispiel 6

PROSITE/PHI-BLAST-Muster	`F-[GSTV]-P-R-L-[G>]`
Übersetzt	`F(G/S/T/V)PRL(G/fc)`

6a. Anonyme Benutzer

Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.009082 Sekunden

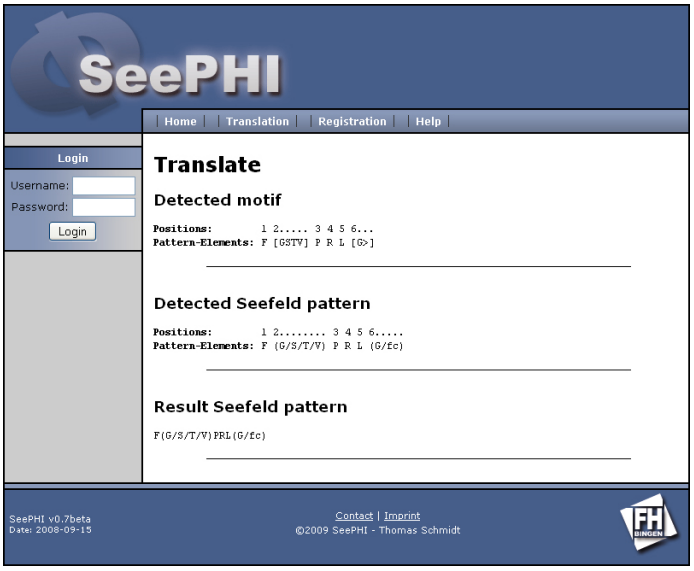
Tabelle 3.11: Interpretationszeit (Übersetzung)

6b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.009609 Sekunden
Schritt 2: Dateiexport	0.011986 Sekunden

Tabelle 3.12: Interpretationszeiten (Übersetzung & Dateiexport)

(a) *Programmeingabe*



(b) *Programmausgabe*

Abbildung 3.11: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 6a.

3.4.3 Anwendungsbeispiel 7

PROSITE/PHI-BLAST-Muster	`[AC]-x-V-x(4)-ED`
Übersetzt	`(A/C)xVxxxx[-]`

7a. Anonyme Benutzer

Vorgang	**Interpretationszeit (*PHP*)**
Übersetzung	0.009251 Sekunden

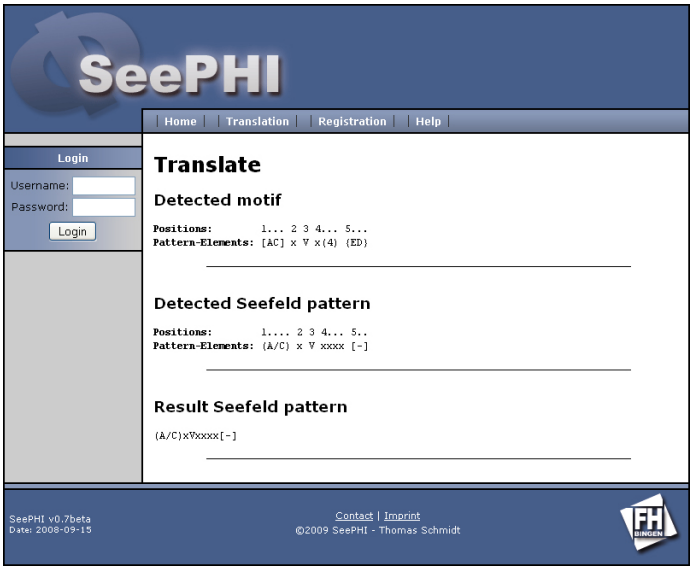
Tabelle 3.13: Interpretationszeit (Übersetzung)

7b. Eingeloggte Benutzer

Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Übersetzung	0.010233 Sekunden
Schritt 2: Dateiexport	0.011869 Sekunden

Tabelle 3.14: Interpretationszeiten (Übersetzung & Dateiexport)

(a) *Programmeingabe*



(b) *Programmausgabe*

Abbildung 3.12: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 7a.

3.4.4 Anwendungsbeispiel 8

PROSITE/PHI-BLAST-Muster	`< A-x-[ST](2)-x(0,1)-V`
Übersetzt	`fnAx(S/T)(S/T)xV`

8a. Anonyme Benutzer

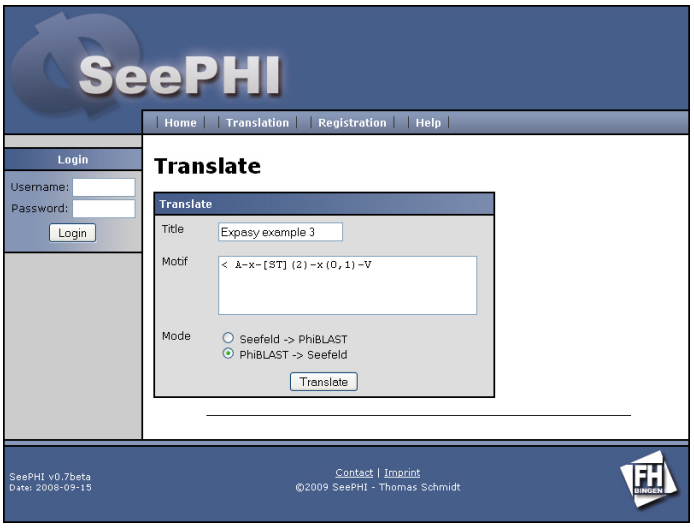
Vorgang	**Interpretationszeit (*PHP*)**
Schritt 1: Request	0.009587 Sekunden
Schritt 2: Request	0.022686 Sekunden
Schritt 3: Übersetzung	0.00913 Sekunden

Tabelle 3.15: Interpretationszeiten (Requests & Übersetzung)

8b. Eingeloggte Benutzer

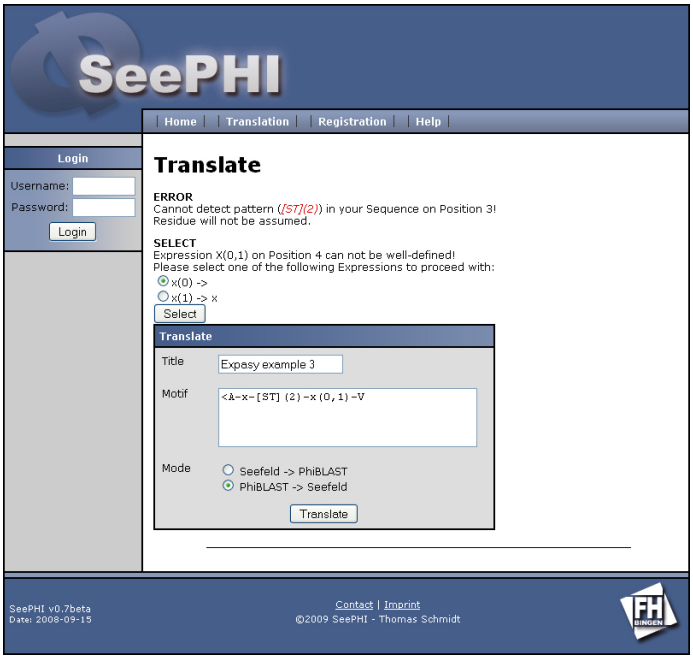
Vorgang	**Interpretationszeit (*PHP-Interpreter*)**
Schritt 1: Request	0.009603 Sekunden
Schritt 2: Request	0.009823 Sekunden
Schritt 3: Übersetzung	0.009582 Sekunden
Schritt 4: Dateiexport	0.011855 Sekunden

Tabelle 3.16: Interpretationszeiten (Requests, Übersetzung & Dateiexport)



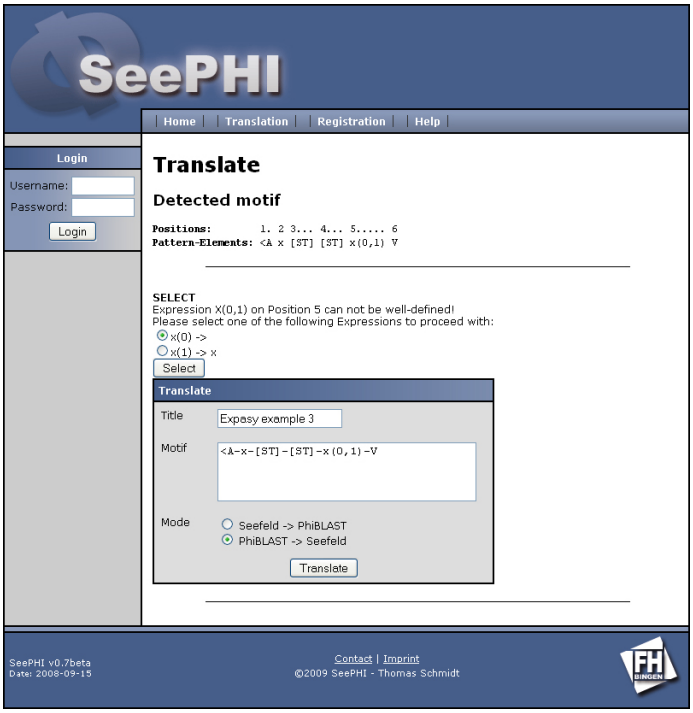
(a) *Programmeingabe*

Abbildung 3.13: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 8a.



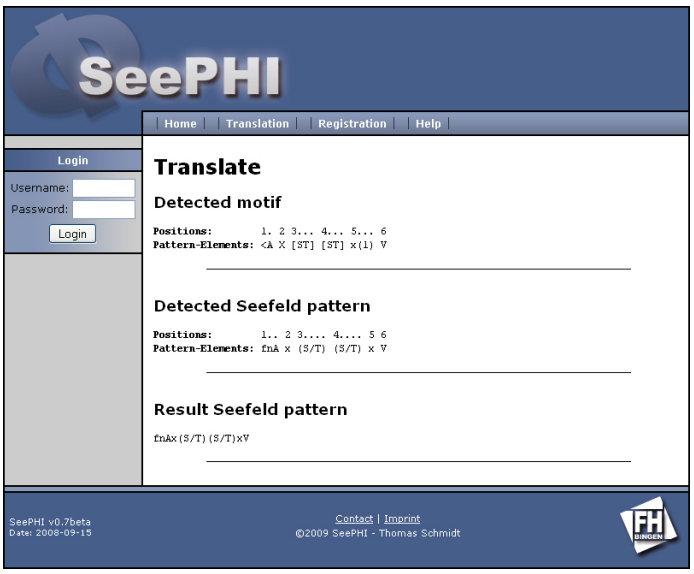
(b) *Request 1*

Abbildung 3.13: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 8a.



(c) *Request 2*

Abbildung 3.13: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 8a.



(d) *Programmausgabe*

Abbildung 3.13: Die Abbildungen zeigen Screenshots der Webanwendung zu Anwendungsbeispiel 8a.

Kapitel 4

Diskussion

4.1 Validierung

Zur Validierung der Übersetzungsroutinen werden verschiedene Sequenzen der *PROSITE-Nomenklatur*, bzw. *Seefeld-Konvention* in die jeweils andere übersetzt. Die daraus resultierende, übersetzte Sequenz wird wiederum zurückübersetzt und mit der Originalsequenz verglichen. Um viele Daten für eine statistische Auswertung zu erhalten, wird für diesen Vorgang eine Klasse zur Automatisierung entwickelt (`validator.class.php`), welche in das *PHP-Basisprogramm* integriert werden kann.
Als Testmuster werden jeweils 500 Muster nach der *Seefeld-Konvention* und der *PROSITE-Nomenklatur* gebildet, übersetzt und ausgewertet.

4.1.1 Validierungsroutine *validator.class.php*

Die Validierungsklasse deckt beide Richtungen der Übersetzungsvalidierung ab, also *Seefeld zu PROSITE zu Seefeld* (`s2p2s`) oder *PROSITE zu Seefeld zu PROSITE* (`p2s2p`). Sie greift direkt auf die jeweilige Übersetzungsroutine zu und übersetzt daher alles, was übersetzt werden kann, ohne Zwischenabfragen, Warn- oder Fehlermeldungen. Zur Generierung einer Instanz erwartet sie die Validierungsmethode (`s2p2s` oder `p2s2p`).
Zunächst wird automatisch eine zufällige Testsequenz erzeugt. Deren Komponenten bestehen aus – für die Übersetzung relevanten – Sprachelementen jeder Nomenklatur. Diese Testsequenzen sollen lediglich Auskunft über die Gültigkeit der Übersetzungen geben und entsprechen keinen realen *Peptid-Mustern*.
Durch Aufruf der Methode `multi_test(`n`)` kann die Anzahl n der zufälligen Testsequenzen gewählt werden. Jede Sequenz wird anschließend in zwei Schritten hin- und zurückübersetzt und direkt mit ihrem Original, über zwei Methoden zur Ermittlung der Ähnlichkeit zweier Zeichenketten, verglichen:

`similar_text()` Diese in *PHP* integrierte Funktion basiert auf einem Algorithmus von *Oliver* (1993, [4]).

Levenshtein Die Levenshtein-Distanz ist „ein Maß für den Unterschied zwischen zwei Zeichenketten bezüglich der minimalen Anzahl der Operationen Einfügen, Löschen und Ersetzen, um die eine Zeichenkette in die andere zu überführen. Benannt ist die Distanz nach dem russischen Wissenschaftler Wladimir Lewenstein, der sie 1965 einführte.“[1] Die *Levenshtein-Distanz* ist ebenfalls als Funktion in *PHP* integriert (`levenshtein()`).

Zusätzlich zu diesen beiden Vergleichsalgorithmen werden weitere Informationen zum Vergleich bestimmter Charakteristika ermittelt:

- **Zeichenkettenlängen**
- **Längenunterschiede** der Zeichenketten (Absolut und Prozentual)
- Ermittlung der **prozentualen *Levenshtein-Distanz*** zur Länge der Originalsequenz

Die Validierungsprozedur wird über die Adresse `http://seephi.coniubix.com/test/validator.php` ausgeführt. Dabei werden direkt 500 Testmuster erzeugt und der erste Übersetzungsschritt ausgeführt.

4.1.2 Ergebnisse: `s2p2s`

Gemäß dem oben beschriebenen Ansatz wurde die Validierung ausgehend von einem *Seefeld-Ausdruck* durchgeführt. In jedem der 500 Fälle stimmte der zurückübersetzte *Seefeld-Ausdruck* mit dem ursprünglichen Ausdruck zu 100% überein.

4.1.3 Ergebnisse: `p2s2p`

Vergleich der Sequenzlängen

Abbildung 4.1 zeigt alle Sequenzlängen des Original-Testmusters, sowie die der zugehörigen, zurückübersetzten Sequenzen.
Es fällt auf, dass die Länge der zurückübersetzen Sequenz fast ausnahmslos kleiner ist, als die der Originalsequenz. Trüge man die Punkte in einem Histogramm auf, so würde deutlich, dass die Charakteristik der Originalsequenz eine leichte Rechtsverschiebung,

[1] Quelle: Wikipedia ([6]).

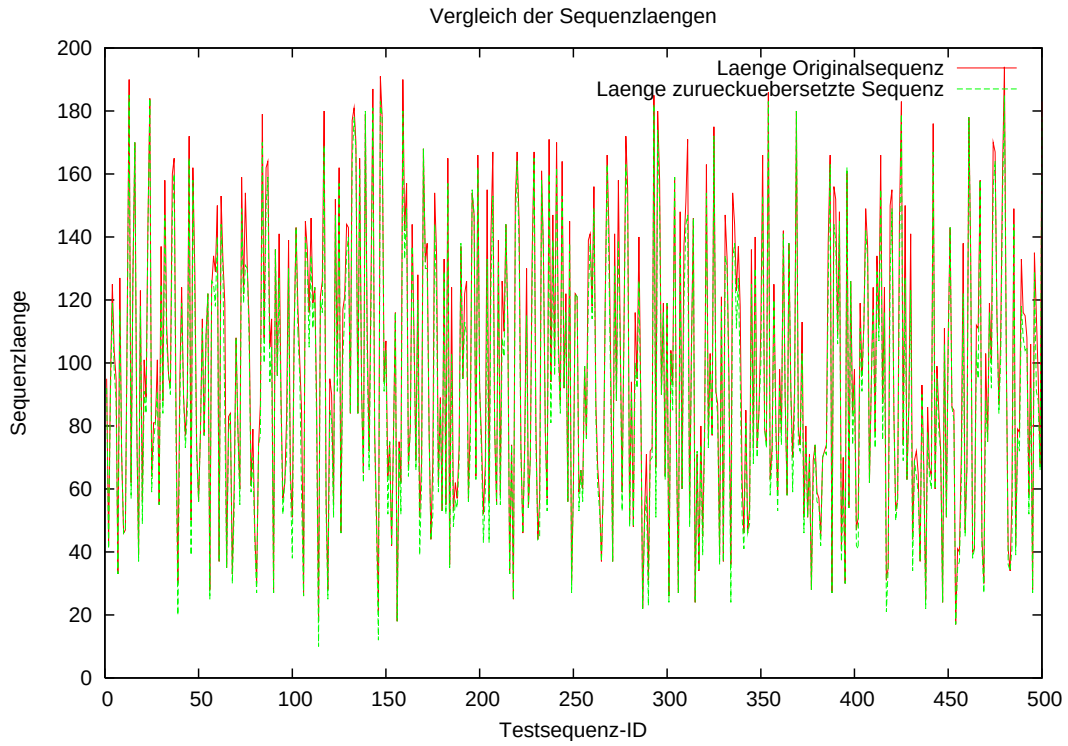


Abbildung 4.1: Die Abbildung zeigt die Längen der Originalsequenz im Vergleich zu den Längen der zurückübersetzten Sequenz. Alle Werte sind voneinander unabhängig und wurden lediglich zur besseren Darstellung über Geraden verbunden.

gegenüber der Charakteristik der zurückübersetzten Sequenz, aufweist. Durch Ermittlung der arithmetischen Mittelwerte ($\overline{x}_{\text{arithm,ori}} = 98,09$ und $\overline{x}_{\text{arithm,reüb}} = 92,896$) und der Standardabweichungen ($\sigma_{x,\text{ori}} = 43,3202$ und $\sigma_{x,\text{reüb}} = 41,8846$) wird diese Annahme unterstützt.

Ist die zurückübersetzte Zeichenkette kürzer als das Original, enthält aber dennoch alle Informationen der Originalsequenz, so kann man von einer Vereinfachung der Original-Zeichenkette sprechen. Stichprobenartige Untersuchungen der 500 Testmuster zeigten eine erfolgreiche Übersetzung ohne Informationsverlust.

Analog dazu zeigt der Vergleich der Sequenzlängen-Differenzen (siehe Abbildung 4.2) den gleichen Effekt. Aufgetragen wurden zum Einen alle ermittelten absoluten Dif-

ferenzwerte, diese beschreiben, dass die zurückübersetzte Sequenz im Vergleich zur Originalsequenz beispielsweise „-3 Zeichen länger“ (also 3 Zeichen kürzer) ist, zum Anderen der prozentuale Anteil der Differenz an der Gesamtlänge der Originalsequenz.

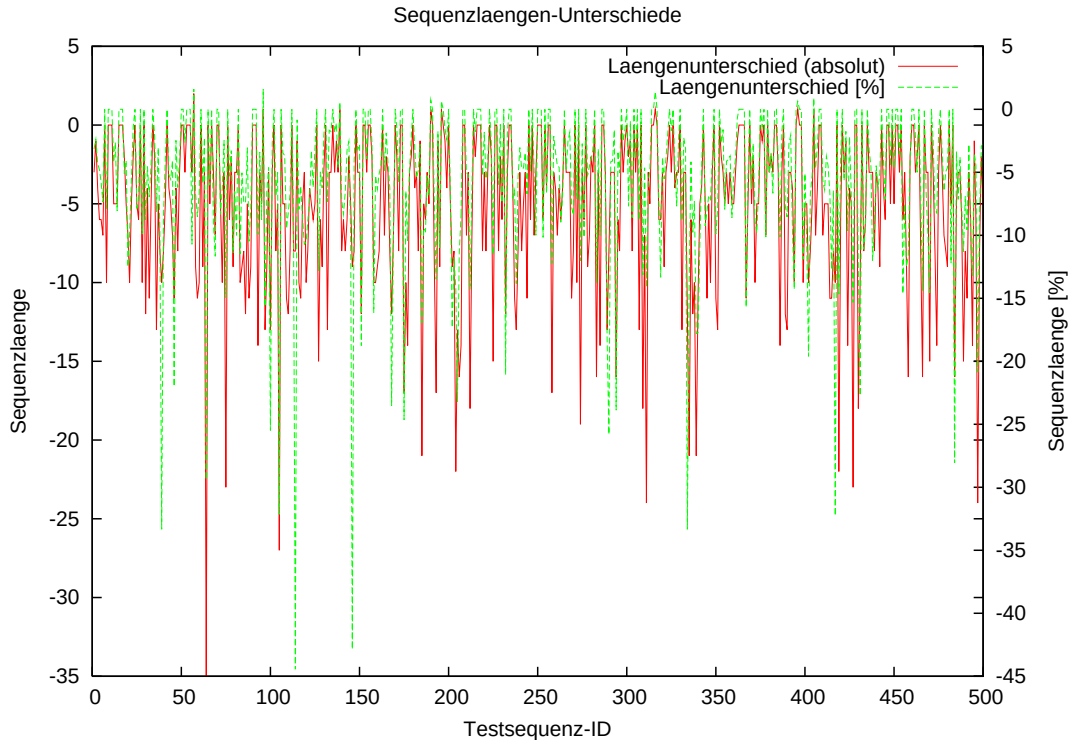


Abbildung 4.2: Die Abbildung zeigt die Längendifferenzen der Originalsequenz im Vergleich zu denen der zurückübersetzten Sequenz. Alle Werte sind voneinander unabhängig und wurden lediglich zur besseren Darstellung über Geraden verbunden.

Fast immer ist die Differenz negativ, was bedeutet, dass die zurückübersetzte Sequenz kürzer ist, als die Originalsequenz (Vereinfachung, siehe oben).
Ferner kann man aus dem prozentualen Wert jeder Differenz zur Originalsequenzlänge deuten, dass die Längendifferenz unabhängig von der Sequenzlänge ist, da die prozentualen Werte recht stark gestreut sind.

Vergleich der Ähnlichkeiten

In Abbildung 4.3 und 4.4 werden die Ergebnisse der beiden Ähnlichkeitsvergleiche dargestellt.

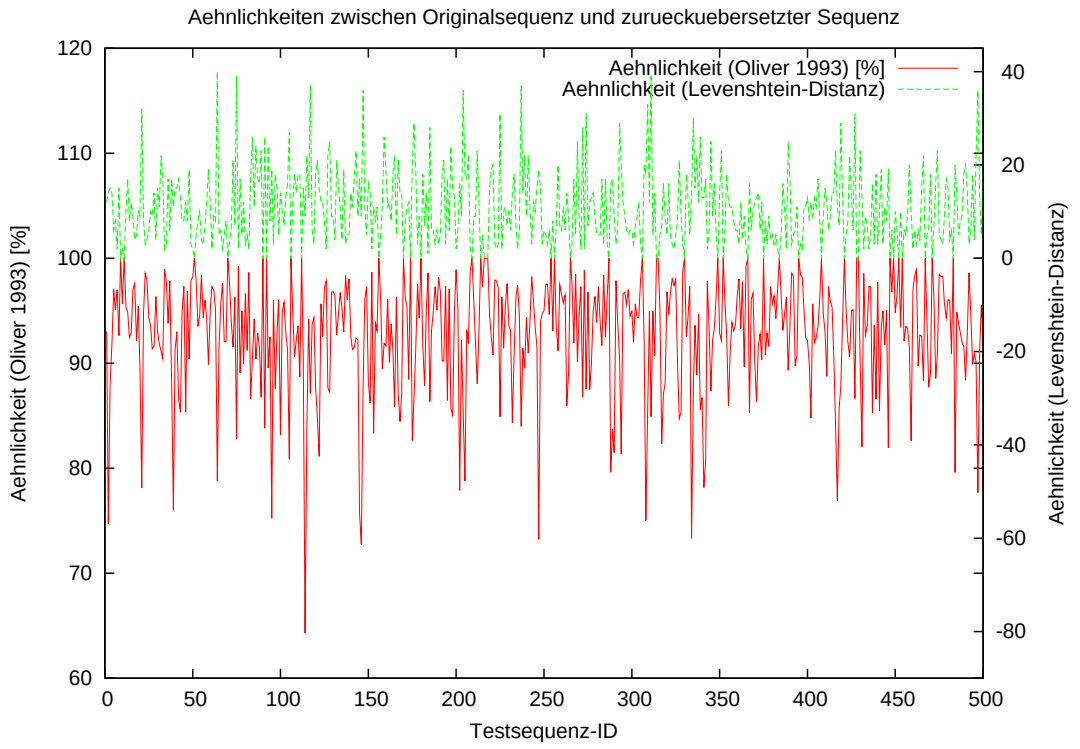


Abbildung 4.3: Die Abbildung zeigt die Ergebnisse der beiden verwendeten Methoden zur Bestimmung der Ähnlichkeiten zwischen Originalsequenz und zurückübersetzter Sequenz. Alle Werte sind voneinander unabhängig und wurden lediglich zur besseren Darstellung über Geraden verbunden.

Zu erkennen ist, dass die Charakteristik der prozentualen Ähnlichkeit nach Oliver ([4]) umgekehrt äquivalent ist zur Charakteristik der Levenshtein-Distanz. Die Charakteristiken deuten eine Symmetrie an.
Nach Levenshtein entspricht eine Distanz von 0 einer hundertprozentigen Übereinstimmung der Sequenzen. Die Signifikanz der Unterschiede zeigt sich nach der Methode von Oliver deutlicher, da sich der prozentuale Wert aus der Sequenzlänge errechnet.

Algorithmus	**Mittelwert $\overline{x}_{\mathbf{arithm}}$**	**Standardabweichung (σ_x)**
Oliver	92,8368%	5,5339%
Levenshtein	10,598	7,9838

Tabelle 4.1: Die Tabelle zeigt Mittelwerte und Standardabweichung der beiden Ähnlichkeitsvergleiche über 500 Testmuster.

Die nach Oliver voneinander unterschiedlichsten Sequenzen liegen bei Testmuster 114 (Oliver = 64,29%; Levenshtein = 9).

PROSITE	`M-x(2)-x(2)-H-P(1)`
Seefeld	`MxxxxHP`
PROSITE	`M-x(4)-H-P`

Tabelle 4.2: Die Tabelle zeigt die voneinander unterschiedlichsten Sequenzen nach Oliver.

Die nach Levenshtein voneinander unterschiedlichsten Sequenzen liegen bei Testmuster 64 (Levenshtein = 40; Oliver = 78,82%).

PROSITE	`<{B<ZC}-x(3)-x(5)-x(4)-x(2)-{HTSDEUF}-Y-[YFW]-{VUIP}-G-T(4)-{GTHIUWV}-S(1)-x(5)-x(4)-x(2)-{RTE}-x(1)-R(1)-A(1)-{GD>SFE}`
Seefeld	`fn^(B/fn/Z/C)xxxxxxxxxxxxxx^(H/T/S/D/E/U/F)Y@^(V/U/I/P)GTTTT^(G/T/H/I/U/W/V)Sxxxxxxxxxxx^(R/T/E)xRA^(G/D/fc/S/F/E)`
PROSITE	`<{B<ZC}-x(14)-{HTSDEUF}-Y-[FWY]-{VUIP}-G-T(4)-{GTHIUWV}-S-x(11)-{RTE}-x-R-A-{GD>SFE}`

Tabelle 4.3: Die Tabelle zeigt die voneinander unterschiedlichsten Sequenzen nach Levenshtein.

Zum besseren Vergleich wird die Levenshtein-Distanz, welche die minimale Anzahl der Operationen um die eine Zeichenkette in die andere zu überführen angibt, prozentual zur Zeichenkettenlänge ermittelt. Demnach ergibt eine hohe Levenshtein-Distanz in einer kurzen Zeichenkette eine größere Abweichung als die gleiche Distanz in einer langen Sequenz (siehe Abbildung 4.4).

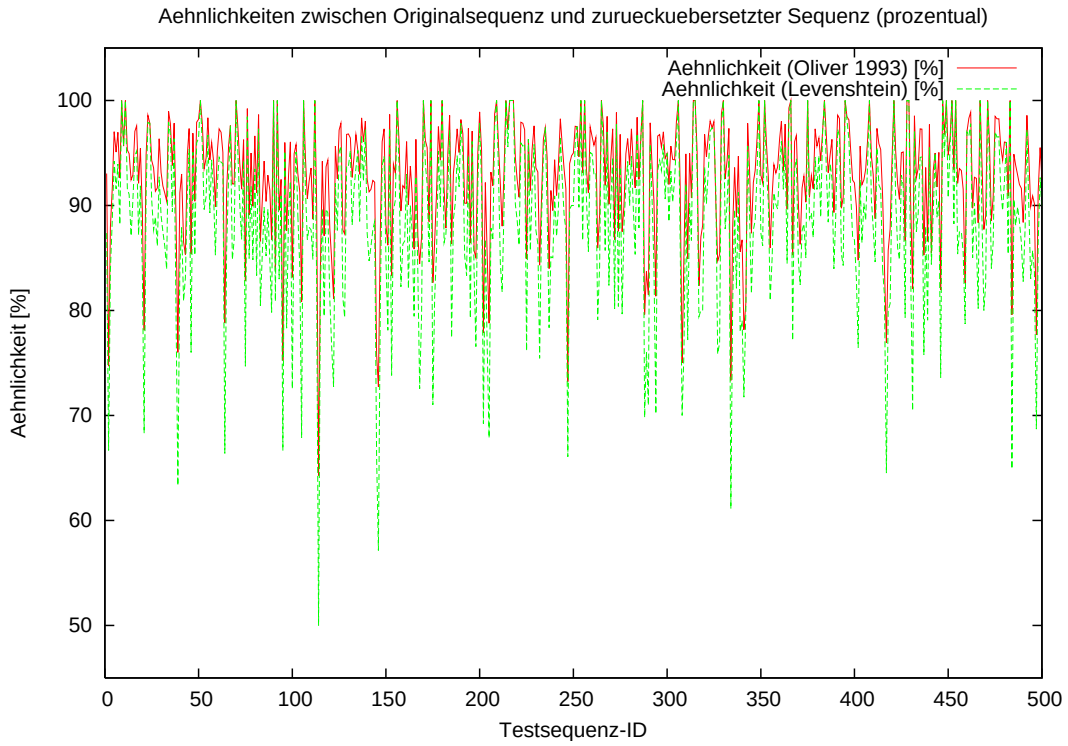


Abbildung 4.4: Die Abbildung zeigt die Ergebnisse der beiden verwendeten Methoden zur Bestimmung der Ähnlichkeiten zwischen Originalsequenz und zurückübersetzter Sequenz. Dazu wurde die Levenshtein-Distanz prozentual zur Länge der Originalsequenz ermittelt. Alle Werte sind voneinander unabhängig und wurden lediglich zur besseren Darstellung über Geraden verbunden.

Abbildung 4.4 stellt die prozentual ermittelten Levenshtein-Distanzen zur Länge der Originalsequenz im Vergleich zur Ähnlichkeitsermittlung nach Oliver dar. Dabei streuen die auf der Levenshtein-Distanz basierenden Werte stärker, stellen jedoch die gleiche Charakteristik dar wie der Ähnlichkeitsvergleich nach Oliver.

Algorithmus	**Mittelwert $\overline{x}_{\mathbf{arithm}}$**	**Standardabweichung (σ_x)**
Oliver	92,8368%	5,5339%
Levenshtein [%]	88,8105	8,0326

Tabelle 4.4: Die Tabelle zeigt Mittelwerte und Standardabweichung der beiden Ähnlichkeitsvergleiche über 500 Testmuster (prozentual).

Längendifferenz und Ähnlichkeit

Nachfolgend soll verdeutlicht werden, wie groß der Zusammenhang zwischen der Ähnlichkeit der Zeichenketten und der Längendifferenz zwischen Original- und zurückübersetzter Sequenz ist. Beide Eigenschaften sind in Abbildung 4.5 gegeneinander aufgetragen.

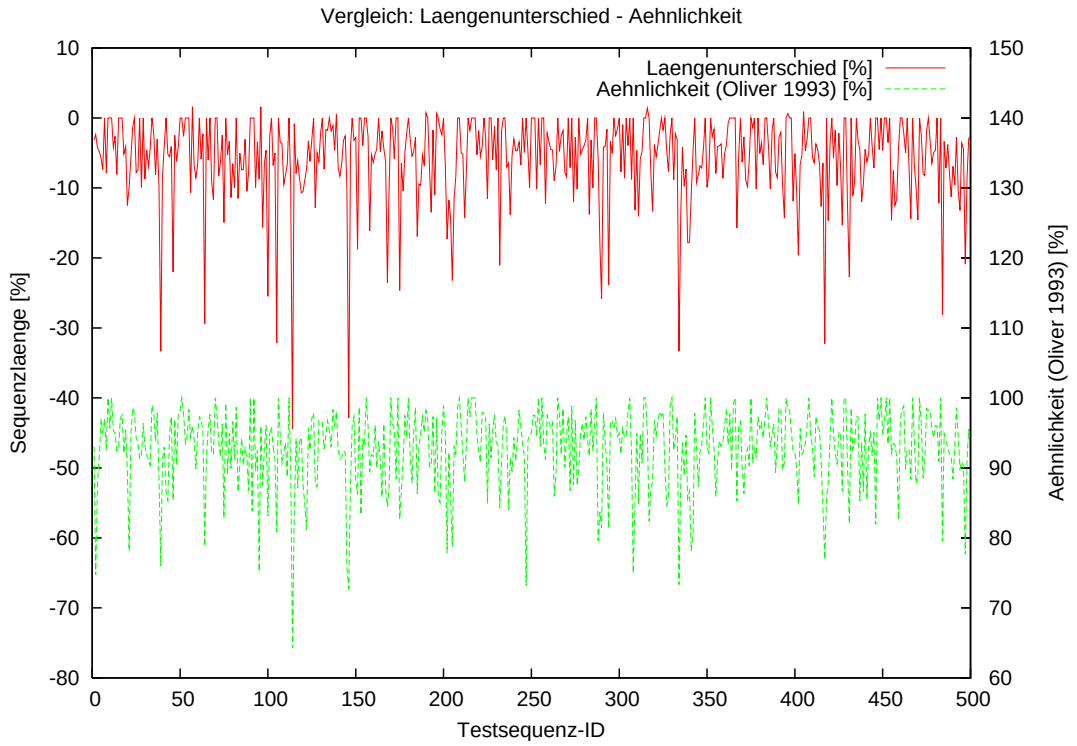


Abbildung 4.5: Die Abbildung zeigt die Längendifferenz von Original- zu zurückübersetzter Sequenz im Vergleich zur Ähnlichkeit nach Oliver. Alle Werte sind voneinander unabhängig und wurden lediglich zur besseren Darstellung über Geraden verbunden.

Überwiegend wird festgestellt, dass an den Stellen, an denen ein großer Unterschied zwischen den Sequenzen festgestellt wurde auch die Längendifferenz wesentlich größer ist. Ein Beispiel für den umgekehrten Fall wäre `x(2)-[ED]-x(3) -> xx[-]xxx -> x(2)-[DE]-x(3)`. Hier liegt eine Längendifferenz von 0% vor, dennoch weicht die

zurückübersetzte Sequenz von der Originalsequenz ab. Durch die Übersetzung von `[ED]` in `[-]` geht die angegebene Reihenfolge von `E` und `D` verloren. Zurückübersetzt ergibt sich die alphabetische Reihenfolge `[DE]`, was jedoch nicht die Qualität des *PROSITE-Ausdrucks* mindert.
Die großen Differenzen in der Länge der Sequenzen entstehen durch eine Vereinfachung bei den Übersetzungsschritten (siehe oben).

4.1.4 Fazit

Die Validierung erfolgte in zwei Teilen (`s2p2s` & `p2s2p`) zu je zwei Schritten:

1. Bau der 500 Testsequenzen und Übersetzung
2. Zurückübersetzung, Ähnlichkeitsvergleiche und Dateiausgabe

Jeder dieser Schritte dauerte ca. 1,5 Sekunden und ist damit ein Indikator für die Leistungsfähigkeit des implementierten Ansatzes, der Webanwendung.
Die Validierung hat gezeigt, dass die Übersetzung von *Seefeld* nach *PROSITE* und wieder zurück (`s2p2s`) ohne Informationsverluste abläuft. Sämtliche zurückübersetzten *Seefeld-Ausdrücke* entsprachen ihrer jeweiligen Originalsequenz. Dies zeigt zum Einen, dass die implementierte Übersetzungsprozedur in beide Richtungen funktioniert. Zum Anderen verdeutlicht diese Tatsache, wie restriktiv die *Seefeld-Konvention* tatsächlich ist. Bereits Eingangs wurde erläutert, dass aus biologischer Sicht eine viel höhere Genauigkeit gefordert ist, um die Bindungseigenschaften eines Moleküls (*Peptid-Liganden*) abzubilden. Der übliche Spielraum, den man bei Proteinen möglicherweise hat, beispielsweise entweder 2, 3 oder 4 unbekannte Aminosäuren an einer Position (*PROSITE-Ausdruck*: `x(2,4)`), macht für *Peptid-Liganden* wenig Sinn.
Ein weiterer Grund für die hohen Übereinstimmungen ist, dass zahlreiche Spracheigenschaften, die von der *PROSITE-Nomenklatur* nicht unterstützt werden, beim Validieren von vornherein ausgelassen wurden, wie zum Beispiel *phosphorylierte Aminosäuren*: `[Y:po]`. Lediglich die – für beide Nomenklaturen gemeinsam – relevanten Spracheigenschaften wurden zur Generierung der Testsequenzen genutzt.
Die Übersetzungen von *PROSITE* nach *Seefeld* und zurück haben in der Validierung viele Unterschiede gezeigt. Diese sind auf eine Vereinfachung der zufällig generier-

ten Originalsequenz zurückzuführen (zum Beispiel: `x(2)-x(3) -> xxxxx -> x(5)`); bewirken also eine maßgebliche Änderung der Sequenzlänge (große Sequenzlängendifferenz). Zudem konnten weitere Ähnlichkeitsunterschiede zwischen zurückübersetzter und Originalsequenz gezeigt werden, die jedoch keinen Qualitätsverlust zur Originalsequenz darstellen. Dies wurde durch manuell durchgeführte Stichprobenuntersuchungen, auf Richtigkeit der Übersetzungen, geprüft.

4.2 Probleme & Verbesserungsvorschläge

4.2.1 PHP-Basisprogramm

Problem: Unsicheres *CAPTCHA* bei der Registrierung

Das zur Registrierung neuer Benutzer erforderliche *CAPTCHA*[2] besteht aus einer fünfstelligen Zahlenfolge die dynamisch als Bild genriert wird. Der Benutzer muss die Zahlenfolge erkennen und zusätzlich zu den anderen erforderlichen Formulardaten eingeben. Über ein `hidden`-Feld im Registrierungsformular wird diese Zahlenfolge *MD5-verschlüsselt* (irreversibel) übertragen. Die vom Benutzer eingegebene Zahlenfolge wird in der durch das Formular aufgerufenen Routine ebenfalls *MD5-verschlüsselt* und mit der per `hidden`-Feld übertragenen *MD5-Zeichenfolge* verglichen. Bei Übereinstimmung wird der Benutzer als Mensch authentifiziert.
Obwohl die *MD5-Verschlüsselung* irreversibel ist, also die ursprüngliche Zeichenkette nicht wiederhergestellt werden kann, existieren im Internet zahlreiche Datenbanken, welche *MD5-verschlüsselte* Zeichenketten, mit der dazugehörigen Originalzeichenkette, enthalten. Da es sich bei dem Inhalt des *CAPTCHAs* nur um eine fünfstellige Zahlenfolge handelt, kann im verwendeten *CAPTCHA* jede Zahlenfolge relativ einfach aus einer *MD5-Datenbank* rekonstruiert werden.
Die Unterscheidung zwischen Mensch und Maschine nach dem verwendeten Ansatz kann also umgangen werden, da die benötigte, verschlüsselte Zahlenfolge direkt und

[2] *CAPTCHA* ist ein Akronym für *Completely Automated Public Turing test to tell Computers and Humans Apart. CAPTCHAs* werden verwendet, um zu entscheiden, ob das Gegenüber ein Mensch oder eine Maschine ist.
Quelle: Wikipedia [7])

maschinell aus dem Registrierungsformular ausgelesen und anschließend mithilfe einer Datenbank rekonstruiert werden kann. Die zurückgewonnene Zahlenfolge muss dann mit den restlichen Formularelementen an die Registrierungsprozedur übertragen werden, was ebenfalls automatisch erfolgen kann. Als Ergebnis würden „Spam-Mails“ mit Registrierungsinformationen an beliebige E-Mail-Adressen versendet werden. Zudem verlangsamen diese *gültigen* Anfragen den Server unnötig, da *HTTP*-, Datenbank- und Mailserver betroffen sind.
Folgende Maßnahmen würden das Problem lösen:

- Neben Zahlen könnten auch Buchstaben und bestimmte Sonderzeichen zum Einsatz kommen.
- Keine Übertragung von *CAPTCHA-Daten* von einer Webseite auf die andere (Formular (GET, POST), *PHP-Sessions*).
- Alle *CAPTCHA-Bilder* sollten im Voraus generiert und zwischengespeichert (Dateisystem oder Datenbank) werden.
- Zu jedem *CAPTCHA* muss die entsprechende Zeichenfolge (intern), die es als Bild darstellt, gespeichert werden.
- Beim ersten Anzeigen des Registrierungsformulars wird gespeichert, dass eines der *CAPTCHAs* aus dem Pool „in Gebrauch“ ist, zusammen mit einem Timestamp und der IP des Benutzers.
- Nach dem Senden des Formulars wird geschaut, ob diese IP ein *CAPTCHA* in Gebrauch hat und ob die vom Benutzer eingegebene Zeichenfolge mit der zum *CAPTCHA* zugehörigen überein stimmt.
- Der Timestamp ermöglicht ein ungültig-werden des *CAPTCHAs*.

Problem: Unsichere Datenbankanbindung

Für das *PHP-Basisprogramm* wurde die Routine „`db.class.php`“ für das Datenbank-Handling, also Datenbank-Verbindung und *SQL-Abfragen*, entwickelt. Diese nutzt die

PHP-eigenen Funktionen für *MySQL*. Bei *SQL-Abfragen* stehen deshalb Variablen direkt im *SQL-Code*. Bei der Ausführung der Abfrage werden die Variablen zunächst ersetzt durch ihren jeweiligen Wert und dann der gesamte *SQL-Befehl* an *mySQL* gesendet. Im Hinblick auf *Hacking-Methoden* bietet dieser Ansatz eine wesentliche Sicherheitslücke, wenn besagte Variablen schadhaften *SQL-Code* enthalten (*SQL-Injection*). Eine Gefahr diesbezüglich ist in der Momentanen Anwendung beispielsweise in der Registrierungsprozedur gegeben, da hier direkt die Benutzereingaben als Variablen im *SQL-Code* stehen.
Als Lösung bietet *PHP* ab Version 5.1 das *PHP-Database-Object* (*PDO*). Hier hat man die Möglichkeit eine Datenbank-Abfrage zu definieren und diese *Vorzuverarbeiten*. Die Variablen werden erst in einem zweiten Schritt an die bereits an *MySQL* gesendete *SQL-Abfrage* übergeben, welche dann ausgeführt wird.
Ein positiver Nebeneffekt dieser Lösung ist die Beschleunigung von mehreren gleichen hintereinander geschalteten Abfragen, da lediglich Variablen an *MySQL* gesendet werden müssen und der eigentliche Befehl bereits vorverarbeitet wurde.

Verzeichnisrechte

Ein mögliches Problem stellen die Benutzerberechtigungen auf bestimmte Verzeichnisse dar. Einige Verzeichnisberechtigungen wurden, um die Anwendung zum Laufen zu bringen, zunächst auf zu großzügig gesetzt. Dies betrifft die Verzeichnisse, in die aus *SeePHI* Dateien gespeichert werden. Inwiefern diese eine Sicherheitslücke darstellen ist unklar.
Mittelfristig ist eine Reorganisation der Verzeichnisrechte (und -Eigentümer) erforderlich.

Bessere Statistik/Benutzerfreundlichkeit

Über eine Art „Job-Tabelle“ könnten gemachte Anfragen von registrierten Benutzern gespeichert und schließlich jederzeit rekonstruiert, also erneut übersetzt werden.

4.2.2 Übersetzungsroutine

Vollständige Aminosäureklassenidentifizierung

Gegenwärtig werden Aminosäureklassen ausschließlich identifiziert, wenn sie alleine an einer Position der zu übersetzenden Sequenz stehen. Steht eine weitere Aminosäure zur Auswahl wird die Klasse nicht erkannt.

Beispiele (*PROSITE zu Seefeld zu PROSITE*)	**Status**
`[ILMV] -> # -> [ILMV]`	Implementiert
`[GILMV] -> (G/#) -> [GILMV]`	Nicht implementiert
`[ILGMV] -> (G/#) -> [GILMV]`	Nicht implementiert

Tabelle 4.5: Die Tabelle zeigt Beispiele zur vollständigen Aminosäureklassenidentifizierung.

Hätte diese Funktion bereits zur aktuellen Programmversion zur Verfügung gestanden, hätte die Validierung Ähnlichkeitsunterschiede zwischen dem zurückübersetzten *Seefeld-Ausdruck* und seinem jeweiligen Original zur Folge aufgezeigt. Die Sequenzlängen jedoch hätten sich im Vergleich nicht geändert (Sequenzlängendifferenz = 0%).

Vollständige multiple Positionen

In der aktuellen Version von *SeePHI* bezieht sich Erkennung von mehreren hintereinanderliegenden Aminosäuren in der Sequenz, nach der Syntax `x(n)`, lediglich auf einzelne Aminosäuren. Mehrere in Frage kommende Aminosäuren an einer Position werden nicht erkannt. Auch die Erkennung von Aminosäureklassen erfolgt momentan gänzlich nicht.

Beispiele (*PROSITE zu Seefeld zu PROSITE*)	**Status**
`x(2) -> xx -> x(2)`	Implementiert
`F(4) -> FFFF -> F(4)`	Implementiert
`[AELP](2) -> (A/E/L/P)(A/E/L/P) -> [AELP](2)`	Nicht implementiert
`[FWY](3) -> @@@ -> [FWY](3)`	Nicht implementiert
`[FGWY](2) -> (@/G)(@/G) -> [FGWY](2)`	Nicht implementiert

Tabelle 4.6: Die Tabelle zeigt Beispiele zu vollständigen multiplen Positionen.

Kapitel 5

Zusammenfassung

Ziel dieser Studienarbeit war die Entwicklung und Implementierung einer webbasierten Übersetzungsroutine von regulären Ausdrücken nach der *Seefeld-Konvention* in die *PHI-BLAST/PROSITE-Nomenklatur* und umgekehrt.
Nach Einarbeitung in die zugrunde liegende Syntax jeder Nomenklatur wurde eine Kompatibiliätstabelle erstellt, welche Übersetzungsschwächen, wie eventuellen Informationsverlust, aufdeckt. Es wurde festgestellt, dass eine Überstetzung von der einen Nomenklatur in die andere grundsätzlich möglich ist.
Die eigentliche Übersetzungsroutine wurde in ein speziell entwickeltes *Web-Grundgerüst* eingebettet, welches eine minimale Standardfunktionalität bereitstellt. Hierzu wurde eine sinnvolle Datei- und Verzeichnisstruktur erstellt, sowie eine *MySQL-Datenbank* entworfen und integriert. Als Programmiersprache wurde objektorientiertes *PHP* gewählt. Alle verwendeten Klassen der Anwendung sind in Form von *UML-Diagrammen* enthalten.
Die Implementierung der Übersetzungsroutine erfolgte in einer allgemeinen Klasse, welche aus dem Hauptprogramm aufgerufen wird. Je nach Übersetzungsart – ob von *PROSITE* nach *Seefeld* oder umgekehrt – wird ein entsprechendes Modul geladen, welches die spezifischen Übersetzungsfunktionen bereitstellt.
Anhand von verschiedenen Anwendungsbeispielen wurden die Programmfunktionen verdeutlicht. Zahlreiche Screenshots geben einen Eindruck von der Interaktion zwischen Benutzer und Anwendung.
In Kapitel 4 (*Diskussion*) wurde eine Validierung des Programms durchgeführt. Hierzu wurden jeweils 500 zufällige und gültige Testsequenzen beider Nomenklaturen erzeugt,

in die jeweils andere übersetzt und wieder zurückübersetzt. Die zurückübersetzte Sequenz wurde anschließend mit ihrer jeweiligen Ursprungssequenz nach zwei verschiedenen Ähnlichkeits-Analyseverfahren verglichen. Als Fazit konnte man erkennen, dass die Übersetzung in beide Richtungen funktioniert, obwohl teils große Ähnlichkeitsunterschiede zu verzeichnen waren.
Einige Probleme und Verbesserungsvorschläge wurden genannt. Dabei sollten in naher Zukunft entdeckte Sicherheitslücken der Webanwendung beseitigt werden. Zudem wurde eine Übersetzungsschwäche aufgedeckt, die ebenfalls in einer nachfolgenden Version von *SeePHI* gelöst sein sollte.
Bei abschließender Betrachtung kann die bis hier entwickelte Anwendung noch weiter ausgebaut werden. Beispiele für weitere Programmfunktionen wären *PDF-Support* oder die Anbindung an Protein-Ligand-Datenbanken. Dennoch entstand eine sinnvolle Bioinformatik-Anwendung, die zukünftig wahrscheinlich an Bedeutung gewinnen wird.

Literaturverzeichnis

[1] Aasland, R. ; Abrams, C. ; Ampe, C. ; Ball, L. J. ; Bedford, M. T. ; Cesareni, G. ; Gimona, M. ; Hurley, J. H. ; Jarchau, T. ; Lehto, V. P. ; Lemmon, M. A. ; Linding, R. ; Mayer, B. J. ; Nagai, M. ; Sudol, M. ; Walter, U. ; Winder, S. J.: Normalization of nomenclature for peptide motifs as ligands of modular protein domains. In: *FEBS Letters* 513 (2002), Feb, Nr. 1, S. 141–144

[2] ExPASy.ch: *ScanProsite tool manual — Advanced Scan mode (Settings).* `http://www.expasy.ch/tools/scanprosite/scanprosite-doc.html`. Version: 2009. – [Online; Stand 17. Januar 2009]

[3] IUPAC/IUBMB: *Nomenclature and Symbolism for Amino Acids and Peptides.* `http://www.chem.qmul.ac.uk/iupac/AminoAcid/AA1n2.html`. Version: 2009. – [Online; Stand 17. Januar 2009]

[4] Oliver, J. J.: Decision Graphs — An Extension of Decision Trees / Department of Computer Science, Monash University, Clayton Victoria 3168 Australia. 1993 (92/173). – Forschungsbericht. – Shortened appeared in AI and Statistics 1993

[5] Otte, L. ; Wiedemann, U. ; Schlegel, B. ; Pires, J. R. ; Beyermann, M. ; Schmieder, P. ; Krause, G. ; Volkmer-Engert, R. ; Schneider-Mergener, J. ; Oschkinat, H.: WW domain sequence activity relationships identified using ligand recognition propensities of 42 WW domains.

[6] Wikipedia: *Levenshtein-Distanz — Wikipedia, Die freie Enzyklopädie.* `http://de.wikipedia.org/w/index.php?title=Levenshtein-Distanz&oldid=53443013`. Version: 2008. – [Online; Stand 16. Januar 2009]

[7] Wikipedia: *CAPTCHA — Wikipedia, Die freie Enzyklopädie.* `http://de.wikipedia.org/w/index.php?title=CAPTCHA&oldid=55072151`. Version: 2009. – [Online; Stand 17. Januar 2009]

Anhang A

Umgebung

A.1 Server

Der genutzte Webspace läuft auf einem entfernten Server mit Root-Zugang auf eine virtuelle Maschine. Virtualisierung ermöglicht es auf einem einzelnen Computer/Server mehrere (virtuelle) Computer/Server zu emulieren. So können mehrere Benutzer einen *Root-Zugang* zu scheinbar eigenständigen Servern haben, die physikalisch jedoch auf ein und derselben Hardware liegen.

A.1.1 Zugangsdaten

Server	`www.coniubix.com`
Erreichbarkeit	`seephi.coniubix.com`
FTP-Login	`seephiadmin`
FTP-Passwort	`seephi&pw16`

Tabelle A.1: Die Tabelle zeigt die Zugangsdaten des genutzten Webservers.

A.1.2 Technische Daten

Betriebssystem	`SUSE Linux 10.3`
HTTP-Server	`Apache 2`
PHP	`PHP 5`
Datenbank	`MySQL`

Tabelle A.2: Die Tabelle zeigt die technischen Daten des genutzten Webserver.

A.2 E-Mail-Konten

Für den Schriftwechsel mit Benutzern, sowie dem automatischen Versenden von Registrierungsbenachrichtigungen, wurde ein E-Mail-Konto eingerichtet, erreichbar unter `seephi@coniubix.com`.

Zugangsdaten

Login	`seephi@coniubix.com`
Passwort	`seepmail&pw29`

Tabelle A.3: Die Tabelle zeigt die Zugangsdaten des genutzten E-Mail-Kontos.

Zusätzlich wurde ein E-Mail-Konto auf dem Server der FH-Bingen eingerichtet, um den akademischen Hintergrund zu verdeutlichen: `seephi@fh-bingen.de`.

Anhang B

Übersetzungsklasse

B.1 Klasse „translator.class.php“

```
1  <?php
2  class translator {
3      var $mainObj;
4      var $title = "";
5      var $motif = "";
6      var $mode = "s2p";
7      var $login = "";
8
9      var $oFilename;
10
11     var $messages = array("error" => array(), # This is a two-dimensional array
           : $messages["warn"][0].
12                           "warn" => array(),
13                           "select" => array());
14
15     var $translated = array();
16
17
18     function translator($mainObj) {
19         $this->mainObj = $mainObj;
20
21         $this->check_parameters();
22
23         if($this->messages["error"])
24             return 0;
25
26         if($this->mode == "s2p")
27             $this->s2p();
```

```
28          else $this->p2s();
29      }
30
31
32
33      function p2s() {
34          include_once $this->mainObj->incPath."p2s.inc.php";
35          $this->motif = clean_pattern($this->motif);
36          $this->translated = detect_pattern($this->title, $this->motif, $this->
                mode, $this->login, $this->messages);
37      }
38
39
40
41      function s2p() {
42          include_once $this->mainObj->incPath."s2p.inc.php";
43          $this->translated = detect_pattern($this->motif);
44      }
45
46
47
48      function print_translated() {
49          if(!empty($this->translated) and empty($this->messages["error"]) and
                empty($this->messages["select"]))
50              print_translated($this->translated);
51      }
52
53
54
55      function print_motifpositions() {
56          if(empty($this->messages["error"]))
57              print_motifpositions($this->motif);
58      }
59
60
61
62      function print_translatepositions() {
63          if(empty($this->messages["error"]) and empty($this->messages["select"])
                ) {
64              if($this->mode == "p2s")
65                  print_seefeldpositions($this->translated);
66              else print_prositepositions($this->translated);
67          }
68      }
```

```
69
70
71
72      function print_errors() {
73          if(!empty($this->messages["error"]))
74              echo implode("\n", $this->messages["error"]);
75      }
76
77
78
79      function print_warnings() {
80          if(!empty($this->messages["warn"]))
81              echo implode("\n", $this->messages["warn"]);
82      }
83
84
85
86      function print_selects() {
87          if(!empty($this->messages["select"]))
88              echo implode("\n", $this->messages["select"]);
89      }
90
91
92
93      function print_convform() {
94          if(!empty($this->messages["error"]) or !empty($this->messages["warn"])
                or !empty($this->messages["select"]))
95              include_once $this->mainObj->incPath."translate.form.inc.php";
96      }
97
98
99
100     function print_save2fileform() {
101         if(!empty($this->translated) and empty($this->messages["error"]) and
                empty($this->messages["select"]) and $this->mainObj->valid_login())
102             include_once $this->mainObj->incPath."save2file.form.inc.php";
103     }
104
105
106
107     function print_send2ncbiform() {
108         if(!empty($this->translated) and empty($this->messages["error"]) and
                empty($this->messages["select"]) and $this->mainObj->valid_login()
                and $this->mode == "s2p")
```

```
109             include_once $this->mainObj->incPath."send2ncbi.form.inc.php";
110     }
111
112
113
114     function check_parameters() {
115         if(isset($_GET["title"]) and !empty($_GET["title"]))
116             $this->title = trim($_GET["title"]);
117         else array_push($this->messages["error"], "<div class=\"error\">ERROR:
                Please set a title for your conversion.</div>");
118         if(isset($_GET["motif"]) and !empty($_GET["motif"]))
119             $this->motif = preg_replace("/\s/", "", $_GET["motif"]);
120         else array_push($this->messages["error"], "<div class=\"error\">ERROR:
                Please set a motif for your conversion.</div>");
121         if(isset($_GET["mode"]) and !empty($_GET["mode"]))
122             $this->mode = $_GET["mode"];
123         if(isset($_GET["output"]) and !empty($_GET["output"]))
124             $this->output = $_GET["output"];
125 //          if(isset($_GET["login"]) and !empty($_GET["login"]))
126 //              $this->login = $_GET["login"];
127         if(isset($_GET["saveas"]) and !empty($_GET["saveas"]))
128             $this->saveas = $_GET["saveas"];
129         if(isset($_GET["oformat"]) and !empty($_GET["oformat"]))
130             $this->oFormat = $_GET["oformat"];
131         if(isset($_GET["ofilename"]) and !empty($_GET["ofilename"]))
132             $this->oFilename = $_GET["ofilename"];
133         else $this->oFilename = $this->title;
134     }
135
136
137
138     function save2file() {
139         $downloadFiles = array();
140         if(isset($this->saveas) and !empty($this->translated) and empty($this->
                messages["error"]) and empty($this->messages["select"]) and $this->
                mainObj->valid_login()) {
141             for($i=0; $i<2; $i++) {
142                 if(isset($this->oFormat[$i]) and $this->oFormat[$i] == "ascii")
143                     array_push($downloadFiles, saveas_ascii($this->title, $this
                            ->motif, $this->translated, "/srv/www/vhosts/coniubix.
                            com/subdomains/seephi/httpdocs/translate/odata/".
                            $_SESSION["userid"]."/", $this->oFilename));
144                 if(isset($this->oFormat[$i]) and $this->oFormat[$i] == "latex")
```

```
145                     array_push($downloadFiles, saveas_latex($this->title, $this
                            ->motif, $this->translated, "/srv/www/vhosts/coniubix.
                            com/subdomains/seephi/httpdocs/translate/odata/".
                            $_SESSION["userid"]."/", $this->oFilename));
146 //                  if(isset($this->oFormat[$i]) and $this->oFormat[$i] == "pdf")
147 //                      array_push($downloadFiles, saveas_pdf($this->title, $this
    ->motif, $this->translated, "/srv/www/vhosts/coniubix.com/subdomains/seephi
    /httpdocs/translate/odata/".$_SESSION["userid"]."/", $this->oFilename));
148                 }
149             }
150
151             if(isset($downloadFiles) and !empty($downloadFiles)) {
152                 echo "<h2>Your files...</h2>\n";
153                 foreach($downloadFiles as $currFile) {
154                     $fileLink = str_replace('/srv/www/vhosts/coniubix.com/
                            subdomains/seephi/httpdocs/', $this->mainObj->url,
                            $currFile);
155                     echo "<a href=\"$fileLink\">".basename($currFile)."</a><br />\n
                            ";
156                 }
157                 echo "<br /> <hr />\n";
158             }
159         }
160 }
161 ?>
```

Anhang C

Datenbank

C.1 Datenbank-Server

Datenbank-Server	*MySQL 5.0.45*
Server	Localhost via UNIX socket (*SUSE Linux 10.3*)
MySQL-Zeichensatz	`UTF-8 Unicode (utf8)`
SeePHI-DB-Login	`seephiadmin`
SeePHI-DB-Passwort	`KRz&b1#`

Tabelle C.1: Die Tabelle zeigt die technischen Daten des *MySQL-Datenbankservers*.

C.2 SQL-Code

C.2.1 Erstellen der Datenbank

```
1 CREATE DATABASE `seephi` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
2 USE `seephi`;
```

C.2.2 Erstellen der Tabelle „clicklog“

```
1 CREATE TABLE `clicklog` (
2   `click_id` int(10) unsigned NOT NULL auto_increment COMMENT 'ID of the click'
        ,
3   `click_path` tinytext collate utf8_unicode_ci NOT NULL COMMENT 'Path that
        user visited',
4   `click_user` smallint(5) unsigned NOT NULL default '0' COMMENT 'ID of the
        user (if login)',
5   `click_ip` varchar(15) collate utf8_unicode_ci NOT NULL default '
        000.000.000.000' COMMENT 'IP of the User',
6   `click_datetime` datetime NOT NULL default '0000-00-00 00:00:00' COMMENT '
        Datetime of visiting this site',
7   `click_parsetime` float unsigned default NULL COMMENT 'Parsing time of the
        PHP-Script',
8   PRIMARY KEY  (`click_id`)
9 ) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    COMMENT='Contains all clicks of the website (for statistics)'
    AUTO_INCREMENT=1 ;
```

C.2.3 Erstellen der Tabelle „metadata“

```
CREATE TABLE `metadata` (
  `metaid` tinyint(3) unsigned NOT NULL auto_increment COMMENT 'Metadata Index (Primary Key); Enable multiple Meta-Strings to one Keyword',
  `metakey` tinytext collate utf8_unicode_ci NOT NULL COMMENT 'Meta-Keyword',
  `metastring` mediumtext collate utf8_unicode_ci NOT NULL COMMENT 'Meta-String',
  PRIMARY KEY  (`metaid`)
) ENGINE=MyISAM AUTO_INCREMENT=6 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci COMMENT='Contains the Metadata of the Project' AUTO_INCREMENT=6 ;


INSERT INTO `metadata` (`metaid`, `metakey`, `metastring`) VALUES (1, 'name="autor"', 'content="Thomas Schmidt"'),
(2, 'name="description"', 'content="SeePHI - Converter for Regular Expressions (Seefeld-Convention to PROSITE/PHI-BLAST and back)"'),
(3, 'name="keywords"', 'content="Seefeld, Convention, PROSITE, PHI-BLAST, Converter, convert, Regular, Expressions, regex, fh-bingen, bioinformatics"'),
(4, 'name="robots"', 'content="index, follow"'),
(5, 'http-equiv="Content-Style-Type"', 'content="text/css"');
```

C.2.4 Erstellen der Tabelle „register“

```
CREATE TABLE `register` (
  `reg_id` smallint(5) unsigned NOT NULL auto_increment COMMENT 'ID of the registration procedure',
  `reg_email` varchar(80) collate utf8_unicode_ci NOT NULL default '' COMMENT 'Registration E-Mail',
  `reg_password` varchar(32) collate utf8_unicode_ci NOT NULL default '' COMMENT 'Temp. Registration Password',
  `reg_confirmkey` varchar(14) collate utf8_unicode_ci NOT NULL default '' COMMENT 'Temp. Registration Key',
  PRIMARY KEY  (`reg_id`)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci COMMENT='Contains all registration procedures of the website' AUTO_INCREMENT=1;
```

C.2.5 Erstellen der Tabelle „users“

```
CREATE TABLE `users` (
  `usr_id` smallint(5) unsigned NOT NULL auto_increment,
  `usr_name` varchar(40) character set utf8 collate utf8_unicode_ci NOT NULL
      default '',
  `usr_pass` varchar(32) NOT NULL default '',
  `usr_firstlogin` datetime NOT NULL default '0000-00-00 00:00:00',
  `usr_lastlogin` datetime NOT NULL default '0000-00-00 00:00:00',
  `usr_numlogins` smallint(5) unsigned NOT NULL default '0',
  `usr_email` varchar(60) NOT NULL default '',
  `usr_regemail` varchar(60) NOT NULL default '',
  PRIMARY KEY  (`usr_id`)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```