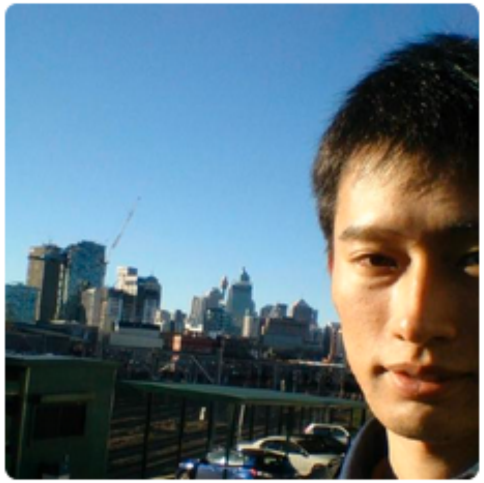
Designing Games of Theorems

Who am I?



Yutaka Ng
yutakang

Block or report user

CVUT, CTU, CIIRC

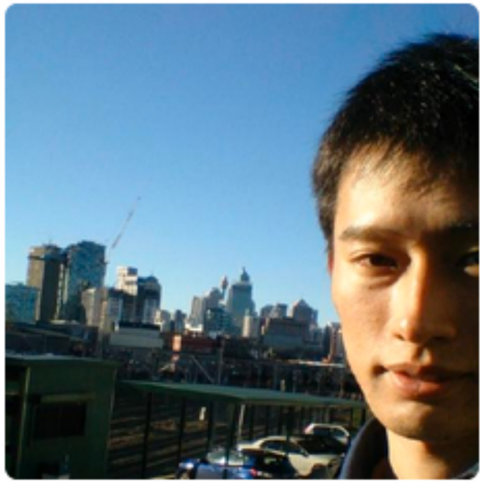
What do I like?

- automating proof search
- in expressive logic (HOL)
- using heuristics / ML

What did I develop?

- PSL/PaMpeR for Isabelle/HOL

Proof Strategy Language (PSL) for Isabelle/HOL



Yutaka Ng
yutakang

Block or report user

CVUT, CTU, CIIRC

meta-tool approach

tactics

quickcheck

sledgehammer

programming language

extensible (Eisbach)

PSL

runtime tactic generation

efficient proof generation

extensive proof search

parallel search

low memory usage

native Isabelle proof script

easy installation

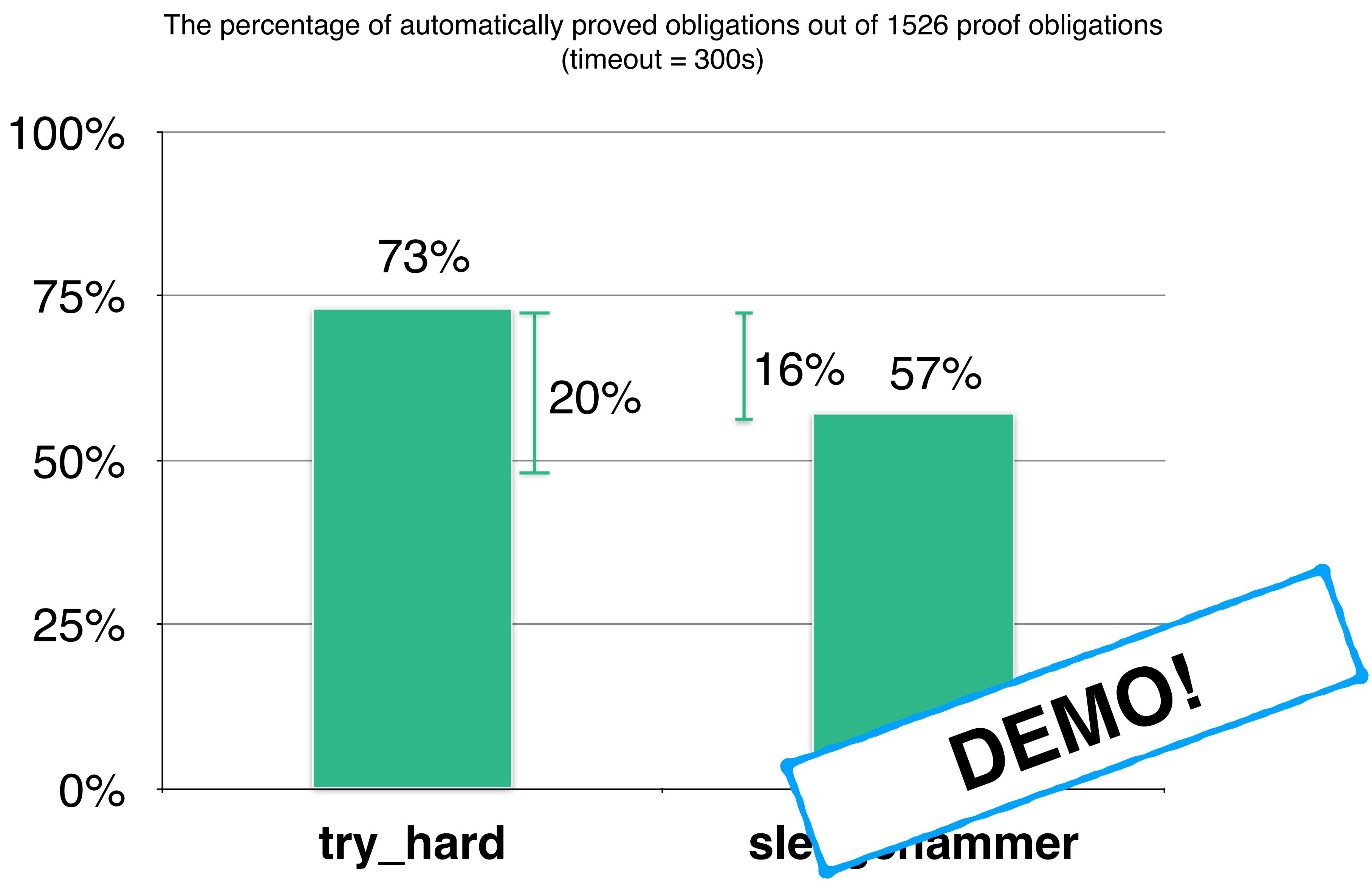
almost no code clutter!!

try_hard: the default strategy

```
strategy Try_Hard =
Ors [Thens [Subgoal, Basic],
      Thens [DInductTac, Auto_Solve],
      Thens [DCaseTac, Auto_Solve],
      Thens [Subgoal, Advanced],
      Thens [DCaseTac, Solve_Many],
      Thens [DInductTac, Solve_Many] ]
```

```
strategy Basic =
  Ors [
      Auto_Solve,
      Blast_Solve,
      FF_Solve,
      Thens [IntroClasses, Auto_Solve],
      Thens [Transfer, Auto_Solve],
      Thens [Normalization, IsSolved],
      Thens [DInduct, Auto_Solve],
      Thens [Hammer, IsSolved],
      Thens [DCases, Auto_Solve],
      Thens [DCoinduction, Auto_Solve],
      Thens [Auto, RepeatN(Hammer), IsSolved],
      Thens [DAuto, IsSolved]]
```

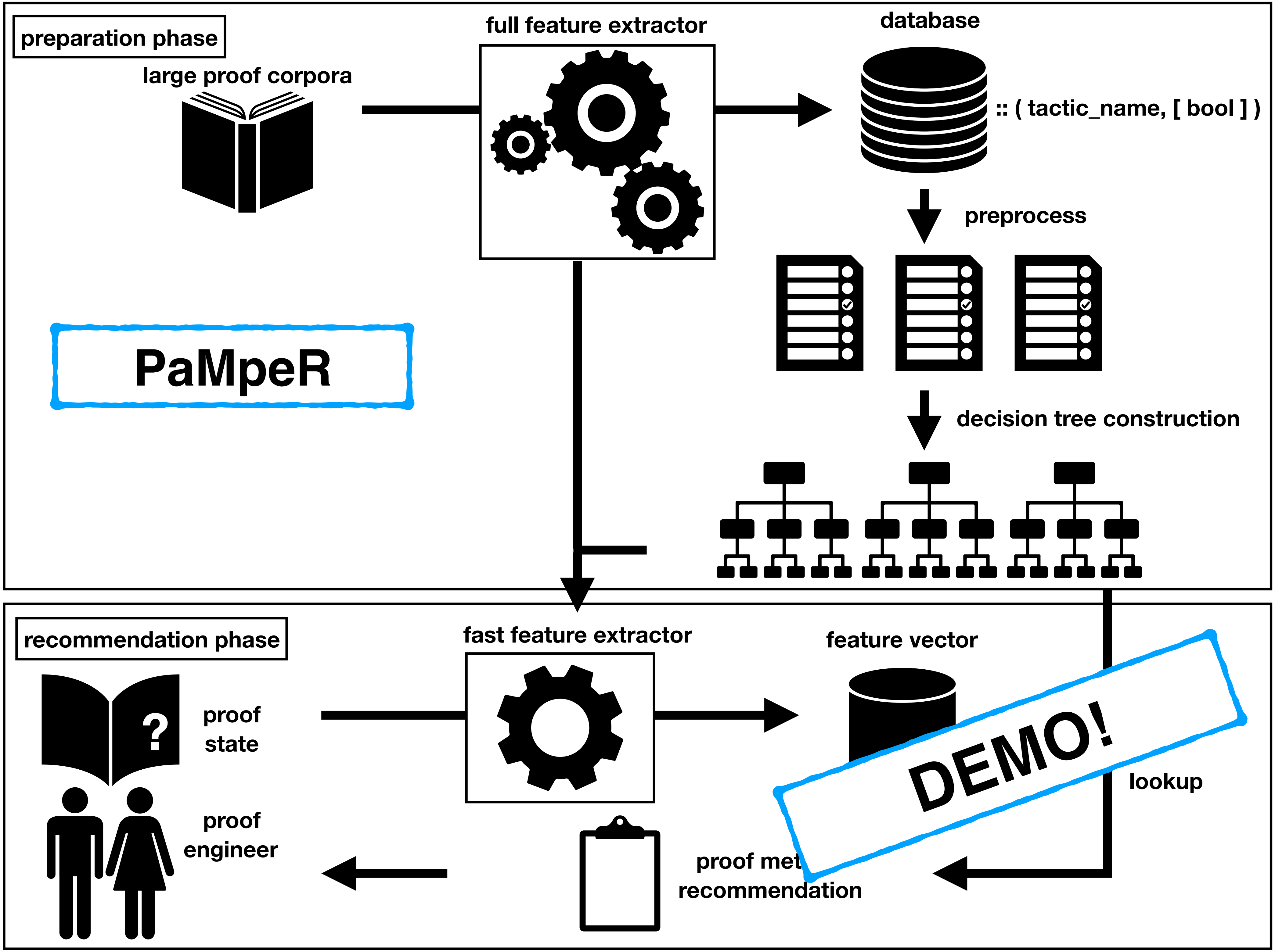
try_hard vs sledgehammer



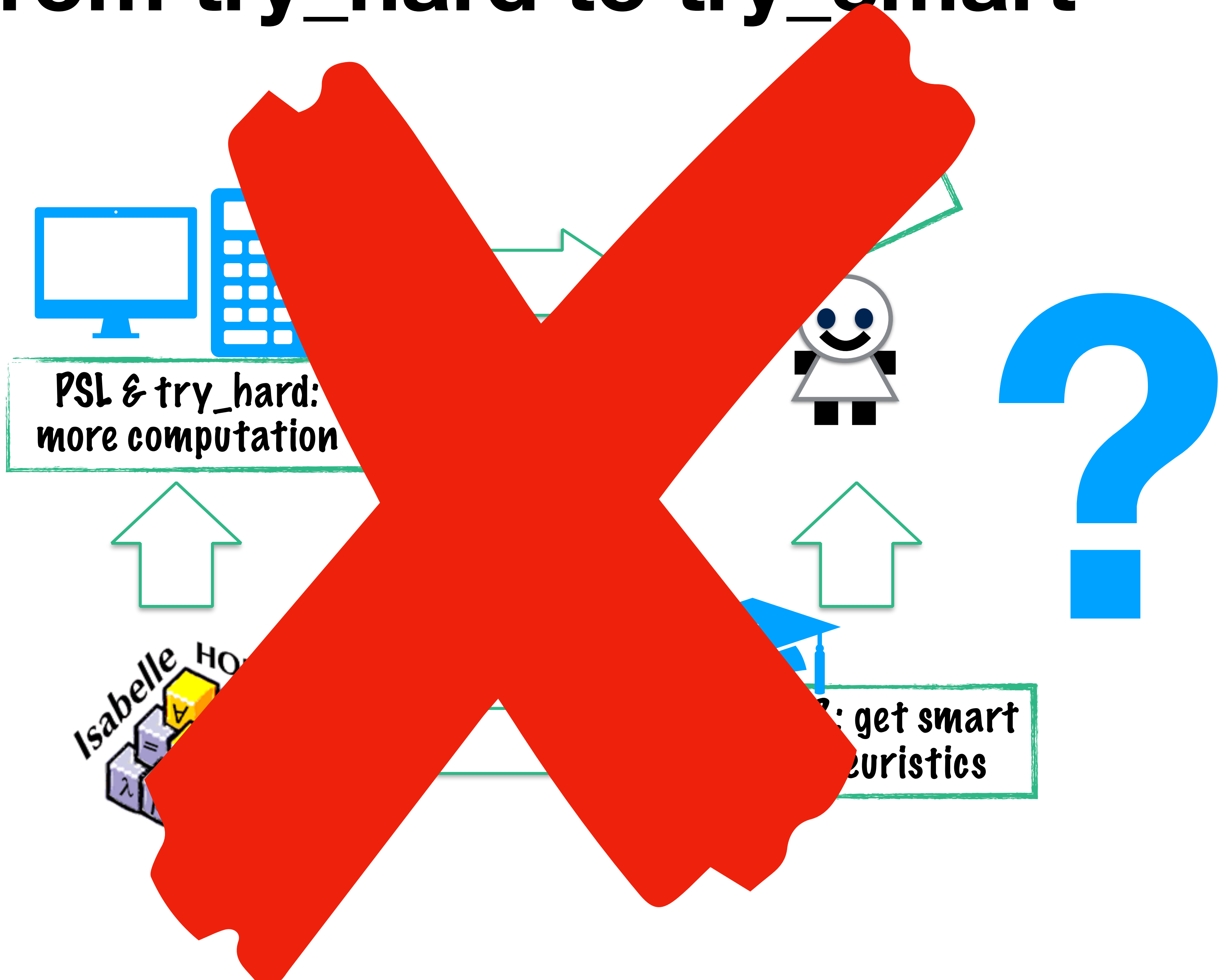
try_hard: the default strategy

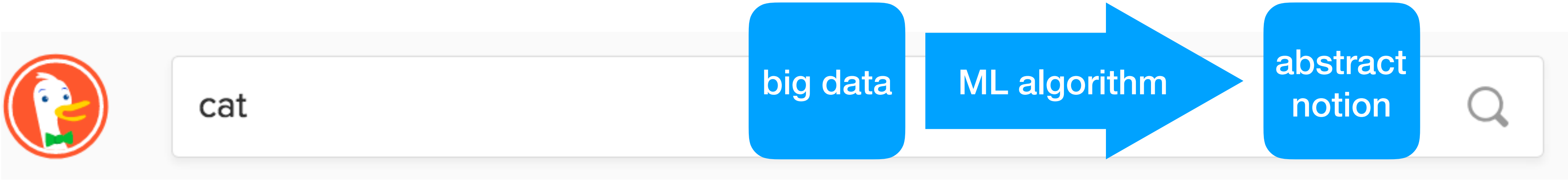
```
strategy Try_Hard =
Ors [Thens [Subgoal, Basic],
       Thens [DInductTac, Auto_Solve],
       Thens [DCaseTac, Auto_Solve],
       Thens [Subgoal, Advanced],
       Thens [DCaseTac, Solve_Many],
       Thens [DInductTac, Solve_Many] ]
```

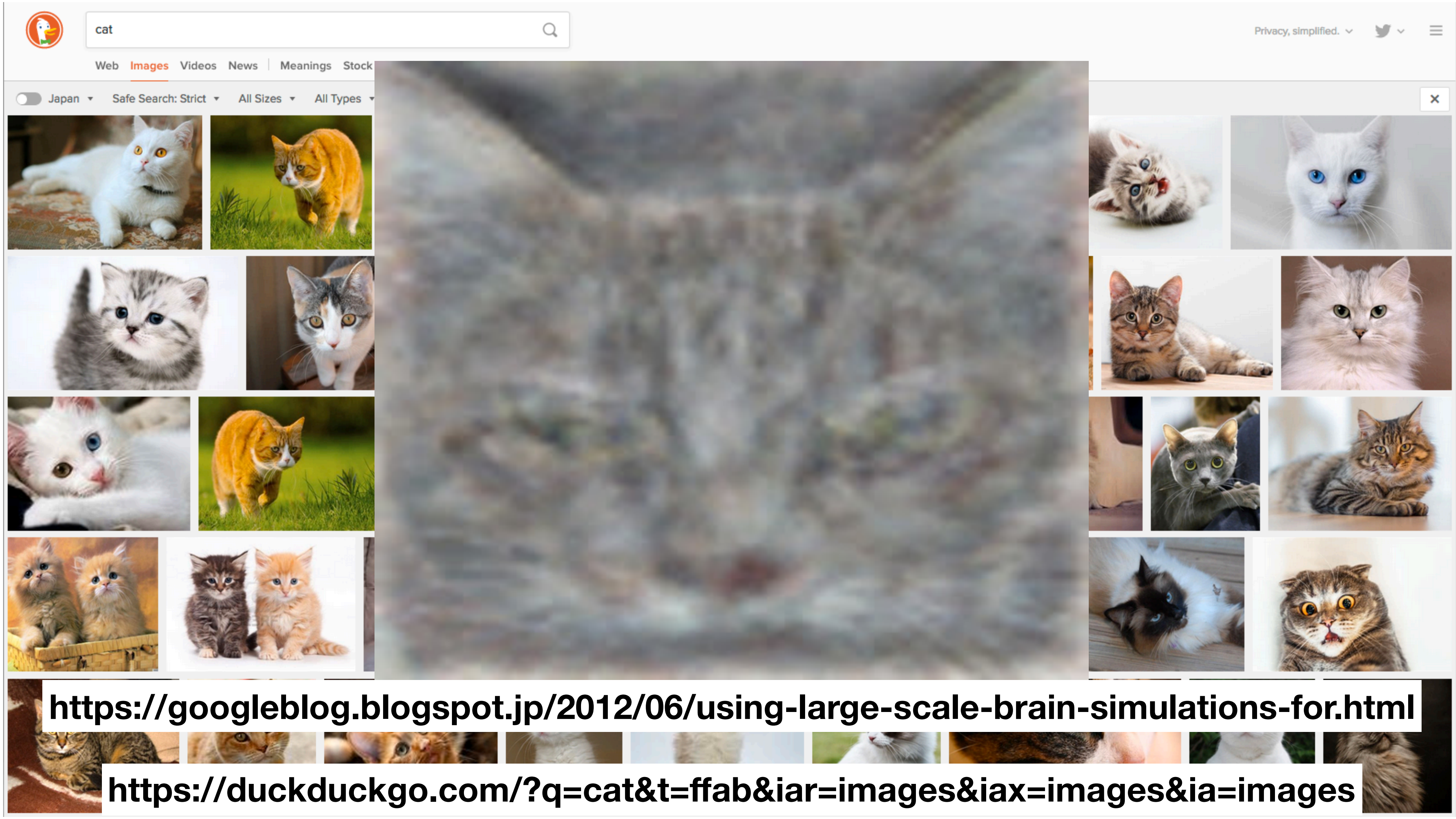
```
strategy Basic =
  Ors [
     Auto_Solve,
     Blast_Solve,
     FF_Solve,
     Thens [IntroClasses, Auto_Solve],
     Thens [Transfer, Auto_Solve],
     Thens [Normalization, IsSolved],
     Thens [DInduct, Auto_Solve],
     Thens [Hammer, IsSolved],
     Thens [DCases, Auto_Solve],
     Thens [DCoinduction, Auto_Solve],
     Thens [Auto, RepeatN(Hammer), IsSolved],
     Thens [DAuto, IsSolved]]
```


preparation phase
large proof corpora
full feature extractor
database
:: (tactic_name, [bool])
preprocess
decision tree construction
PaMpeR
recommendation phase
fast feature extractor
feature vector
proof
state
proof
engineer
lookup
DEMO!
proof met
recommendation

from try_hard to try_smart







https://googleblog.blogspot.jp/2012/06/using-large-scale-brain-simulations-for.html

https://duckduckgo.com/?q=cat&t=ffab&iar=images&iax=images&ia=images

many proofs → ML algorithm → abstract proof

```
primrec my_append :: "'a list ⇒ 'a ... r ...) where
append_Nil: "[] @@ ys = ys"
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"
```

```
lemma "([1::nat]   @@ [2]      ) @@ [3]       = [1]       @@ ([2]      @@ [3]      )"by auto
lemma "([1::int]   @@ [2]      ) @@ [3]       = [1]       @@ ([2]      @@ [3]      )"by auto
lemma "([1,2::nat] @@ [3,4]    ) @@ [5,6]     = [1,2]     @@ ([3,4]    @@ [5,6]    )"by auto
lemma "([λx. x+1]  @@ [λx. x+2]) @@ [λx. x+3] = [λx. x+1] @@ ([λx. x+2] @@ [λx. x+3])"by auto
lemma "([''a'']    @@ [''b'']  ) @@ [''c'']   = [''a'']   @@ ([''b'']  @@ [''c'']  )"by auto
lemma "([''d'']    @@ [''e'']  ) @@ [''f'']   = [''d'']   @@ ([''e'']  @@ [''f'']  )"by auto
```

```
lemma "(x @@ y) @@ z = x @@ (y @@ z)"
```

```
apply auto
Failed to apply proof method:
goal (1 subgoal):
 1. (x @@ y) @@ z = x @@ y @@ z
```

```
apply (induct x)
 apply auto

goal:
No subgoals!
```

Higher-Order functions

type class

polymorphism

dependent types

universal quantifier

lambda abstraction

concise formula that can cover lots of concrete cases

small data set for each problem

different proof for general case

Large Proof Corpora?

Transfer learning? ✗ (?)

Security. Performance. Proof.

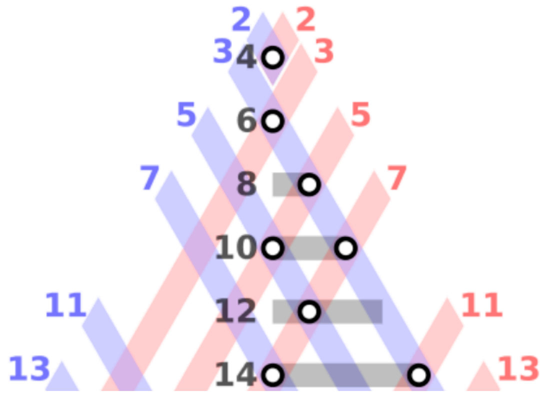
The seL4 proofs in Isabelle

OpenTheory Project

opentheory • metis • chess

http://www.gilith.com/opentheory/

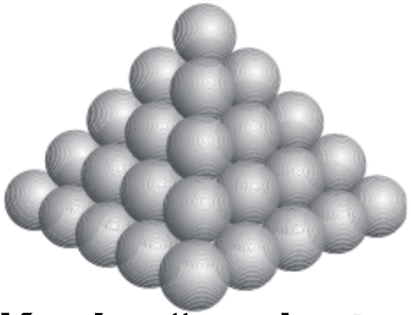
?



Goldbach's conjecture

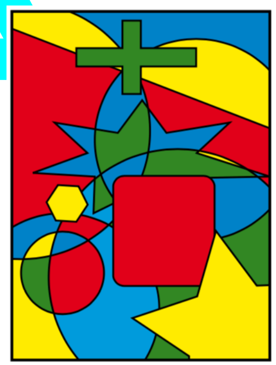
https://en.wikipedia.org/wiki/File:Goldbach_partitions_of_the_even_integers_from_4_to_50_rev4b.svg

"Every even integer greater than 2 can be expressed as the sum of two primes."



The Kepler "conjecture" in HOL Light

http://annals.math.princeton.edu/wp-content/uploads/annals-v162-n3-p01.pdf



Four color theorem in Coq

https://en.wikipedia.org/wiki/File:Four_Colour_Map_Example.svg

different logics in different provers

different proof corpora about different problems

poor proof automation
for expressive logics

artificial intelligence
for theorem proving!

We need big data!

really?

only small dataset available
because of expressiveness

Games

AlphaGo (Zero) problems similar to proving [Silver+2016]

- Node evaluation
- Policy decisions

http://cl-informatik.uibk.ac.at/teaching/ss18/mltp/02.pdf

Really? Self-play?

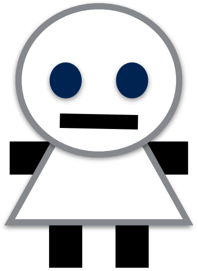
I want to train my prover using self-play so that it can prove Goldbach's conjecture.

But how? Proof search is not a 2-player game.

The one that finds a proof of Goldbach's conjecture first is the winner.

If one prover finds a proof, that's it. It is only 1 iteration.

But how do you train provers, so that one prover can eventually find a proof.



For each iteration, I create a set of not-so-difficult conjectures. The one that proves more conjectures is the winner.

But how do you create not-so-difficult conjectures?

random?

But randomly created conjectures are not always good training data.

Conjectures with difficult proofs are important ones.

Not really. You need a mechanism to create many conjectures that are relevant to Goldbach's conjecture.

I can produce conjectures by mutating Goldbach's conjecture.

That might work for a small number of conjectures. Not for many conjectures.

How?

The more conjectures you create, the more valuable they should be.

Games

AlphaGo (Zero) problems similar to proving [Silver+2016]

- Node evaluation
- Policy decisions

http://cl-informatik.uibk.ac.at/teaching/ss18/mltp/02.pdf

Really? Self-play?

I want to t … conjecture.

… ayer game.

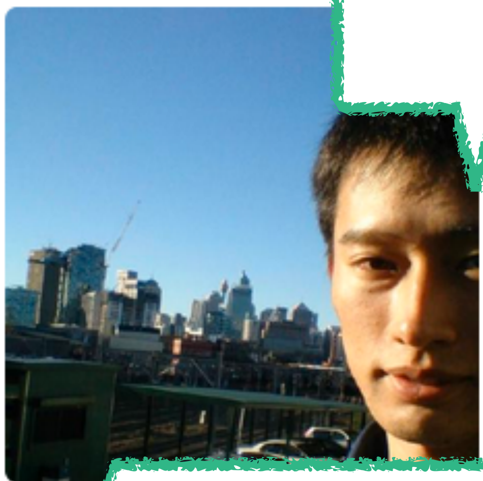
The … ner.

Research hypothesis:

subgoals proved during heuristic (incomplete) proof search are useful to train provers.

each

The



Game of Theorems!

always good training data.

Yuta

yutak

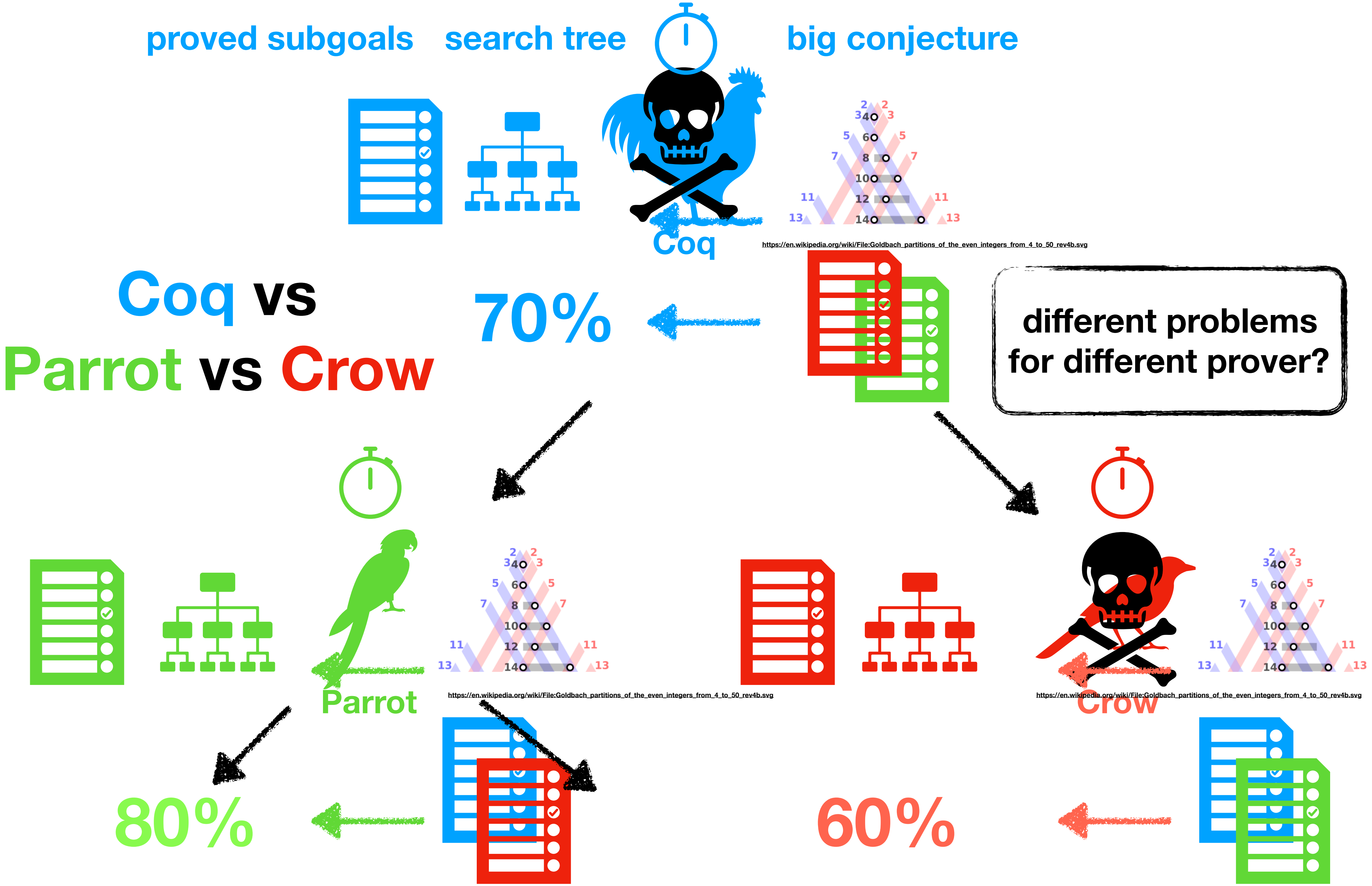
Block or

The more iterations it goes through, the higher the quality of problems should be!

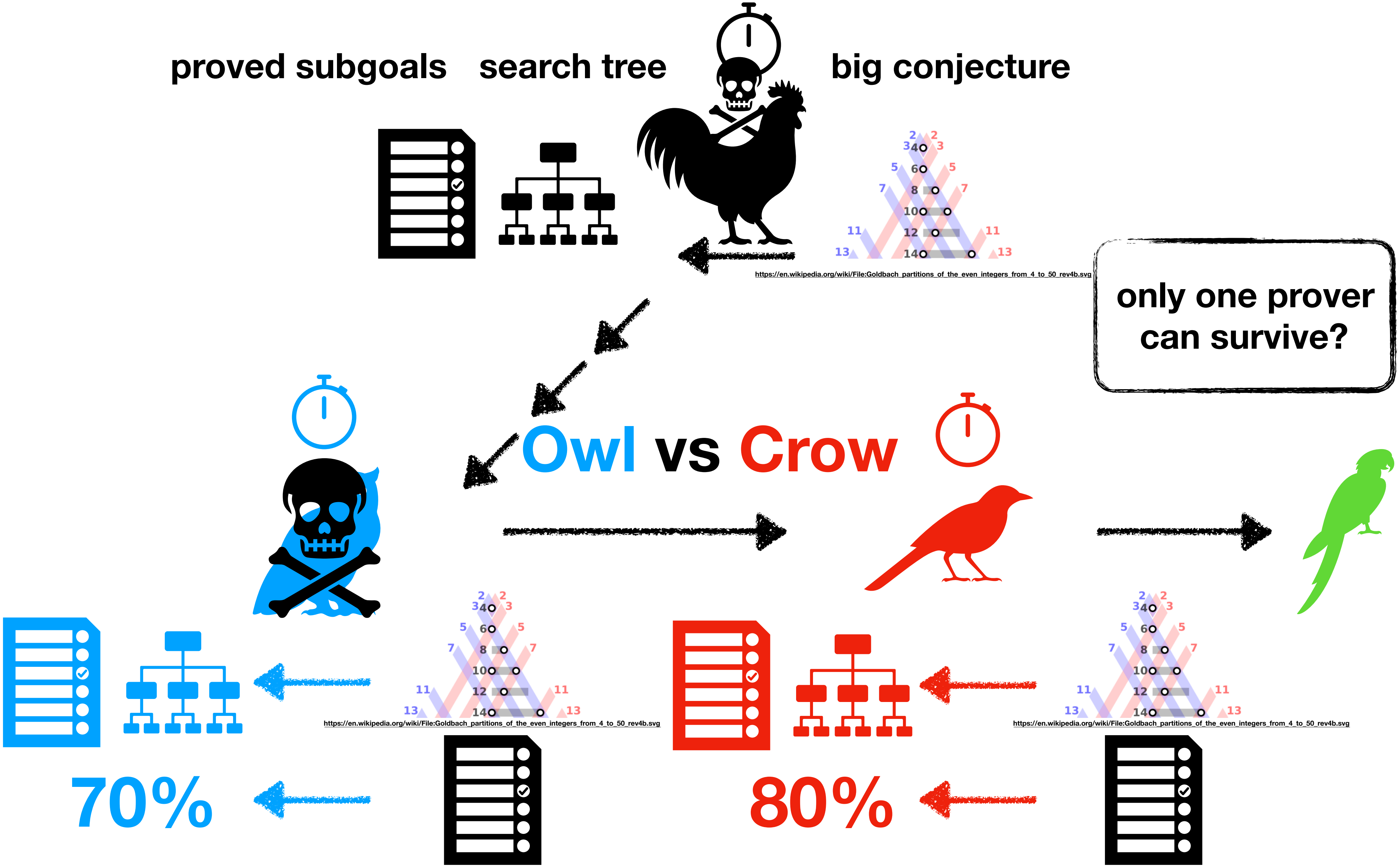
CVUT, CTU, CIIRC

…e more conjectures you create, the more valuable they should be.

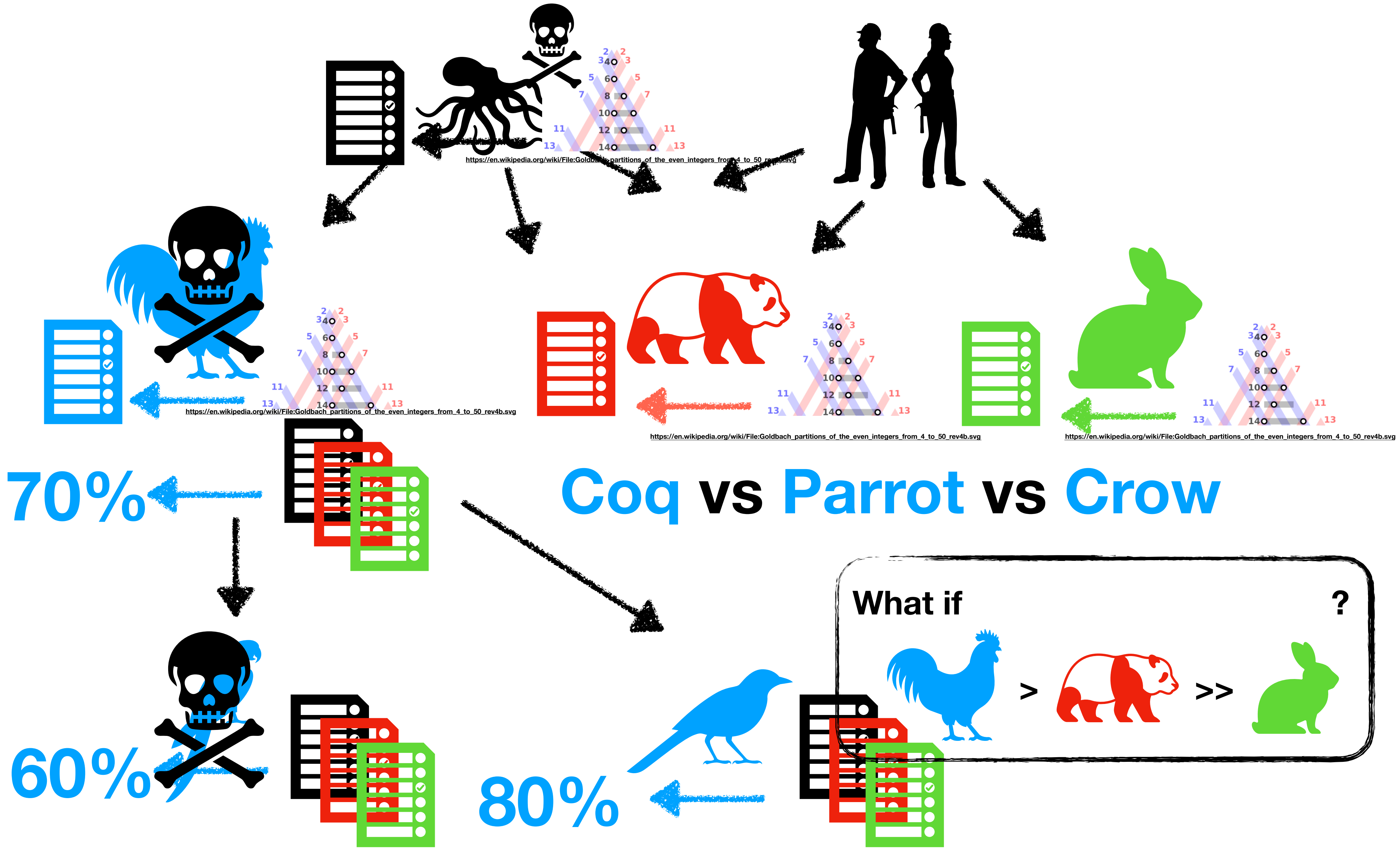
Game of Theorems 1



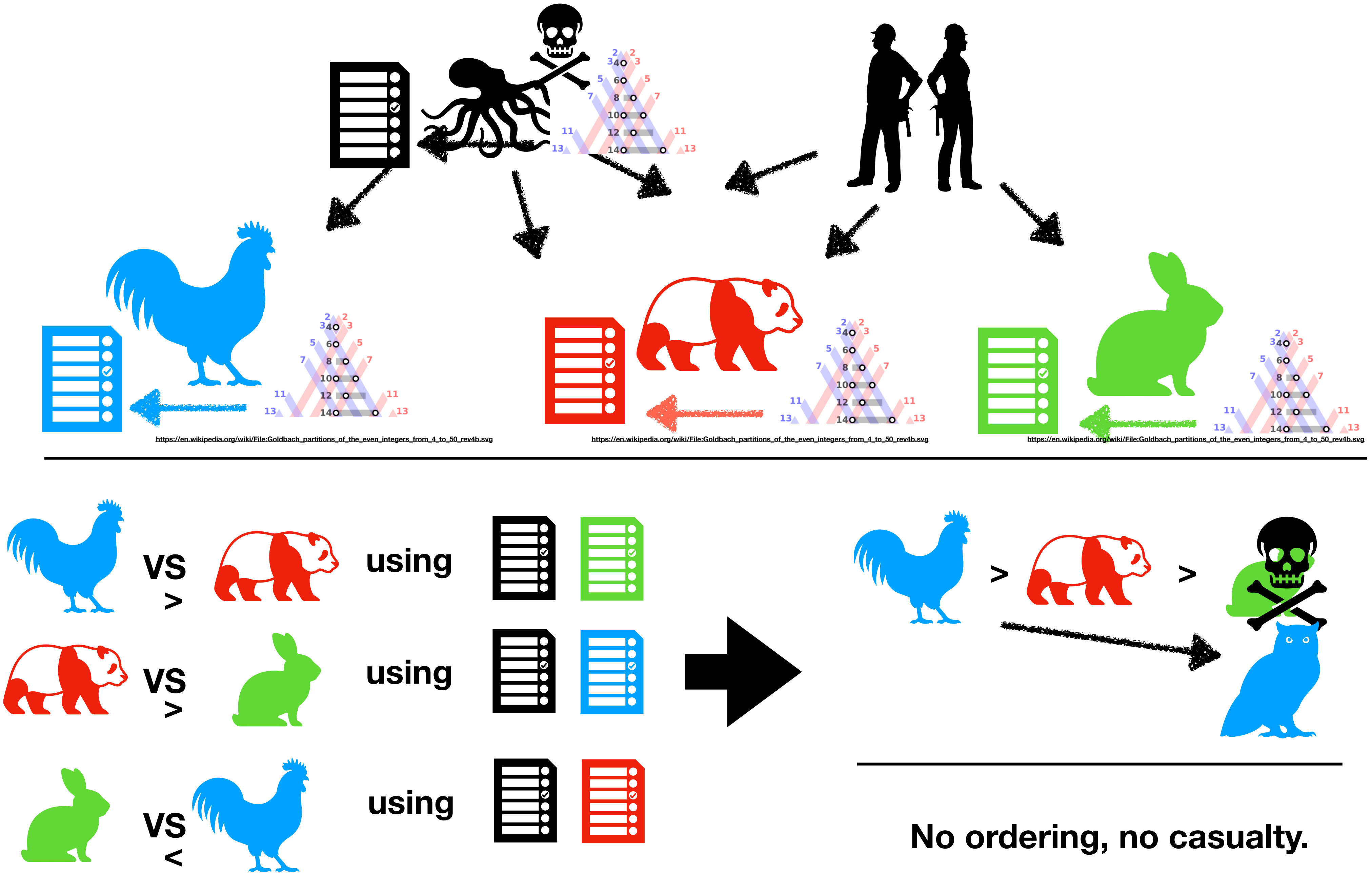
Game of Theorems 2



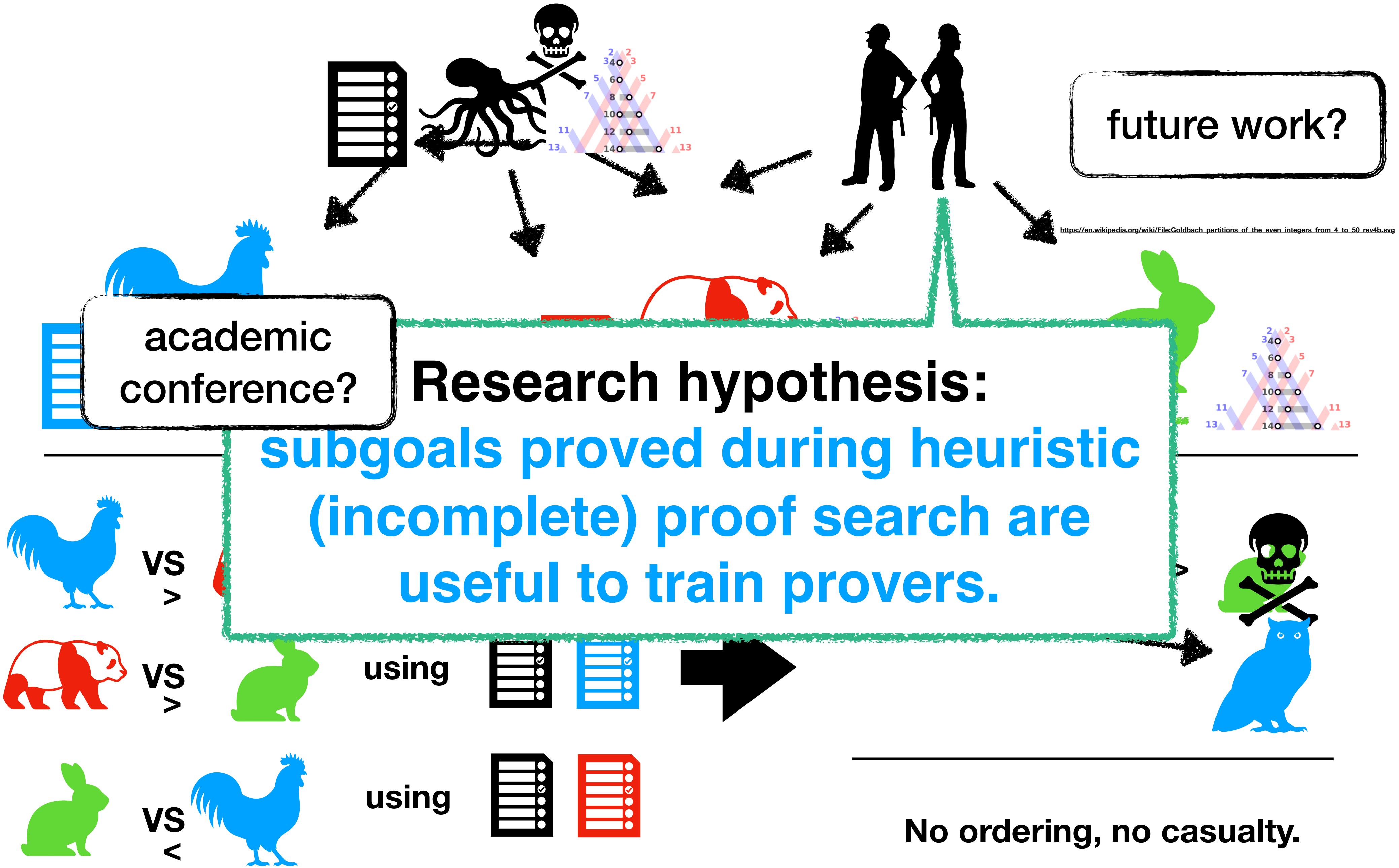
Game of Theorems 3



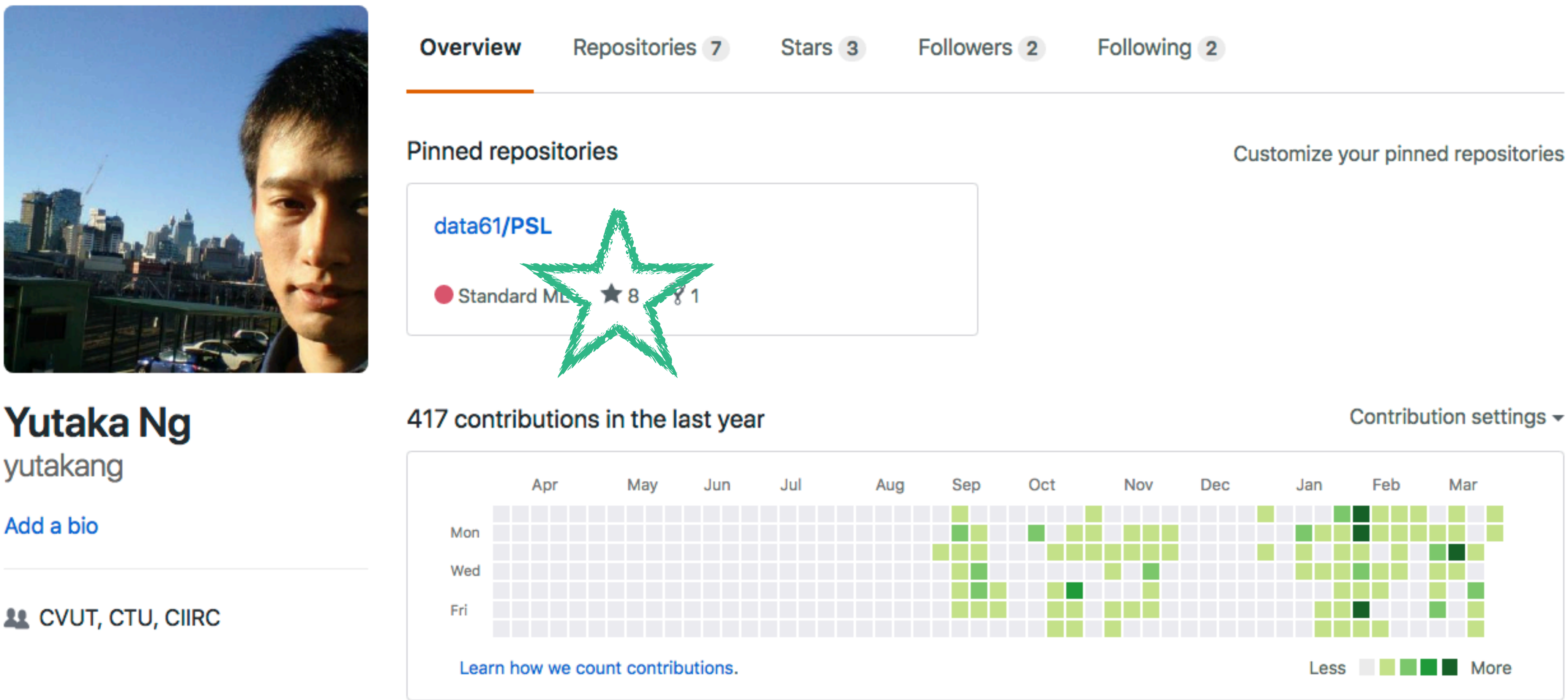
Game of Theorems 4



Game of Theorems 4



Thanks,


Overview
Repositories 7
Stars 3
Followers 2
Following 2
Pinned repositories
Customize your pinned repositories
data61/PSL
Standard ML
8
1
Yutaka Ng
yutakang
Add a bio
CVUT, CTU, CIIRC
417 contributions in the last year
Contribution settings
Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar
Mon
Wed
Fri
Learn how we count contributions.
Less
More