Prince of Parsea

OH HAI

I'm masak

Part I: Saving the princess

'Prince of Persia' by Brøderbund (1989)

classic platform game

complete the levels, get the girl

(princess not in another castle)

evil vizier

PoP traps

spikes

falling blocks

those jaw thingies

increasingly savvy guards

time

P6Regex traps

literals

quantifiers

subrules

lookarounds

charclasses

anchors

alternations

conjunctions

concat

main analogy

user input -> levels -> WIN/LOSE

user input -> regex -> MATCH/FAIL

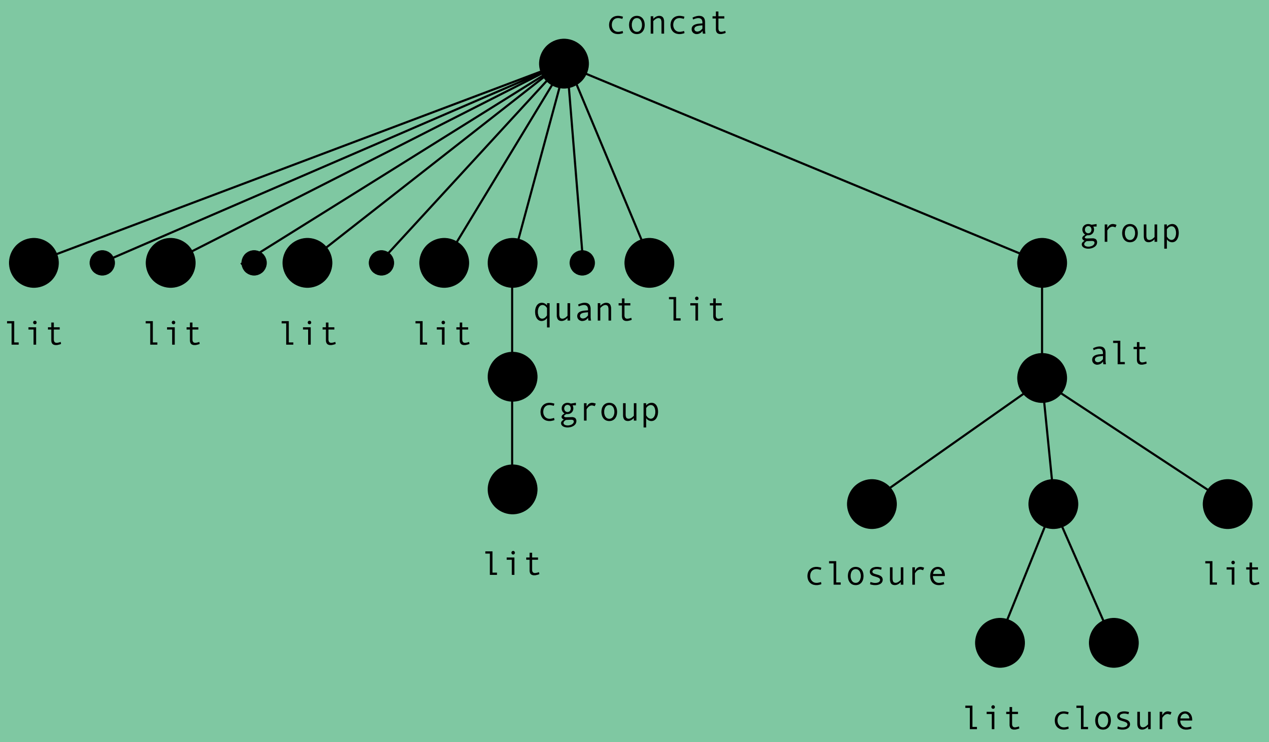
```
mm/ The quick brown fox(es)? jump[<?{$0}>|s<!{$0}>|ed] /
```

```
mm/ The quick brown fox(es)? jump[<?{$0}>|s<!{$0}>|ed] /
```

```
Matches:  The quick brown fox jumps
          The quick brown fox jumped
          The quick brown foxes jump
          The quick brown foxes jumped

Fails on: The quick brown fox jump
          The quick brown foxes jumps
```

```
mm/ The quick brown fox(es)? jump[<?{$0}>|s<!{$0}>|ed] /
```



```
mm/ The quick brown fox(es)? jump[<?{$0}>|s<!{$0}>|ed] /
```

a regex is a level in a game

a grammar is a game

.kv

Part II: Sublanguages

When someone says
"I want a programming language in which
I need only say what I wish done,"
give him a lollipop.

Alan Perlis, 1982

generalist langs/specialist langs

large langs/small langs

"little languages"

"DSLs"

many

SQL

XPath

Graphviz

Haskell's 'do' notation

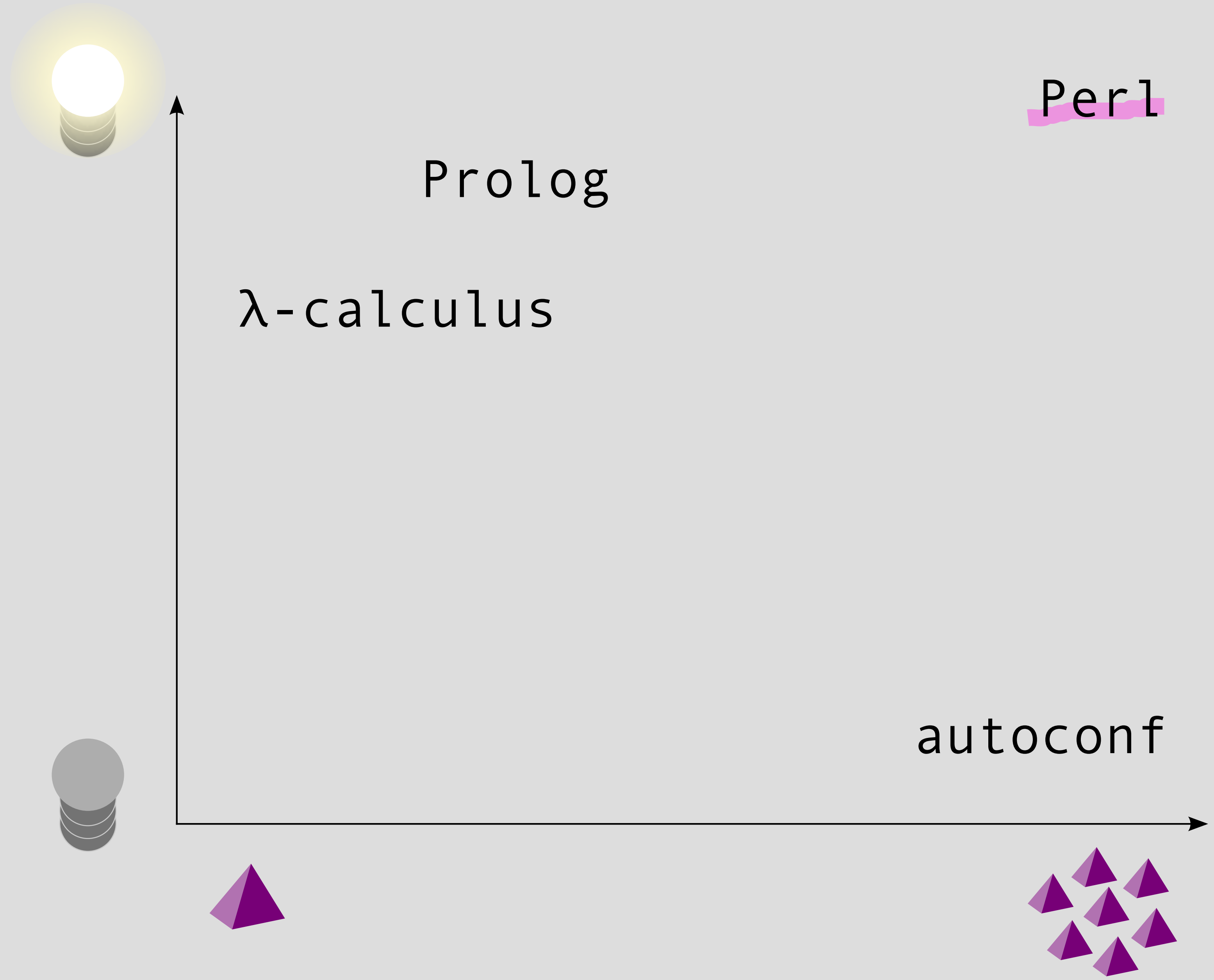
I need only say what I wish done

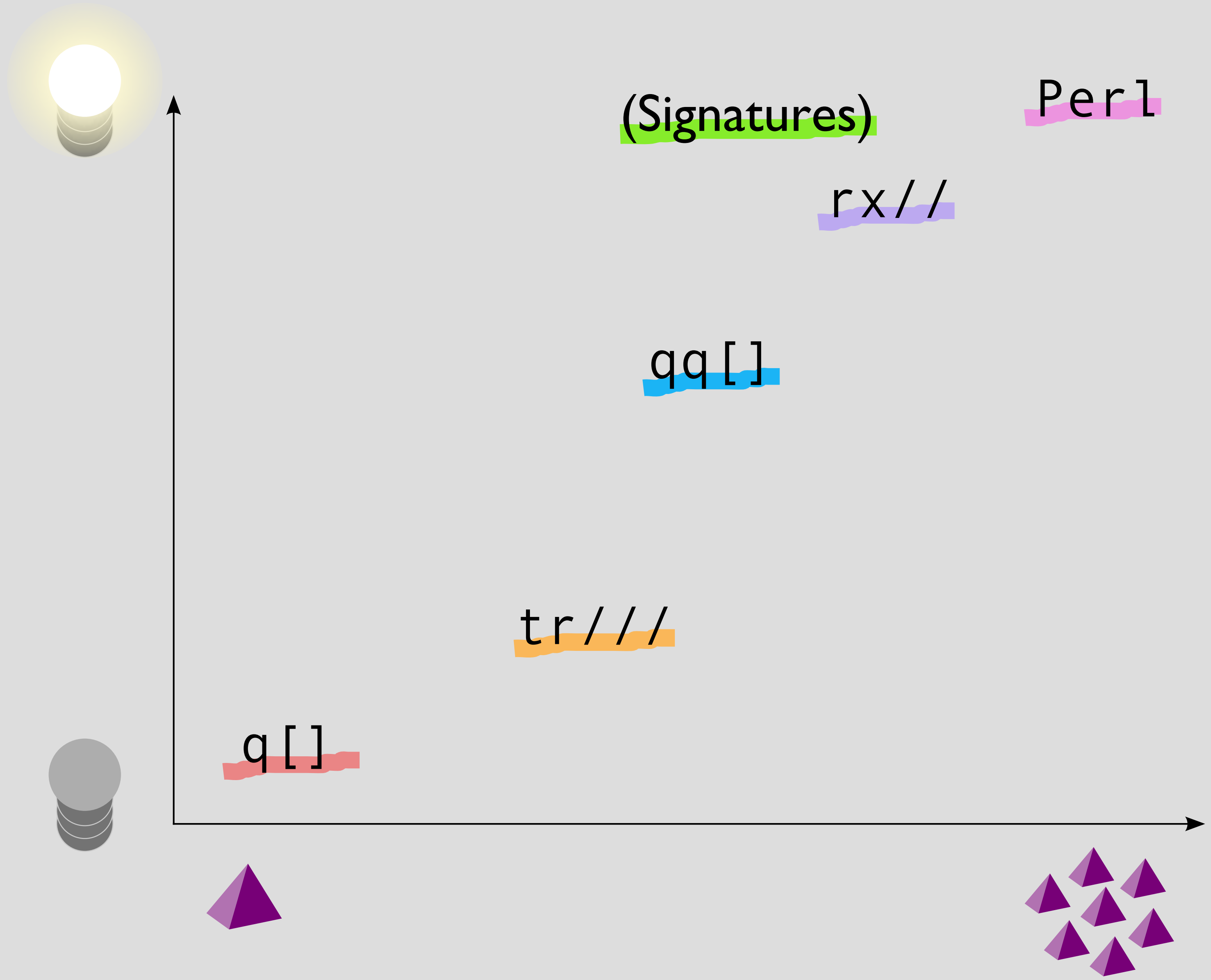
Embrace your inner lollipop

Perl helps you do that

Perl is full of DSLs already

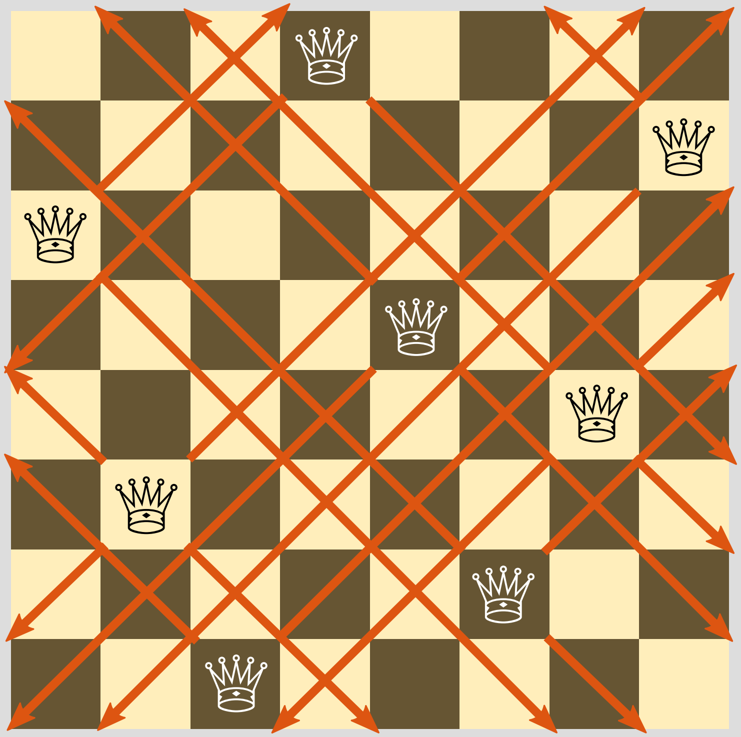
(Perl 6 formalizes them)

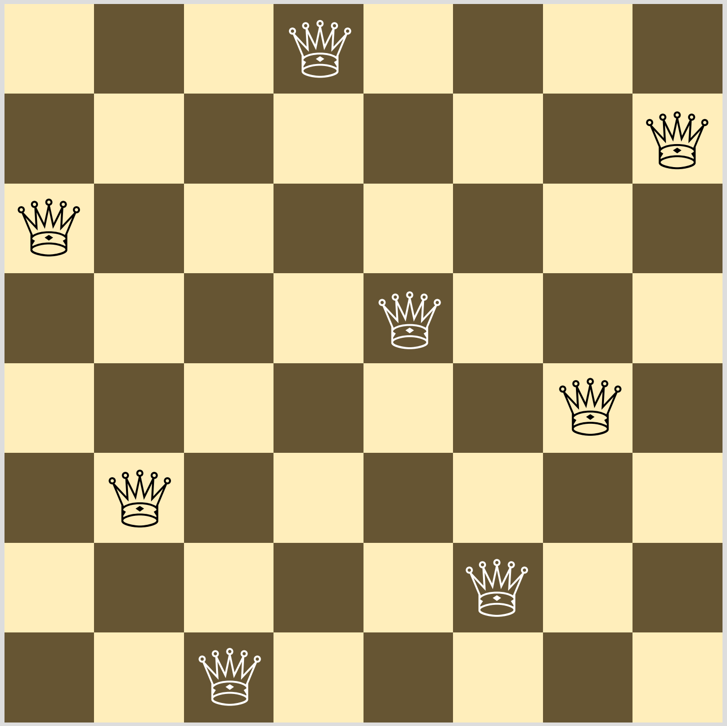

Perl
Prolog
λ-calculus
autoconf


(Signatures)
Perl
rx//
qq[]
tr///
q[]

8-queens

vizier

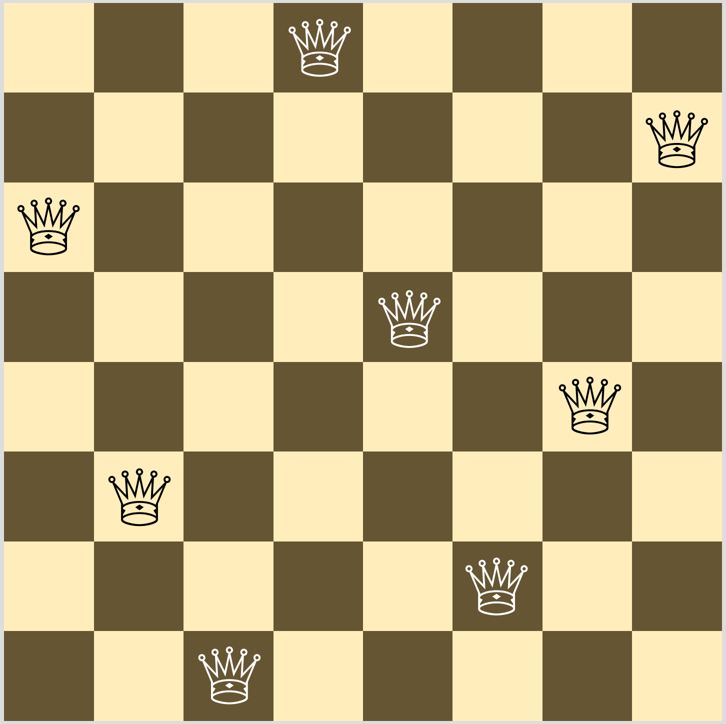




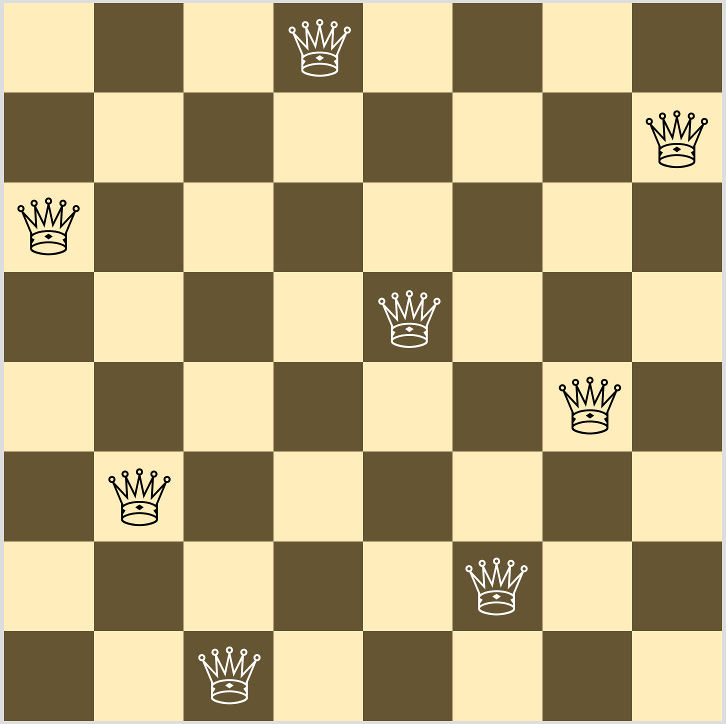
```
for 1..8 -> $pos1 {
  for 1..8 -> $pos2 {
    for 1..8 -> $pos3 {
      for 1..8 -> $pos4 {
        for 1..8 -> $pos5 {
          for 1..8 -> $pos6 {
            for 1..8 -> $pos7 {
              for 1..8 -> $pos8 {
                # print solution
              }
            }
          }
        }
      }
    }
  }
}
```

longing for a "meta-for"

but it gets worse!



```
for 1..8 -> $pos1 {
  for 1..8 -> $pos2 {
    next if $pos2     == $pos1;
    for 1..8 -> $pos3 {
      next if $pos3     == $pos1;
      next if $pos3     == $pos2;
      for 1..8 -> $pos4 {
        next if $pos4     == $pos1;
        next if $pos4     == $pos2;
        next if $pos4     == $pos3;
        for 1..8 -> $pos5 {
          next if $pos5     == $pos1;
          next if $pos5     == $pos2;
          next if $pos5     == $pos3;
          next if $pos5     == $pos4;
          for 1..8 -> $pos6 {
            next if $pos6     == $pos1;
            next if $pos6     == $pos2;
            next if $pos6     == $pos3;
            next if $pos6     == $pos4;
            next if $pos6     == $pos5;
            for 1..8 -> $pos7 {
              next if $pos7     == $pos1;
              next if $pos7     == $pos2;
              next if $pos7     == $pos3;
              next if $pos7     == $pos4;
              next if $pos7     == $pos5;
              next if $pos7     == $pos6;
              for 1..8 -> $pos8 {
                next if $pos8     == $pos1;
                next if $pos8     == $pos2;
                next if $pos8     == $pos3;
                next if $pos8     == $pos4;
                next if $pos8     == $pos5;
                next if $pos8     == $pos6;
                next if $pos8     == $pos7;
                # print solution
              }
            }
          }
        }
      }
    }
  }
}
```
]



```
for 1..8 -> $pos1 {
  for 1..8 -> $pos2 {
    next if $pos2     == $pos1;
    next if $pos2 - 1 == $pos1;
    next if $pos2 + 1 == $pos1;
    for 1..8 -> $pos3 {
      next if $pos3     == $pos1;
      next if $pos3 - 2 == $pos1;
      next if $pos3 + 2 == $pos1;
      next if $pos3     == $pos2;
      next if $pos3 - 1 == $pos2;
      next if $pos3 + 1 == $pos2;
      for 1..8 -> $pos4 {
        next if $pos4     == $pos1;
        next if $pos4 - 3 == $pos1;
        next if $pos4 + 3 == $pos1;
        next if $pos4     == $pos2;
        next if $pos4 - 2 == $pos2;
        next if $pos4 + 2 == $pos2;
        next if $pos4     == $pos3;
        next if $pos4 - 1 == $pos3;
        next if $pos4 + 1 == $pos3;
        for 1..8 -> $pos5 {
          next if $pos5     == $pos1;
          next if $pos5 - 4 == $pos1;
          next if $pos5 + 4 == $pos1;
          next if $pos5     == $pos2;
          next if $pos5 - 3 == $pos2;
          next if $pos5 + 3 == $pos2;
          next if $pos5     == $pos3;
          next if $pos5 - 2 == $pos3;
          next if $pos5 + 2 == $pos3;
          next if $pos5     == $pos4;
          next if $pos5 - 1 == $pos4;
          next if $pos5 + 1 == $pos4;
          for 1..8 -> $pos6 {
            next if $pos6     == $pos1;
            next if $pos6 - 5 == $pos1;
            next if $pos6 + 5 == $pos1;
            next if $pos6     == $pos2;
            next if $pos6 - 4 == $pos2;
            next if $pos6 + 4 == $pos2;
            next if $pos6     == $pos3;
            next if $pos6 - 3 == $pos3;
            next if $pos6 + 3 == $pos3;
            next if $pos6     == $pos4;
            next if $pos6 - 2 == $pos4;
            next if $pos6 + 2 == $pos4;
            next if $pos6     == $pos5;
            next if $pos6 - 1 == $pos5;
            next if $pos6 + 1 == $pos5;
            for 1..8 -> $pos7 {
              next if $pos7     == $pos1;
              next if $pos7 - 6 == $pos1;
              next if $pos7 + 6 == $pos1;
              next if $pos7     == $pos2;
              next if $pos7 - 5 == $pos2;
              next if $pos7 + 5 == $pos2;
              next if $pos7     == $pos3;
              next if $pos7 - 4 == $pos3;
              next if $pos7 + 4 == $pos3;
              next if $pos7     == $pos4;
              next if $pos7 - 3 == $pos4;
              next if $pos7 + 3 == $pos4;
              next if $pos7     == $pos5;
              next if $pos7 - 2 == $pos5;
              next if $pos7 + 2 == $pos5;
              next if $pos7     == $pos6;
              next if $pos7 - 1 == $pos6;
              next if $pos7 + 1 == $pos6;
              for 1..8 -> $pos8 {
                next if $pos8     == $pos1;
                next if $pos8 - 7 == $pos1;
                next if $pos8 + 7 == $pos1;
                next if $pos8     == $pos2;
                next if $pos8 - 6 == $pos2;
                next if $pos8 + 6 == $pos2;
                next if $pos8     == $pos3;
                next if $pos8 - 5 == $pos3;
                next if $pos8 + 5 == $pos3;
                next if $pos8     == $pos4;
                next if $pos8 - 4 == $pos4;
                next if $pos8 + 4 == $pos4;
                next if $pos8     == $pos5;
                next if $pos8 - 3 == $pos5;
                next if $pos8 + 3 == $pos5;
                next if $pos8     == $pos6;
                next if $pos8 - 2 == $pos6;
                next if $pos8 + 2 == $pos6;
                next if $pos8     == $pos7;
                next if $pos8 - 1 == $pos7;
                next if $pos8 + 1 == $pos7;
                # rint solutions
              }
            }
          }
        }
      }
    }
  }
}
```

recursion

```
use strict;

# The classical 8 queens puzzle
# Place 8 queens on a chess board without any of them threatening each other
# Returns true if the first argument equals any of the subsequent ones,
# otherwise returns false
sub any_equals {
   my $value = shift;

   while (my $other_value = shift) {
       return 1 if $value == $other_value;
   }

   return &apos;&apos;;
}

# Returns true if the first argument differs (absolutely) by one from the
# second, or by two from the third, or... and so on, otherwise returns
# false
sub any_aligns {
   my $value = shift;
   my $difference = 0;

   while (my $other_value = shift) {
       ++$difference;
       return 1 if abs($value - $other_value) == $difference;
   }

   return &apos;&apos;;
}

sub generate_solutions {
   my $levels_left = shift;
   my @values_so_far = @_;

   for my $column (1..8) {
       next if any_equals($column, @values_so_far);
       next if any_aligns($column, @values_so_far);

       if ($levels_left > 1) {
           generate_solutions($levels_left - 1, $column, @values_so_far);
       }
       else {
           print join ' ', ($column, @values_so_far);
           print "\\n";
       }
   }
}

generate_solutions(8);
```

```
sub g{my$l=pop;for$c(1..8){my$d;grep++$d==abs$c-$_|$c==$_,@_
or$l<7&!g($c,@_,$l+1)||print "@{[$c,@_]}\\n"}}g
```

my sublanguage

```
my 1..8 $pos1;
my 1..8 $pos2;
my 1..8 $pos3;
my 1..8 $pos4;
my 1..8 $pos5;
my 1..8 $pos6;
my 1..8 $pos7;
my 1..8 $pos8;
my 1..8 $pos1;
my 1..8 $pos2 where {    $pos2     != $pos1
                      && $pos2 - 1 != $pos1
                      && $pos2 + 1 != $pos1 };
my 1..8 $pos3 where {    $pos3     != $pos1
                      && $pos3 - 2 != $pos1
                      && $pos3 + 2 != $pos2
                      && $pos3     != $pos2
                      && $pos3 - 1 != $pos2
                      && $pos3 + 1 != $pos2 };
# ...
```

```
use distinct;

my 1..8 $pos1;
my 1..8 $pos2 where {    $pos2 - 1 != $pos1
                      && $pos2 + 1 != $pos1 };
my 1..8 $pos3 where {    $pos3 - 2 != $pos1
                      && $pos3 + 2 != $pos2
                      && $pos3 - 1 != $pos2
                      && $pos3 + 1 != $pos2 };
# ...
```

```
use distinct;

my 1..8 $1;
my 1..8 $2 where {    $2 - 1 != $1
                   && $2 + 1 != $1 };
my 1..8 $3 where {    $3 - 2 != $1
                   && $3 + 2 != $2
                   && $3 - 1 != $2
                   && $3 + 1 != $2 };
# ...
```

```
use distinct;

my 1..8 $1;
my 1..8 $2 where {
    all map {    $2 - $_ != $[2 - $_]
              && $2 + $_ != $[2 + $_] }, 1 };
my 1..8 $3 where {
    all map {    $3 - $_ != $[3 - $_]
              && $3 + $_ != $[3 + $_] }, 1, 2 };
# ...
```

```
use distinct;

for 1..8 -> $n {
    my 1..8 $[$n] where {
        all map {    $[$n] - $_ != $[$n - $_]
                  && $[$n] + $_ != $[$n + $_] }, 1 };
}
```

I have a module

(it's very slow)

(but it works!)

```
      S E N D
+     M O R E
-----------
    M O N E Y
```

```
my 0..9 $D;
my 0..9 $E where ($N + $R + $!C1 % 10);
my        $Y = ($D + $E) % 10;
my        $!C1 = ($D + $E) div 10;
my 0..9 $N;
my 0..9 $R;
my        $!C2 = ($N + $R + $!C1) div 10;
my 0..9 $O where ($S + $M + $!C3) % 10;
my        $!C3 = ($E + $O + $!C2) div 10;
my 1..9 $S;
my 1..9 $M where $!C4;
my        $!C4 = ($S + $M + $!C3) div 10;
```

Perl 6's KILLER FEATURE

declarative

optimization

exotic control flow

making exotic control flow feel natural

sublanguages

slangs

lollipops

Perl 6 regexes just a *part* of this

How many Prolog programmers does it take to change a lightbulb?

"No."

Part III: Saving yourself

'Prince of Persia: The Sands of Time'

by Ubisoft (2003)

core problem

restarting something finished

call sub

sub returns result

"sub, could you give me more result?"

sub goes "huh?"

solutions

coroutines

Ruby's 'yield'

Perl 6's 'gather'

continuations

call/cc

setjmp/longjmp

streams

STD.pm6 does this

cheating

other core problem

it's all wrong!

regexes aren't regular expressions

regexes are fundamentally corrupt

Thompson engine

controversial

protothreads

except

```
mm/ The quick brown fox(es)? jump[<?{$0}>|s<!{$0}>|ed] /
```

STD.pm6 combines decl/proc

declarative prefix

worlds meet

☺ happy ☺

live demo?

thank you