Machine Learning for Isabelle/HOL

This work was supported by the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).

Yutaka Nagashima
University of Innsbruck
Czech Technical University

git clone https://github.com/data61/PSL

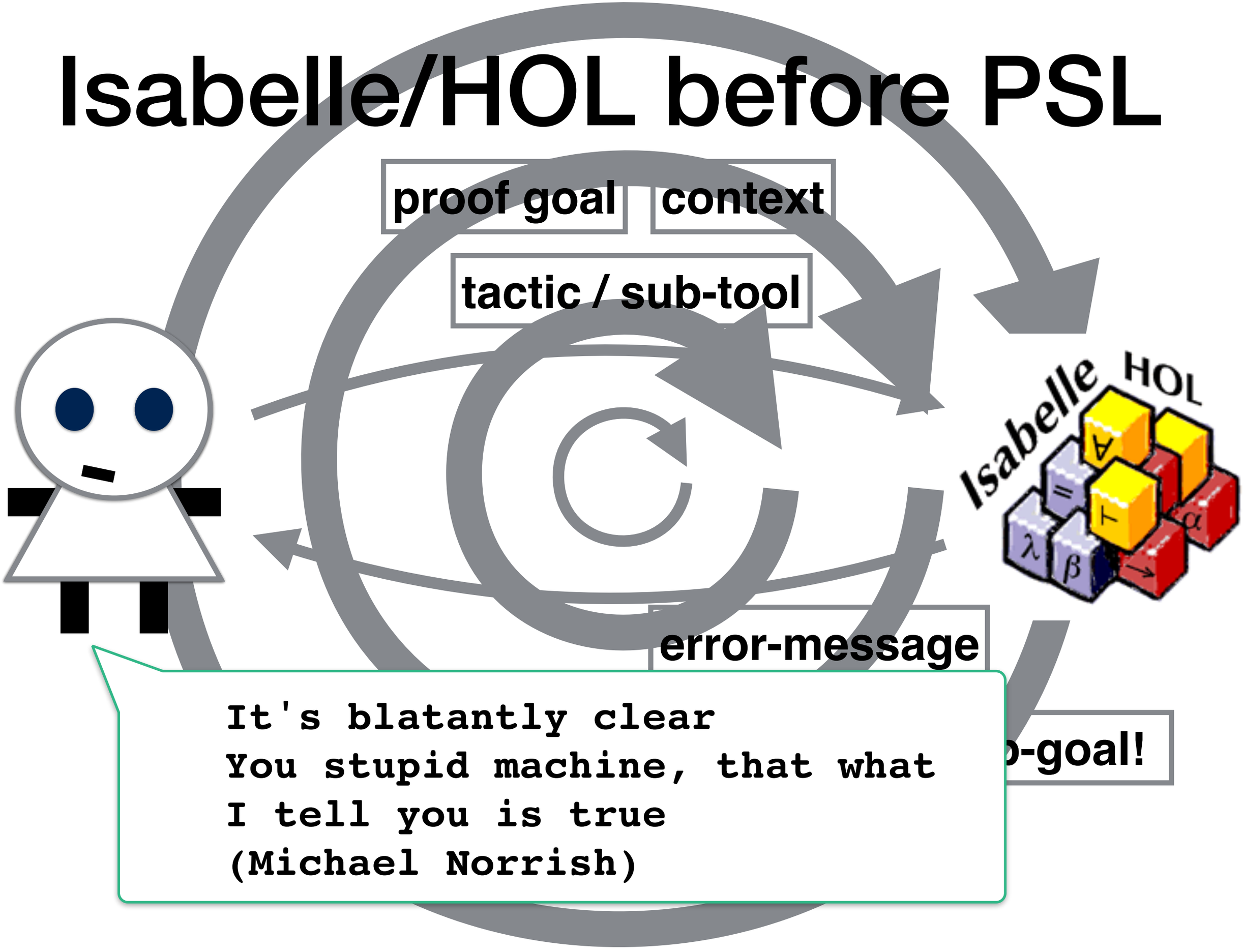
CZECH INSTITUTE
OF INFORMATICS
ROBOTICS AND
CYBERNETICS
CTU IN PRAGUE

Contents

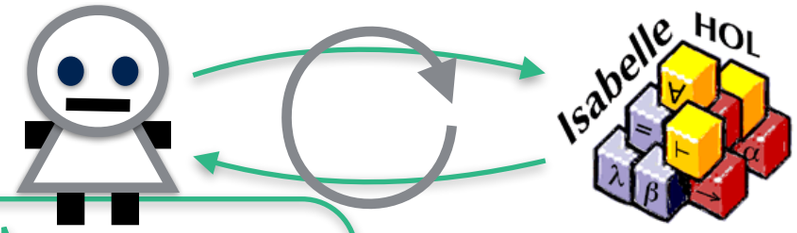
- Part I: Machine Learning for Isabelle/HOL
  - PSL (Proof Strategy Language)
  - PaMpeR (Proof Method Recommendation)
  - PGT: (Proof Goal Transformer)
- Part II: Isabelle/ML
- Exercises

https://github.com/data61/PSL

Isabelle/HOL before PSL



Tactics 1



goal → preprocess → goal → imp → goal

principle of explosion

False → imp → P

Case 1

new goal → imp → goal

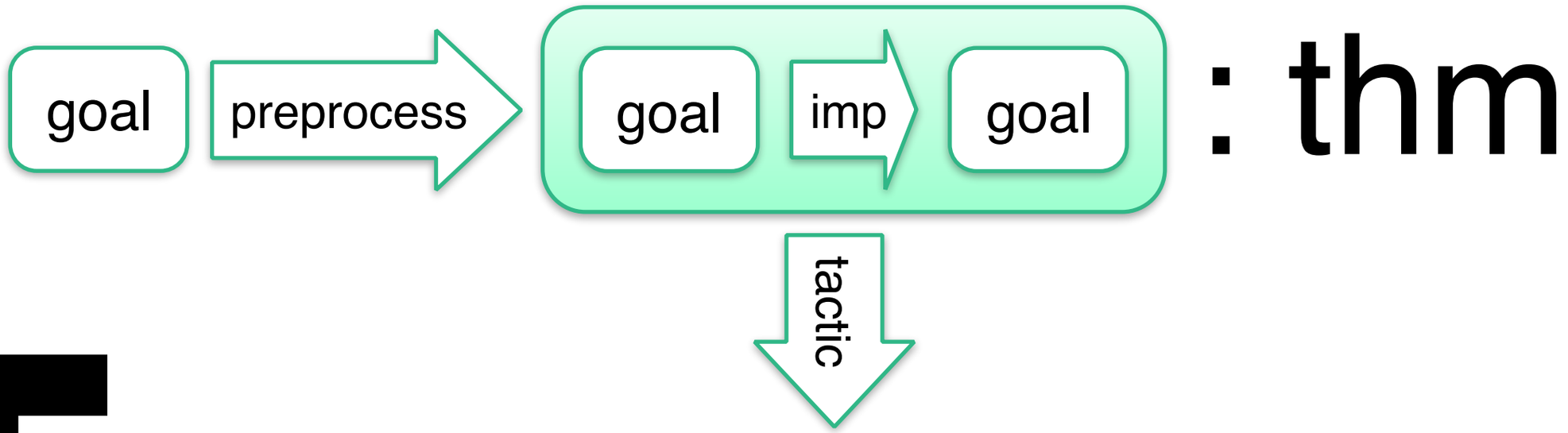
Case 2

goal

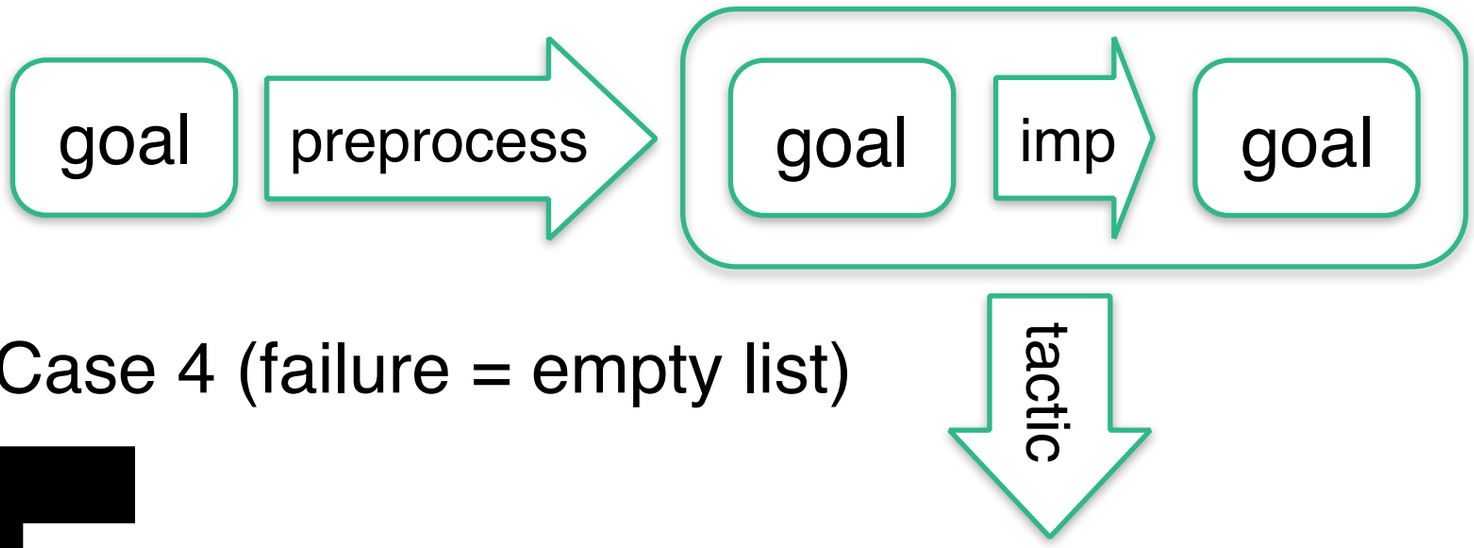
Case 3

subgoal 1 → imp → subgoal 2 → imp → ○ ○ ○ → imp → goal

Tactics 2


goal
preprocess
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 3

our original goal

(w ∧ x => y ∧ z => z)

our current proof obligati

apply (erule conjE)

(y ∧ z => w => x => _)

(w ∧ _)

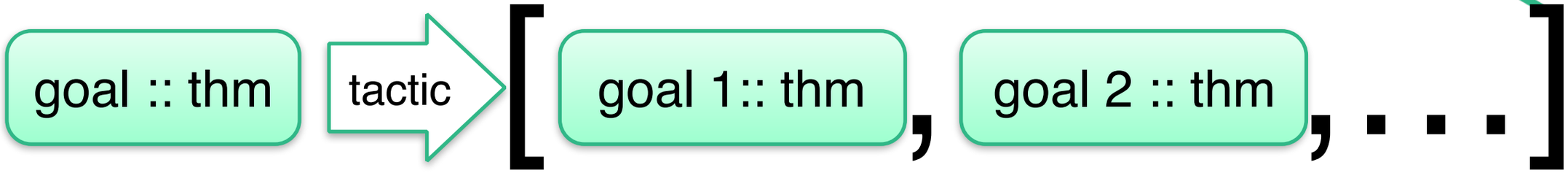
apply (rule conjE, assumption)

sequential combinator that admits backtracking (= THEN)

(assumption)

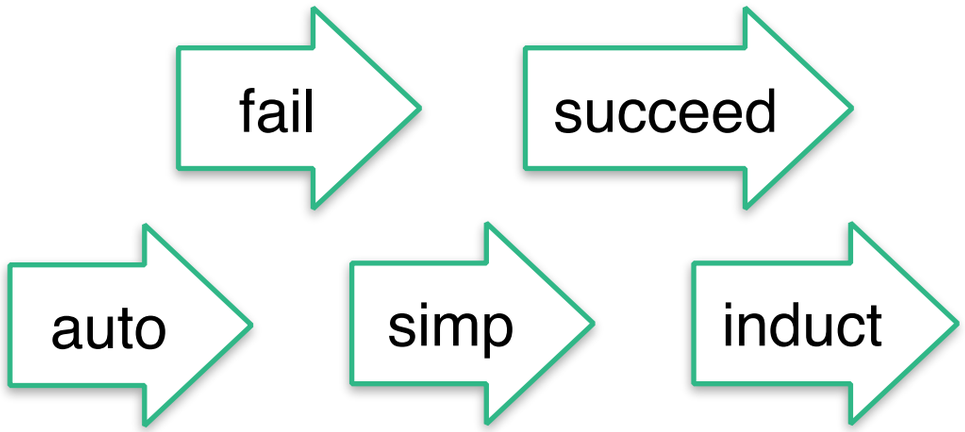
[ ]++[(w ∧ x => y ∧ z => z)]

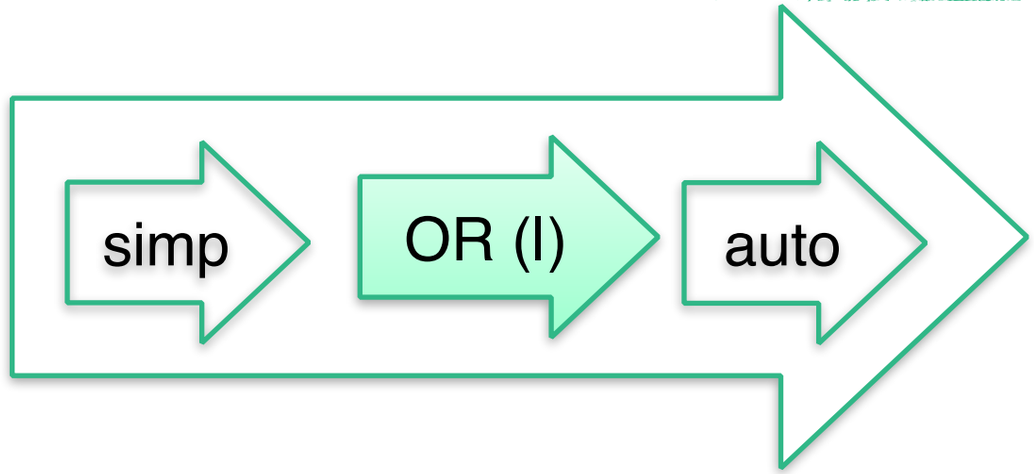
Tactics 4

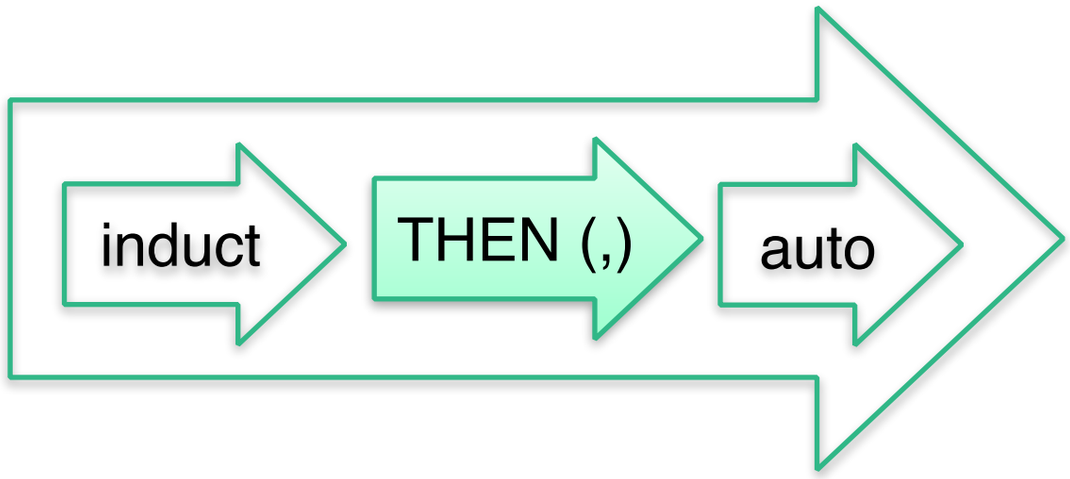


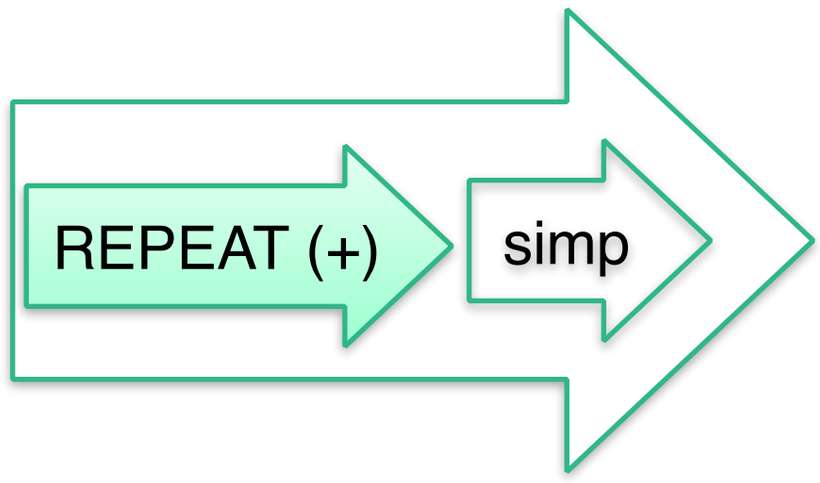
Lazy

fun tactic :: thm -> [thm]





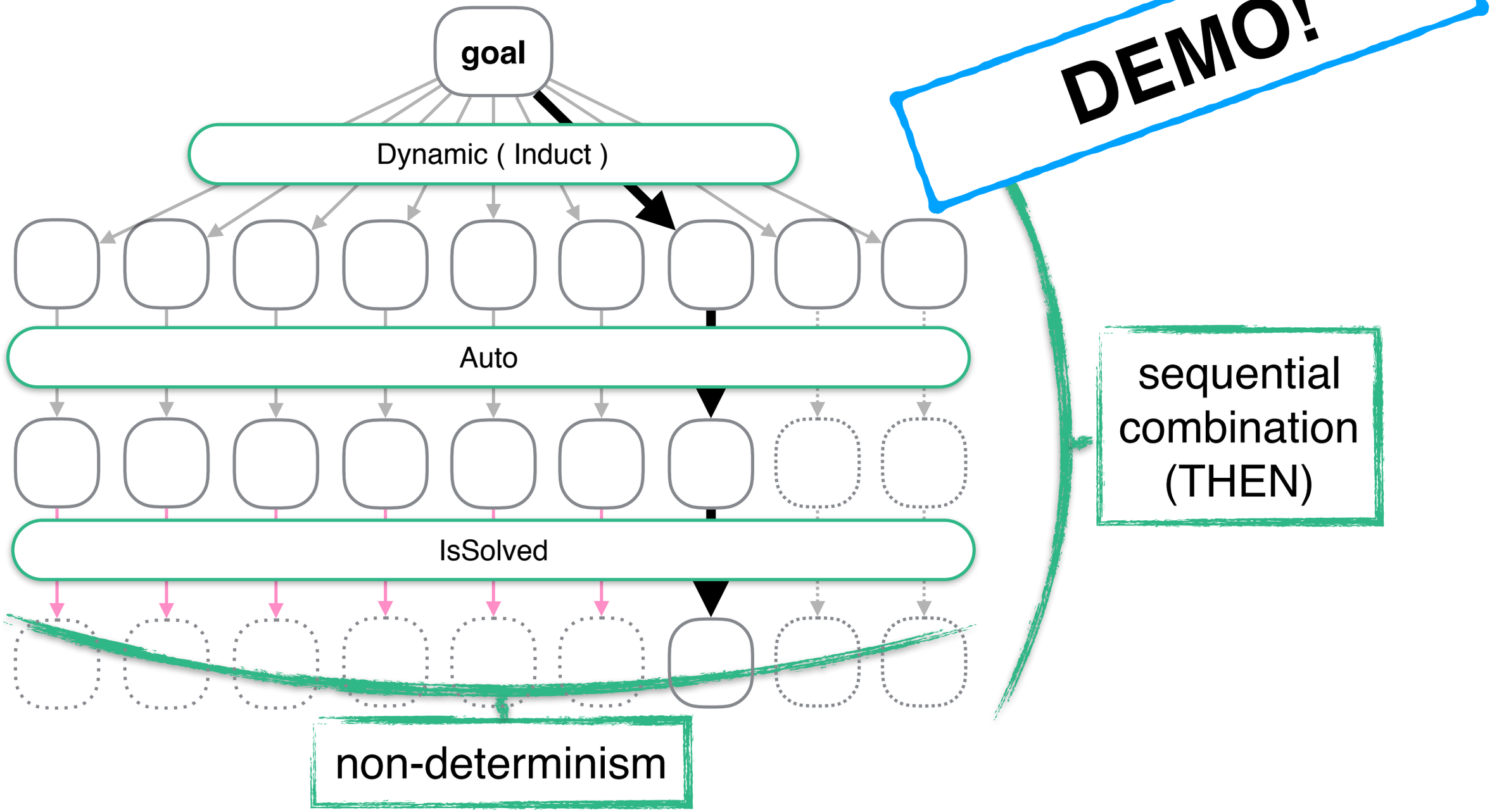




Prove "map f (sep x xs) = sep (f x) (map f xs)" using **PSL**

strategy DInd = Thens [Dynamic(Induct), Auto, IsSolved]

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



try_hard: the default strategy

```
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]]
```

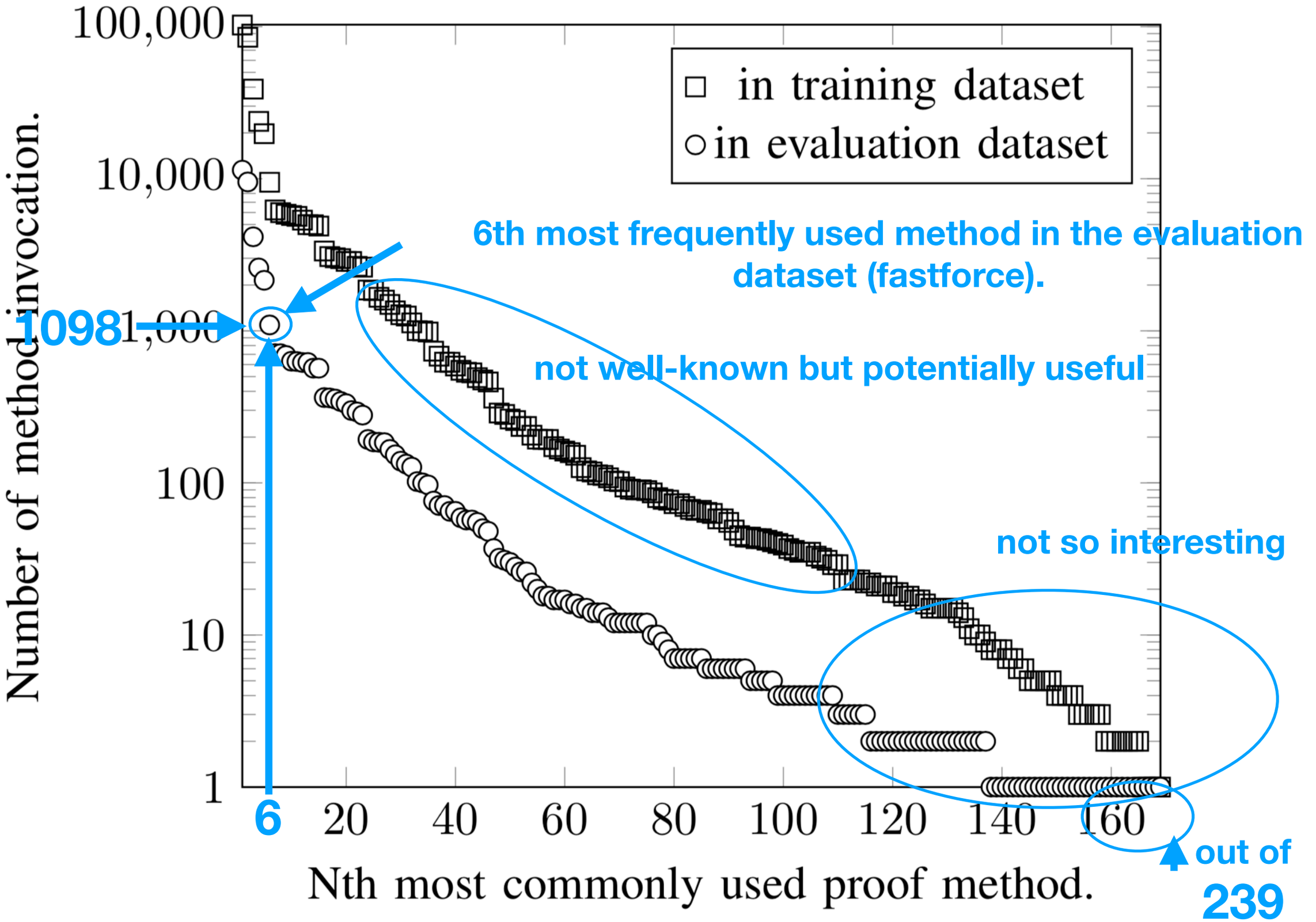
```
strategy Try_Hard =
Ors [Thens [Subgoal, Basic],
      Thens [DInductTac, Auto_Solve],
      Thens [DCaseTac, Auto_Solve],
      Thens [Subgoal, Advanced],
      Thens [DCaseTac, Solve_Many],
      Thens [DInductTac, Solve_Many] ]
```

16 percentage point performance improvement compared to sledgehammer

but the search space explodes

PaMpeR: Proof Method Recommendation

Class imbalance for tactic usage



preparation phase

large proof corpora

AFP and standard library

full feature extractor

6021 CPU hours

108 assertions

?

database (425334 data points)

:: (tactic_name, [bool])

preprocess

decision tree construction

How does PaMpeR work?

recommendation phase

proof state

proof engineer

fast feature extractor

feature vector

lookup

proof method recommendation

Feature extractor?

```
fun sep::"'a ⇒ 'a list ⇒ 'a list" where
  "sep a [] = []" |
  "sep a [x] = [x]" |
  "sep a (x#y#zs) = x # a # sep a (y#zs)"
```

DEMO!

automatically proves and saves many auxiliary lemmas in the context
sep.simps, sep.induct, sep.elims, etc.

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
```

assertion 27: if the outermost constant is the HOL equality? ✓

assertion 32: if the outermost constant is the HOL existential quantifier? ✗

assertion 93: if the goal has a term of type “real”? ✗

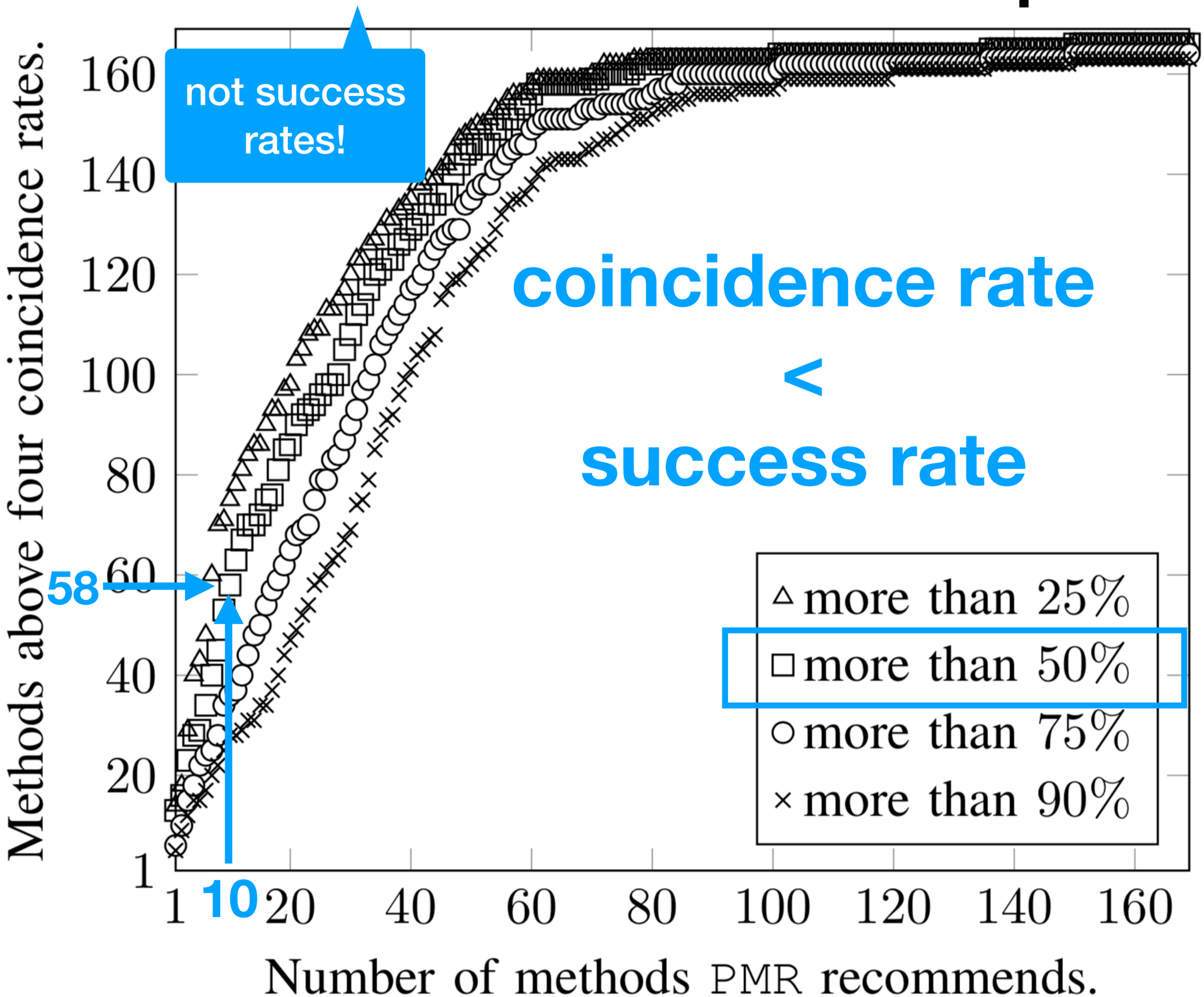
assertion 10: the context has a related recursive simplification rule? ✓

assertion 58: the context has a constant defined with the “fun” keyword? ✓

resulting feature vector: [...,1,...,1,...0,...,1,...0,...]

10th 27th 32nd 58th 93rd

Coincidence rates of PaMpeR



Success story

High coincidence rates for specialized proof methods

despite the severe class imbalance.

Too good to be true?

PaMpeR's feature extraction can be slow.

Some specialized methods are rarely useful.

Arguments of proof methods?

How to state proof goals,
so that Isabelle methods can work effectively?

PSL with PGT

proof goal sub-optimal for proof automation

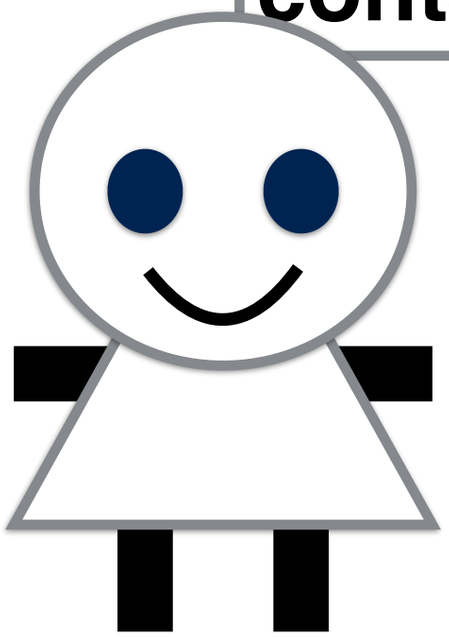
context

PGT strategy

proof goal

context

tactic / sub-tool



proof for the original goal, and auxiliary lemma optimal for proof automation

proved theorem / subgoals /

DEMO!

```
goal (1 subgoal):
 1. itrev xs [] = rev xs
```

goal

```
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
```

!?

```
goal (2 subgoals):
 1. (⋀Nil. itrev xs Nil = rev xs @ Nil) ⟹
    itrev xs [] = rev xs
 2. ⋀Nil. itrev xs Nil = rev xs @ Nil
```

Conjecture

Fastforce

```
apply fastforce
```

```
goal (1 subgoal):
 1. ⋀Nil. itrev xs Nil = rev xs @ Nil
```

Quickcheck

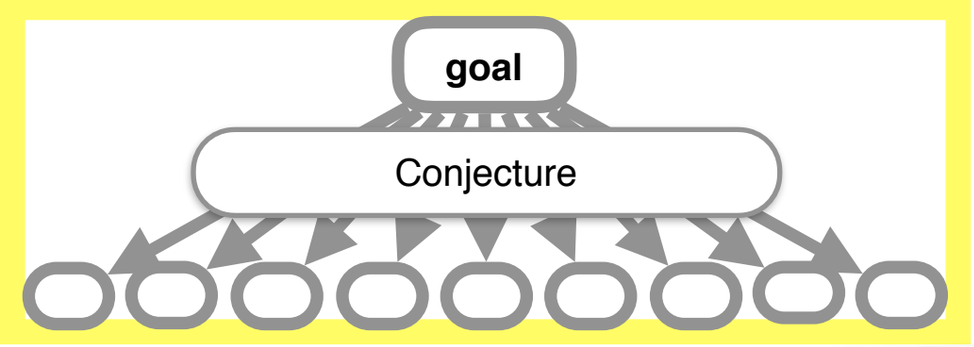
```
goal (1 subgoal):
 1. ⋀Nil. itrev xs Nil = rev xs @ Nil
```

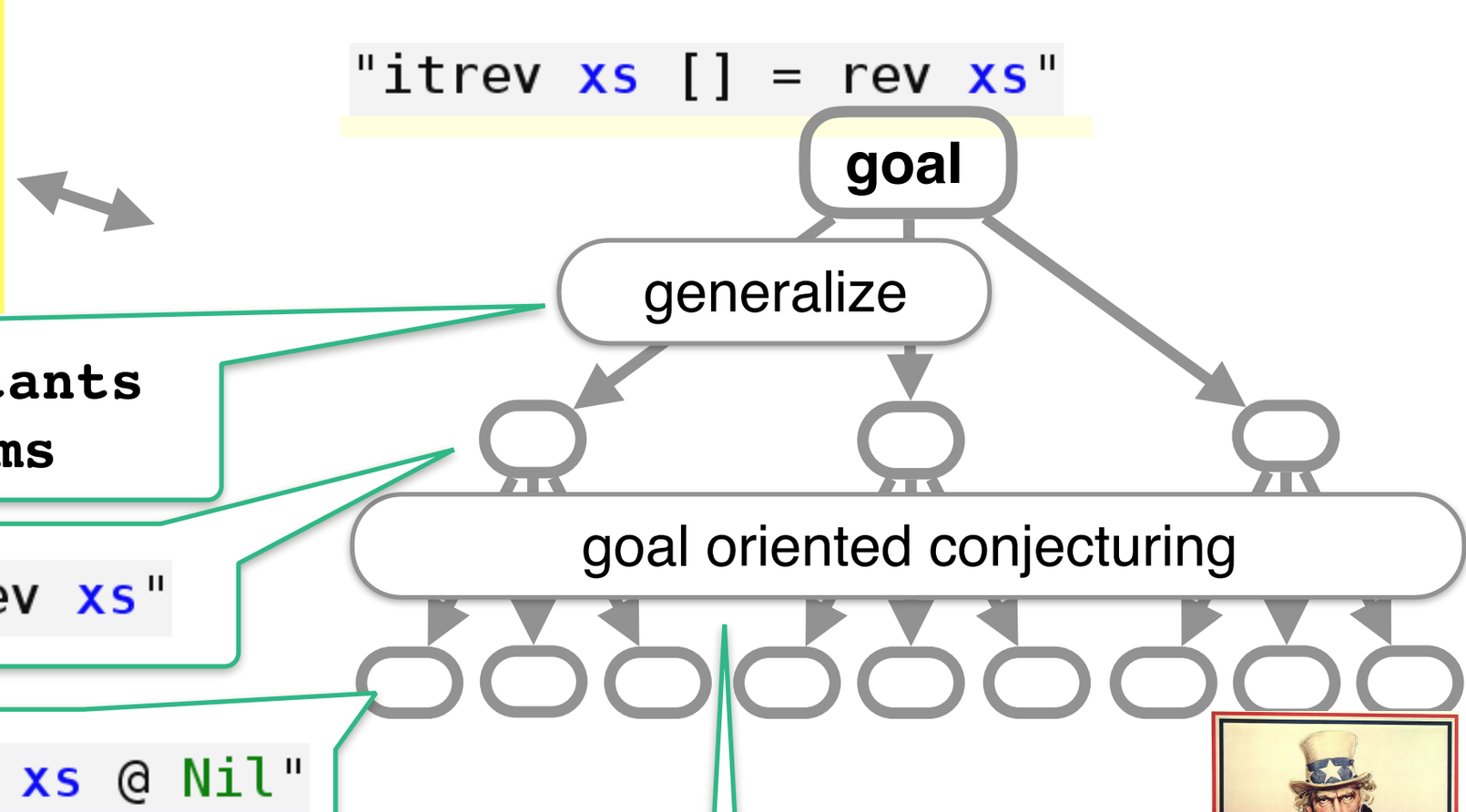
DInd

DInd

```
apply (induct xs)
apply auto
done
```

```
theorem itrev ?xs [] = rev ?xs
```





generalize over constants and common sub-terms

```
"⋀Nil. itrev xs Nil = rev xs"
```

```
"⋀Nil. itrev xs Nil = rev xs @ Nil"
```

```
List.rev.simps(2): rev (?x # ?xs) = rev ?xs @ [?x]
```

@

1. pick up constants that appear in the definitions of the constants appearing in the generalized conjectures. `rev`
2. replace a sub-term of generalized conjectures using the constants, the sub-term itself, and universally quantified variables. `rev xs` `Nil as a variable`
3. repeat 2. in a top-down manner within each syntax tree of generalized conjecture.

@

5 min. break

Next session is about Isabelle/ML

(I assume you know the basics of Haskell, such as monads, applicatives, and functors.)

Haskell vs (Isabelle/)ML

Haskell	ML
referential transparency	referential transparency
statically typed	statically typed
lazy evaluation	strict evaluation
type class (e.g. monad)	structure
monad transformer	ML functor

Isabelle/HOL architecture

PIDE / jEdit

Isar

extensible

HOL

Meta-logic

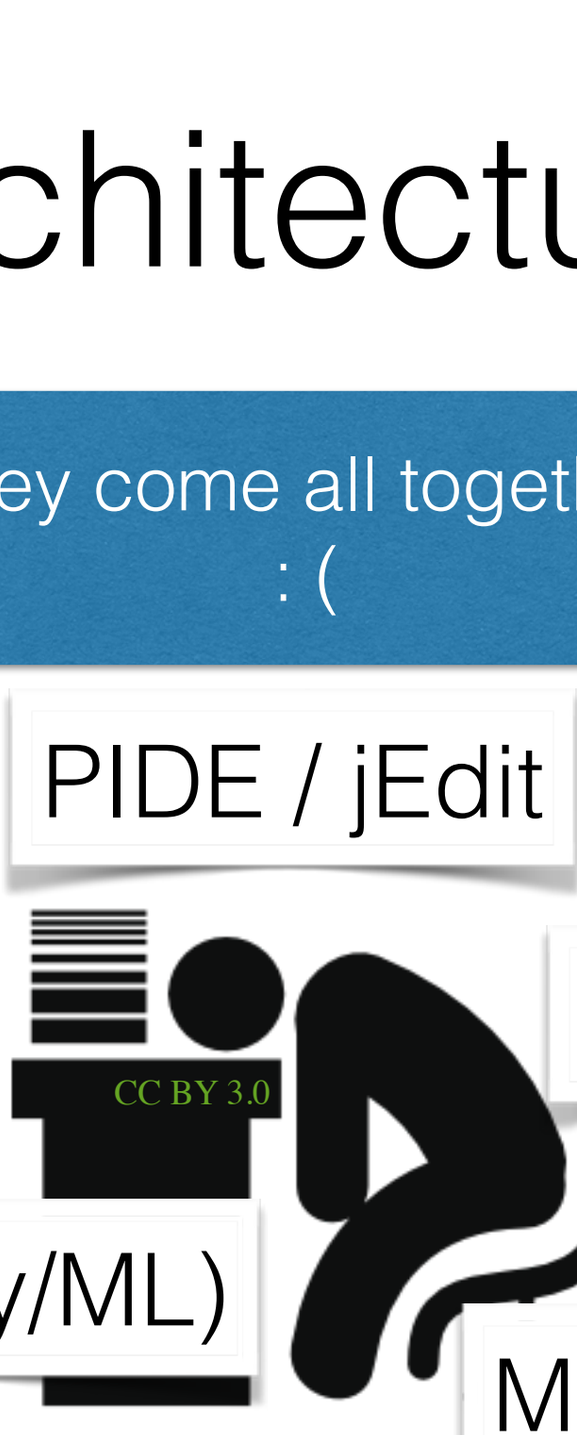
ML (Poly/ML)

You can access all the layers!
:)

They come all together!
: (

PIDE / jEdit

Isar



HOL

ML (Poly/ML)

Meta-logic

Where/Whom to ask for help

The Isabelle/Isar Implementation
by Makarius Wenzel

ML for the working programmer
by Laurence C. Paulson (open access)

The Isabelle Cookbook (outdated)
A Gentle Tutorial for Programming on the ML-Level of Isabelle (draft)
by Christian Urban

Isabelle mailing list
https://lists.cam.ac.uk/pipermail/cl-isabelle-users/index.html

The Isabelle/Isar Reference Manual
Makarius Wenzel

Isabelle/HOL itself
Jump to the definition with Ctrl+click. Isabelle's source code is mostly self-explanatory.

IRC channel #isabelle on freenode

ML in Isabelle/jEdit

import ML file

ML code snippet in cartouche

tracing to print out strings (useful for debugging)

@{assert} for unit test

the function “nth” is defined in the structure “List”

access Isabelle terms within ML code snippet using @{term} anti-quotation

ML_val to insert codesnippet within a proof

DEMO!

ML values in the Output panel

```
theory Demo imports Main
begin
ML_file "Empty.ML"
ML‹ val tracing_returns_unit = tracing "Use @{print} and @{print_tracing} as well";
    val assert_for_unit_test = @{assert} (List.nth ([1,2,3], 0)= 1);
    val true_in_HOL1         = @{term "True"};
    val true_in_HOL2         = Const ("HOL.True", @{typ "bool"})
    val assert_for_unit = @{assert} (true_in_HOL1 = true_in_HOL2)›
lemma "True ∧ True"
  apply - ML_val‹(*Use ML_val within a proof.*)› by auto
```

```
Use @{print} and @{print_tracing} as well
val tracing_returns_unit = (): unit
val assert_for_unit_test = (): unit
val true_in_HOL1 = Const ("HOL.True", "bool"): term
val true_in_HOL2 = Const ("HOL.True", "bool"): term
val assert_for_unit = (): unit
```

ML is 1.5 languages

ML = core language + module language

core language

module language

values	≒	structures
types	≒	signatures
functions	≒	functors

why module language?

better abstraction
(and data protection)

Example 1: Monads in ML

```
signature MONAD_MIN =
sig
  type 'a monad_min;
  val return : 'a -> 'a monad_min;
  val bind : 'a monad_min ->
       ('a -> 'b monad_min) -> 'b monad_min;
end;
```

```
struct ListMonadMin =
struct
  type 'a monad_min = 'a list;
  fun return x              = [x];
  fun bind seq func    =
        flat (map func seq);
end;
```

functor mk_Monad (ListMonadMin)

```
signature MONAD =
sig
  type 'a monad;
  include MONAD_MIN;
  val >>= : …;  val fail : …;  val >=> : …;
  val liftM : …; val filterM : …;
  val forever: …; val join : …;
end;
```

```
struct ListMonad : MONAD =
struct
  The functor automatically produces this
struct.
end;
```

Example 2: Monads in ML

```
signature MONAD_MIN =
sig
  type 'a monad_min;
  val return : 'a -> 'a monad_min;
  val bind : 'a monad_min ->
       ('a -> 'b monad_min) -> 'b monad_min;
end;
```

```
struct ListMonadMin =
struct
  type 'a monad_min = 'a list;
  fun return x              = [x];
  fun bind seq func    =
        flat (map func seq);
end;
```

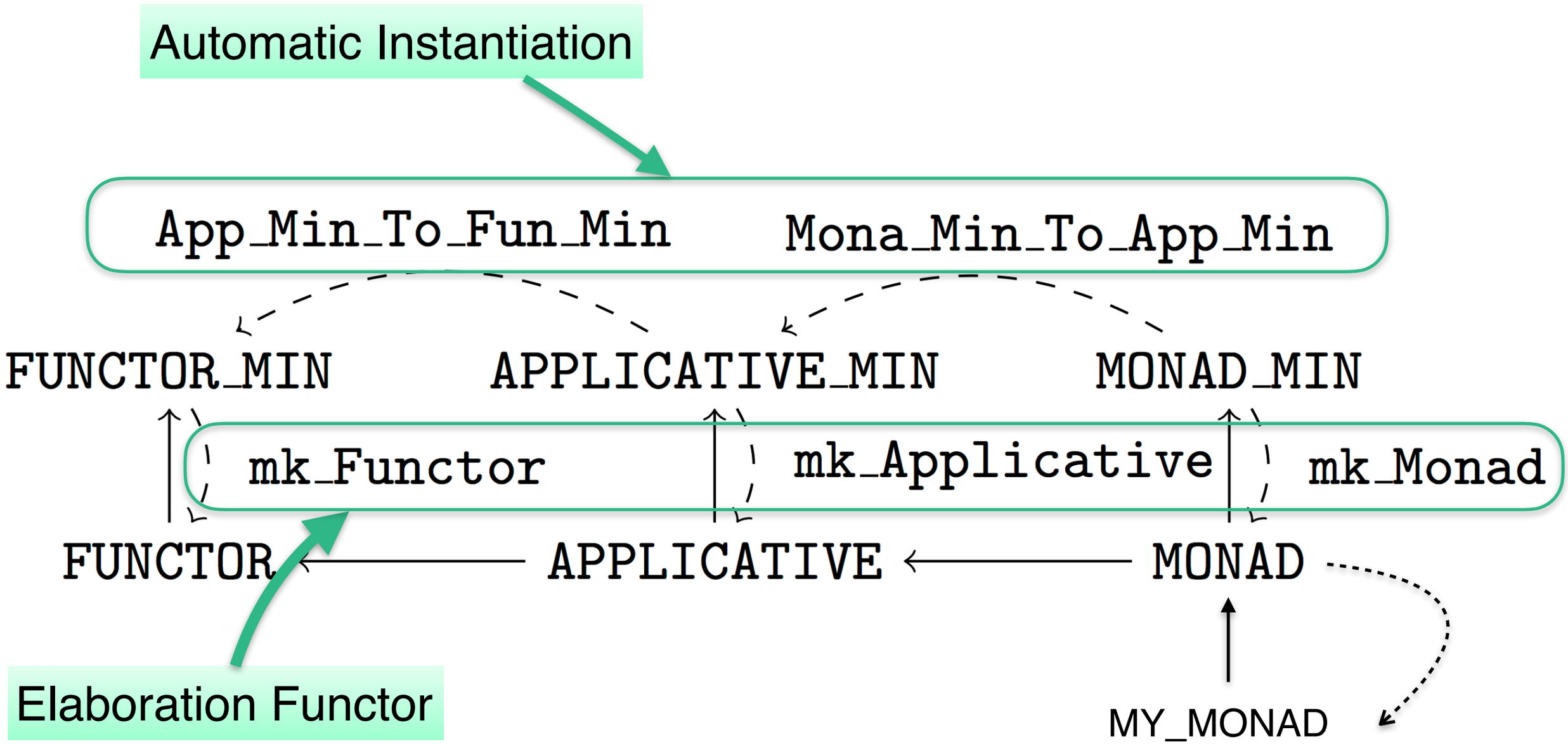
functor mk_Monad (ListMonadMin)

subtyping by "include" not by "structure"

```
signature MONAD =
sig
  type 'a monad;
  include MONAD_MIN APPLICATIVE;
  sharing type monad = applicative;
  val >>= : …;  val fail : …;  val >=> : …;
  val liftM : …; val filterM : …;
  val forever: …; val join : …;
end;
```

```
struct ListMonad : MONAD =
struct
  The functor automatically produces this
struct.
  The functor instantiates the list type as a
member of applicative and functor
(ArrowApply, Arrow, Category in future)
while deriving their default definitions.
end;
```

Hierarchical Constructor Classes



For more details, read our 2-page paper,
"Close Encounters of the Higher Kind"
available at https://arxiv.org/abs/1608.03350

(my personal) Isabelle/ML tips

The standard library offers many useful functions.
See, for example, “~~/src/Pure/library.ML"

Be aware of the following structures:
Thm in “~~/src/Pure/thm.ML”
Term in “~~/src/Pure/term.ML”
Logic in “~~/src/Pure/logic.ML”

Be functional. Don’t use references.
(I am not even going to explain them.)

Be aware of proof context. (@{context})

Don’t try to understand everything at once.

Thank you!

Leave a star at GitHub for PSL!

Project proposals

- **Proof Marathon with Tons of Inductive Problems (TIP)**
- **Algorithmic Conjecturing for Isabelle/HOL**
- **Multi-Output Regression Tree Construction with Cost-Complexity Pruning in Isabelle/ML**
- **Random forest in Isabelle/ML**
- **Compiler for the Register Machine from Hell**
- **BIGNAT - Specification and Verification**
- **Matching First-Order Terms - Soundness and Completeness**
- **Propositional Logic - Soundness and Completeness**
- **The Euclidean Algorithm - Inductively**
- **Tries**
- **Tseitin Transformation - Verification and Optimization**

3 min. break Next session is exercise.

Exercise (Innsbruck/Exercise.thy)

- Pre-requisite: git clone https://github.com/data61/PSL
- Exercise I: Prove lemmas using PSL
- Exercise II-a: Complete the minimal definition of Seq_Min and produce the complete definition of lazy sequence as a monad using the elaboration function provided in the accompanying file (Innsbruck/"Constructor_Class.ML").
- Exercise II-b: Prove "True ∨ False” using the tactic method.
- (Exercise II-c: Develop a ML functor that emulates the writer monad transformer.)