# Appendix A

# Vector map

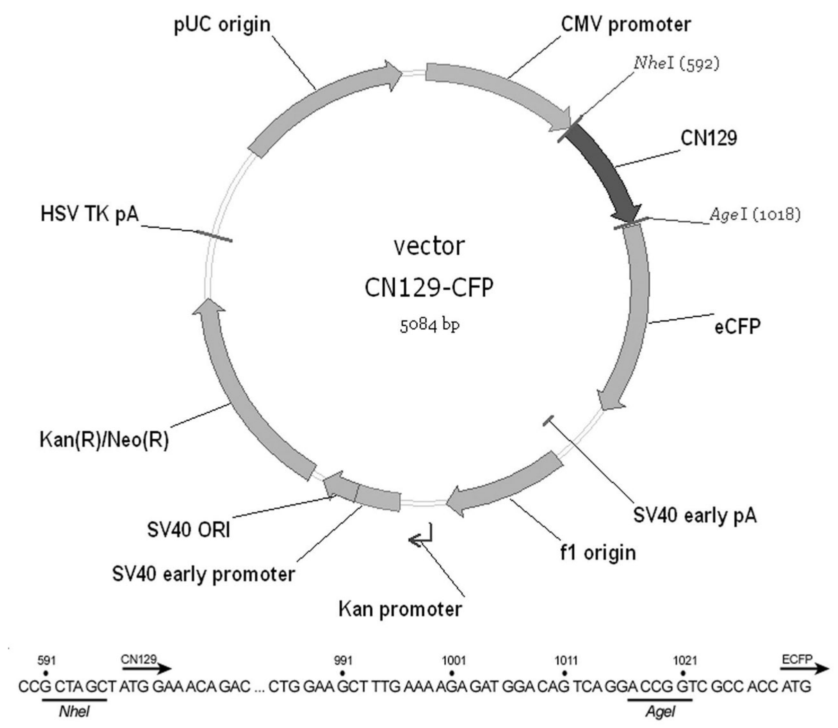

Figure A.1: Vector map of human full length CN129-CFP. Depicted is the vector map and a partial sequence. The sequence illustrates the utilised restriction sites for cloning and the flanking regions of the vector pECFP-N1 and the CN129 insert. The start codons of CN129, eCFP and the positions within the vector sequence are indicated. The partial rat and human CN129-CFP constructs and the mutagenised version thereof follow this scheme, for details see Materials and Experimental procedures.

# Appendix B

# Program source codes

## Source code for the program retrieving peptide sequences from swissprot.

```
#
# Input : file via command line attribute
# ( IDs obtained from scansite , plain list )
# Output : index , SP—ID , description , peptide sequence ( s ) as csv
#
use Bio::DB::SwissProt; # loads bioperl modules
# =======
# main
# =======
# Output filename ( putatively > 1 sequence / protein )
my $dateiname="Sequences_raw.csv";
my @ID_List;
my $ID ="";
my $motif =
  "........[AVLISE]..[AVLIF][AVLI]..[AVLI][AVLIF]..[AVILSE]....";
my $ptl = 25; # peptide— length
# replacement for separator from db— entries
# ( avoid conflict with cvs - format )
my $sep=";"; # separator ( cvs - format )
my $rsep=","; # replacement
my $tseq;
my $seq;
my @outp;
@ID_List = <>;
open(AUS,">$dateiname");
# generate caption for output file
print AUS "No".$sep."ID".$sep."Descr".$sep."RII-BDs\n";
my $database = new Bio::DB::SwissProt;
my $tseq = "";
my $j = 0;
my $i = 1;
foreach (@ID_List){ # parse ID list
  $ID = $ID_List[$j];
  @mdbe = mtf ($ID); # call mtf to fetch sequence from db
  $mdbe[1] =~s/$sep/$rsep/ig; # replace csv- separator if found
  # generate line in output file
  print AUS (1+$j).$sep.join ($sep, @mdbe)."\n";
  print (1+$j)."␣".join ($sep, @mdbe)."\n"; # display line
  ++$j;
}
close (AUS);
# =======
# mtf
# =======
sub mtf{
  chomp($ID=$_[0]);
  print "$ID\n"; # display actual ID in console window ( stdout )
  # get sequence from db by ID ( e.g . ' CN129_HUMAN ')
  $dbe = $database->get_Seq_by_id($ID);
  $seq = $dbe->seq();
  $dc = $dbe->desc();
  @mv = $seq =~ m/$motif/ig; # find motifs within sequence
  my $epos = (pos $seq) - 1;
  my $spos = $epos - $ptl+1;
  $outp[0] = $ID; # generate & display output
  print "$outp[0]\n";
  $outp[1] = $dc;
  print "$outp[1]\n";
  $outp[2] = join($sep, @mv);
  print "$outp[2]\n";
  return @outp;
}
```

**Source code for the program filtering difficult sequences.**

```
#
# filter for peptide sequences ( difficult and undesired sequences )
# input : sequence file in csv — format
# ( fields : No ; ID ; Description ; RII —BD Sequence )
# output : two sequence files in csv — format
# ( passed and excluded sequences )
#
# =======
# main
# =======
undef $/;
my $dateiIn = "Sequences.csv"; # Input filename
my $dateiOut = "Excluded_".$dateiIn; # Output filename1 ( excluded )
my $dateiOut2 = "Passed_".$dateiIn; # Output filename2 ( passed seqs .)
my $temp;
my @file1Content;
my @file2Content;
my @file3Content;
my @tCont1; # temporary content
my $sep=";"; # separator of fields ( csv )
my $comment =""; # comment, reason of filtering
open(IN,"<␣$dateiIn") or
  die "\ncouldn't␣open␣file␣of␣known␣sequences:␣$dateiIn!!!\n\n";
$temp = <IN>;
close (IN);
@file1Content = split("\n", $temp); # split input file in lines
my $lines= @file1Content;
$file2Content[0]= "Excluded-list;\n"; # generate caption of output file
my $ii=0;
my $iii=0;
for (my $i=0; $i<($lines);$i++){ # parse input file ( lines )
  @tCont1= split($sep,$file1Content[$i]);# split lines in fields
  if (pfilter($tCont1[3])>0){ # apply filter to sequence ( last field )
      # if filtered => output ( excluded ) & comment
      $file2Content[$ii]= $file1Content[$i].$sep.$comment."\n";
      $ii++;
  }
  else {
    # if passed => output ( passed )
    $file3Content[$iii]= $file1Content[$i]."\n";
    $iii++;
  }
} # save output files :
open(AUS,">␣$dateiOut") or
  die "\ncouldn't␣generate␣outputfile:␣$dateiOut!!!\n\n";
print AUS @file2Content;
```

```
close (AUS);
open(AUS,">␣$dateiOut2") or
  die "\ncouldn’t␣generate␣outputfile:␣$dateiOut2!!!\n\n";
print AUS @file3Content;
close (AUS);
# ==========================
#    filter   and   filter    conditions
# ==========================
sub pfilter {
  my $tseq = $_[0];
  my $fseq = 0;
  my $flg = 0;
  my @seqarr;
  my @valarr;
  my $ratio=0;
  my $nAA = 6;
  my $cval= 1.6; # critical value => upper border
  my $c2val= 0.4; # critical value 2 => lower border
  my @outp;
  #
  # -- Aggregation values according to Krchnak and Vagner --
  #
  %AA = ("A" => "1.34", # map values to amino acids
  "C" => "1.09",
  "D" => "0.63",
  "E" => "1.10",
  "F" => "1.07",
  "G" => "0.81",
  "H" => "0.64",
  "I" => "1.58",
  "K" => "1.31",
  "L" => "1.20",
  "M" => "1.15",
  "N" => "0.97",
  "P" => "0.26",
  "Q" => "0.79",
  "R" => "0.46",
  "S" => "0.69",
  "T" => "1.15",
  "V" => "1.77",
  "W" => "1.01",
  "Y" => "1.12",
   );
  @seqarr = split (//, $tseq); # split sequence to single amino acids
  $comment="";
  $back = 0;
  # -----------------
  #    filter    cysteins
  # -----------------
  $fseq = $tseq =~ m/c+/i;
  if ($fseq){
    $flg++;
    $comment = "one␣or␣more␣cysteins";
   }
  # -----------------
  #    filter  >=5  Ala
  # -----------------
  if (!$flg){
    $fseq = $tseq =~ m/AAAAA/i;
    if ($fseq){
      $flg++;
      $comment = "5␣or␣more␣Ala␣in␣row";
    }
  }
  # ------------------
  #    filter   >=5  Glu
  # ------------------
  if (!$flg){
    $fseq = $tseq =~ m/EEEEE/i;
    if ($fseq){
      $flg++;
      $comment = "5␣or␣more␣Glu␣in␣row";
    }
  }
  # ------------------
  #    filter   >=5  Leu
  # ------------------
  if (!$flg){
    $fseq = $tseq =~ m/LLLLL/i;
    if ($fseq){
      $flg++;
      $comment = "5␣or␣more␣Leu␣in␣row";
    }
  }
  # ---------------------
  #  filter   turn   building   AA  1
  # ---------------------
  if (!$flg){
    $fseq = $tseq =~ m/.DG./i;
    if ($fseq){
      $flg++;
      $comment = "warning:␣turn␣building␣pattern␣X-D-G-X";
    }
```

```
}
# --------------------
#  filter   turn   building   AA 2
# --------------------
if (!$flg){
  $fseq = $tseq =~ m/.DS./i;
  if ($fseq){
    $flg++;
    $comment = "warning:␣turn␣building␣pattern␣X-D-S-X";
  }
}
# --------------------
#  filter   difficult   sequences
# --------------------
if (!$flg){
  my $t =0;
  foreach (@seqarr){
    # add  mapped  aa— value  to  end  of  value — array
    push @valarr, $AA{$seqarr[$t]};
    push @outp, $seqarr[$t]; # add aa to  end  of  output — array
    if ($t > ($nAA-1)) { # if "window  is  full  " (+1  element )
      my $trash= shift @valarr; # discard  first  element  of  value — array
      # discard  first  element  of  output — array
      $trash= shift @outp;
    }
    if ($t >= ($nAA-1)){ # since window  is  first  time  "  filled  "
        $ratio = sumarr(@valarr)/$nAA; # calculate mean  via  " summarr "
       if ( $ratio > $cval) { # if higher  than  upper  border  =>  exclude
         $flg++;
         $comment = "diff.␣seq.␣-␣ratio␣(>):␣$ratio␣==>␣";
         $comment = $comment."␣".(join ("", @outp));
        }
        if ( $ratio < $c2val) { # if lower than lower border => exclude
          $flg++;
          $comment = "diff.␣seq.␣-␣ratio␣(<)␣:␣$ratio␣==>␣";
          $comment = $comment."␣".(join ("", @outp));
        }
      }
      $t++;
    }
  }
  # --------------------------------
  # if anything filtered => return is true
  # --------------------------------
  if ($flg){
   $back = 1;
  }
  return $back;
}
# -------
# summarr
# -------
sub sumarr {
  my @temp = @_;
  my $total=0;
  my $tt =0;
  foreach (@temp) {
    $total = $total + $temp[$tt];
    ++$tt;
  }
  return $total;
}
```