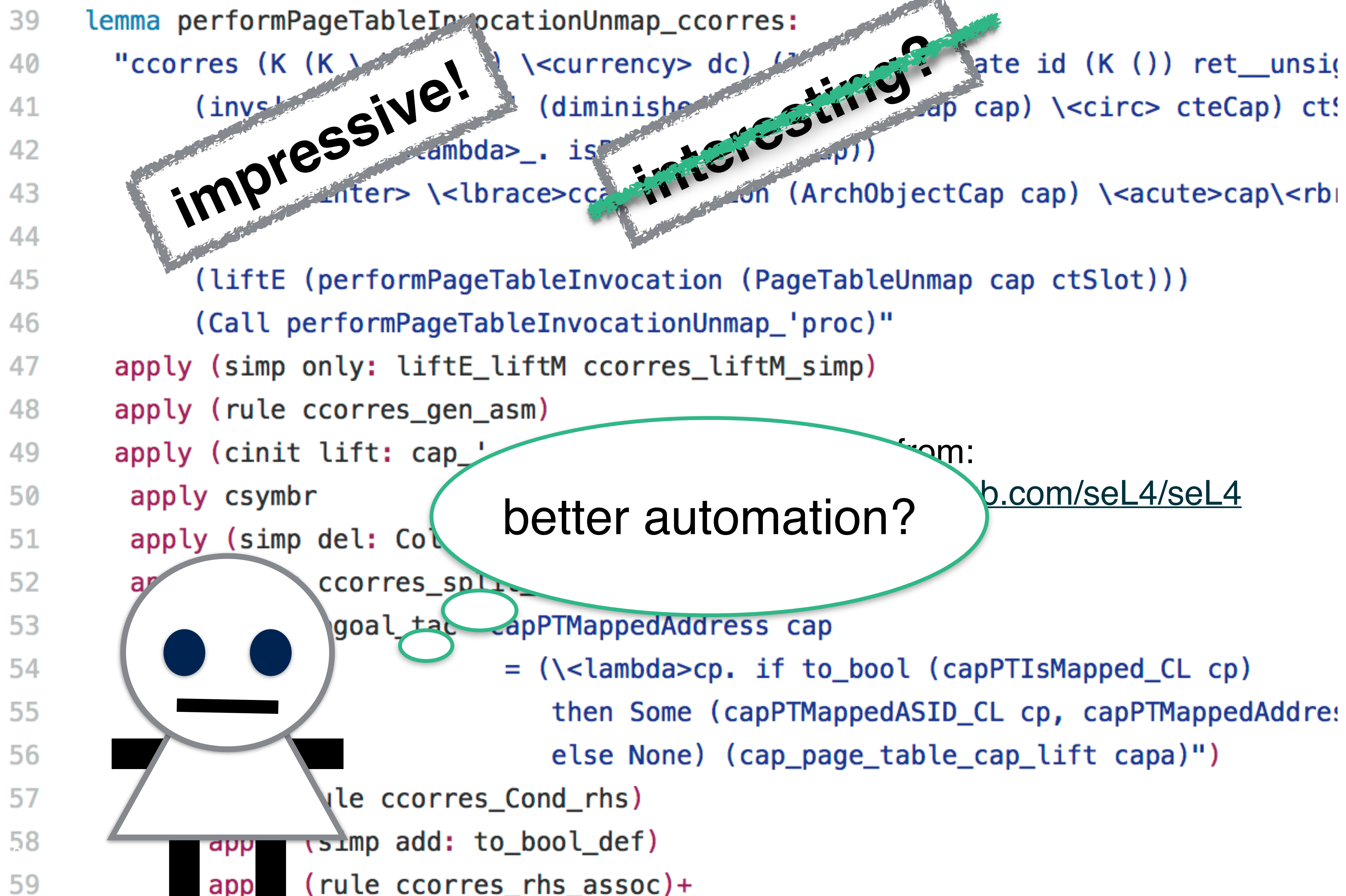
PSL and PaMpeR

Example proof at Data61

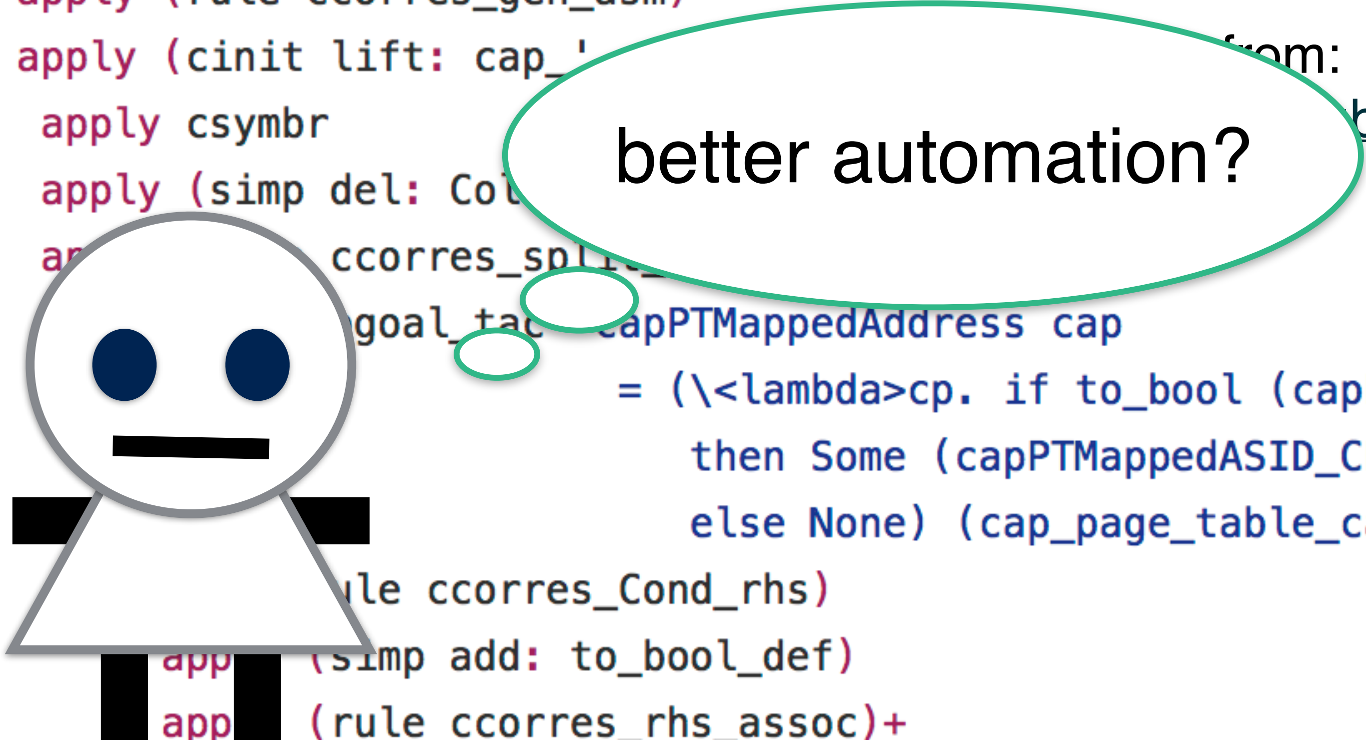


```
lemma performPageTableInvocationUnmap_ccorres:
  "ccorres (K (K \                ) \<currency> dc) (              ate id (K ()) ret__unsig
      (invs'             (diminishe                   ap cap) \<circ> cteCap) ct
                    \<lambda>_. is                  ap))
                 nter> \<lbrace>cc                  on (ArchObjectCap cap) \<acute>cap\<rb

      (liftE (performPageTableInvocation (PageTableUnmap cap ctSlot)))
      (Call performPageTableInvocationUnmap_'proc)"
  apply (simp only: liftE_liftM ccorres_liftM_simp)
  apply (rule ccorres_gen_asm)
  apply (cinit lift: cap_'
   apply csymbr
   apply (simp del: Co
   ap          ccorres_spl
              goal_tac   capPTMappedAddress cap
                           = (\<lambda>cp. if to_bool (capPTIsMapped_CL cp)
                              then Some (capPTMappedASID_CL cp, capPTMappedAddre
                              else None) (cap_page_table_cap_lift capa)")
               ule ccorres_Cond_rhs)
    app       (simp add: to_bool_def)
    app       (rule ccorres_rhs_assoc)+
```

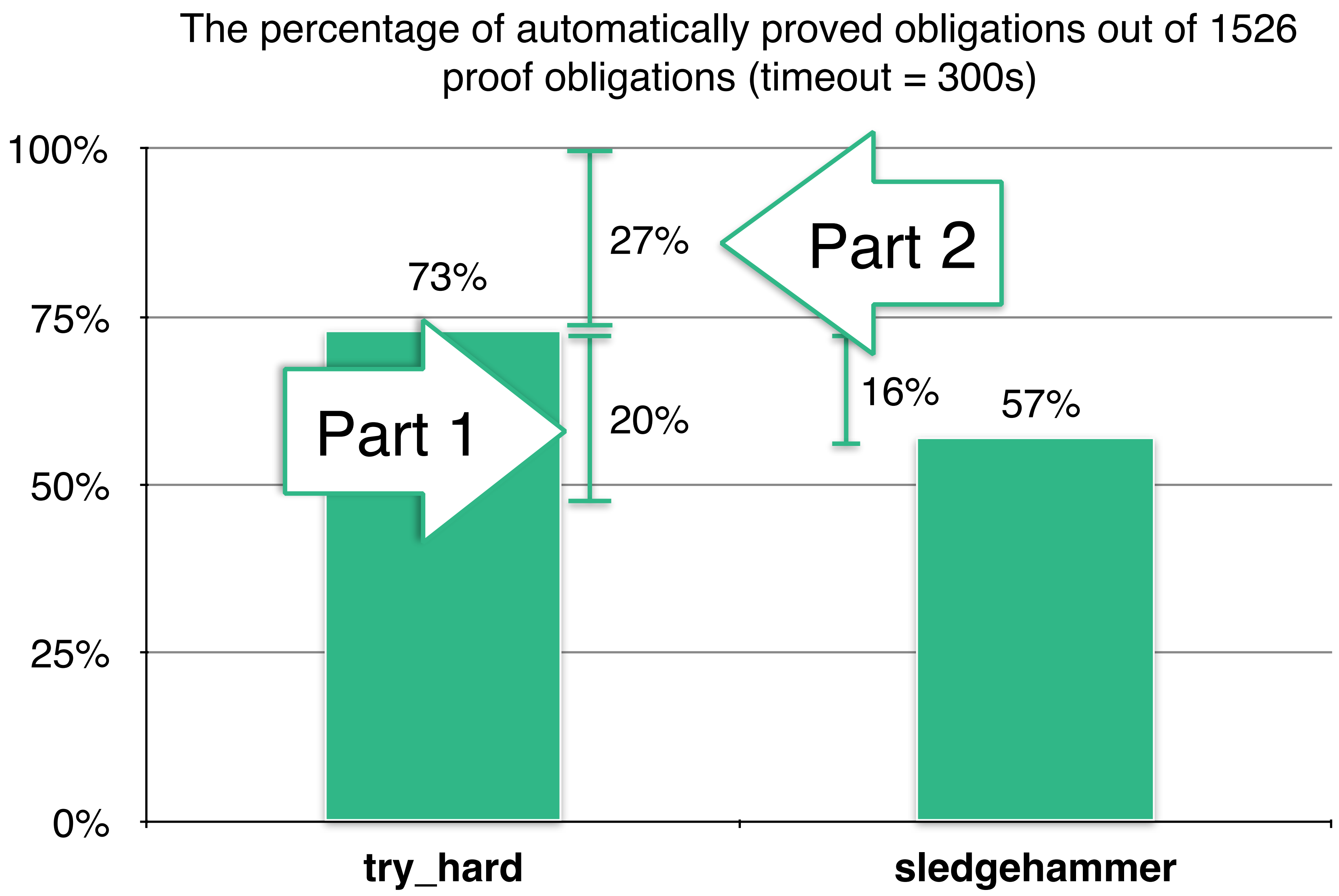
impressive!

~~interesting?~~

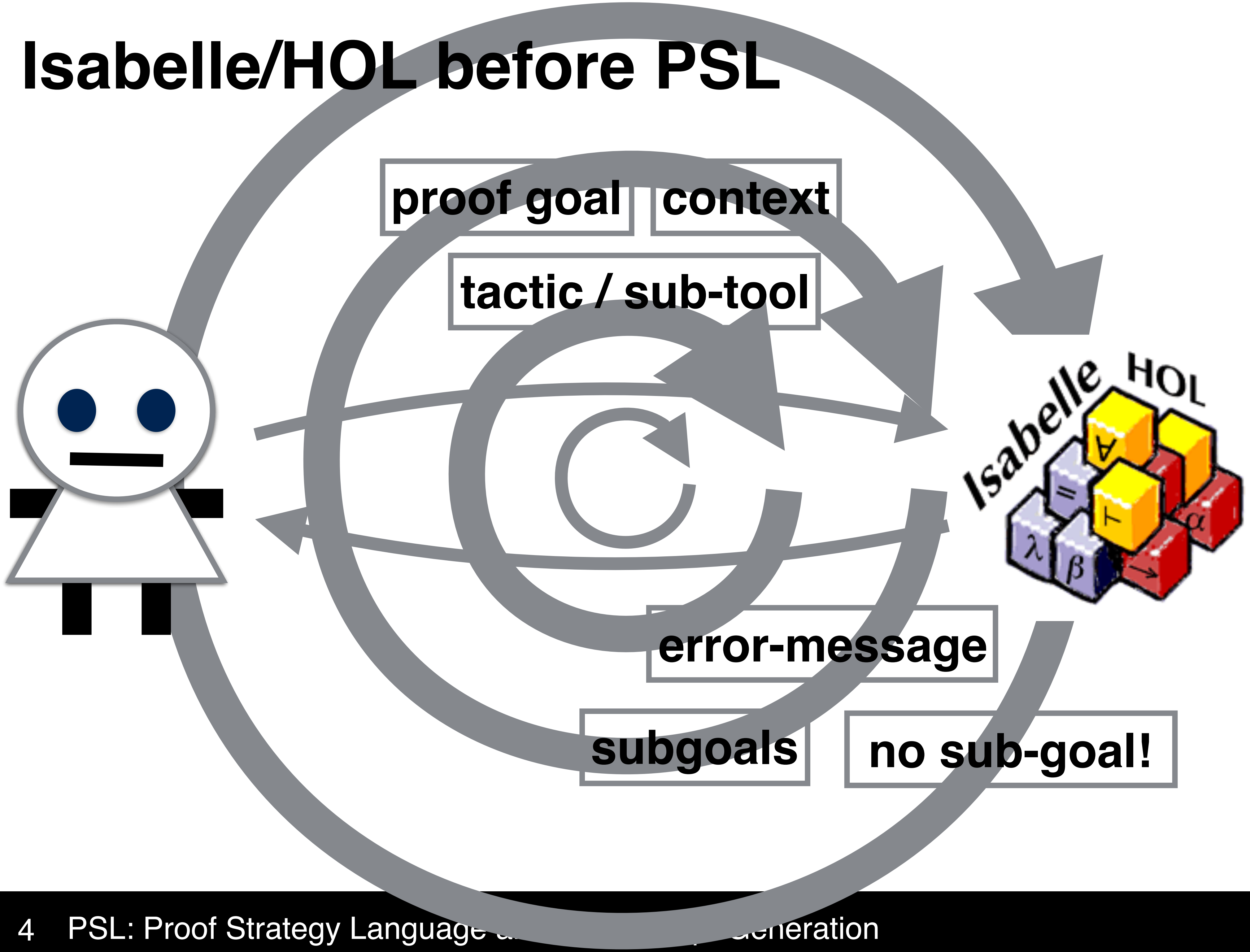
from:
b.com/seL4/seL4



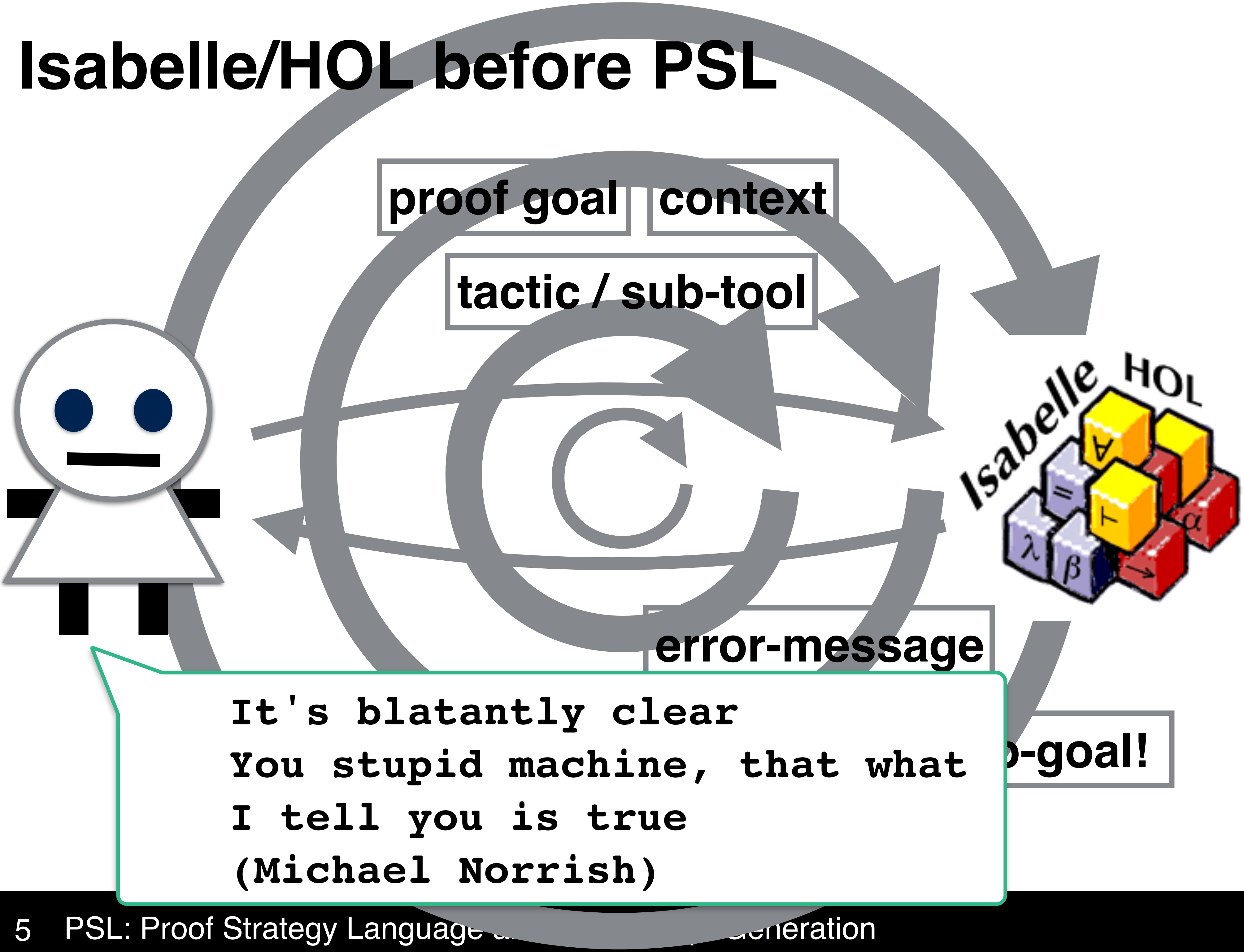
PSL and try-hard for Isabelle/HOL



Isabelle/HOL before PSL



Isabelle/HOL before PSL



It's blatantly clear
You stupid machine, that what
I tell you is true
(Michael Norrish)

PSL (Proof Strategy Language)

meta-tool approach

tactics

quickcheck

sledgehammer

programming language

extensible (Eisbach)

PSL

runtime tactic generation

efficient proof generation

extensive proof search

parallel search

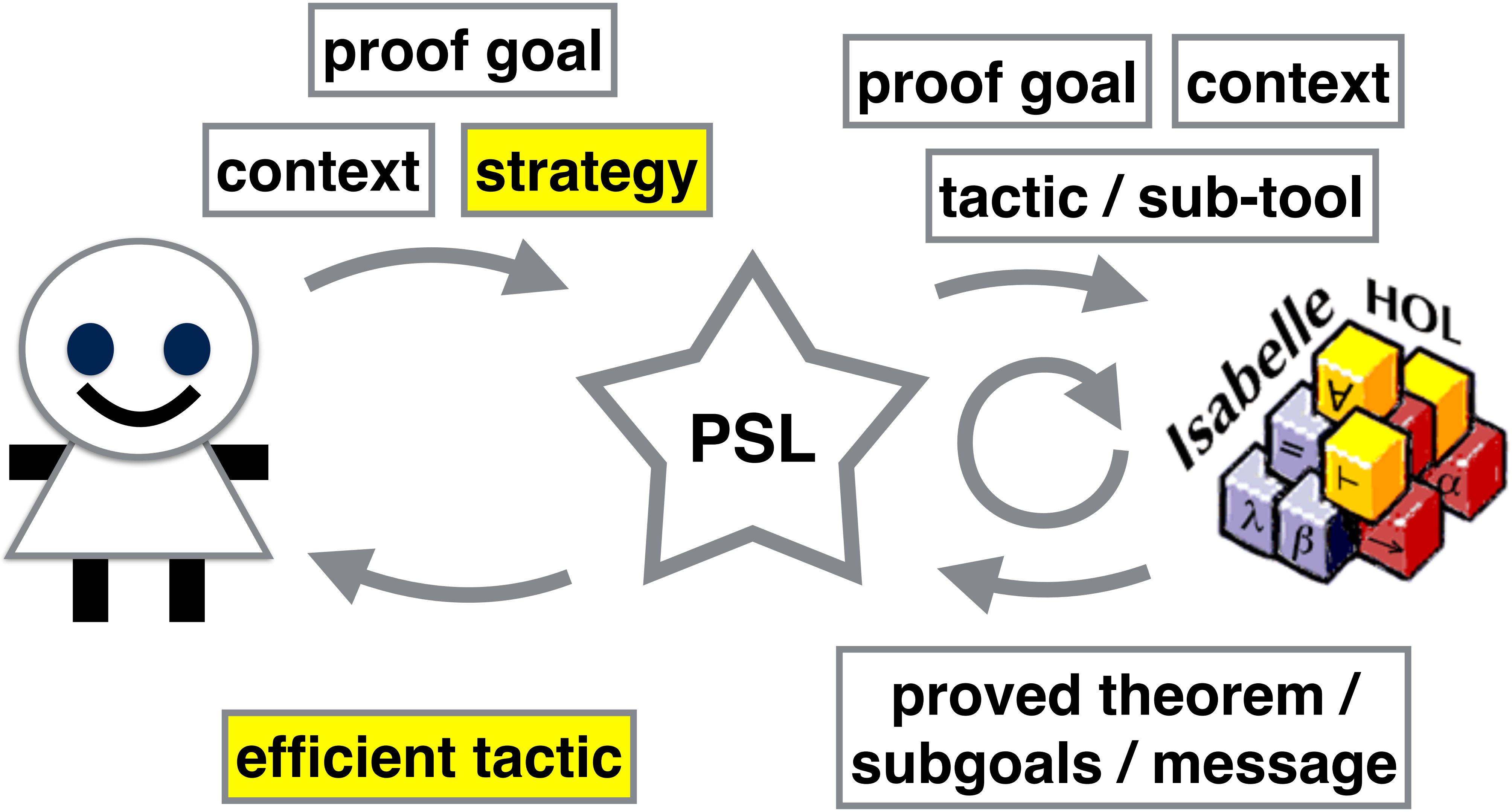
low memory usage

native Isabelle proof script

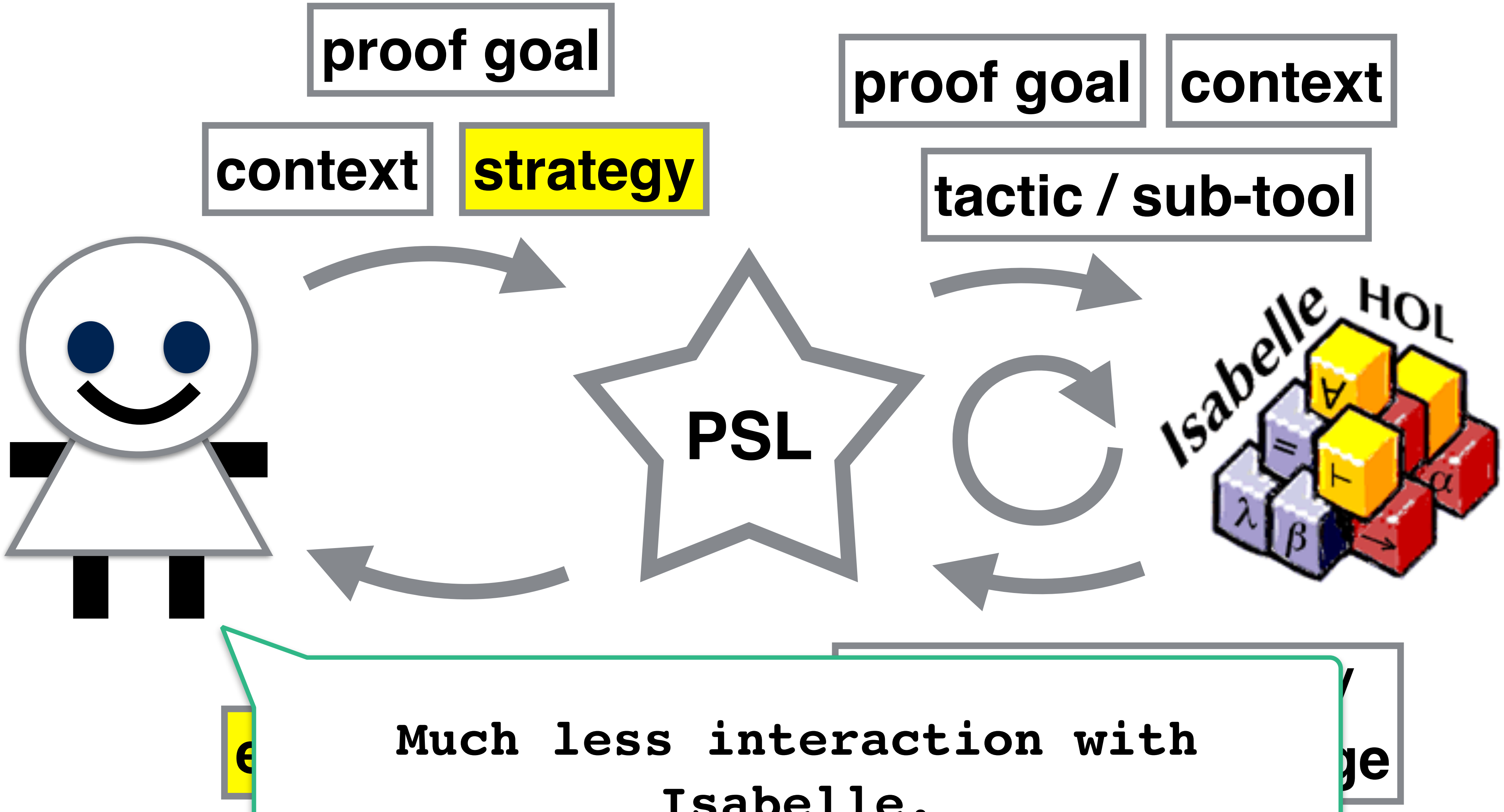
easy installation

almost no code clutter!!

Isabelle/HOL with PSL

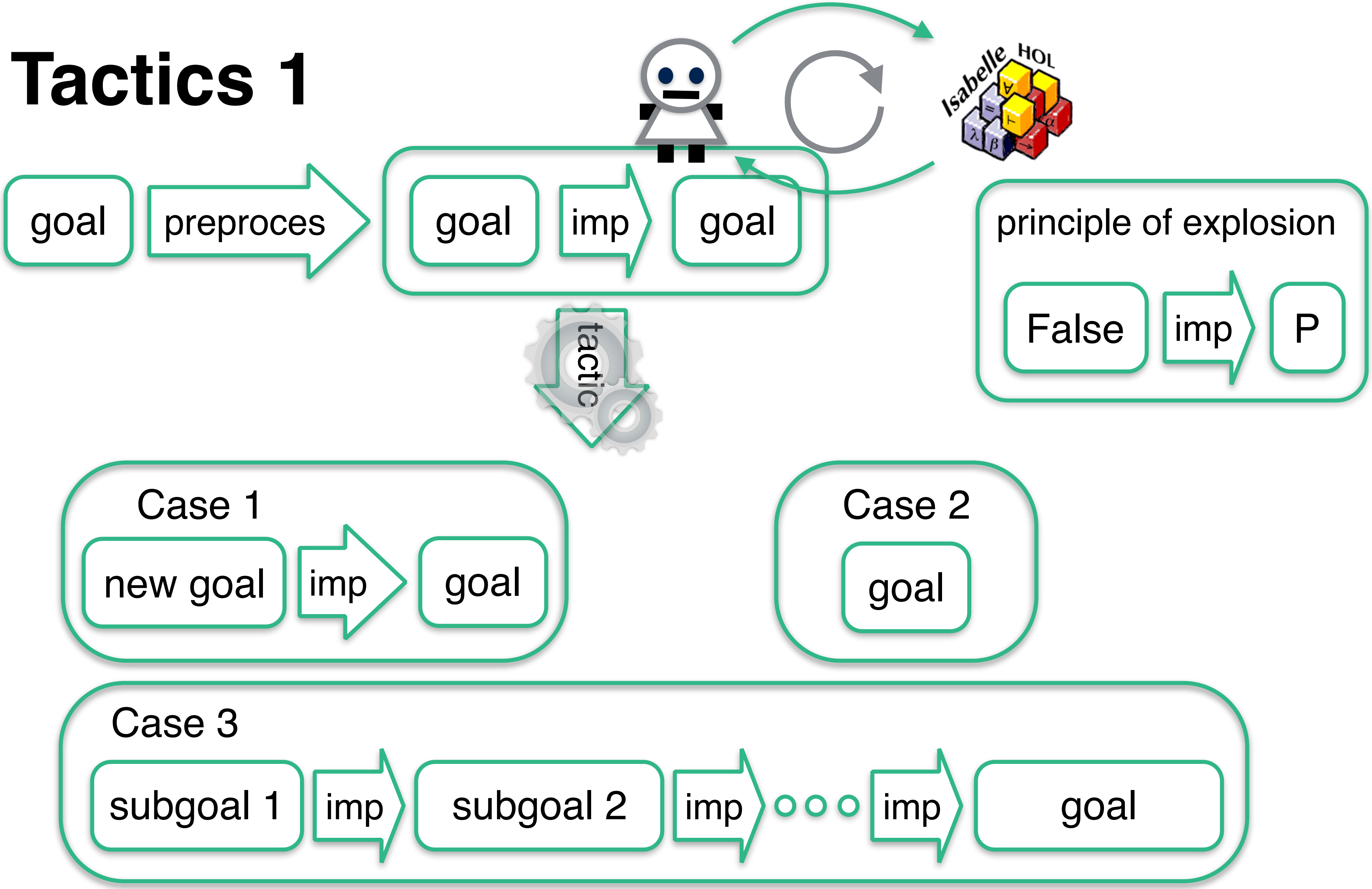


Isabelle/HOL with PSL

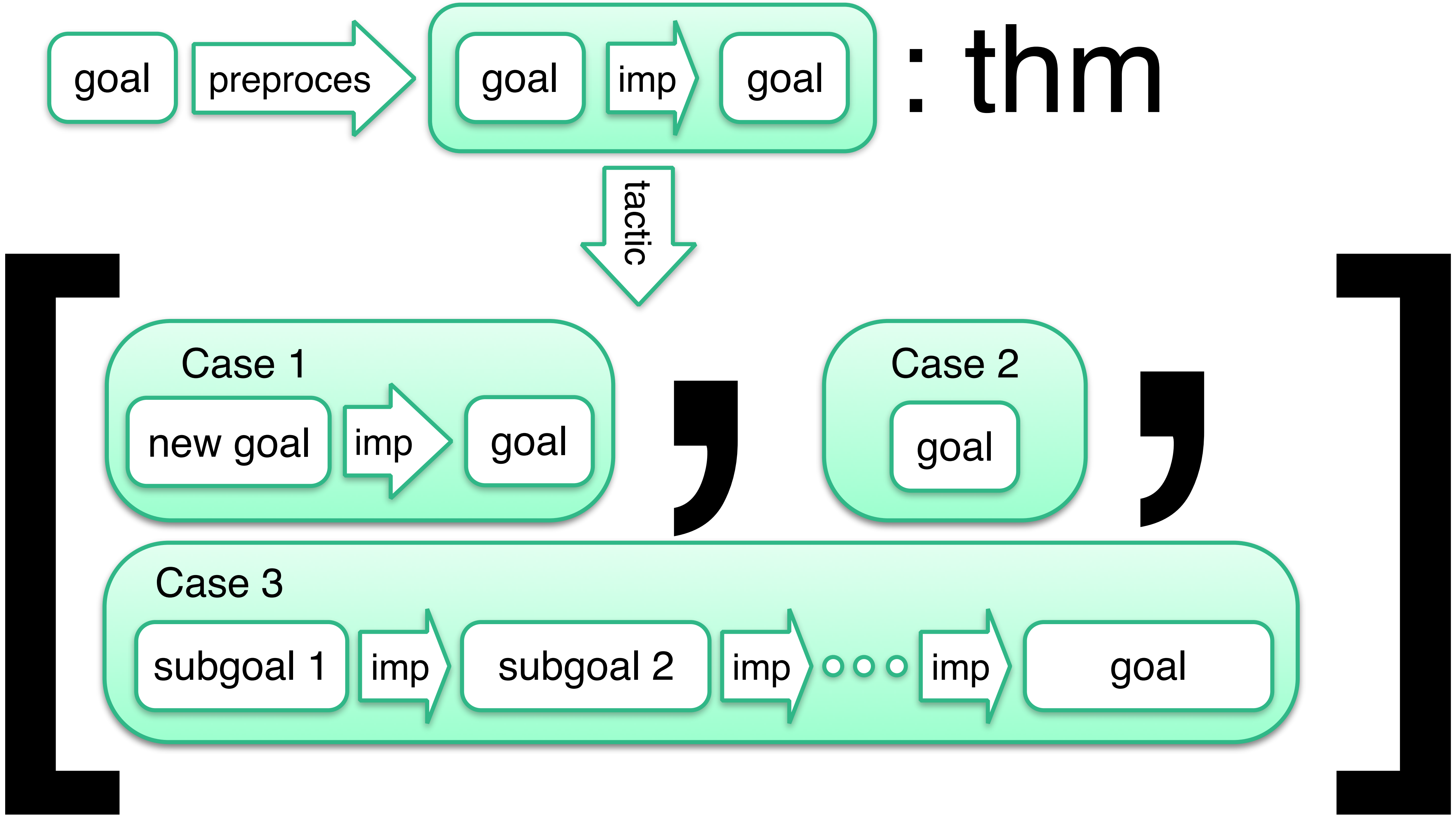


Much less interaction with Isabelle.

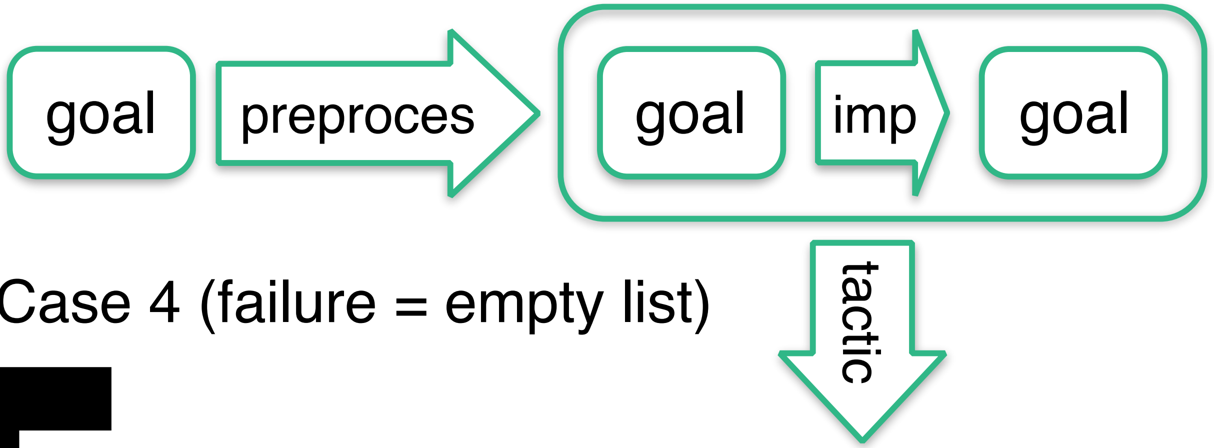
Tactics 1


goal
preproces
goal
imp
goal
Isabelle HOL
principle of explosion
False
imp
P
tactic
Case 1
new goal
imp
goal
Case 2
goal
Case 3
subgoal 1
imp
subgoal 2
imp
imp
goal

Tactics 2


goal
preproces
goal
imp
goal
: thm
tactic
[
Case 1
new goal
imp
goal
,
Case 2
goal
,
Case 3
subgoal 1
imp
subgoal 2
imp
imp
goal
]

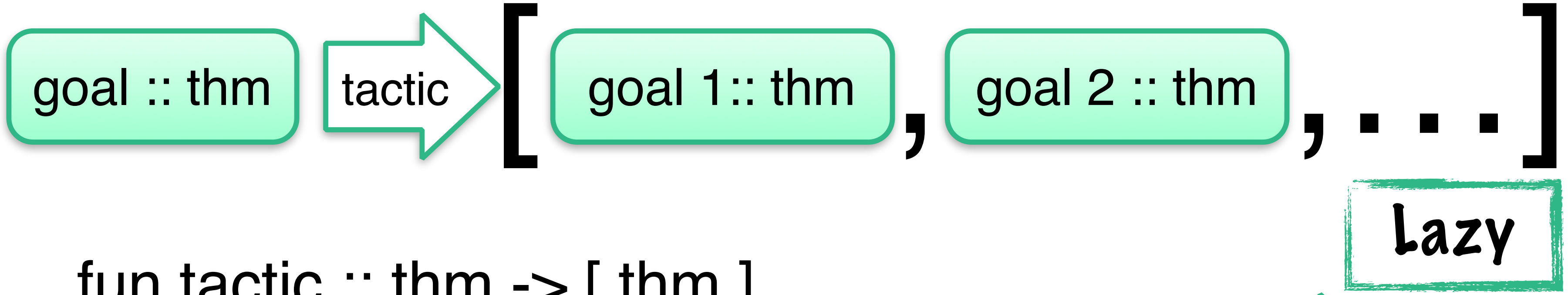
Tactics 2



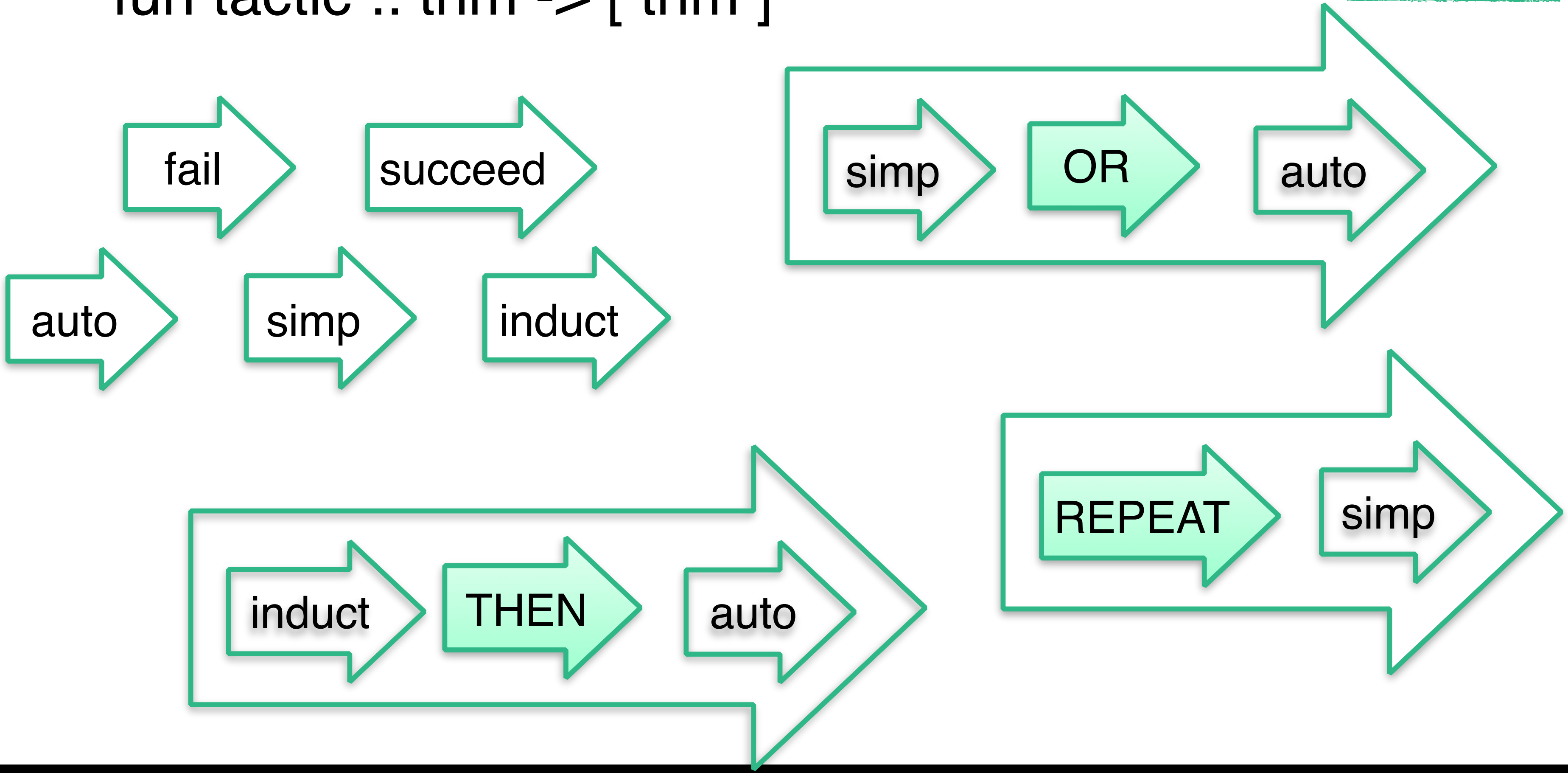
Case 4 (failure = empty list)

[ ]

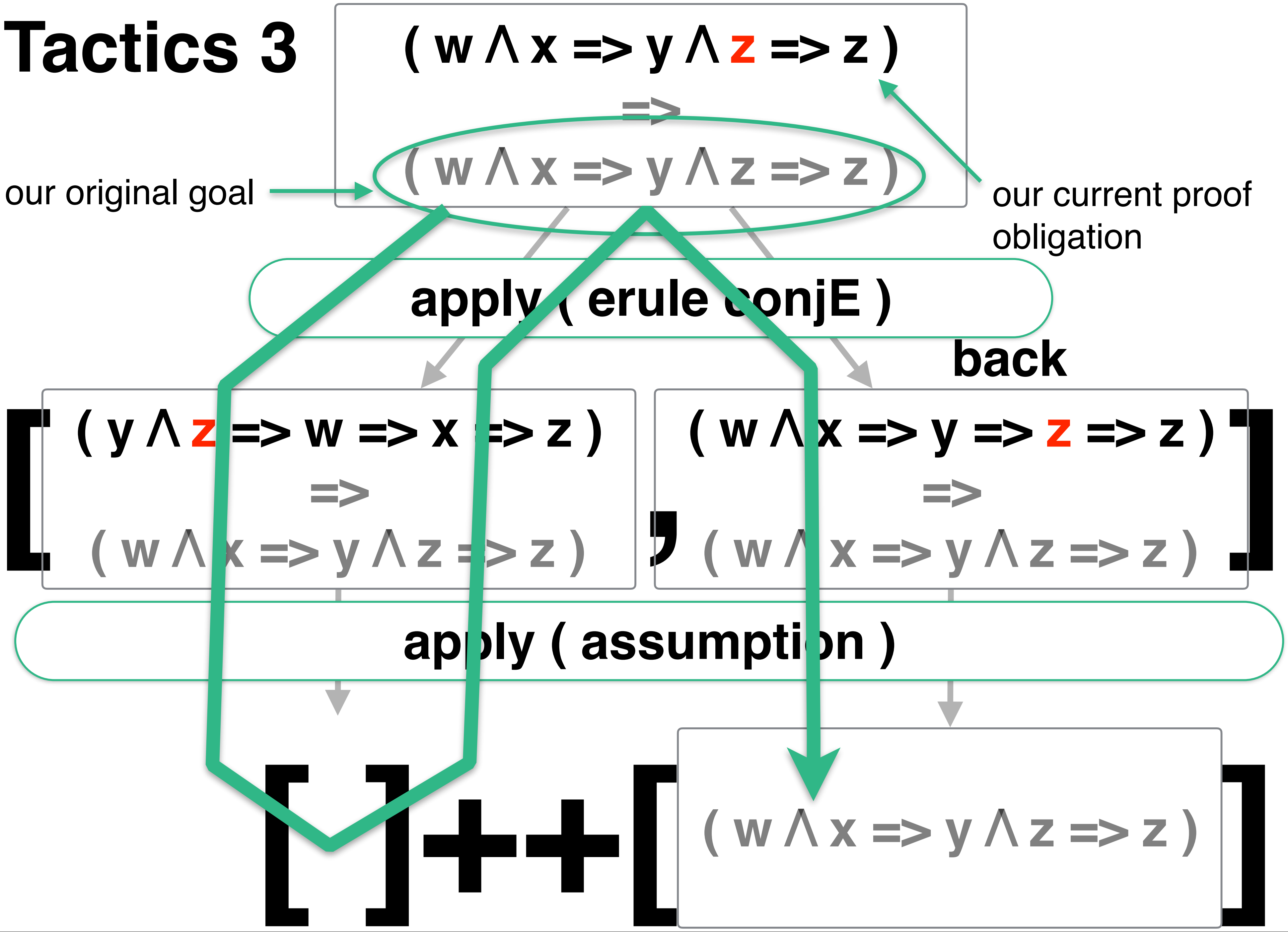
Tactics 4



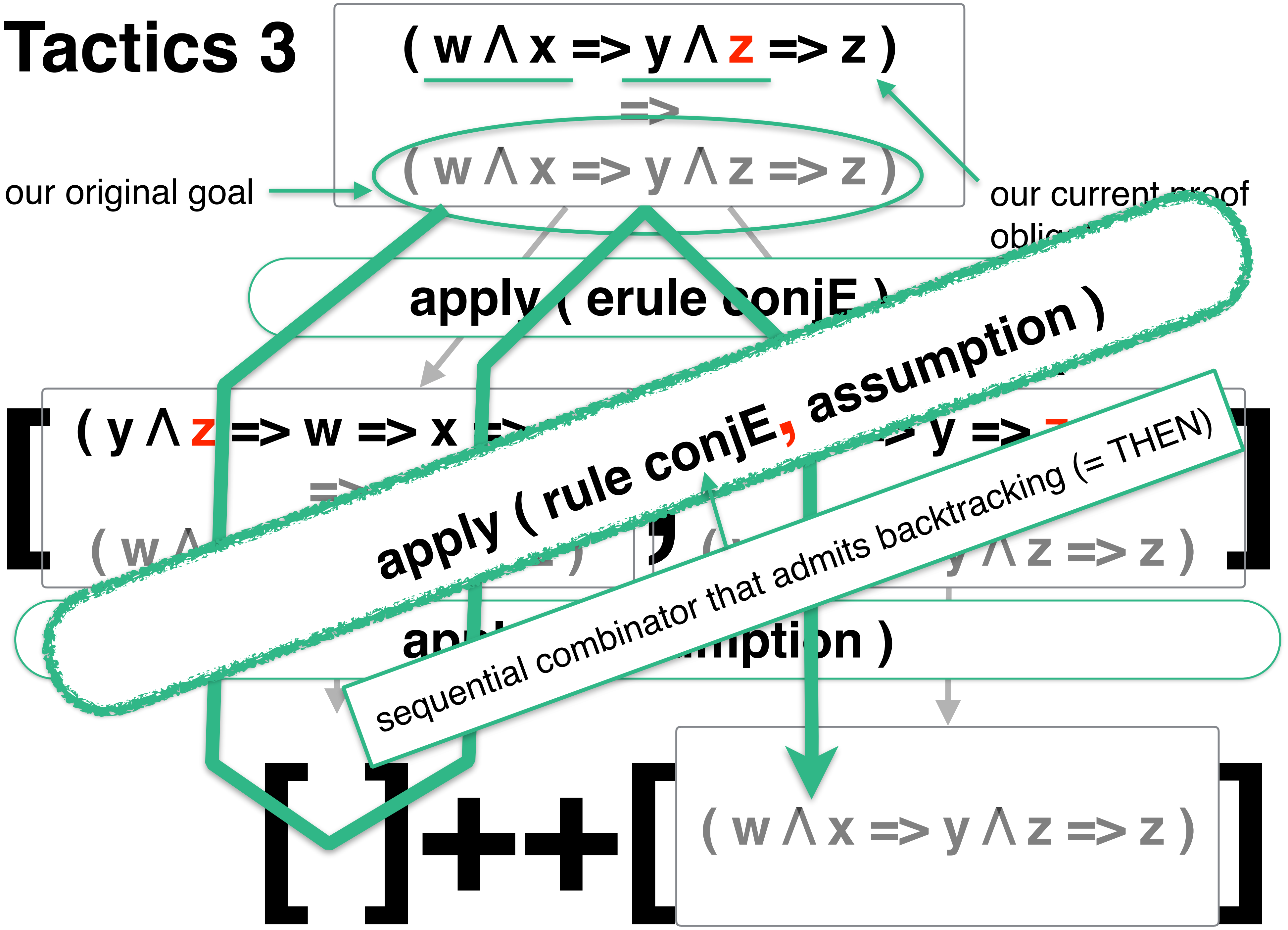
fun tactic :: thm -> [thm]



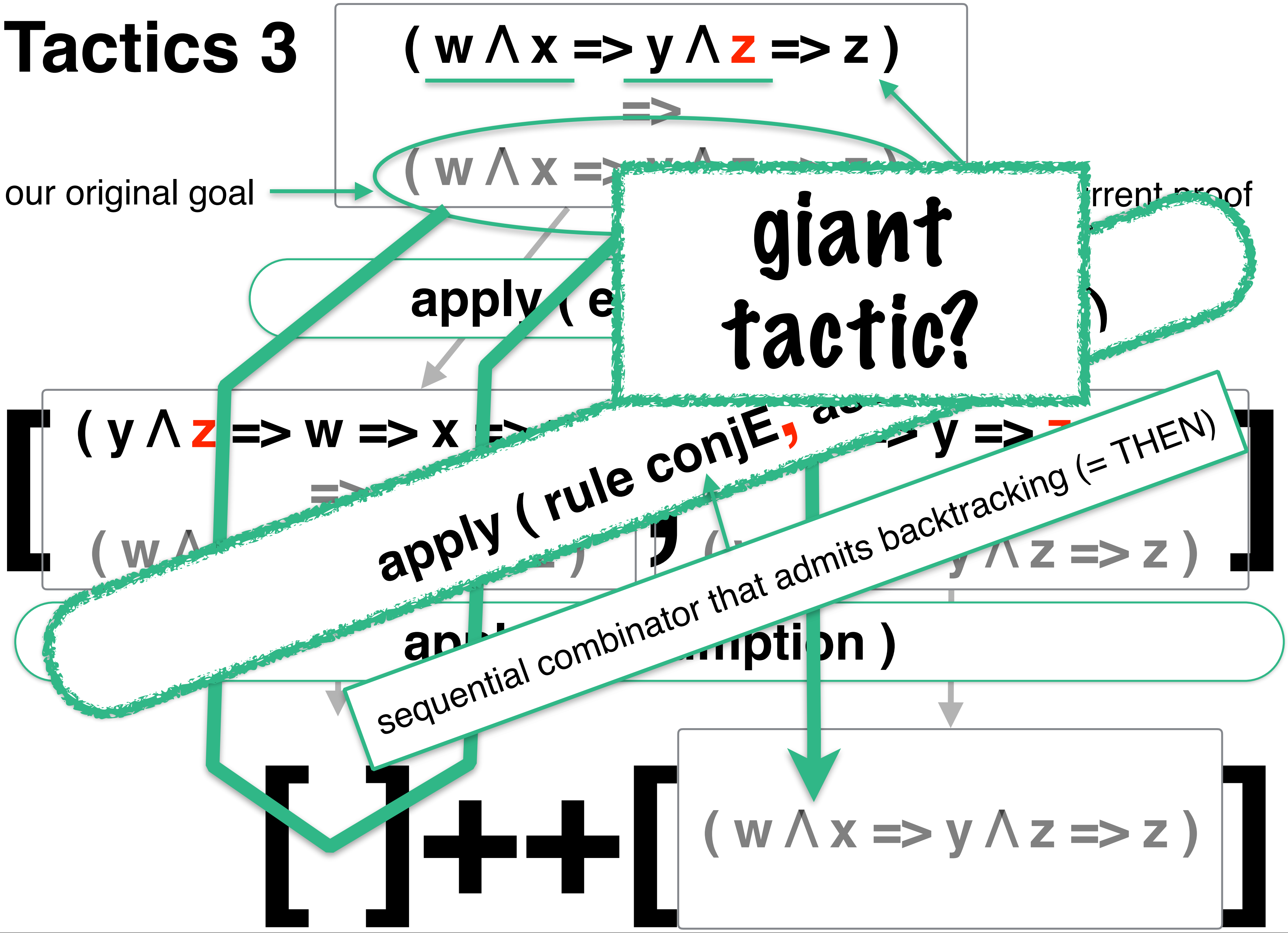
Tactics 3


(w ∧ x => y ∧ z => z)
=>
(w ∧ x => y ∧ z => z)
our original goal
our current proof obligation
apply (erule conjE)
back
[(y ∧ z => w => x => z)
=>
(w ∧ x => y ∧ z => z) ,
(w ∧ x => y => z => z)
=>
(w ∧ x => y ∧ z => z)]
apply (assumption)
[ ] ++ [(w ∧ x => y ∧ z => z)]

Tactics 3

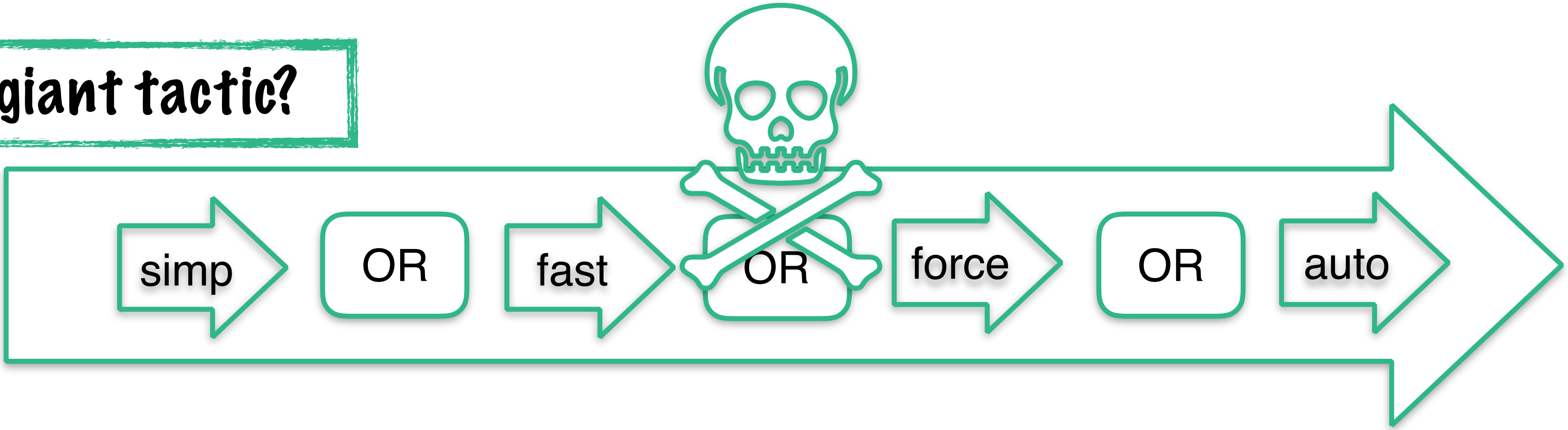

(w ∧ x => y ∧ z => z)
=>
(w ∧ x => y ∧ z => z)
our original goal
our current proof obligation
apply (erule conjE)
[(y ∧ z => w => x => ... => y => z ...]
=>
(w ∧ ... z)
(... y ∧ z => z)
apply (assumption)
apply (rule conjE, assumption)
sequential combinator that admits backtracking (= THEN)
[ ] ++ [(w ∧ x => y ∧ z => z)]

Tactics 3


(w ∧ x => y ∧ z => z)
=>
(w ∧ x =>
our original goal
rrent proof
giant tactic?
apply (e
apply (rule conjE, as
sequential combinator that admits backtracking (= THEN)
[(y ∧ z => w => x =>
=> y => z
(w ∧
y ∧ z => z)]
ap
mption)
[ ] ++ [(w ∧ x => y ∧ z => z)]

Giant tactic

giant tactic?



problem 1: Default tactics are too weak!

problem 2: Giant tactics are too slow!

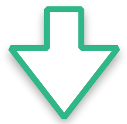
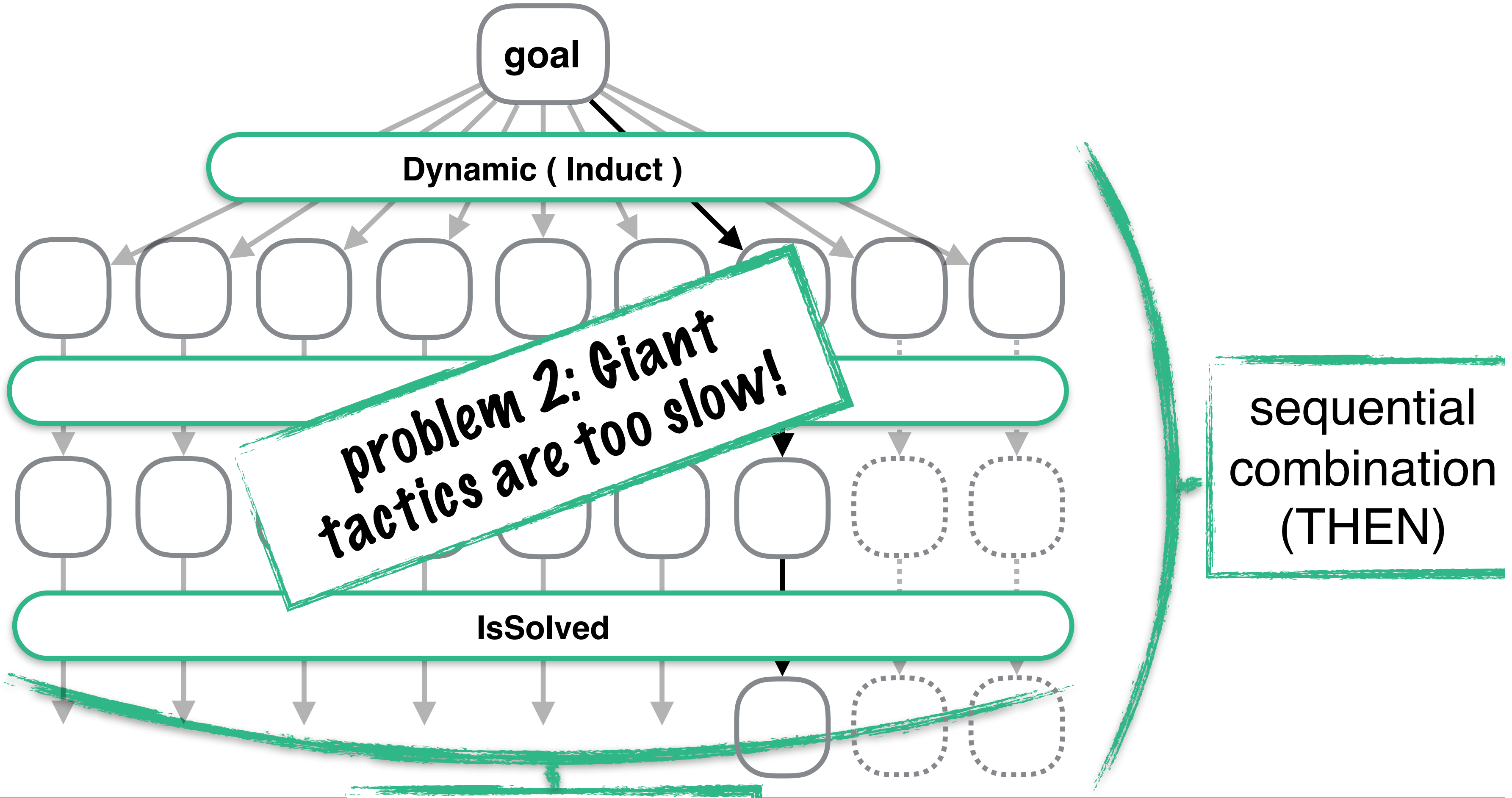
problem 3: Sledgehammer and quick-check are not tactics!

problem 1: Default tactics are too weak!

Thens [Dynamic(Induct), Auto, IsSolved]

runtime interpretation

(InductA ++ InductB ++ …) THEN auto THEN is_solved



goal
Dynamic (Induct)
IsSolved
problem 2: Giant tactics are too slow!
sequential combination (THEN)

non-determinism

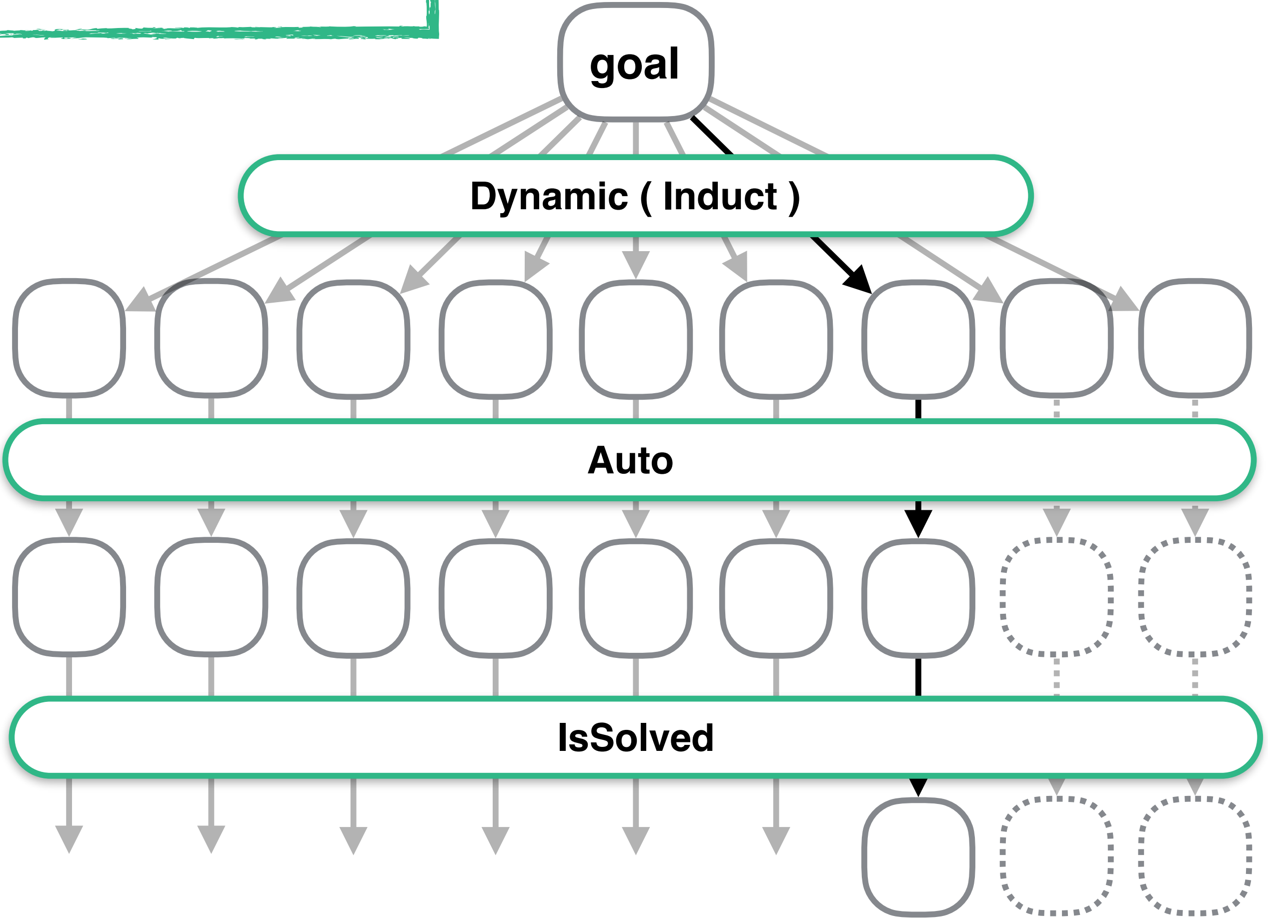
problem 2: Giant tactics are too slow!

truncating backtracked steps is hard!

type tactic = thm -> thm Seq.seq

explicit tree construction?

pointer?



problem 2: Giant tactics are too slow!

truncating backtracked steps is hard!

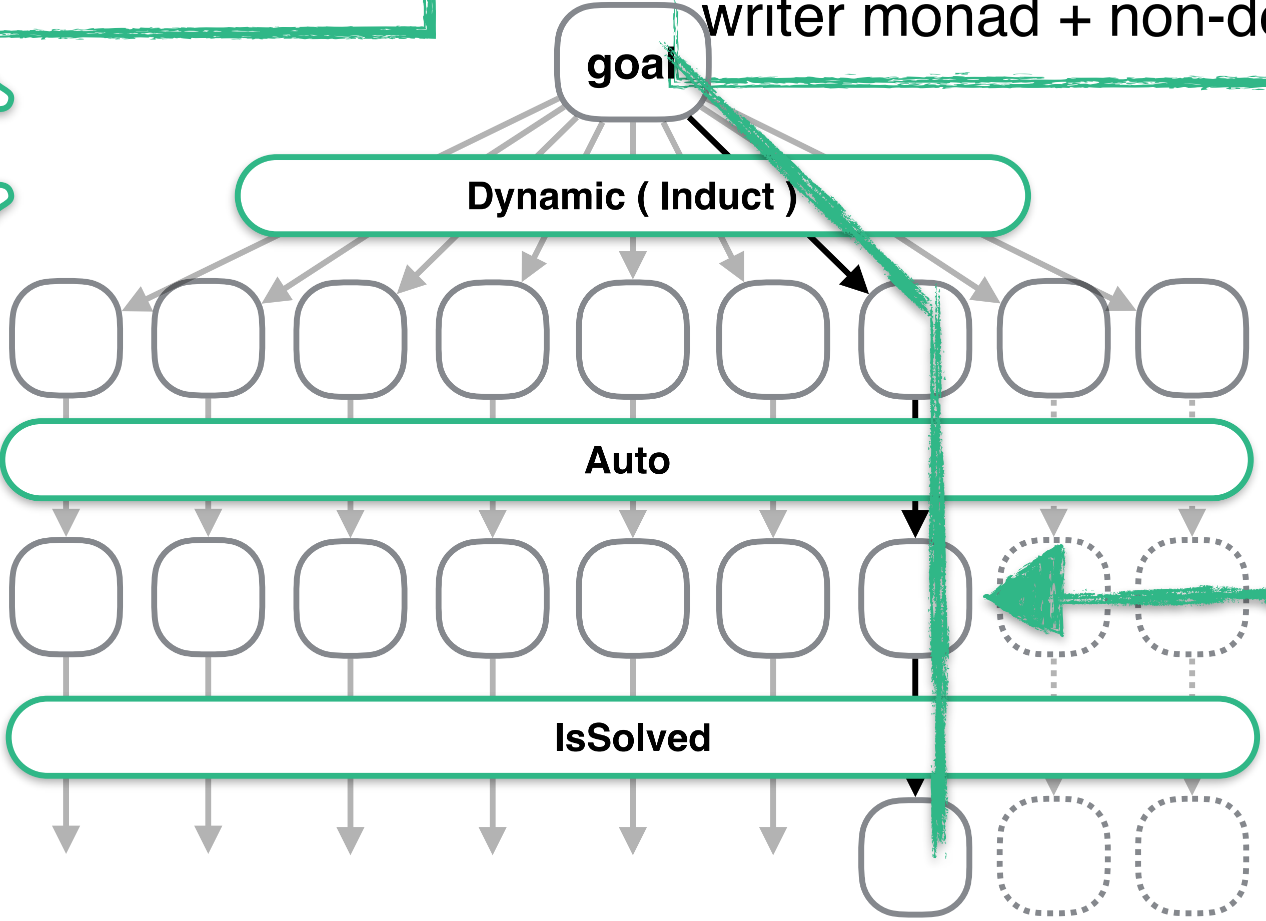
type tactic = thm -> thm Seq.seq

type 'a tactic = 'a -> 'a monad

exp[illegible] [illegible]ee construction?

poin[illegible]?

writer monad + non-deterministic monad

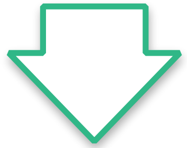


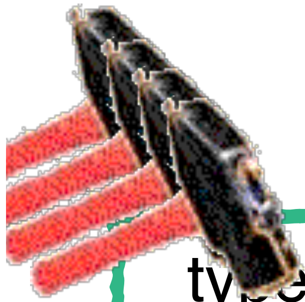
efficient proof scripts as "state"

problem 3: Sledgehammer and quick-check are not tactics!

They work on Proof.state not on thm.

```
type ‘a tactic  = 'a -> ‘a nondet_state_monad
```





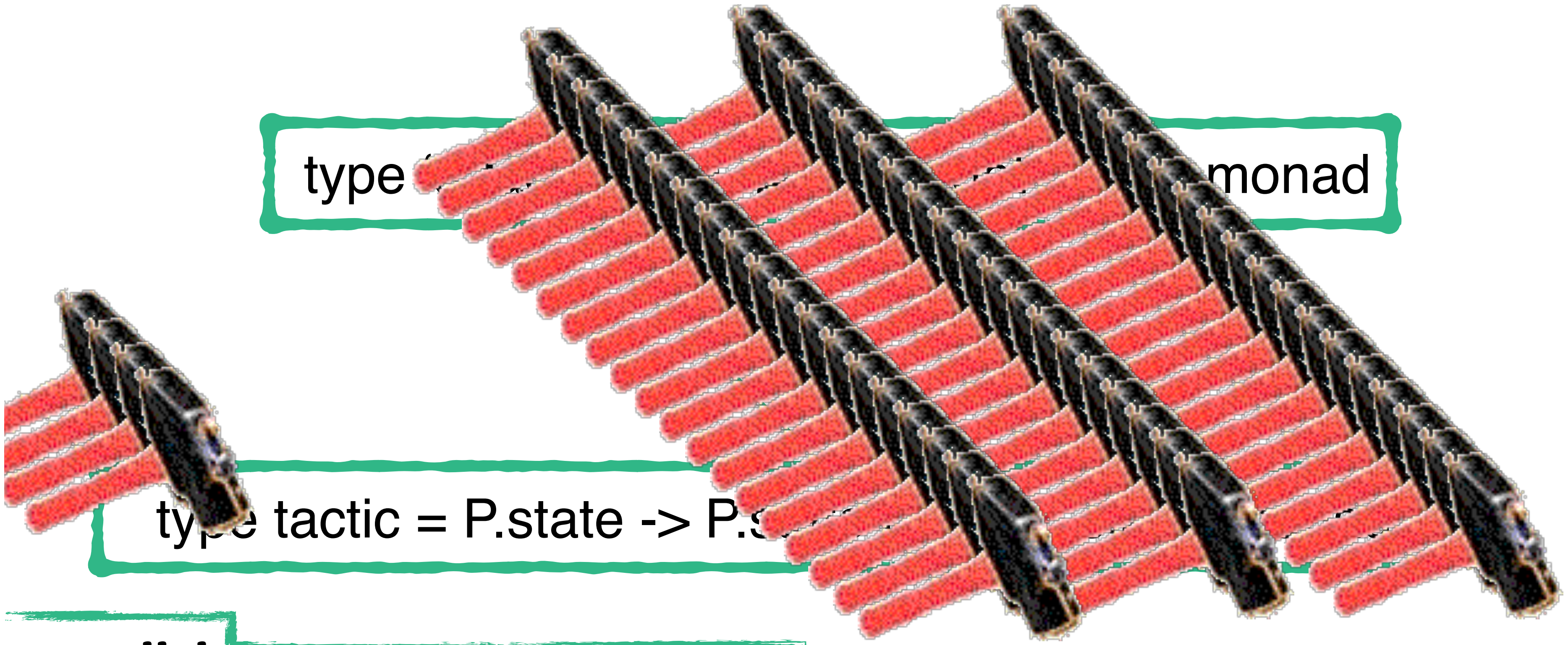
```
type tactic = P.state -> P.state nondet_state_monad
```

persistant hammering

```
Thens [Dynamic (Induct), Thens[Hammer+ , IsSolved]]
```

problem 3: Sledgehammer and quick-check are not tactics!

They work on Proof.state not on thm.



type monad

type tactic = P.state -> P.s

parallel

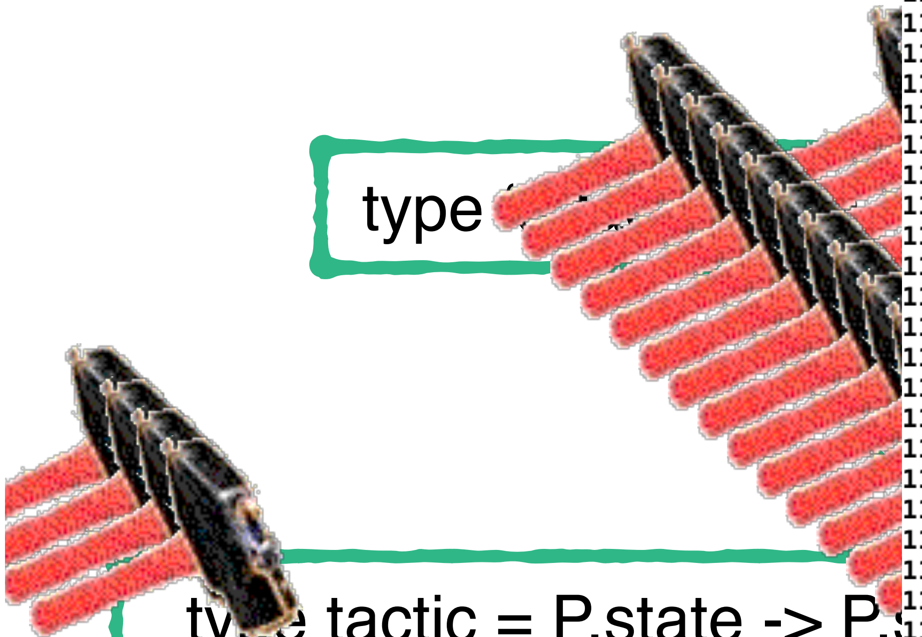
persistant hammering

PThenOne ~~Thens~~ [Dynamic (Induct), Thens[Hammer+ , IsSolved]]

problem 3: Sledgehammer and

They work o

type

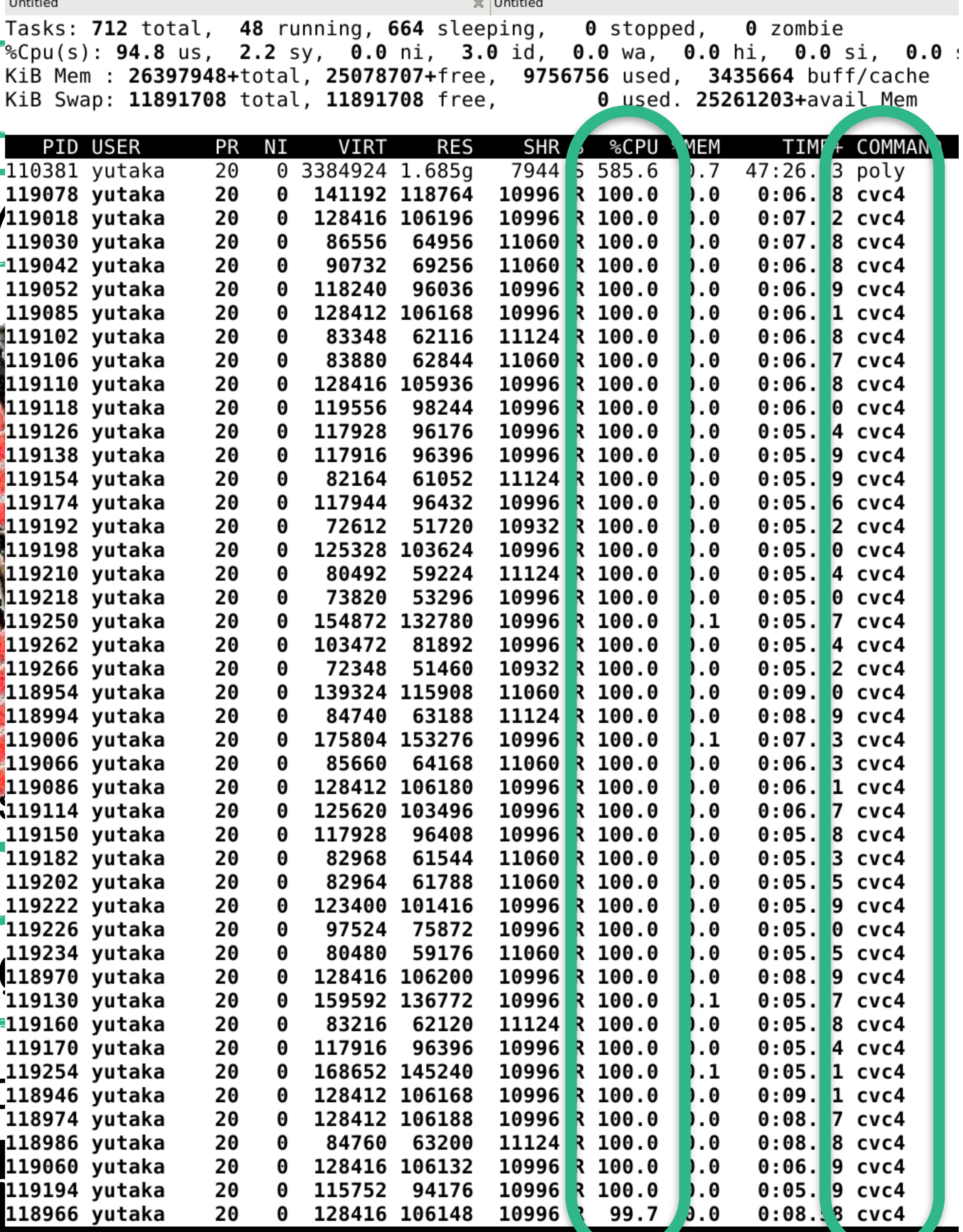


type tactic = P.state -> P.s

parallel

persistant hammerin

PThenOne ~~Thens~~ [Dynamic



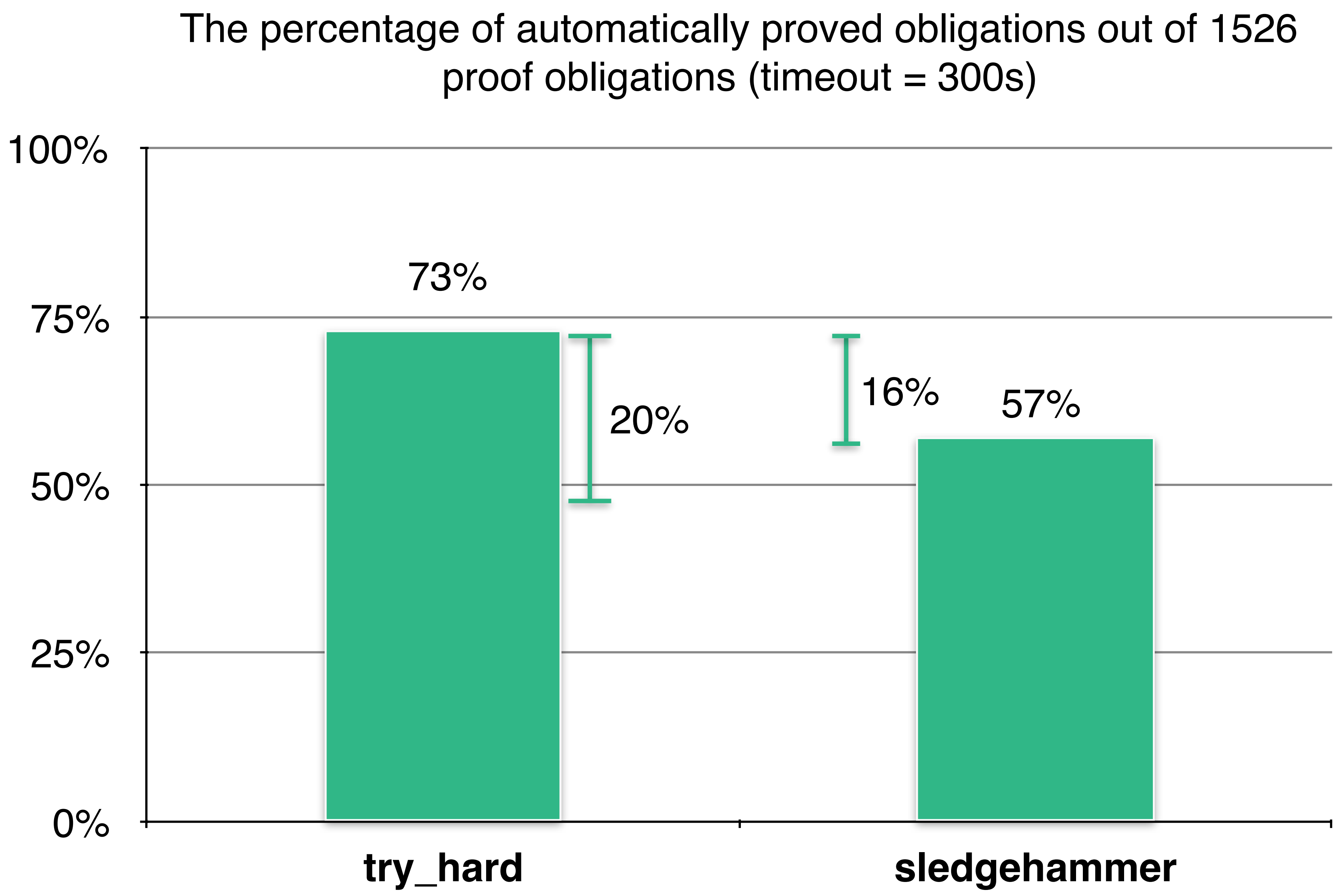
```
Untitled                                              Untitled
Tasks: 712 total,  48 running, 664 sleeping,   0 stopped,   0 zombie
%Cpu(s): 94.8 us,  2.2 sy,  0.0 ni,  3.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
KiB Mem : 26397948+total, 25078707+free,  9756756 used,  3435664 buff/cache
KiB Swap: 11891708 total, 11891708 free,        0 used. 25261203+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAN
110381 yutaka    20   0 3384924 1.685g   7944 S 585.6  .7  47:26.3 poly
119078 yutaka    20   0  141192 118764  10996 R 100.0  .0   0:06.8 cvc4
119018 yutaka    20   0  128416 106196  10996 R 100.0  .0   0:07.2 cvc4
119030 yutaka    20   0   86556  64956  11060 R 100.0  .0   0:07.8 cvc4
119042 yutaka    20   0   90732  69256  11060 R 100.0  .0   0:06.8 cvc4
119052 yutaka    20   0  118240  96036  10996 R 100.0  .0   0:06.9 cvc4
119085 yutaka    20   0  128412 106168  10996 R 100.0  .0   0:06.1 cvc4
119102 yutaka    20   0   83348  62116  11124 R 100.0  .0   0:06.8 cvc4
119106 yutaka    20   0   83880  62844  11060 R 100.0  .0   0:06.7 cvc4
119110 yutaka    20   0  128416 105936  10996 R 100.0  .0   0:06.8 cvc4
119118 yutaka    20   0  119556  98244  10996 R 100.0  .0   0:06.0 cvc4
119126 yutaka    20   0  117928  96176  10996 R 100.0  .0   0:05.4 cvc4
119138 yutaka    20   0  117916  96396  10996 R 100.0  .0   0:05.9 cvc4
119154 yutaka    20   0   82164  61052  11124 R 100.0  .0   0:05.9 cvc4
119174 yutaka    20   0  117944  96432  10996 R 100.0  .0   0:05.6 cvc4
119192 yutaka    20   0   72612  51720  10932 R 100.0  .0   0:05.2 cvc4
119198 yutaka    20   0  125328 103624  10996 R 100.0  .0   0:05.0 cvc4
119210 yutaka    20   0   80492  59224  11124 R 100.0  .0   0:05.4 cvc4
119218 yutaka    20   0   73820  53296  10996 R 100.0  .0   0:05.0 cvc4
119250 yutaka    20   0  154872 132780  10996 R 100.0  .1   0:05.7 cvc4
119262 yutaka    20   0  103472  81892  10996 R 100.0  .0   0:05.4 cvc4
119266 yutaka    20   0   72348  51460  10932 R 100.0  .0   0:05.2 cvc4
118954 yutaka    20   0  139324 115908  11060 R 100.0  .0   0:09.0 cvc4
118994 yutaka    20   0   84740  63188  11124 R 100.0  .0   0:08.9 cvc4
119006 yutaka    20   0  175804 153276  10996 R 100.0  .1   0:07.3 cvc4
119066 yutaka    20   0   85660  64168  11060 R 100.0  .0   0:06.3 cvc4
119086 yutaka    20   0  128412 106180  10996 R 100.0  .0   0:06.1 cvc4
119114 yutaka    20   0  125620 103496  10996 R 100.0  .0   0:06.7 cvc4
119150 yutaka    20   0  117928  96408  10996 R 100.0  .0   0:05.8 cvc4
119182 yutaka    20   0   82968  61544  11060 R 100.0  .0   0:05.3 cvc4
119202 yutaka    20   0   82964  61788  11060 R 100.0  .0   0:05.5 cvc4
119222 yutaka    20   0  123400 101416  10996 R 100.0  .0   0:05.9 cvc4
119226 yutaka    20   0   97524  75872  10996 R 100.0  .0   0:05.0 cvc4
119234 yutaka    20   0   80480  59176  11060 R 100.0  .0   0:05.5 cvc4
118970 yutaka    20   0  128416 106200  10996 R 100.0  .0   0:08.9 cvc4
119130 yutaka    20   0  159592 136772  10996 R 100.0  .1   0:05.7 cvc4
119160 yutaka    20   0   83216  62120  11124 R 100.0  .0   0:05.8 cvc4
119170 yutaka    20   0  117916  96396  10996 R 100.0  .0   0:05.4 cvc4
119254 yutaka    20   0  168652 145240  10996 R 100.0  .1   0:05.1 cvc4
118946 yutaka    20   0  128412 106168  10996 R 100.0  .0   0:09.1 cvc4
118974 yutaka    20   0  128412 106188  10996 R 100.0  .0   0:08.7 cvc4
118986 yutaka    20   0   84760  63200  11124 R 100.0  .0   0:08.8 cvc4
119060 yutaka    20   0  128416 106132  10996 R 100.0  .0   0:06.9 cvc4
119194 yutaka    20   0  115752  94176  10996 R 100.0  .0   0:05.9 cvc4
118966 yutaka    20   0  128416 106148  10996    99.7  .0   0:08.8 cvc4
```

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] ]
```

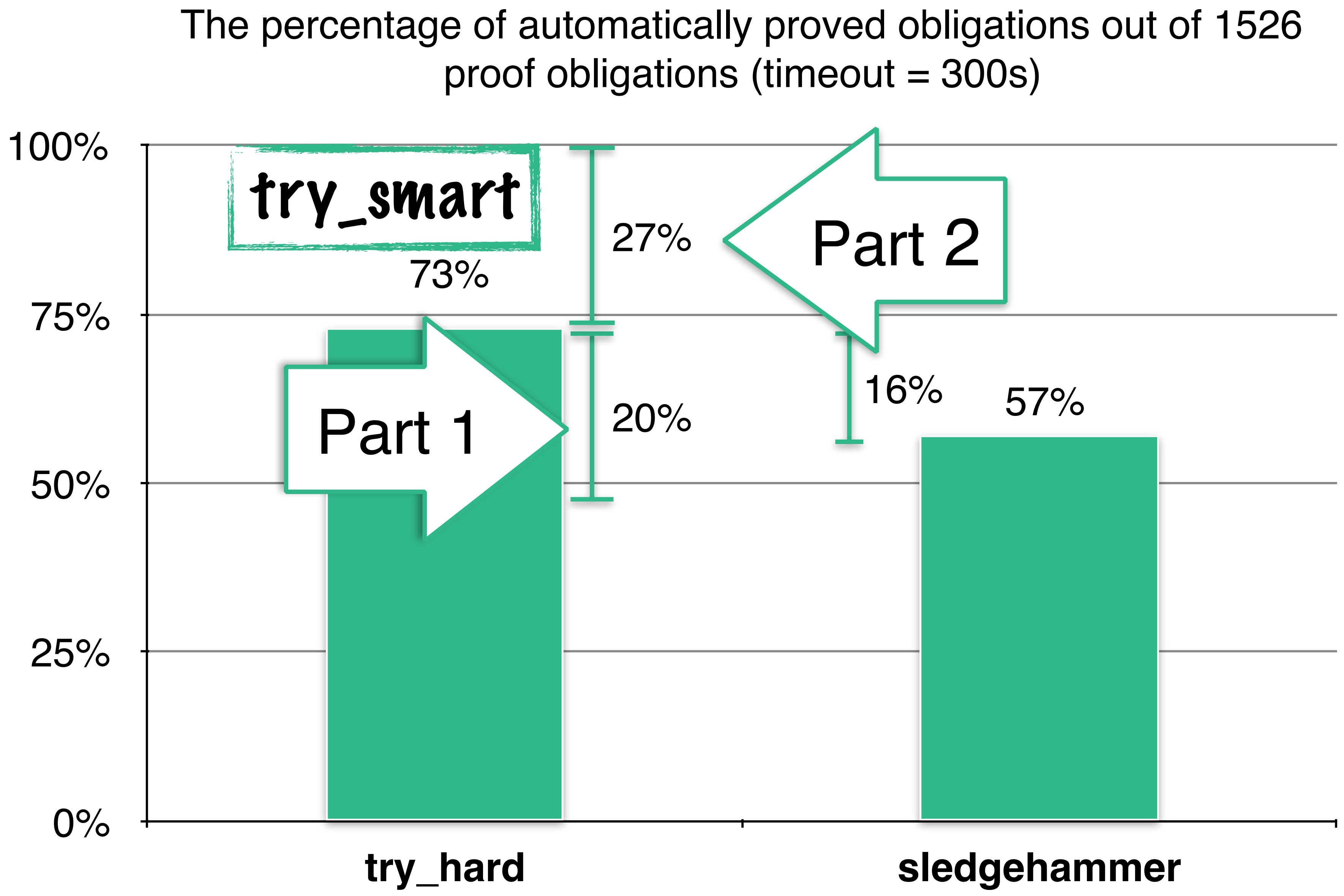
```
strategy Basic =
  Ors [
      Auto_Solve,
      Blast_Solve,
      FF_Solve,
      Thens [IntroClasses, Auto_Solve],
      Thens [Transfer, Auto_Solve],
      Thens [Normalization, IsSolved],
      Thens [DInduct, Auto_Solve],
      Thens [Hammer, IsSolved],
      Thens [DCases, Auto_Solve],
  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

PSL and try-hard for Isabelle/HOL



Demo

PSL and try-hard for Isabelle/HOL



What’s wrong with try_hard?

Huge search space with little intelligence

special purpose tools

```
strategy Basi
  Ors [
       Auto_Sol
       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]]
```

```
tegy 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] ]
```

What's wrong with try_hard?

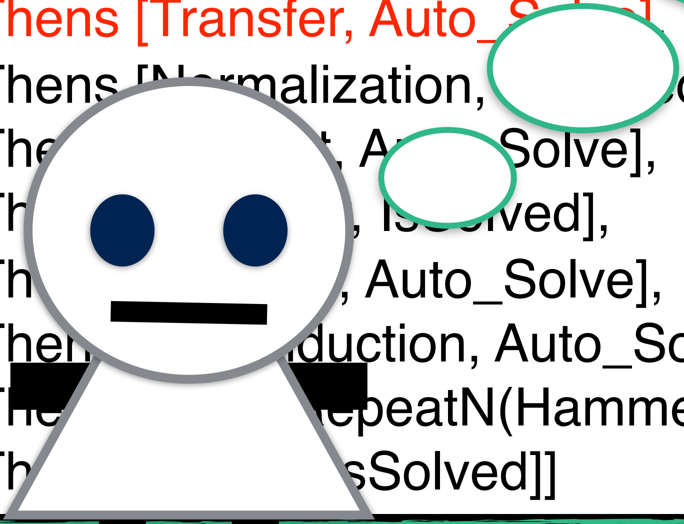
Huge search space with little intelligence

```
strategy Basi
  Ors [
     Auto_Sol
     Blast_Solve,
     FF_Solve,
     Thens [IntroClasses, Auto_
     Thens [Transfer, Auto_S    ]
     Thens [Normalization,          d],
     The           t, A      Solve],
     Th                  , IsSolved],
     Th                  , Auto_Solve],
     Then        duction, Auto_Solve],
     The             peatN(Hammer), IsSolved],
     Th                    sSolved]]
```

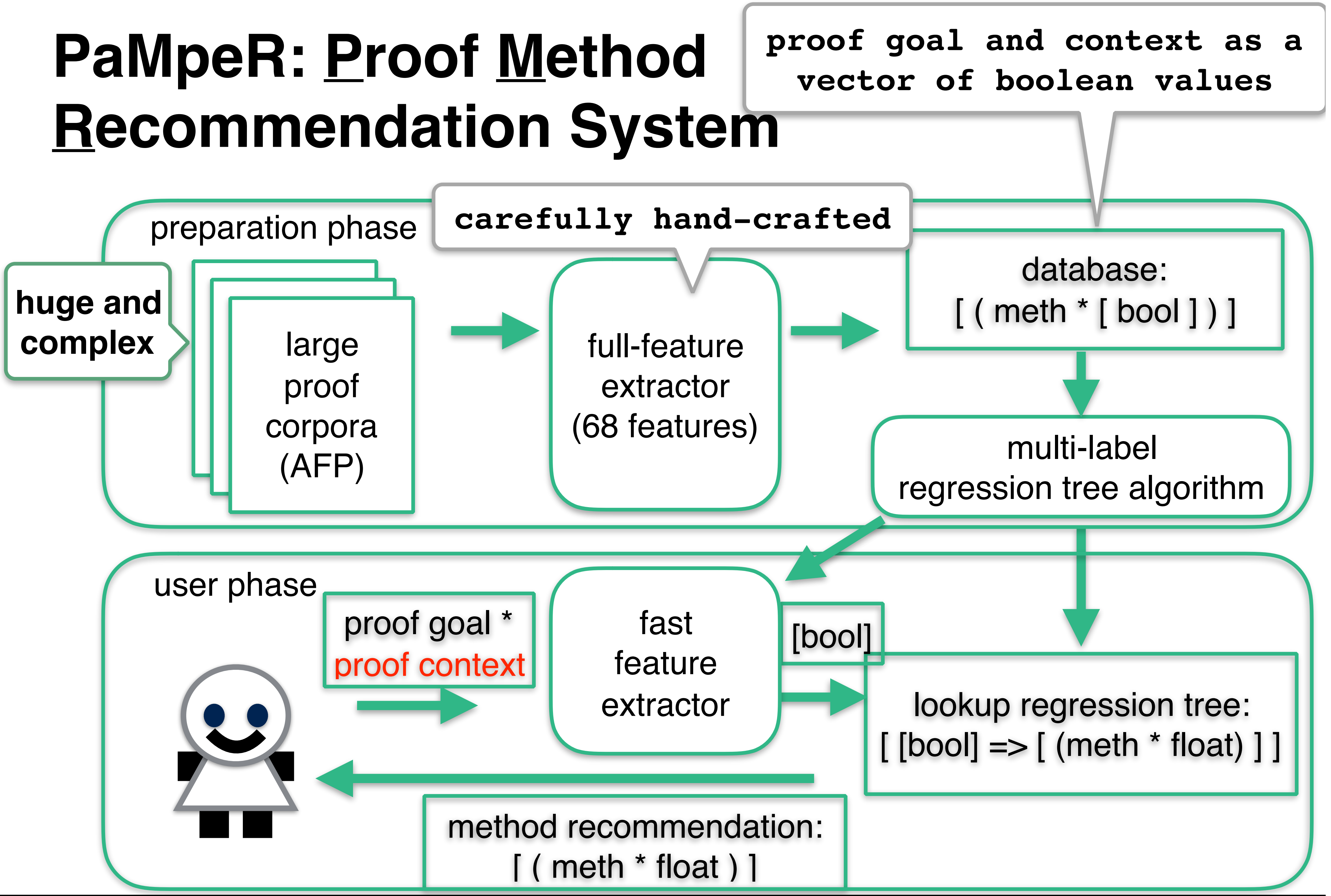
special

```
                                ],
                             nced],
     Thens [DCaseTac, Solve_Many],
     Thens [DInductTac, Solve_Many] ]
```

Can we guess which tool to use based on the meta-information and information in the standard library?



PaMpeR: Proof Method Recommendation System



proof goal and context as a vector of boolean values
preparation phase
carefully hand-crafted
huge and complex
large proof corpora (AFP)
full-feature extractor (68 features)
database: [(meth * [bool])]
multi-label regression tree algorithm
user phase
proof goal * proof context
fast feature extractor
[bool]
lookup regression tree: [[bool] => [(meth * float)]]
method recommendation: [(meth * float)]

Hand-crafted feature ?

defined in Isabelle/HOL

assertions about proof goal

Example1: Is the outermost constant ∧?

```
lemma "P x ∧ Q y"
```

user defined constant ?

assertions about proof goal and its context

Example2: constants related to corecursion?

defined by a user

```
theorem Plus_ZeroL[simp]: "Plus Zero r = r"
```

Hand-crafted feature ?

defined in Isabelle/HOL

assertions about proof goal

Example1: Is the outermost constant V̲?

```
lemma "P x ∧ Q y"
```

user defined constant ?

assertions about proof goal and its context

Example2: constants related to corecursion?

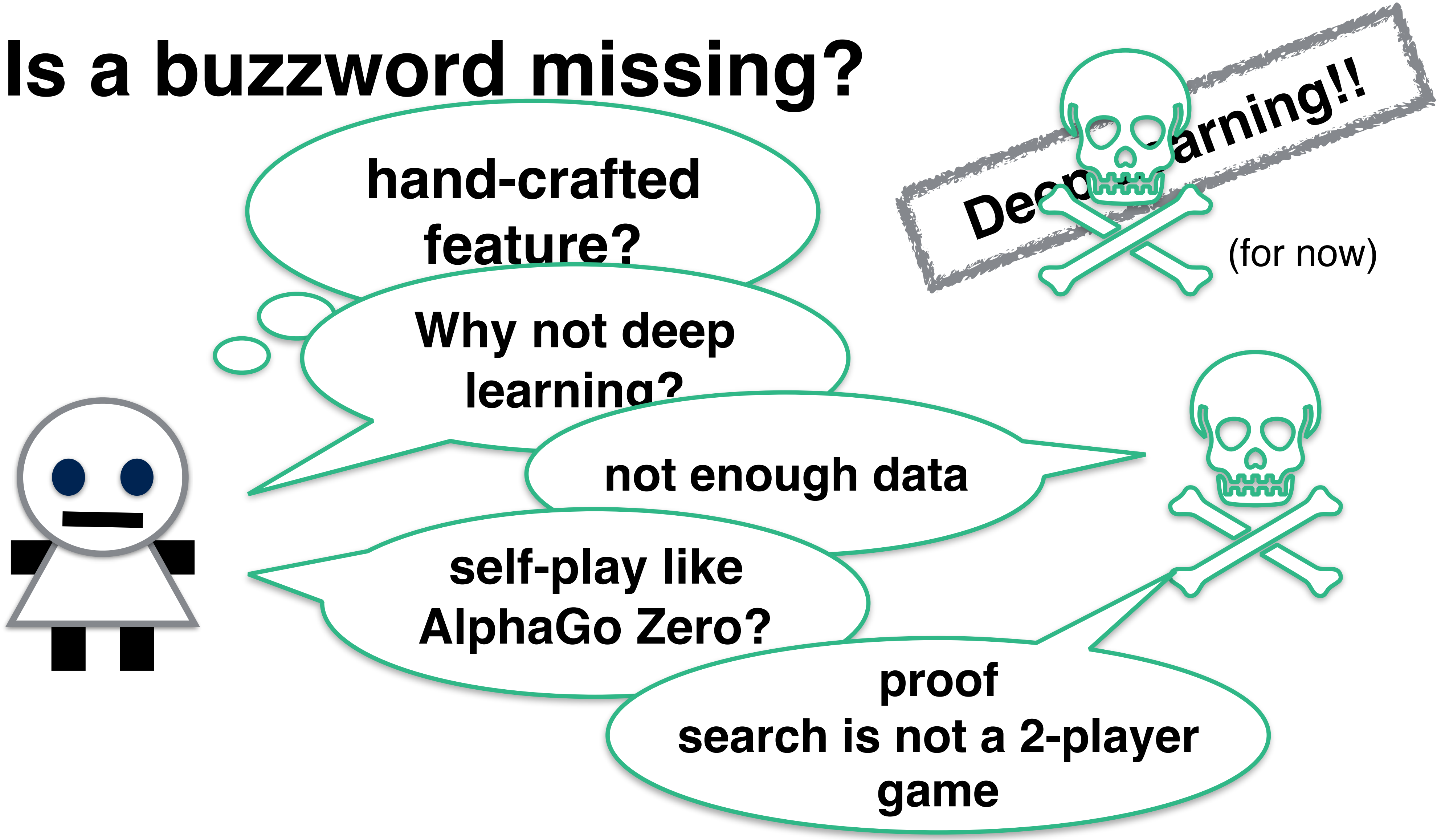
= if the context has theorems called Plus.code, Plus.ctr, Plus.sel.

```
primcorec Plus :: "'a language ⇒ 'a language ⇒ 'a
  "о (Plus r s) = (о r ∨ о s)"
| "∂ (Plus r s) = (λa. Plus (∂ r a) (∂ s a))"
theorem Plus_ZeroL[simp]: "Plus Zero r = r"
```

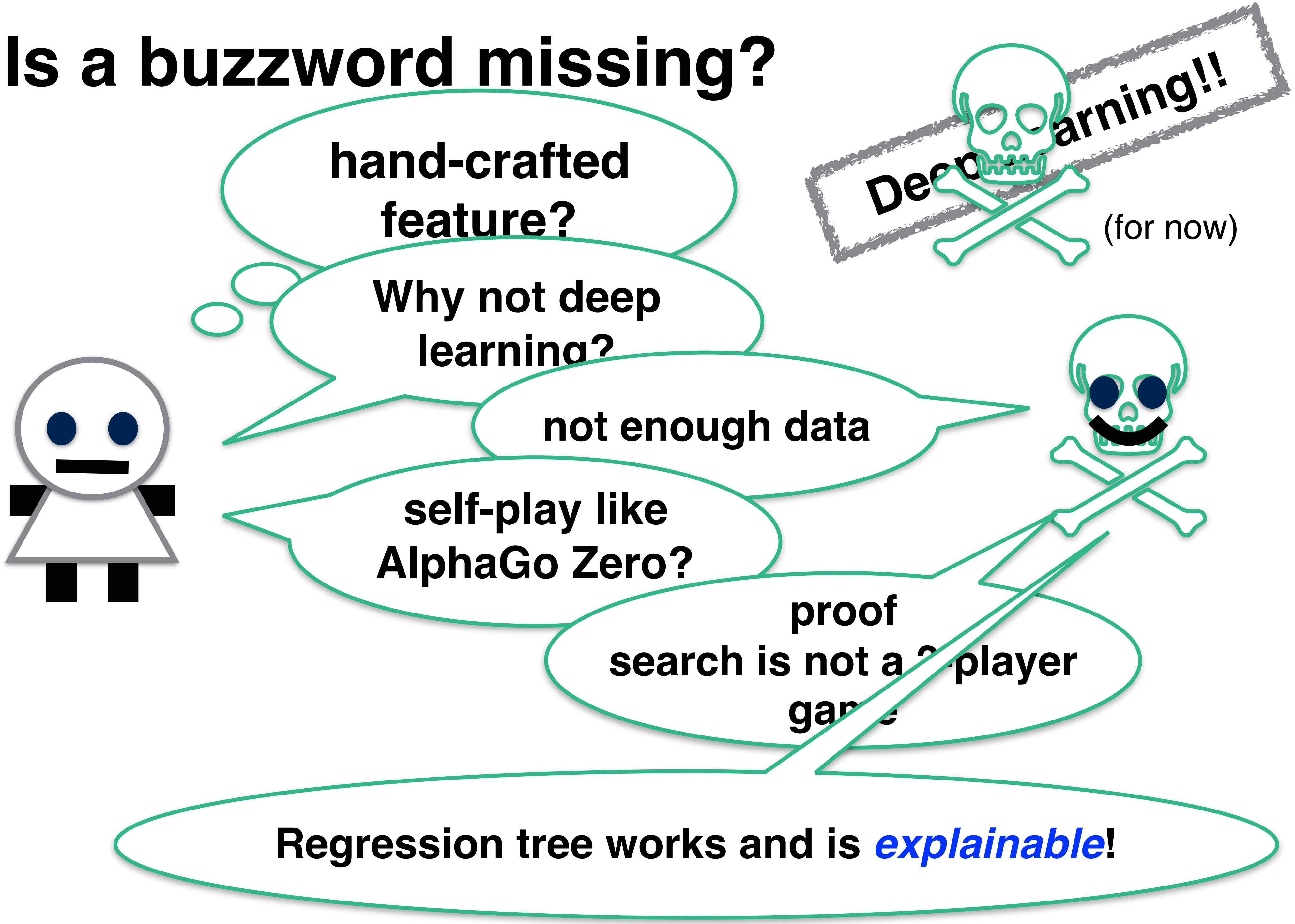
Is a buzzword missing?

Deep Learning!!
hand-crafted feature?
Why not deep learning?
not enough data
self-play like AlphaGo Zero?
proof search is not a 2-player game

Is a buzzword missing?



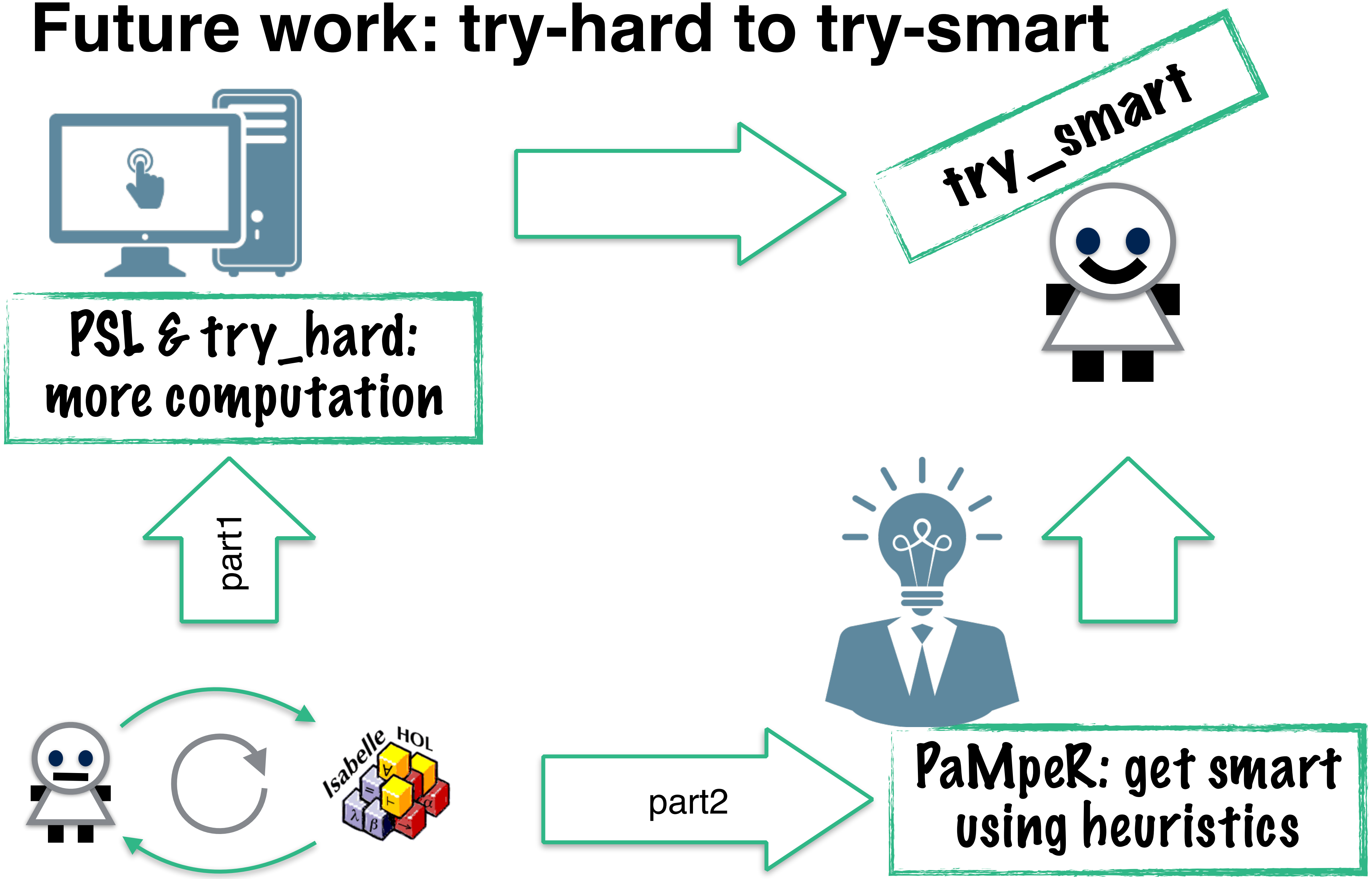
Is a buzzword missing?



Deep Learning!!
(for now)
hand-crafted feature?
Why not deep learning?
not enough data
self-play like AlphaGo Zero?
proof search is not a 2-player game
Regression tree works and is *explainable*!

Demo

Future work: try-hard to try-smart


try_smart
PSL & try_hard: more computation
part1
part2
Isabelle HOL
PaMpeR: get smart using heuristics

Thanks!

meta-tool approach

feature extractor

regression tree

programming language

extensible (Eisbach)

runtime tactic generation

efficient proof generation

extensive proof search

parallel search

native Isabelle proof script

low memory usage

easy installation

almost no code clutter!!