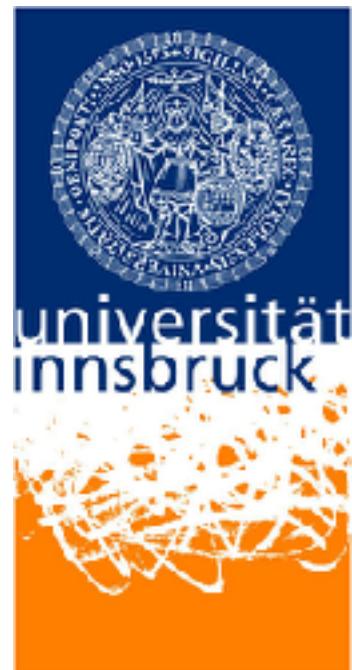


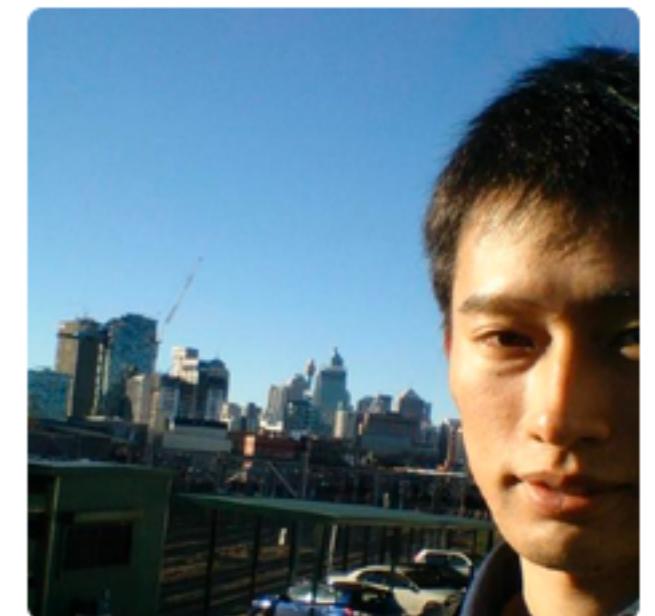
Interactive theorem proving in Isabelle/HOL with AI&ML&DSL



Yutaka Nagashima



**CZECH INSTITUTE
OF INFORMATICS
ROBOTICS AND
CYBERNETICS
CTU IN PRAGUE**



Yutaka Ng

Interactive theorem proving in Isabelle/HOL with AI&ML&DSL



Yutaka Nagashima



**CZECH INSTITUTE
OF INFORMATICS
ROBOTICS AND
CYBERNETICS
CTU IN PRAGUE**



Yutaka Ng

Background

2013 ~ 2017



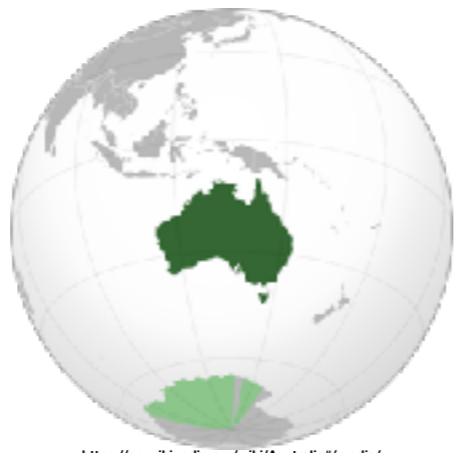
<http://www.cse.unsw.edu.au/~kleing/>

with Prof. Gerwin Klein



Background

2013 ~ 2017



<http://www.cse.unsw.edu.au/~kleing/>

with Prof. Gerwin Klein



Cogent

Background

2013 ~ 2017

Intern &
Engineer



with Prof. Gerwin Klein



Cogent

Background

2013 ~ 2017

Intern &
Engineer



Cogent

PhD in
AI for theorem proving

Background

2013 ~ 2017

Intern &
Engineer



Cogent

with Prof. Gerwin Klein

PhD in
AI for theorem proving



with Prof. Cezary Kaliszyk

2017 ~ 2018

Background

2013 ~ 2017

Intern &
Engineer



Cogent

PhD in
AI for theorem proving



PSL

2017 ~ 2018

with Prof. Gerwin Klein

with Prof. Cezary Kaliszyk

Background

2013 ~ 2017

Intern &
Engineer



Cogent

PhD in
AI for theorem proving



PSL

2017 ~ 2018

with Prof. Cezary Kaliszyk

PaMpeR

Background

2013 ~ 2017

Intern &
Engineer



Cogent

PhD in
AI for theorem proving



PSL

2017 ~ 2018

2018 ~
2020/21



PaMpeR

with Prof. Gerwin Klein

with Prof. Cezary Kaliszyk

with Dr. Josef Urban

Background

2013 ~ 2017

Intern &
Engineer



Cogent

PhD in
AI for theorem proving



PSL

2017 ~ 2018

with Prof. Gerwin Klein

with Prof. Cezary Kaliszyk

2018 ~
2020/21



LiFtEr

with Dr. Josef Urban

Background

2013 ~ 2017

Intern & Engineer



PhD in AI for theorem proving



2017 ~ 2018

A portrait of a man with short brown hair, wearing a black hoodie. He is looking directly at the camera with a neutral expression.

<http://www.cse.unsw.edu.au/~kleing/>
with Prof. Gerwin Klein



Cogent

**2018 ~
2020/21**



<http://ci-informatik.tuuk.ac.at/users/cez>



Isabelle HOL

PSL

PaMpeR



LiFtEr

<http://ai4reason.org/members.html>

smart induct

Background

2013 ~ 2017

Intern &
Engineer



As a contributor.

Cogent



with Prof. Gerwin Klein

PhD in
AI for theorem proving



PSL



PaMpeR

2017 ~ 2018

with Prof. Cezary Kaliszyk

2018 ~
2020/21



LiFtEr

with Dr. Josef Urban

smart_induct

Background

2013 ~ 2017

Intern &
Engineer



As a contributor.

Cogent

As the main developer.

with Prof. Gerwin Klein

PhD in
AI for theorem proving



PSL

2017 ~ 2018

with Prof. Cezary Kaliszyk

2018 ~
2020/21



LiFtEr

with Dr. Josef Urban

smart_induct

DEMO1: PSL

The screenshot shows the Isabelle proof assistant interface with a blue border around the title "DEMO1: PSL".

The main window displays a PSL (PIDE) file named "Example.thy". The code defines a function "sep" with three cases:

```
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)"
```

The third case is highlighted with a yellow background.

At the bottom of the interface, there is a status bar with the following information:

- Proof state:
- Auto update:
- Update:
- Search:
- Zoom: 100%
- Date: 10.42 (221/935)
- File: (isabelle.isabelle,UTF-8-Isabelle) mmr@U... 329/535MB 1:17 AM

The status bar also includes tabs for Output, Query, Sledgehammer, and Symbols.

DEMO1: PSL

The screenshot shows the Isabelle proof assistant interface with a blue border around the title bar.

The code editor window displays a file named "Example.thy" with the following content:

```
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)

value "sep (1::int) [0,0,0]"
```

The cursor is positioned at the end of the third line of the function definition. The status bar at the bottom shows the output "[0, 1, 0, 1, 0]" and the type ":: int list".

At the bottom left, there are tabs for Output, Query, Sledgehammer, and Symbols. At the bottom right, there is a status bar with the text "12.29 (251/963)" and "(isabelle.isabelle,UTF-8-Isabelle) nmr o U.. 286/535MB 1:17 AM".

DEMO1: PSL

The screenshot shows the Isabelle/Isar proof assistant interface. The main window displays a theory file named `Example.thy`. The code defines a function `sep` and a value, and states a lemma. The lemma is currently being edited.

```
File Browser Documentation Example.thy (~-/Workplace/PSL/PGT/)

1 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
2   "sep a []      = []" |
3   "sep a [x]     = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"
5
6 value "sep (1::int) [0,0,0]"
7
8
9
10
11
12
13
14
15
16 Lemma
17   "map f (sep x xs) = sep (f x) (map f xs)"
```

```
proof (prove)
goal (1 subgoal):
  1. map f (sep x xs) = sep (f x) (map f xs)
```

DEMO1: PSL

```
File Browser Documentation Example.thy (~-/Workplace/PSL/PGT/)

1 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
2   "sep a []      = []" |
3   "sep a [x]     = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"
5
6 value "sep (1::int) [0,0,0]"
7
8 strategy DInd = Thens [Dynamic (Induct), Auto, IsSolved]
9
10
11 Lemma
12   "map f (sep x xs) = sep (f x) (map f xs)"
13
14
15
16
17
18
19
20
21
```

my language (PSL)

Proof state Auto update Update Search: 100% 0

Output Query Sledgehammer Symbols

14.58 (310/1067) (isabelle.isabelle,UTF-8-Isabelle) nmr o U.. 291/535MB 1:20 AM

DEMO1: PSL

The screenshot shows the Isabelle/Isar proof assistant interface. The main window displays a file named "Example.thy". The code in the file is as follows:

```
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)"

value "sep (1::int) [0,0,0]"

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

lemma
  "map f (sep x xs) = sep (f x) (map f xs)"
  find_proof DInd

proof (prove)
goal (1 subgoal):
  1. map f (sep x xs) = sep (f x) (map f xs)
```

The cursor is positioned over the lemma statement. The interface includes a toolbar at the top, a vertical scroll bar on the right, and a status bar at the bottom.

DEMO1: PSL

The screenshot shows the Isabelle/Isar proof assistant interface. The main window displays a theory file named `Example.thy`. The code defines a function `sep` and a value, sets a strategy, and proves a lemma. The cursor is positioned at the start of the proof command for the lemma.

```
File Browser Documentation Example.thy (~-/Workplace/PSL/PGT/)

1 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
2   "sep a []      = []" |
3   "sep a [x]     = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"
5
6 value "sep (1::int) [0,0,0]"
7
8 strategy DInd  = Thens [Dynamic (Induct), Auto, IsSolved]
9
10 lemma
11   "map f (sep x xs) = sep (f x) (map f xs)"
12   find_proof DInd
13
14
15
16
17
18
19
20
21
```

Number of lines of commands: 3

apply (induct xs rule: Example.sep.induct)

apply auto

done

DEMO1: PSL

The screenshot shows the Isabelle/Isar proof assistant interface. The main window displays a theory file named "Example.thy". The code defines a function "sep" and a value, sets a strategy, proves a lemma, and concludes with a proof block.

```
File Browser Documentation Example.thy (~-/Workplace/PSL/PGT/)

1 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
2   "sep a []      = []" |
3   "sep a [x]     = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"
5
6 value "sep (1::int) [0,0,0]"
7
8 strategy DInd = Thens [Dynamic (Induct), Auto, IsSolved]
9
10
11 lemma
12   "map f (sep x xs) = sep (f x) (map f xs)"
13   find_proof DInd
14 apply (induct xs rule: Example.sep.induct)
15 apply auto
16 done

proof (prove)
goal:
No subgoals!
```

The interface includes a toolbar at the top, a vertical navigation bar on the left, and a status bar at the bottom. The status bar shows the current proof state, update frequency, search field, zoom level (100%), and system information (20.11 (433/1147), (isabelle,isabelle,UTF-8-Isabelle) in miro U., 255/535MB 1:05 AM).

DEMO1: PSL

human expertise + search => proof script

```
File Browser Documentation Example.thy (~ /Workplace/PSL/PCF)
1 fun sep : Type[0]
2   | "sep" : α
3   | "sep" : α
4   | "sep" : α
5 
6 value "sep (1::int) [0,0,0]"
7 
8 strategy DInd = Thens [Dynamic (Induct), Auto, IsSolved]
9 
10 Lemma
11   "map f (sep x xs) = sep (f x) (map f xs)"
12   find_proof DInd
13   apply (induct xs rule: Example.sep.induct)
14   apply auto
15   done
```

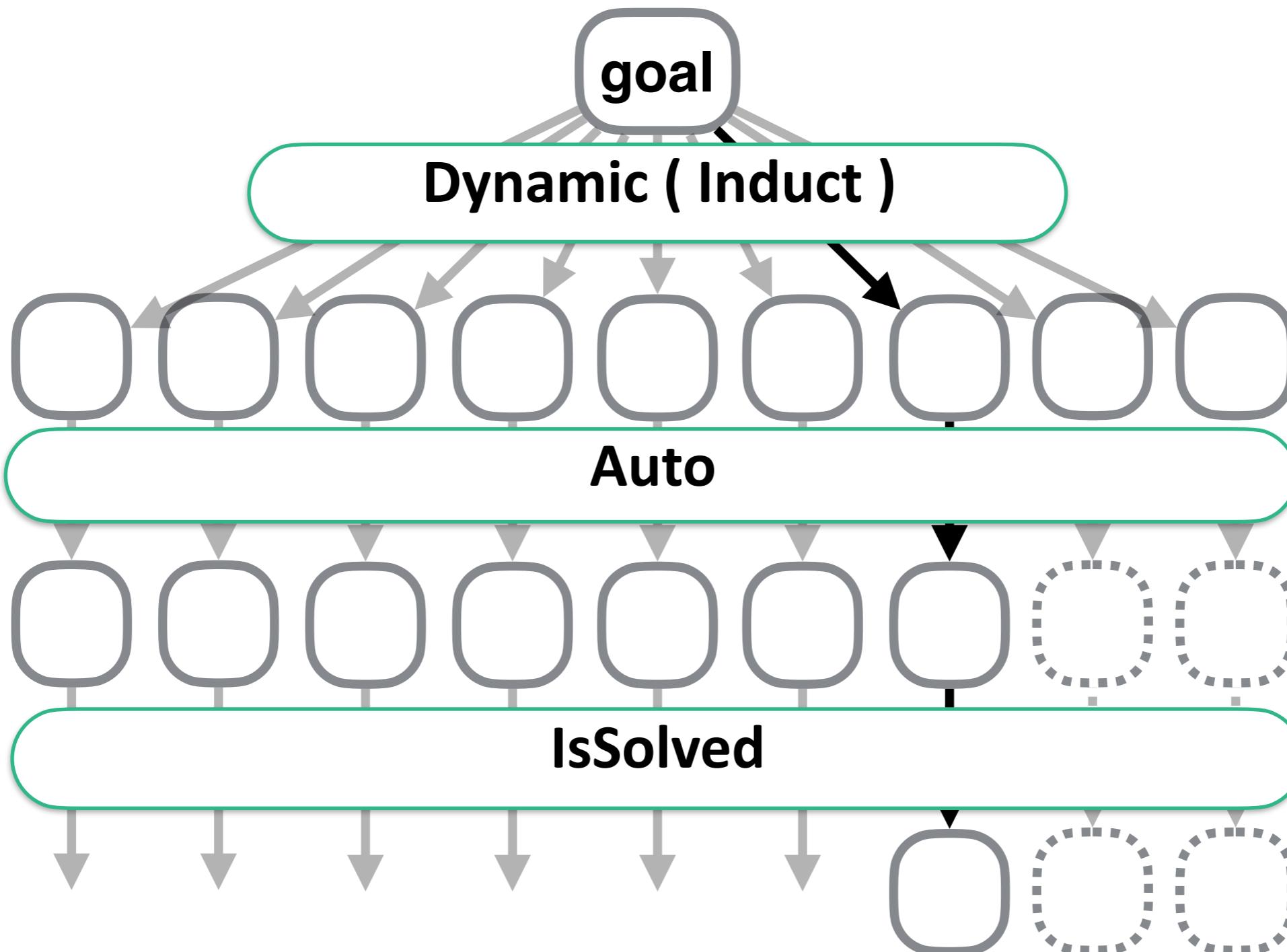
```
proof (prove)
goal:
No subgoals!
```

Proof state Auto update Update Search: 100% C

DEMO1

PSL: Proof Strategy Language

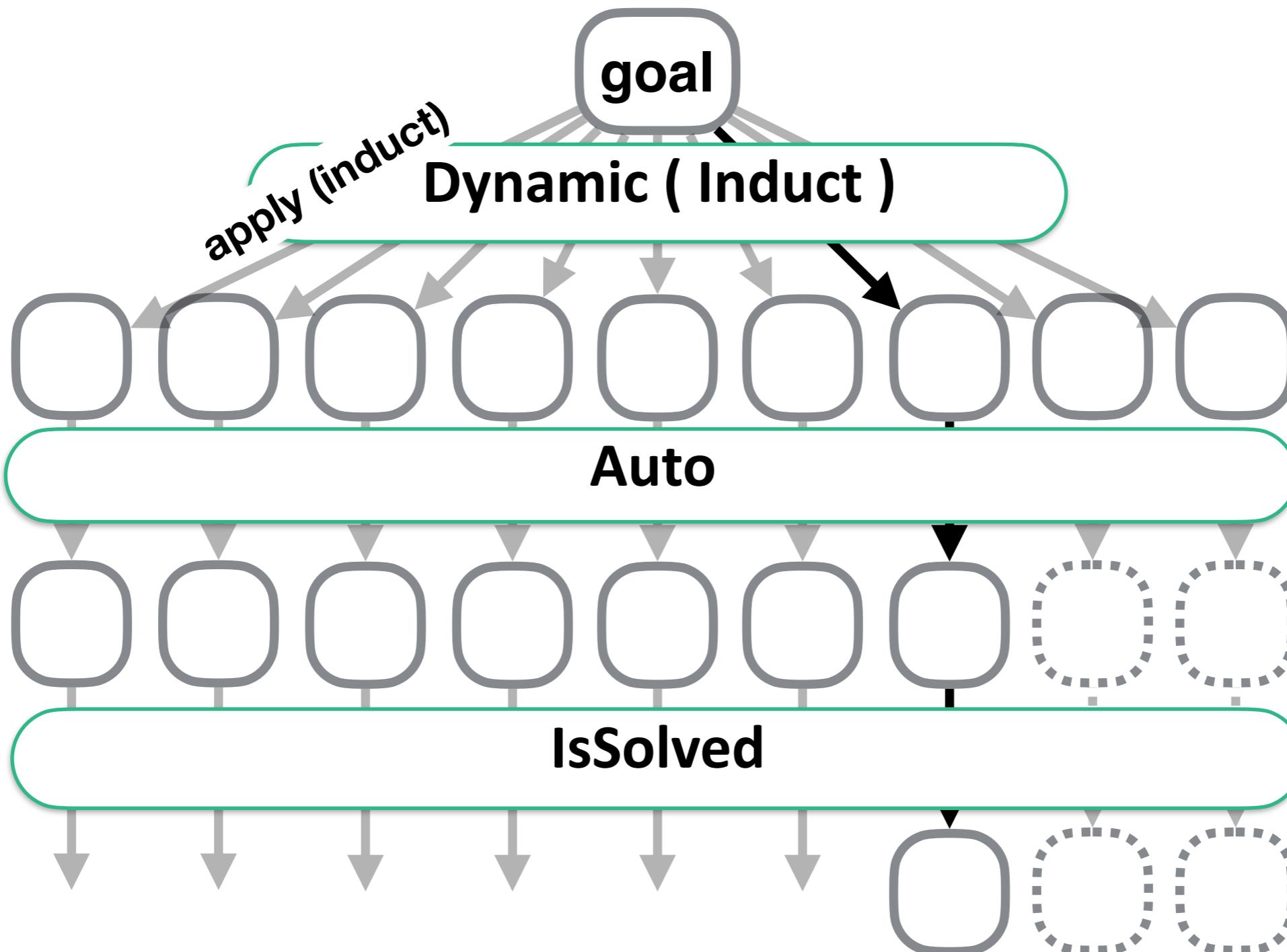
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

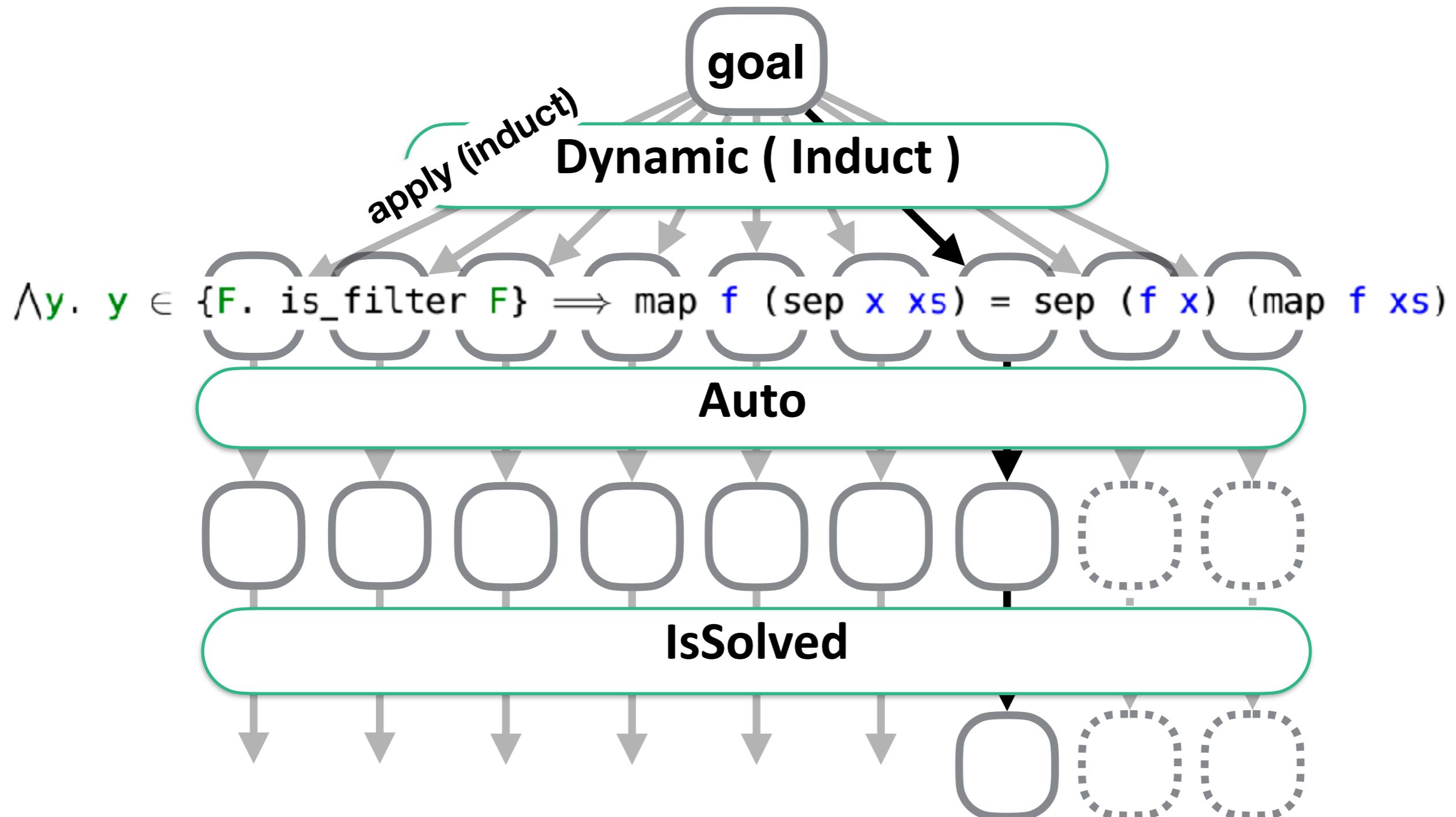
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

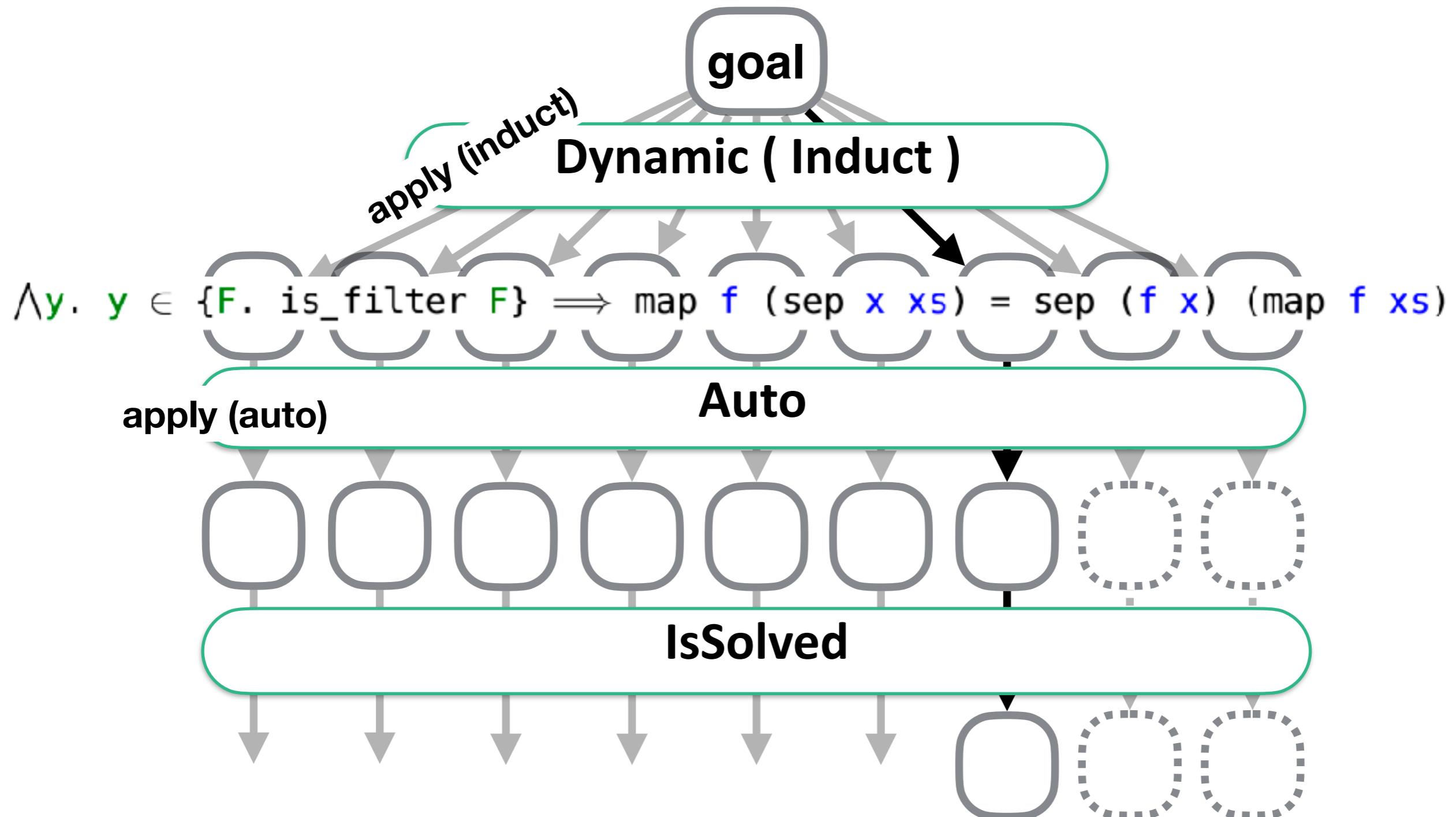
PSL: Proof Strategy Language

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



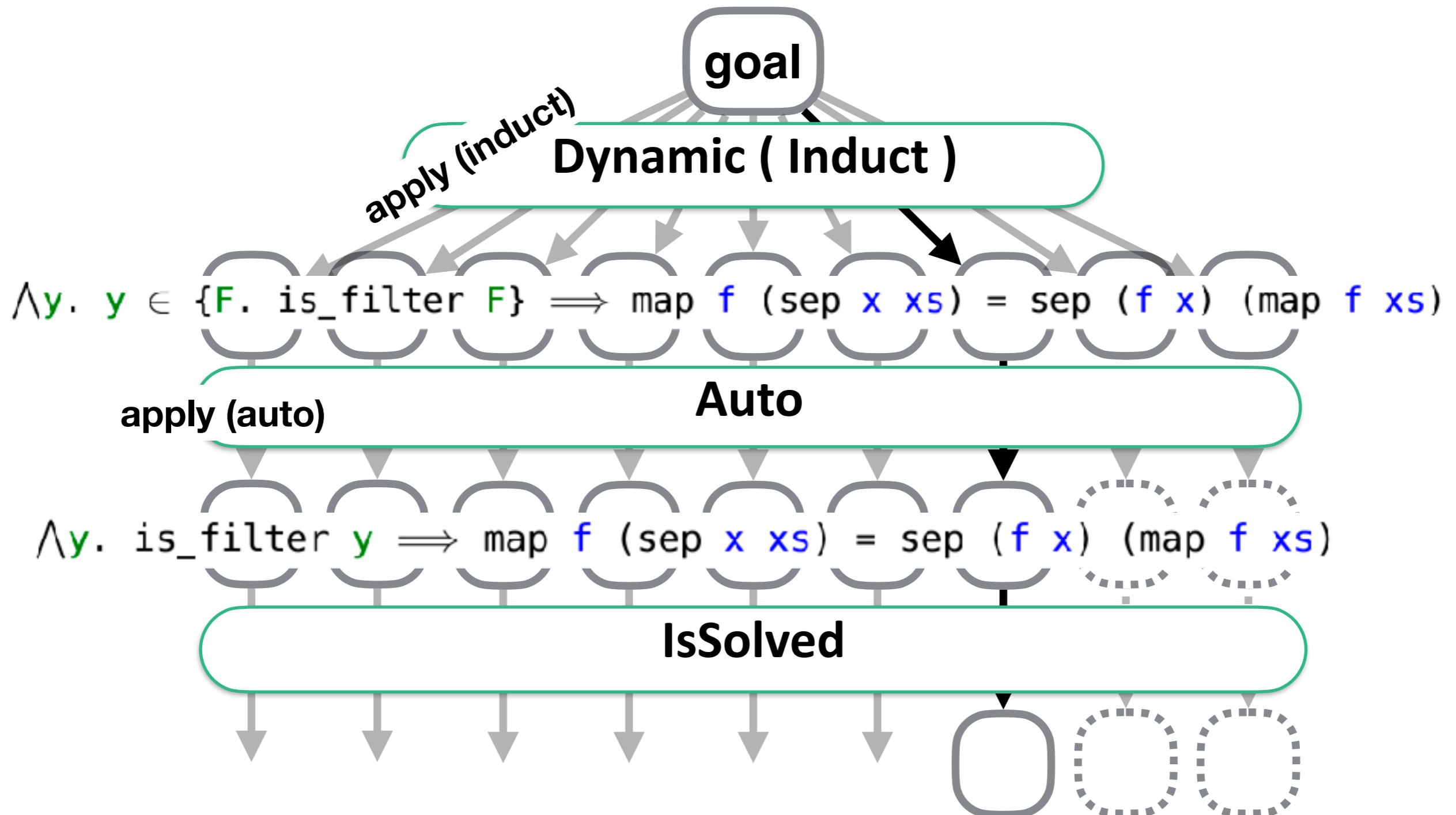
DEMO1 PSL: Proof Strategy Language

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1 PSL: Proof Strategy Language

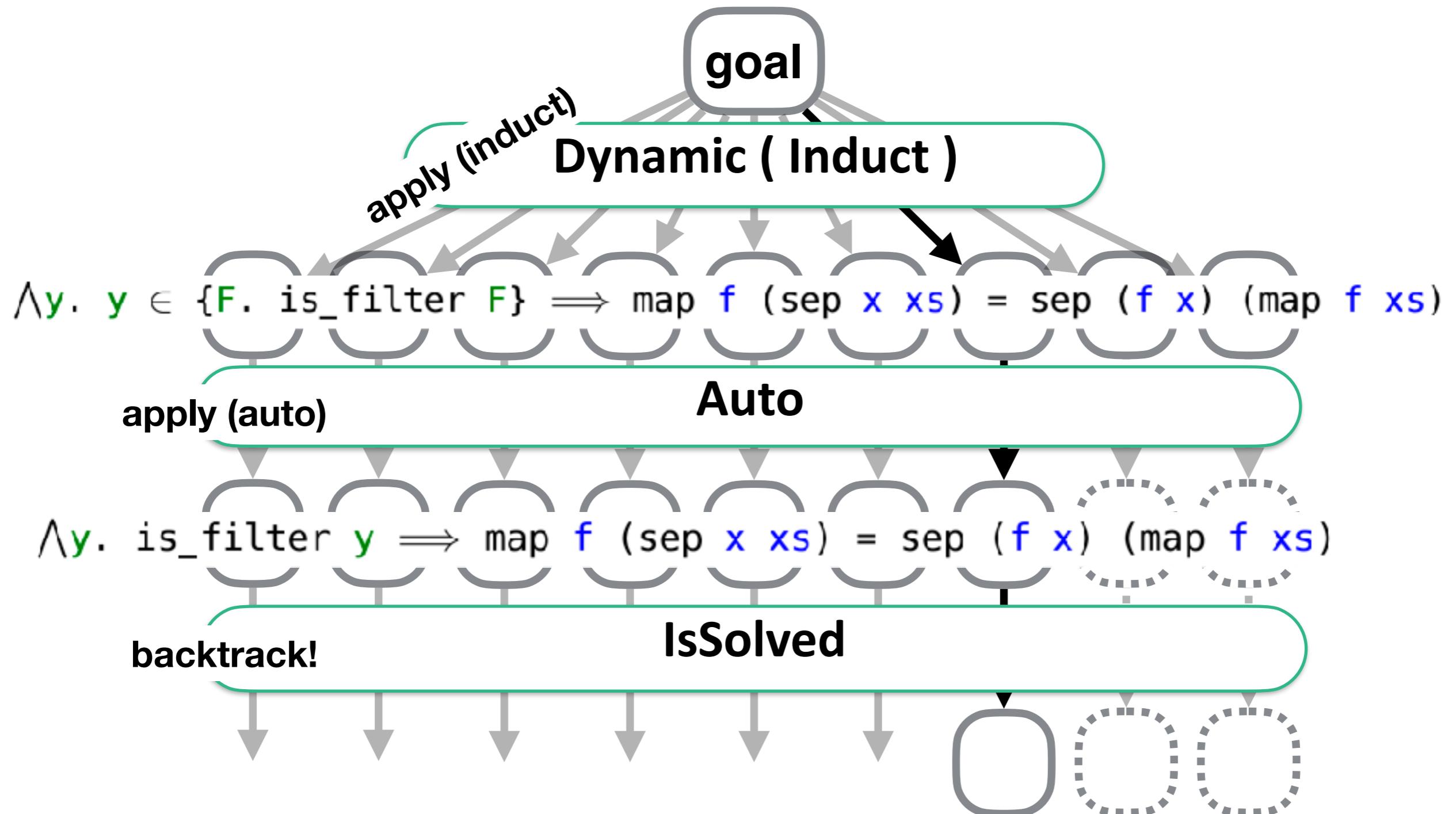
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

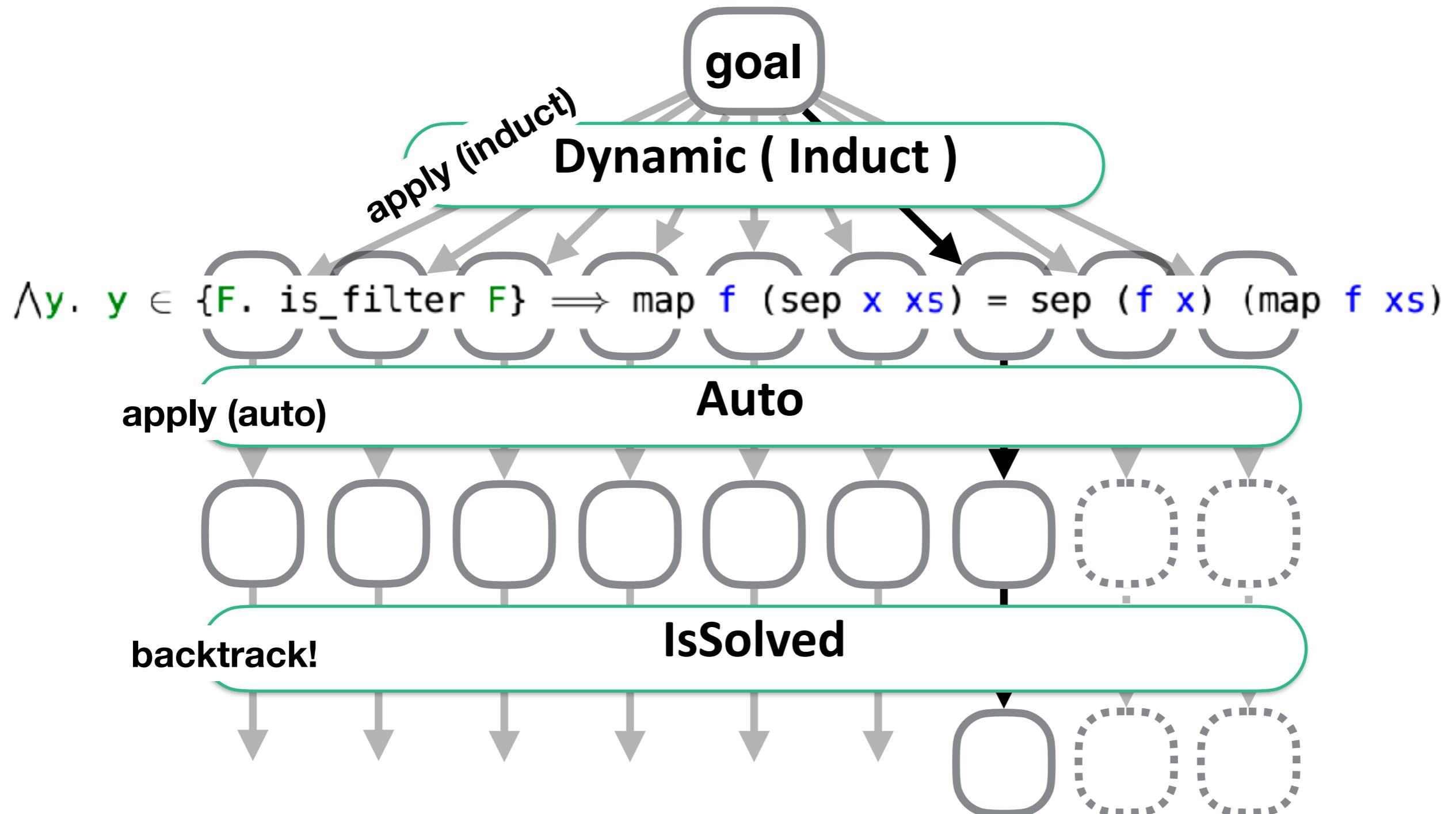
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

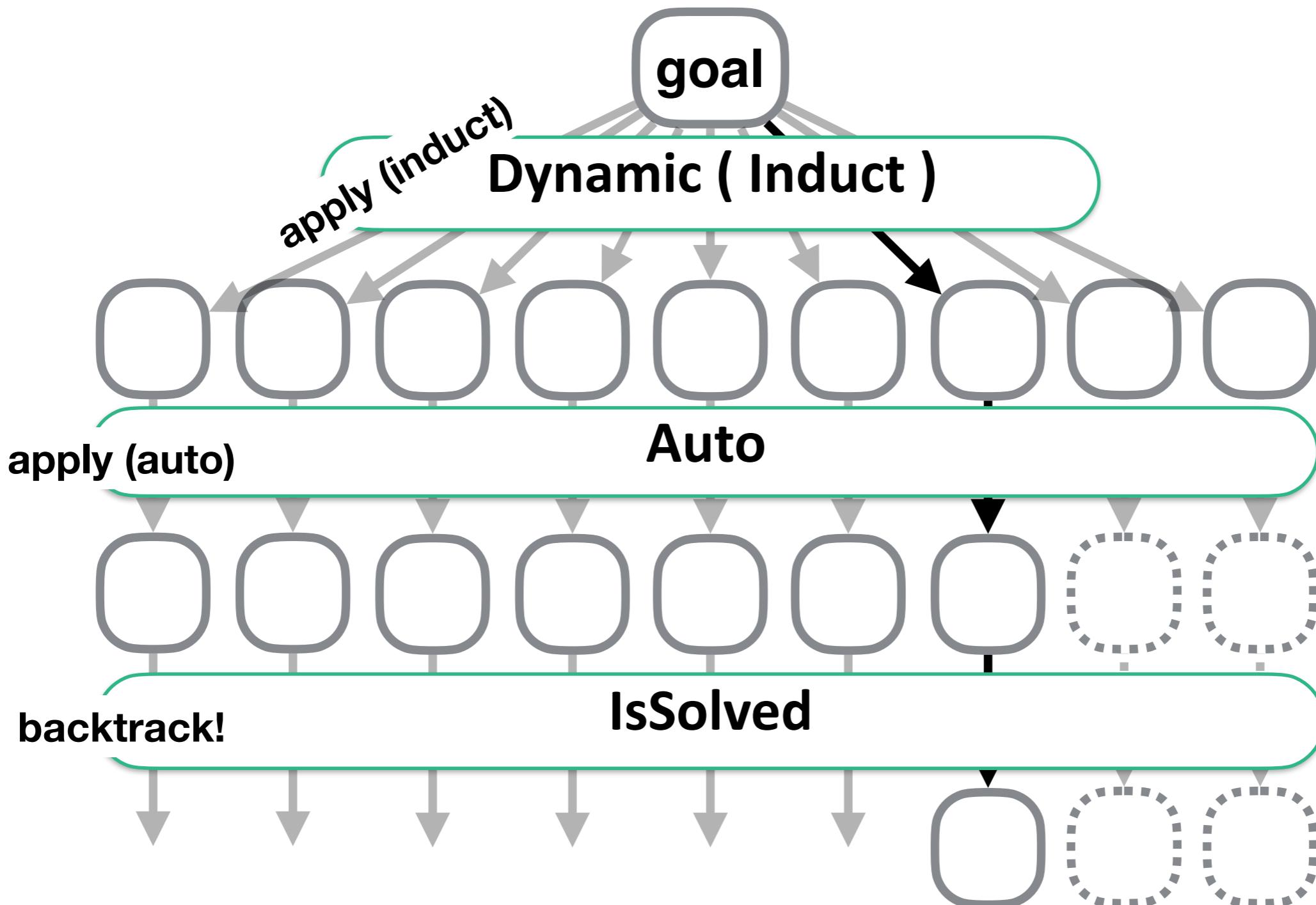
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

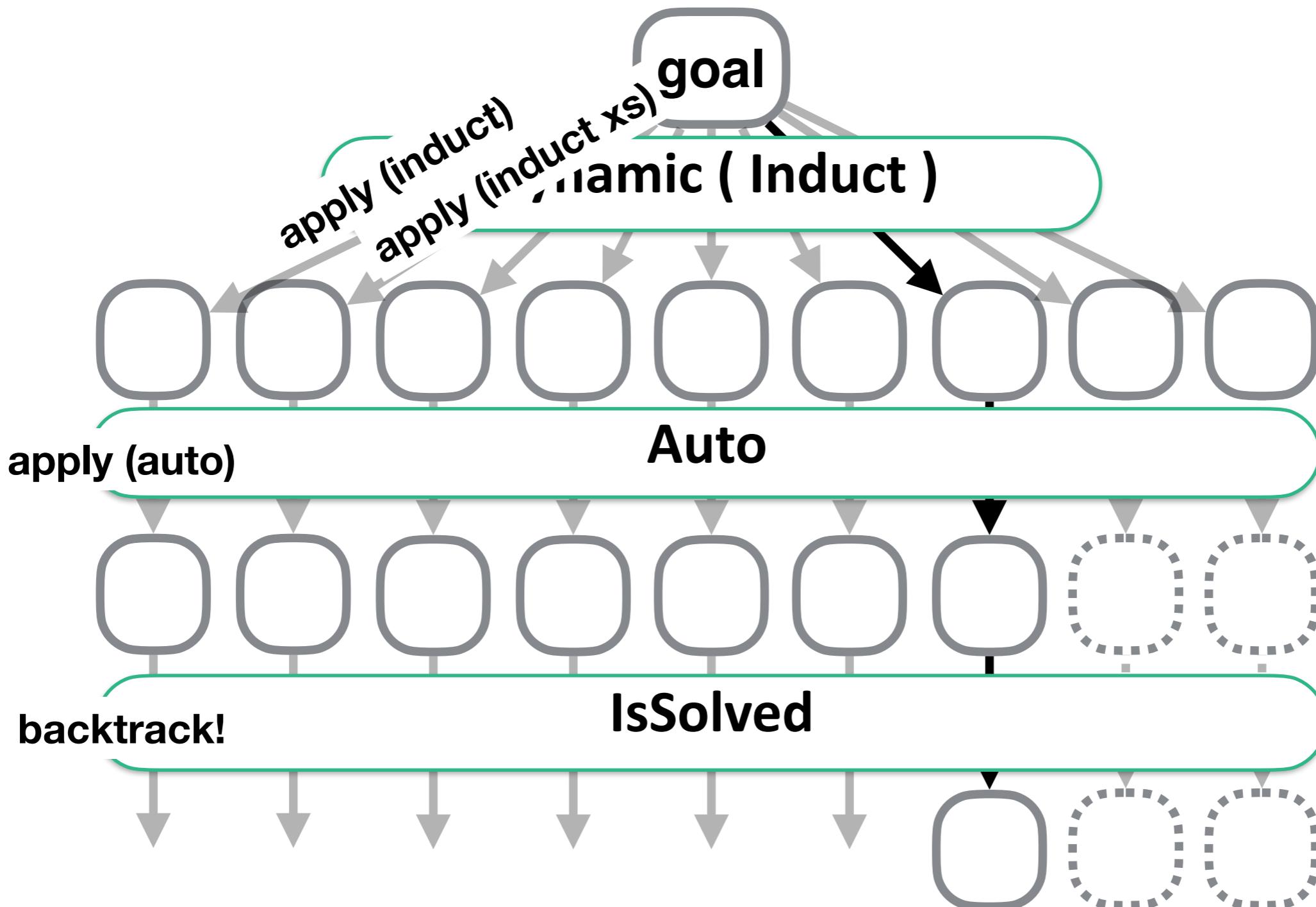
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

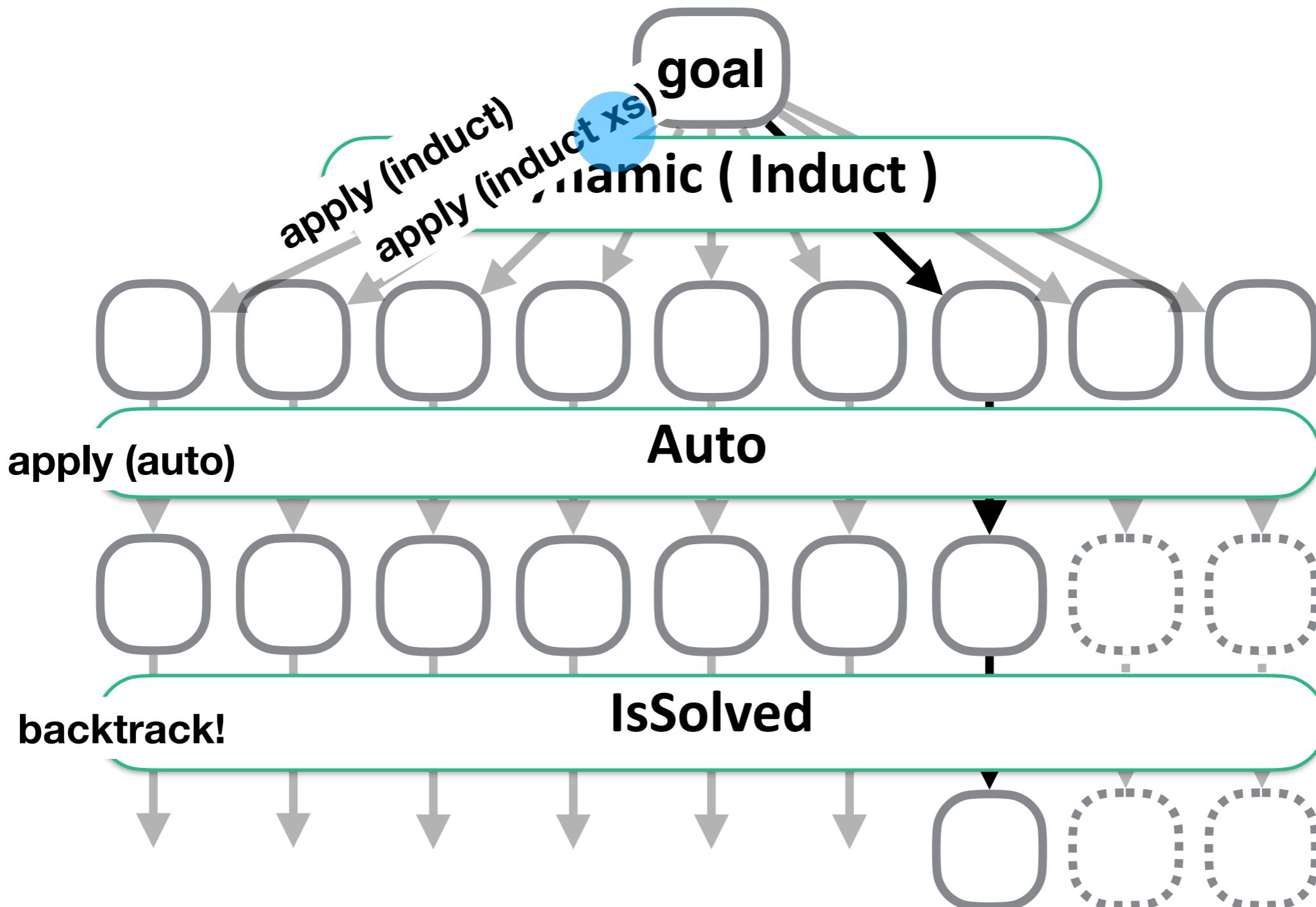
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

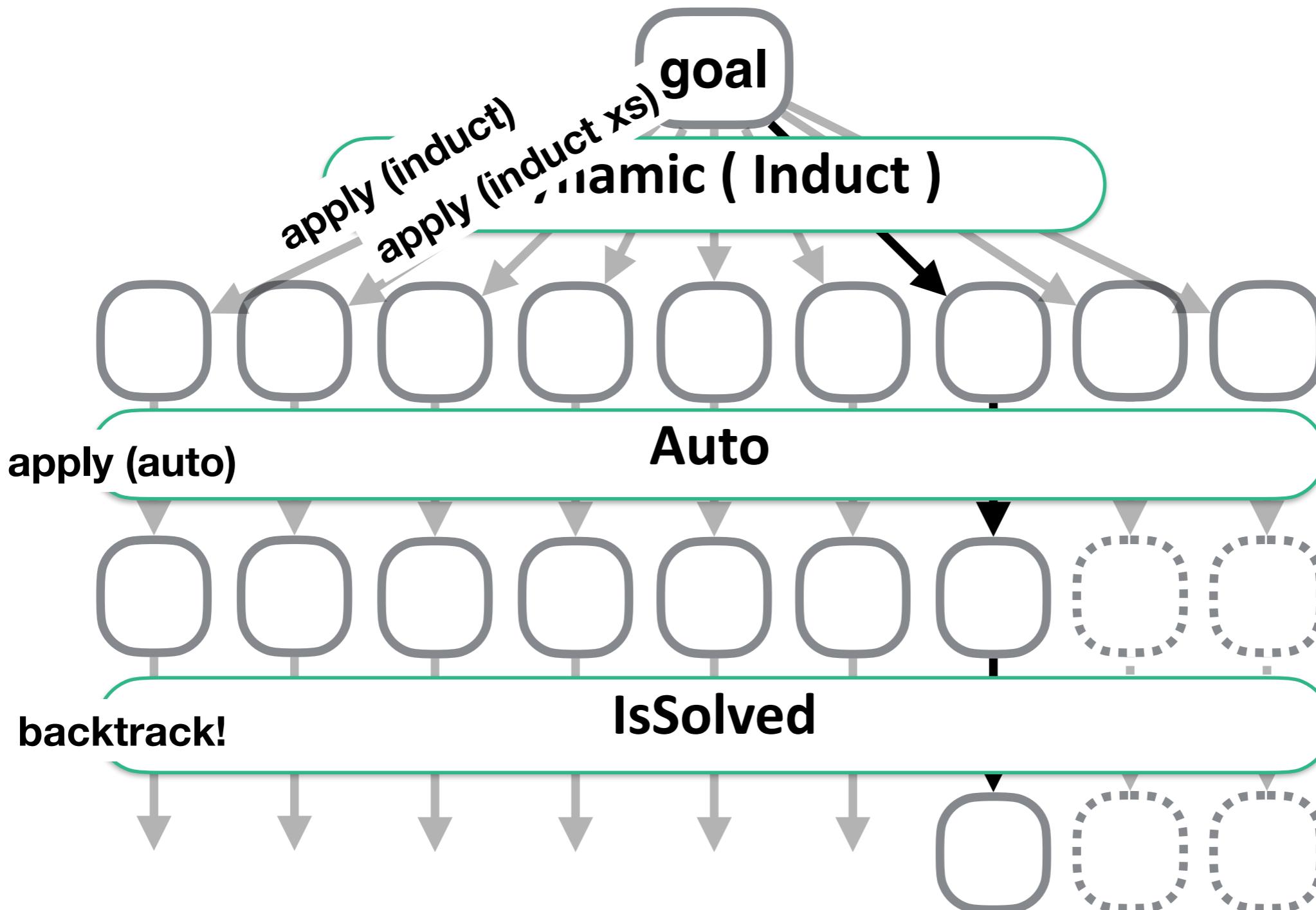
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

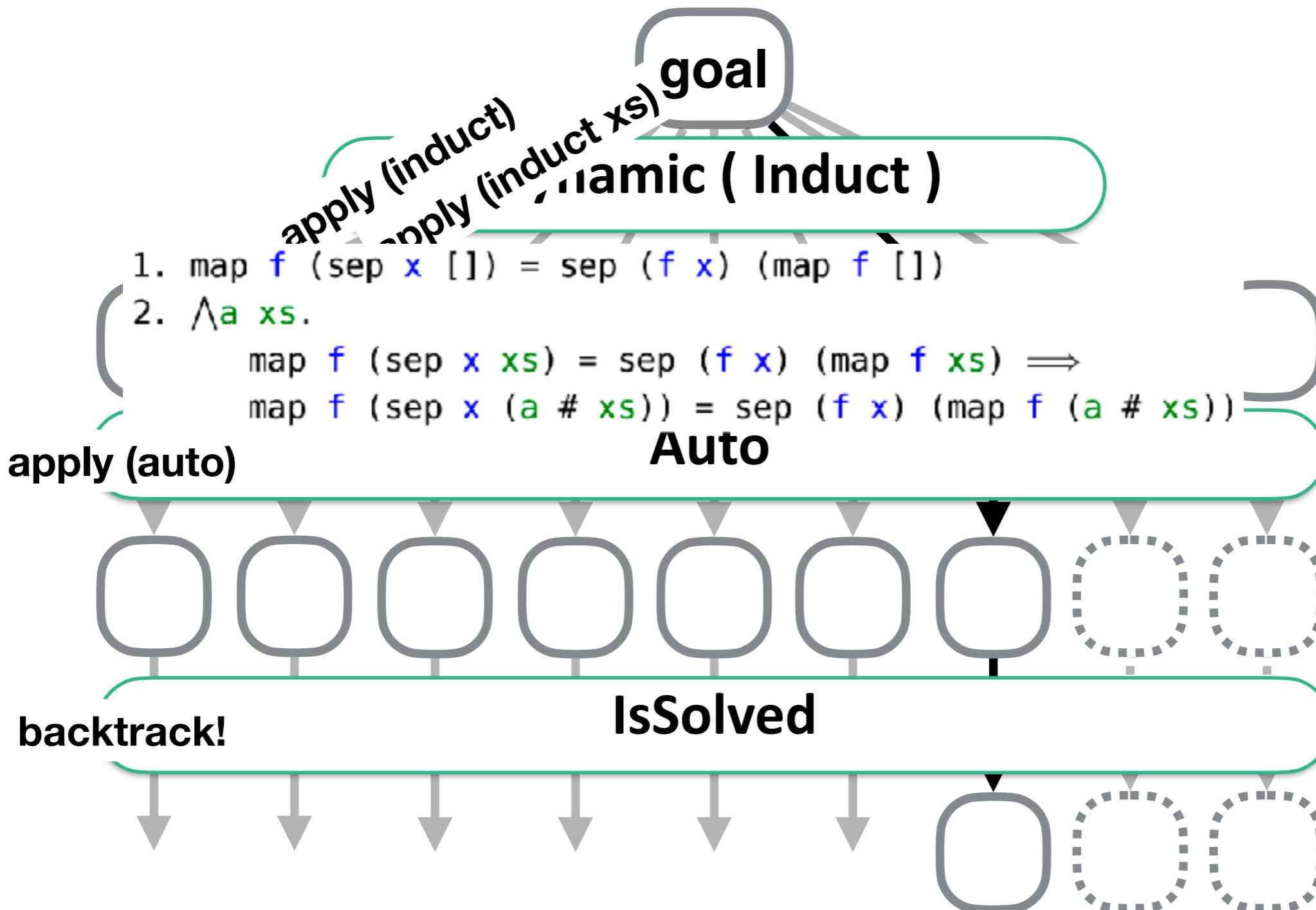
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

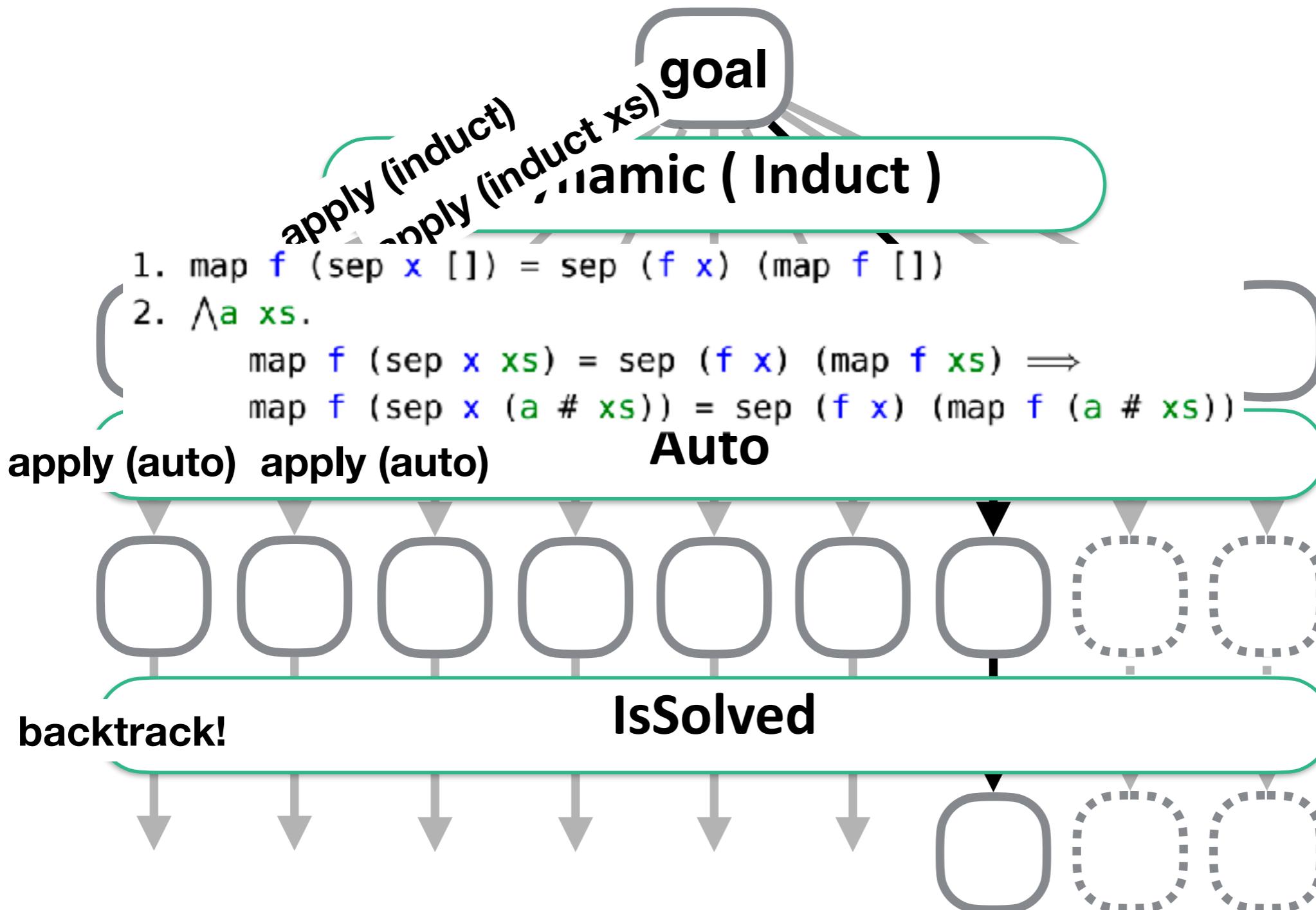
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

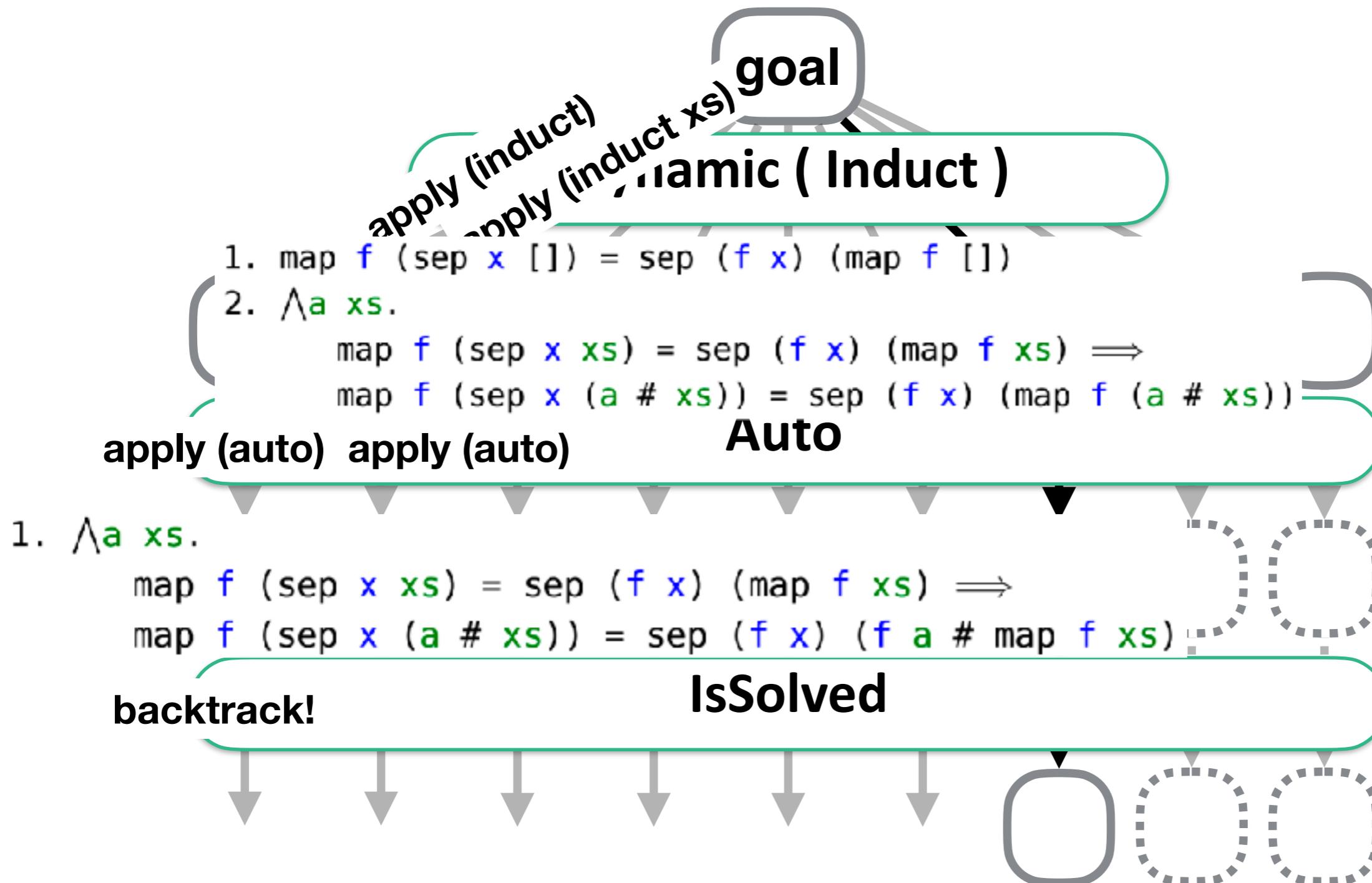
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

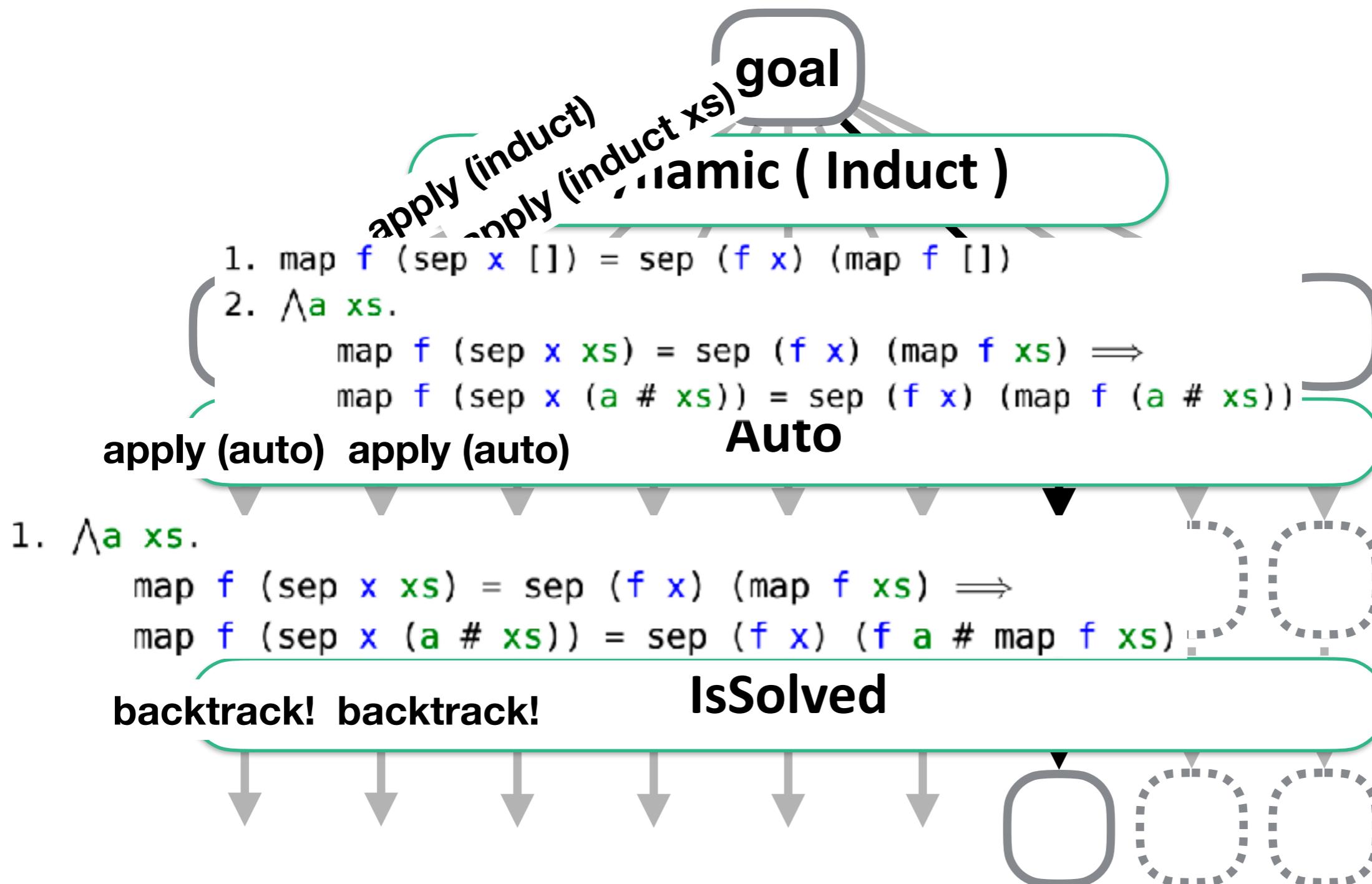
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

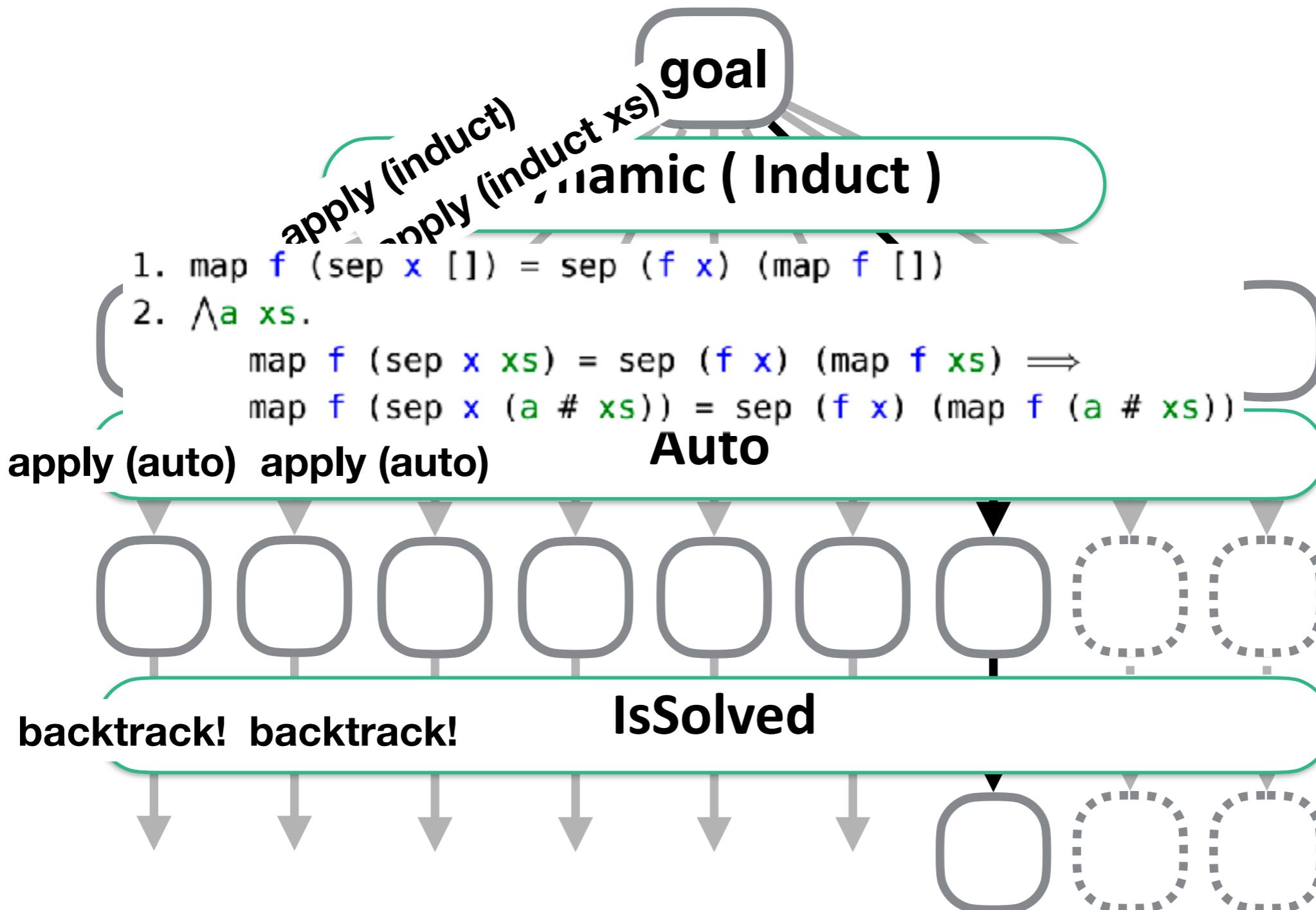
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

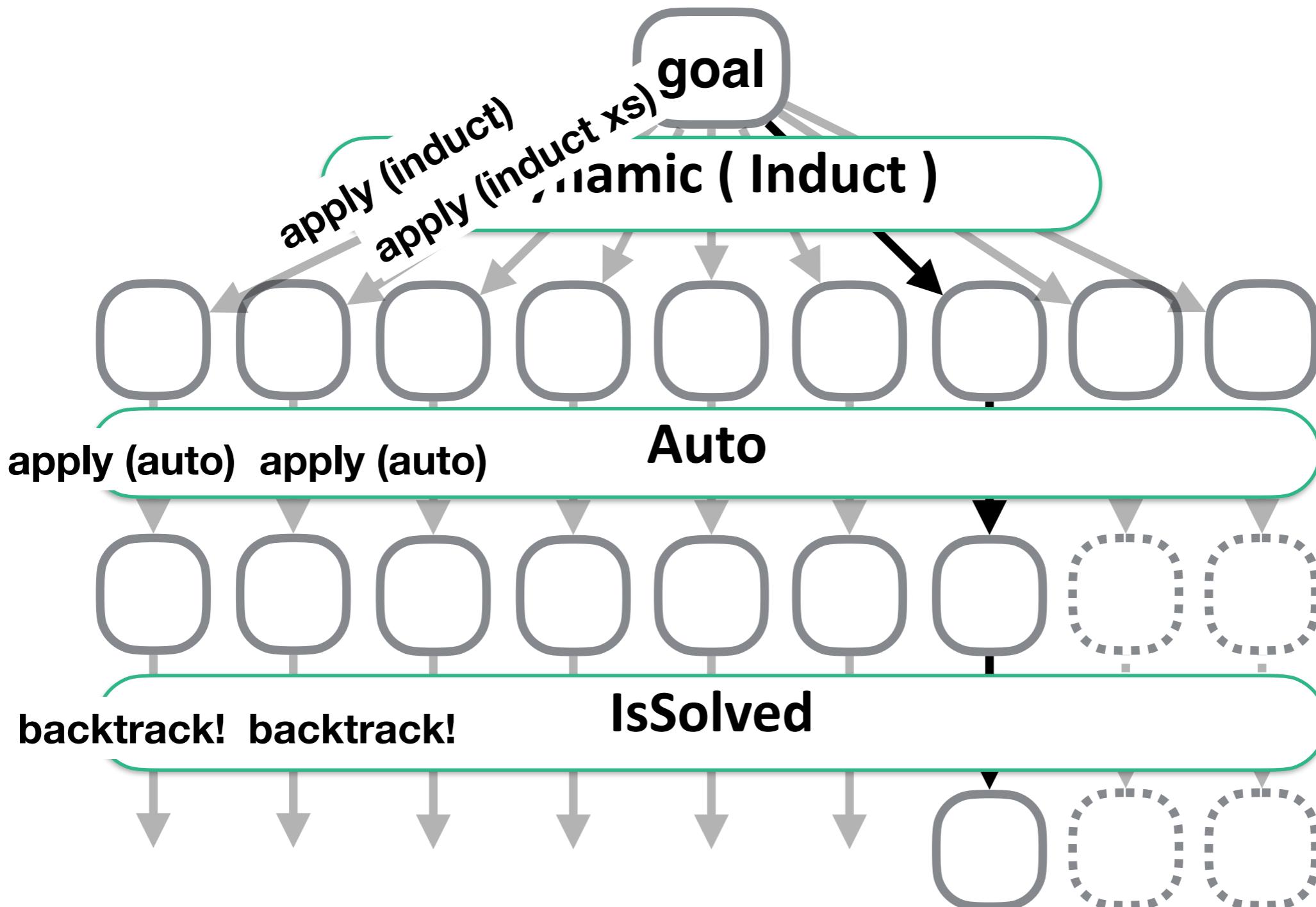
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

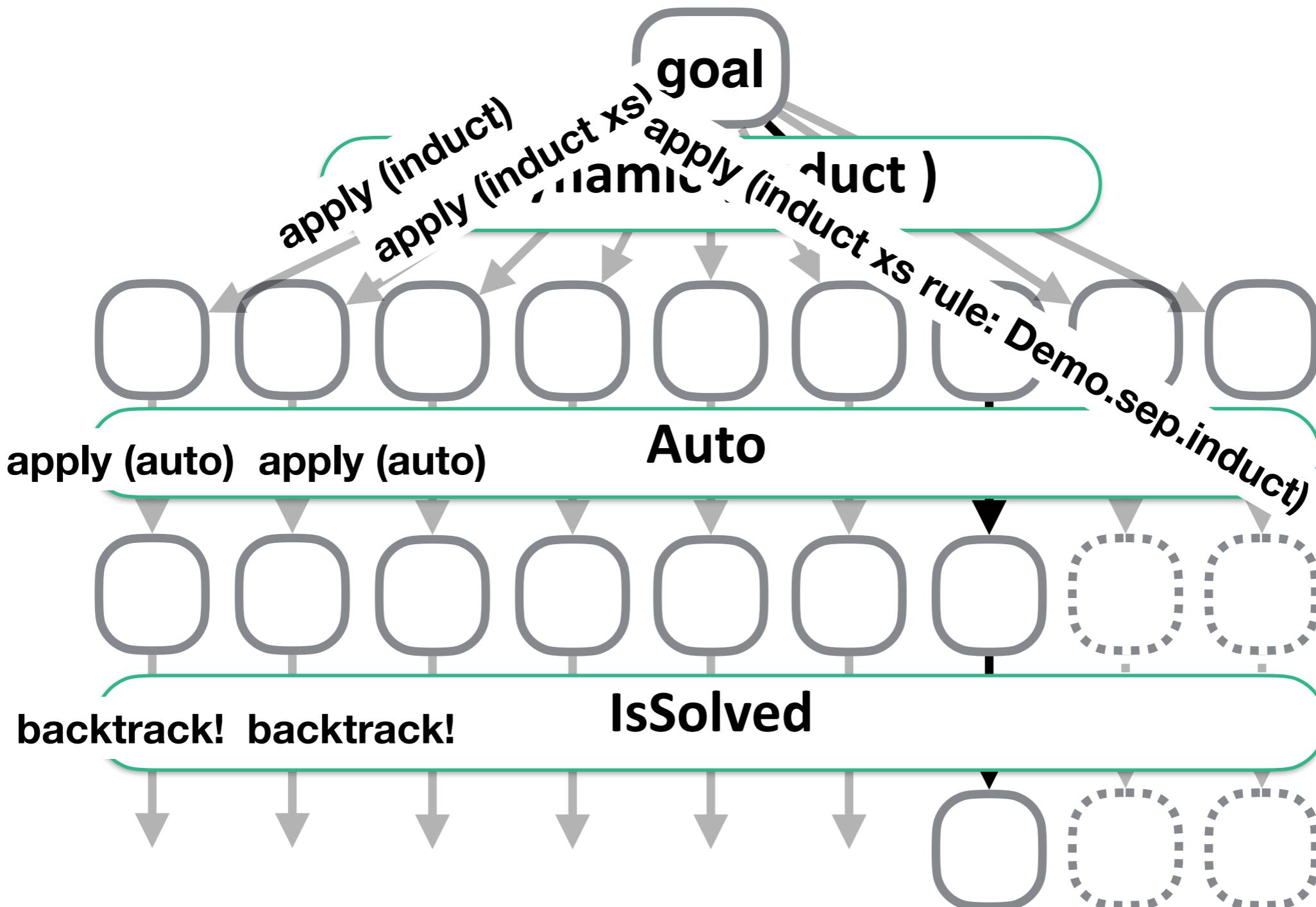
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

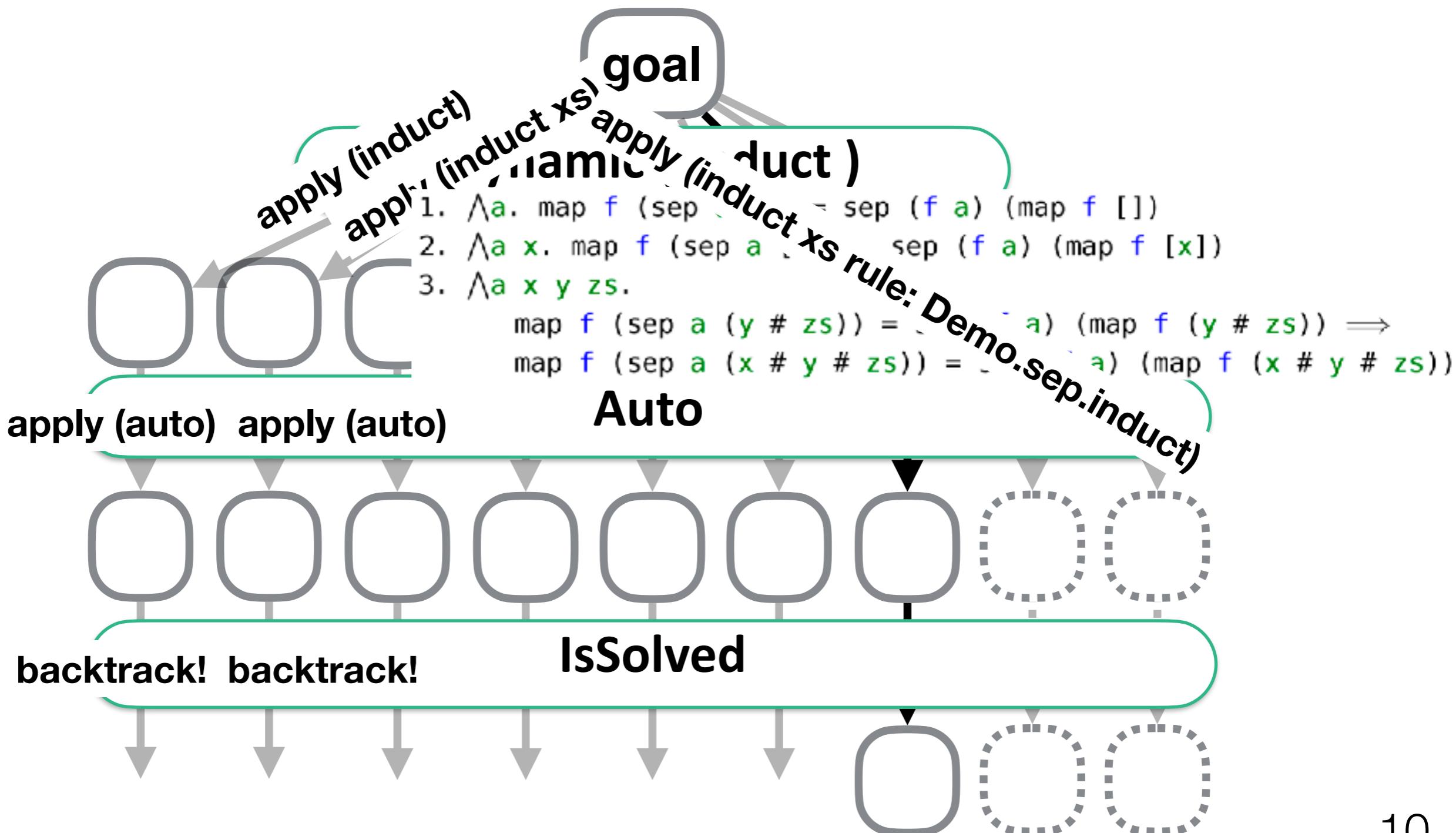
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

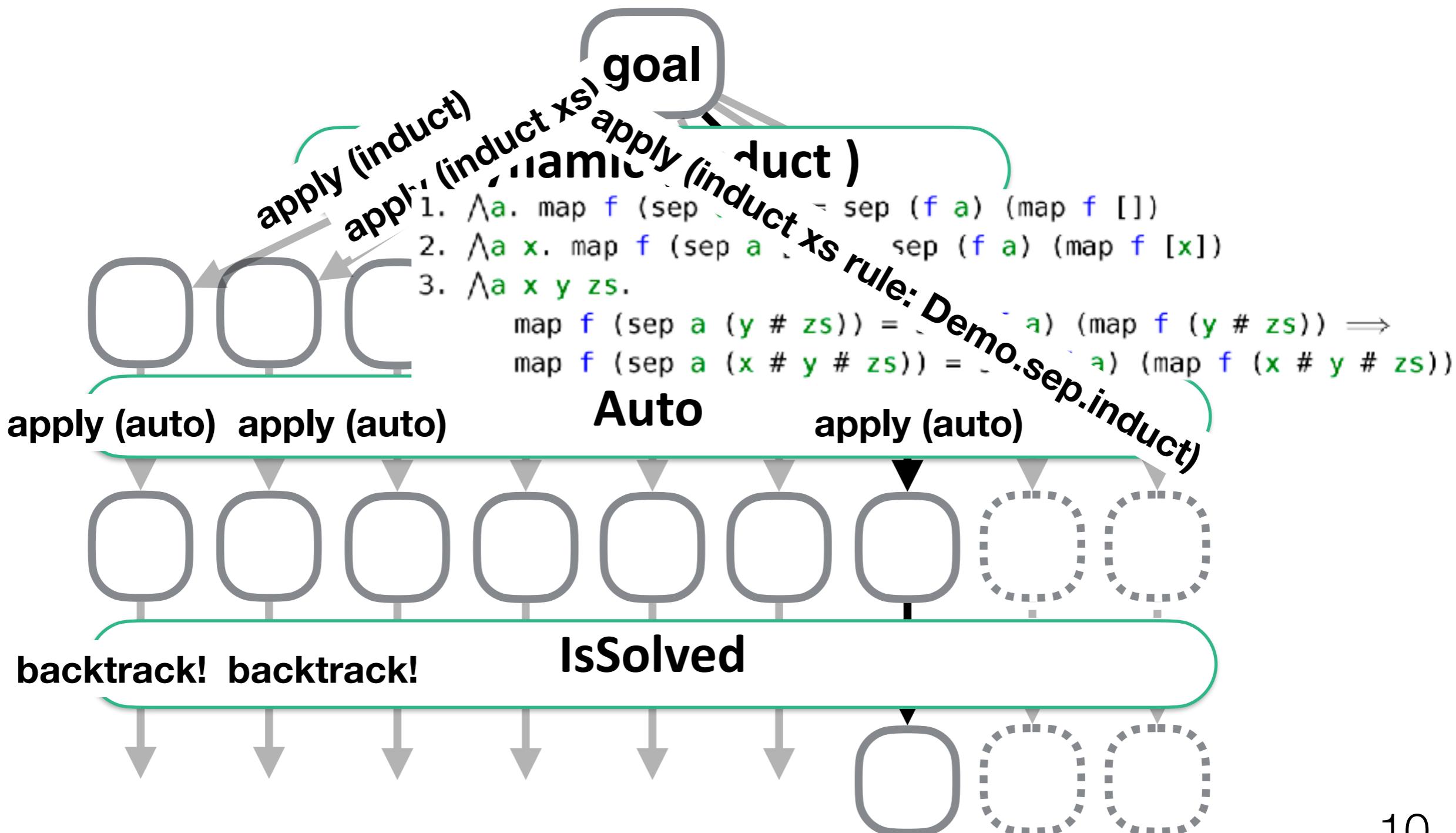
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

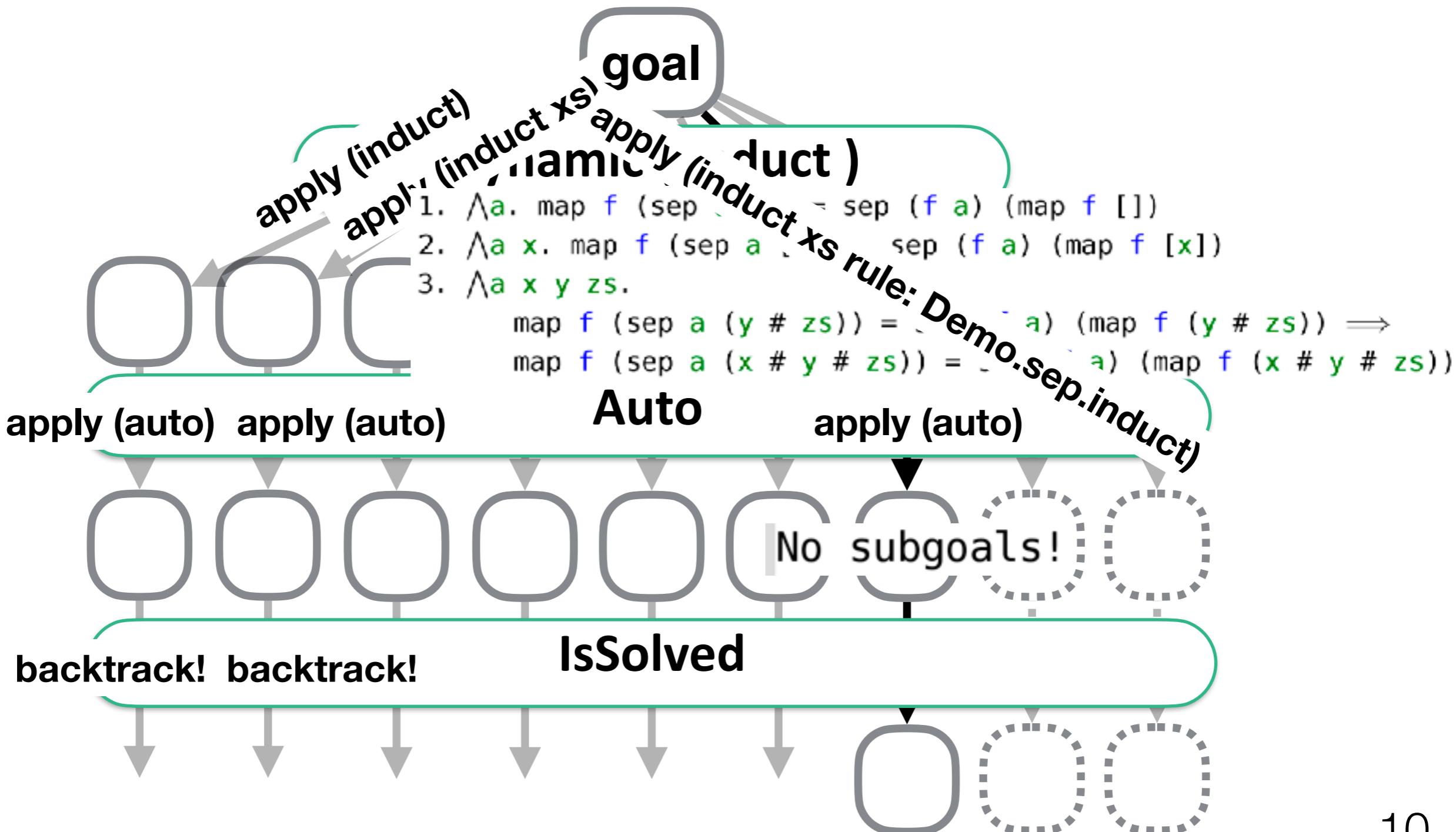
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

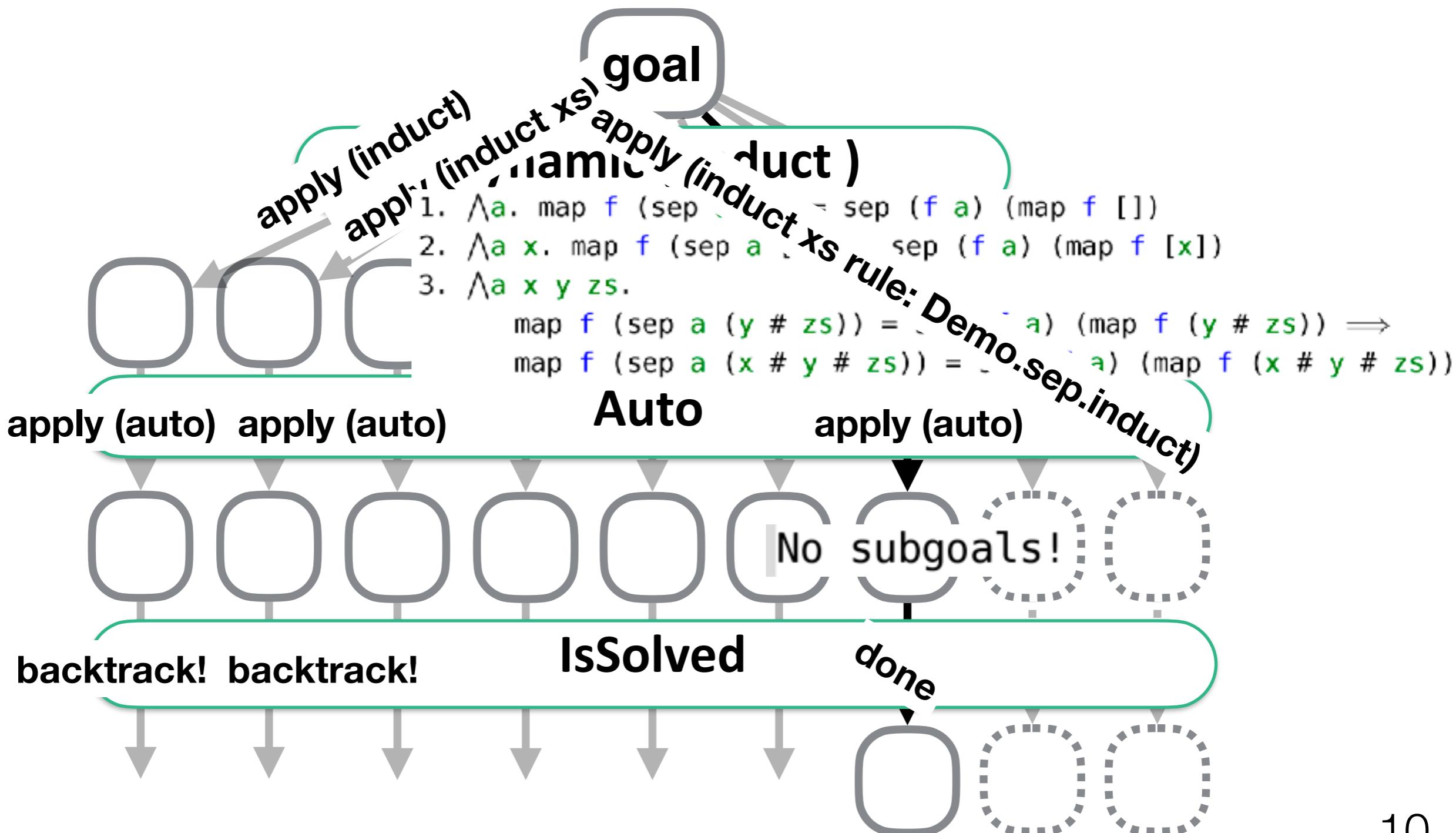
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

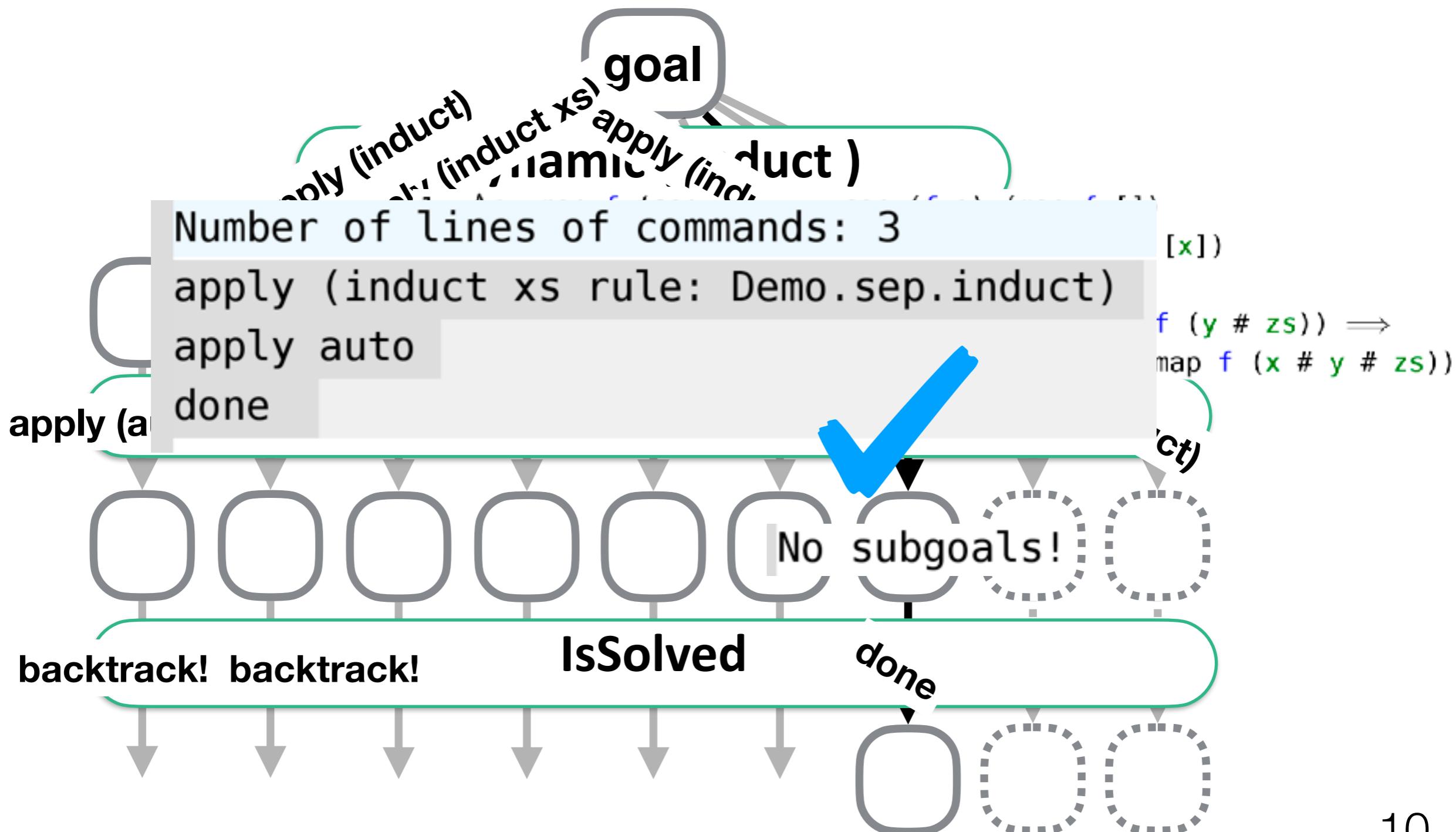
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

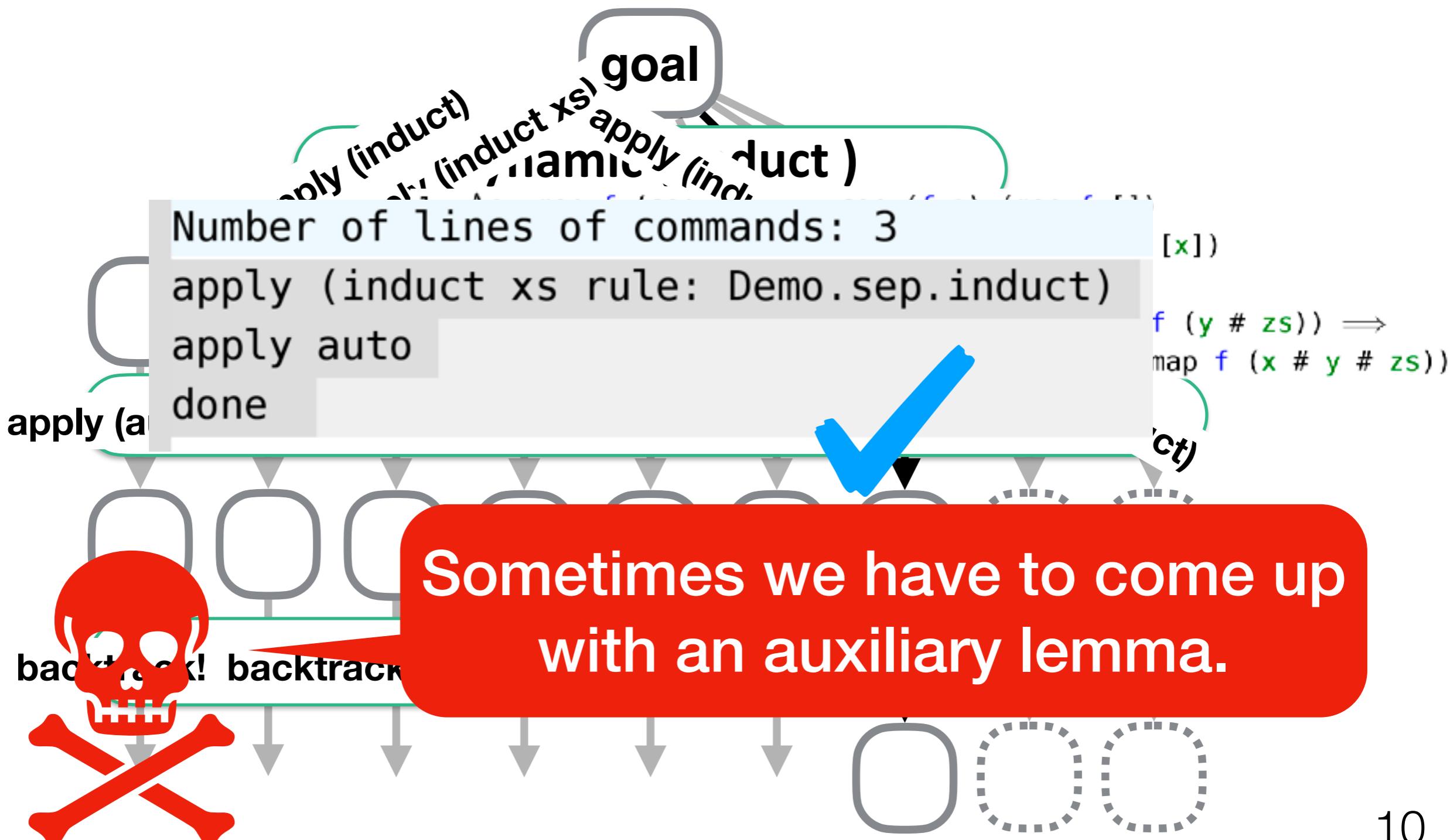
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"  
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO2: PaMpeR

The screenshot shows the Isabelle/HOL proof assistant interface. At the top, there's a toolbar with various icons. Below it, the main window displays a function definition and a lemma:

```
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)"

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

The lemma is highlighted with a yellow bar. The bottom part of the interface shows the proof state:

```
proof (prove)
goal (1 subgoal):
  1. map f (sep x xs) = sep (f x) (map f xs)
```

DEMO2: PaMpeR

```
File Browser Documentation
Demo.thy (-/Workplace/PSL/Innoshmack/)

18 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
19   "sep a [] = []" |
20   "sep a [x] = [x]" |
21   "sep a (x#y#zs) = x # a # sep a (y#zs)"

22 lemma "map f (sep x xs) = sep (f x) (map f xs)"
23 which_method
24
25
```

Promising methods for this proof goal are:

- simp with expectation of 0.411909082795
- auto with expectation of 0.159332679097
- rule with expectation of 0.0874798467373
- induction with expectation of 0.0613706657985
- metis with expectation of 0.052603838226
- induct with expectation of 0.0510162518838
- blast with expectation of 0.0485850209767
- fastforce with expectation of 0.035198219502
- unfold with expectation of 0.0236782046405
- simp all with expectation of 0.0185440627027

Output Query Sledgehammer Symbols
24.15 (5:89/2087) Input/output complete (isabelle,isabelle,UIF-8) innoshmack 307/5 2MB 11:17 PM

DEMO2: PaMpeR

The screenshot shows the Isabelle/HOL proof assistant interface. The main window displays a code editor with the following content:

```
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)"

lemma "map f (sep x xs) = sep (f x) (map f xs)"
  which_method
  why_method |induct|
```

The code is color-coded: blue for keywords like `fun`, `where`, `lemma`, and `map`; green for type annotations; and red for identifiers like `x`, `y`, `zs`, `f`, and `sep`. The line `|induct|` is highlighted with a red background.

At the bottom of the interface, there are several tabs: `Output`, `Query`, `Sledgehammer`, and `Symbols`. The status bar at the bottom right shows the current session: (isabelle,isabelle,UIF-8) in mode U.. 87/512MB 11:18 PM.

Because it is not true that the context has locally defined assumptions.

Because the underlying proof context has a recursive simplification rule related to

DEMO2: PaMpeR

```
File Browser Documentation
Demain.thy (-/Workplace/PSL/Innoshack/)

18 fun sep :: "'a ⇒ 'a list ⇒ 'a list" where
19   "sep a [] = []" |
20   "sep a [x] = [x]" |
21   "sep a (x#y#zs) = x # a # sep a (y#zs)"

22 Lemma "map f (sep x xs) = sep (f x) (map f xs)"
23 which_method
24 why_method induct
25 rank_method coinduction
```

coinduction 123 out of 239

Proof state Auto update Update Search: 100%

Output Query Sledgehammer Symbols

26,26 (6.35/2084) (isabelle,isabelle,UIF-8) innoshack 100/512MB 11:18 PM

preparation phase

**How does
PaMpeR work?**

recommendation phase

preparation phase

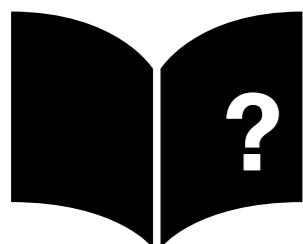
large proof corpora



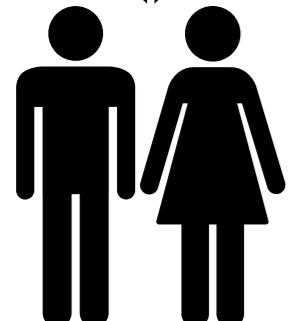
AFP and standard library

How does PaMpeR work?

recommendation phase



**proof
state**



**proof
engineer**

preparation phase

large proof corpora



AFP and standard library



STATISTICS

Archive of Formal Proofs (<https://www.isa-afp.org>)

Statistics

Number of Articles: 468

Number of Authors: 313

Number of lemmas: ~128,900

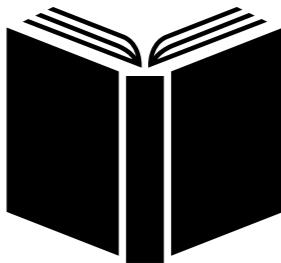
Lines of Code: ~2,170,300

Most used AFP articles:

Name	Used by ? articles
1. Collections	15
2. List-Index	14
3. Coinductive	12

preparation phase

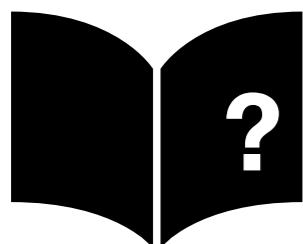
large proof corpora



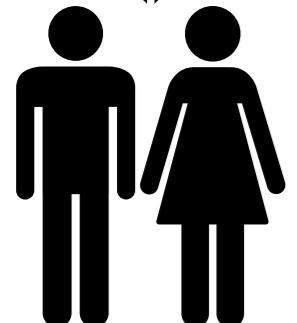
AFP and standard library

How does PaMpeR work?

recommendation phase



**proof
state**



**proof
engineer**

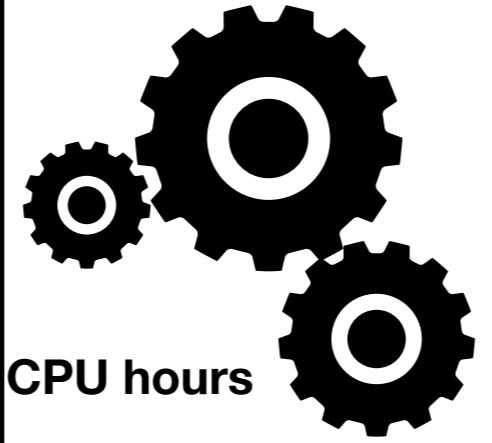
preparation phase

large proof corpora



AFP and standard library

full feature extractor

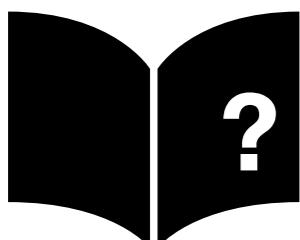


6021 CPU hours

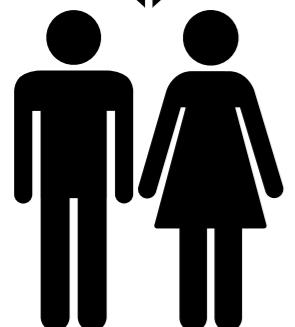
108 assertions

How does
PaMpeR work?

recommendation phase



proof
state



proof
engineer

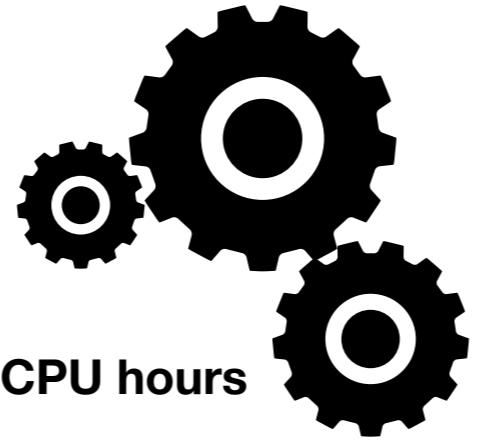
preparation phase

large proof corpora

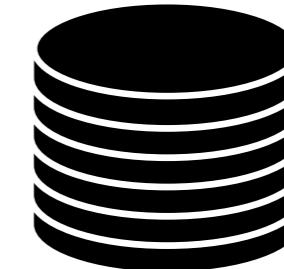


AFP and standard library

full feature extractor



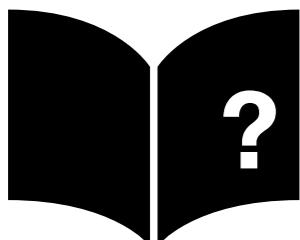
database (425334 data points)



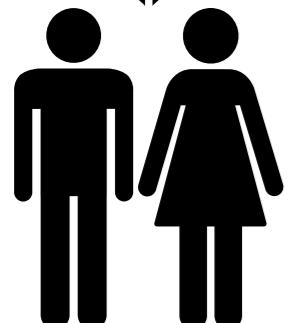
:: (tactic_name, [bool])

How does
PaMpeR work?

recommendation phase



proof
state



proof
engineer

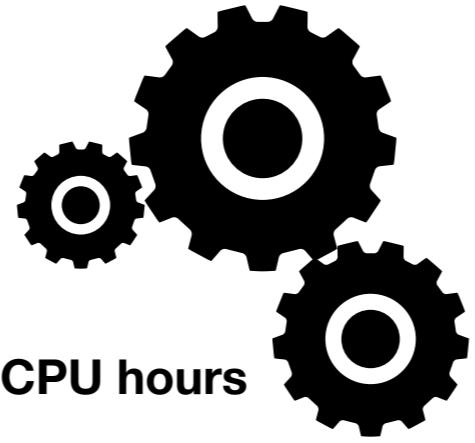
preparation phase

large proof corpora

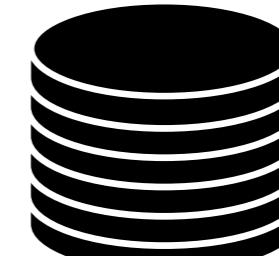


AFP and standard library

full feature extractor

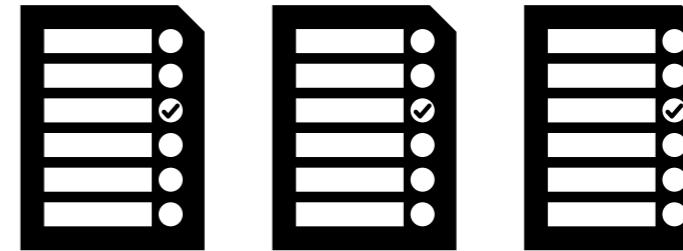


database (425334 data points)

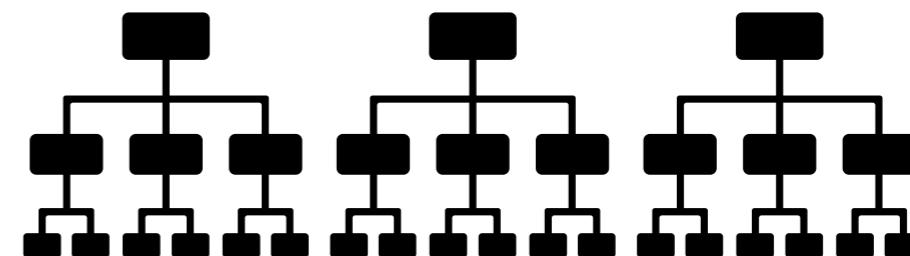


:: (tactic_name, [bool])

↓ preprocess

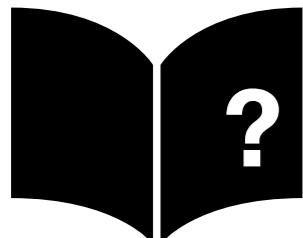


↓ decision tree construction

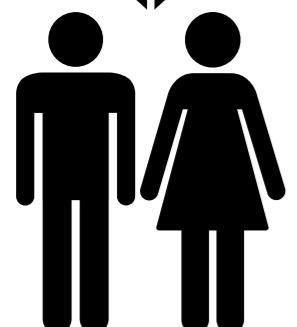


How does
PaMpeR work?

recommendation phase



proof
state



proof
engineer

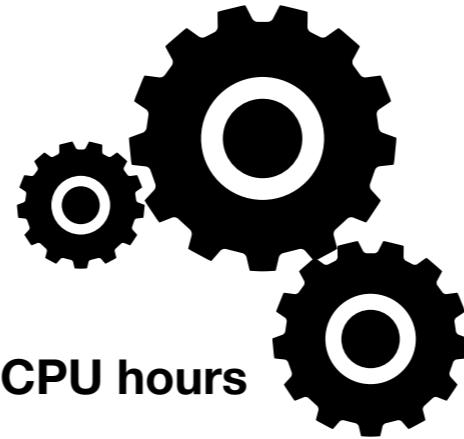
preparation phase

large proof corpora

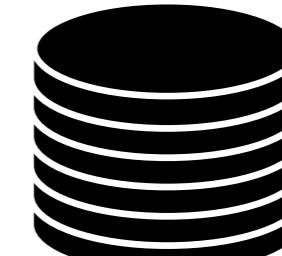


AFP and standard library

full feature extractor

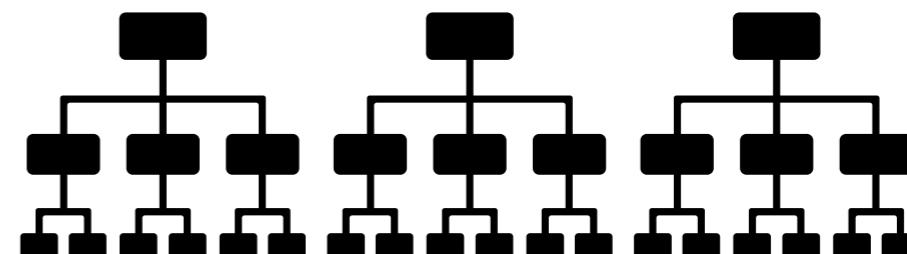


database (425334 data points)



:: (tactic_name, [bool])

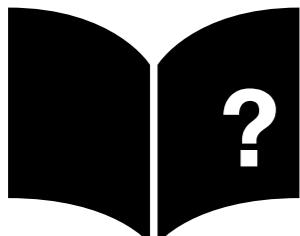
↓ preprocess



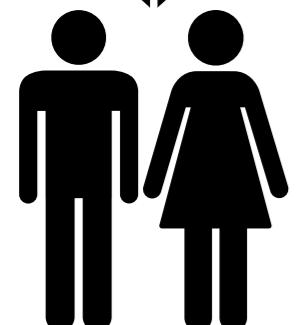
How does
PaMpeR work?

recommendation phase

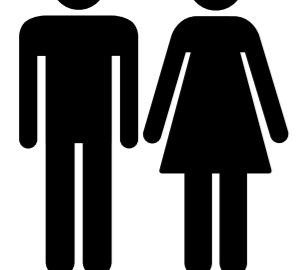
fast feature extractor



proof
state



proof
engineer



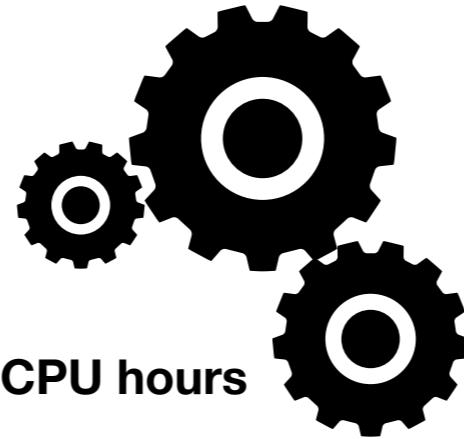
preparation phase

large proof corpora

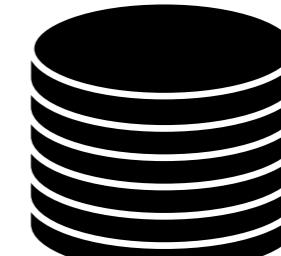


AFP and standard library

full feature extractor

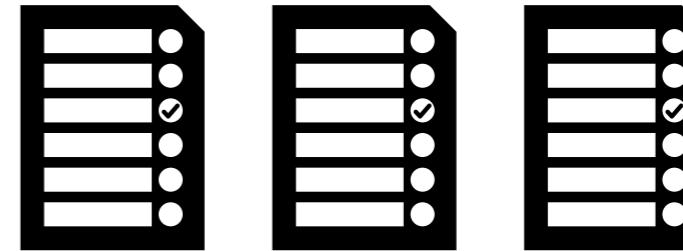


database (425334 data points)

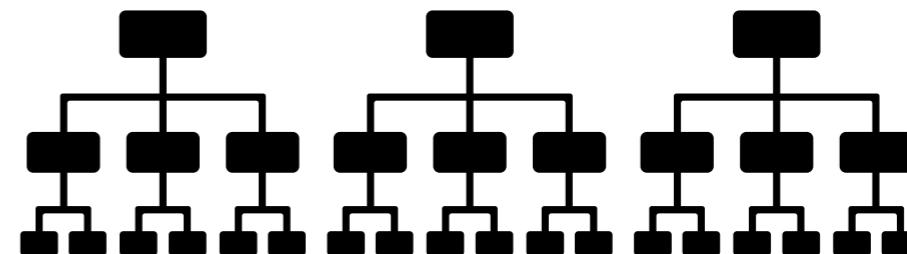


:: (tactic_name, [bool])

↓ preprocess



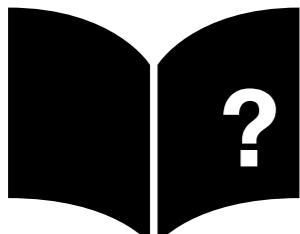
↓ decision tree construction



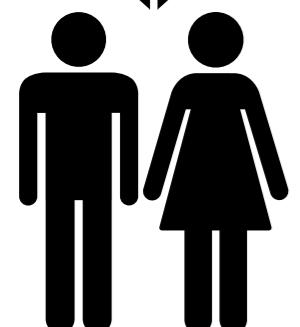
How does
PaMpeR work?

recommendation phase

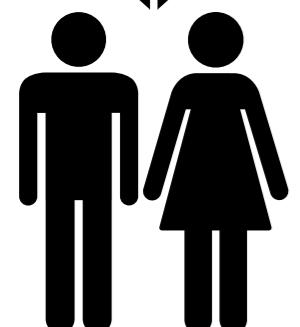
fast feature extractor



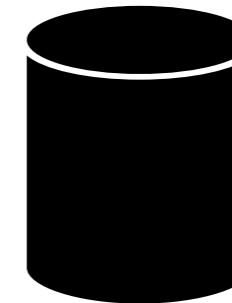
proof state

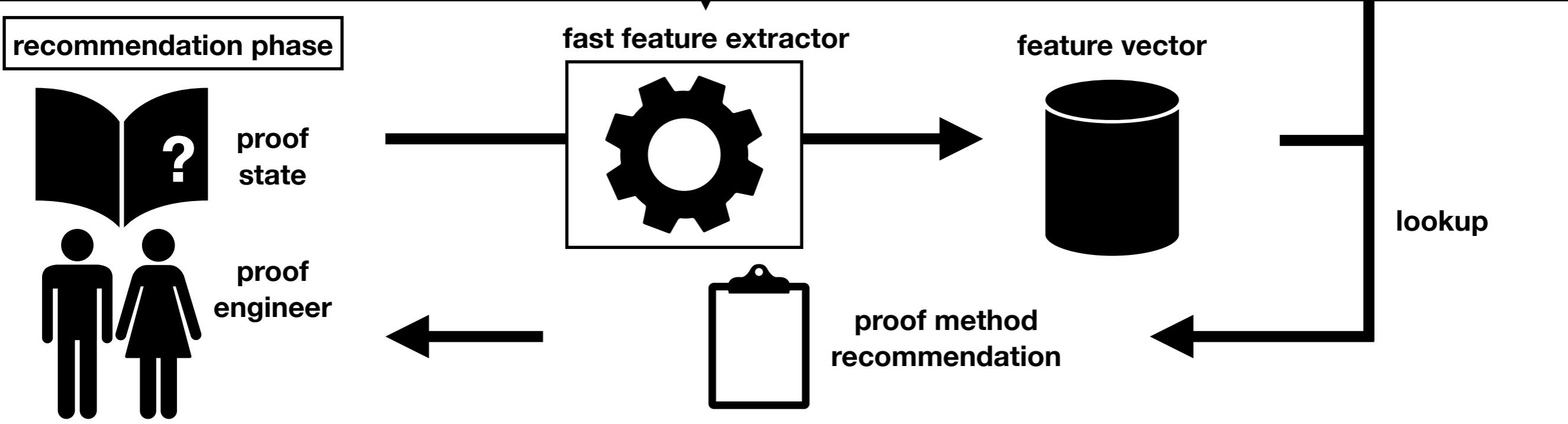
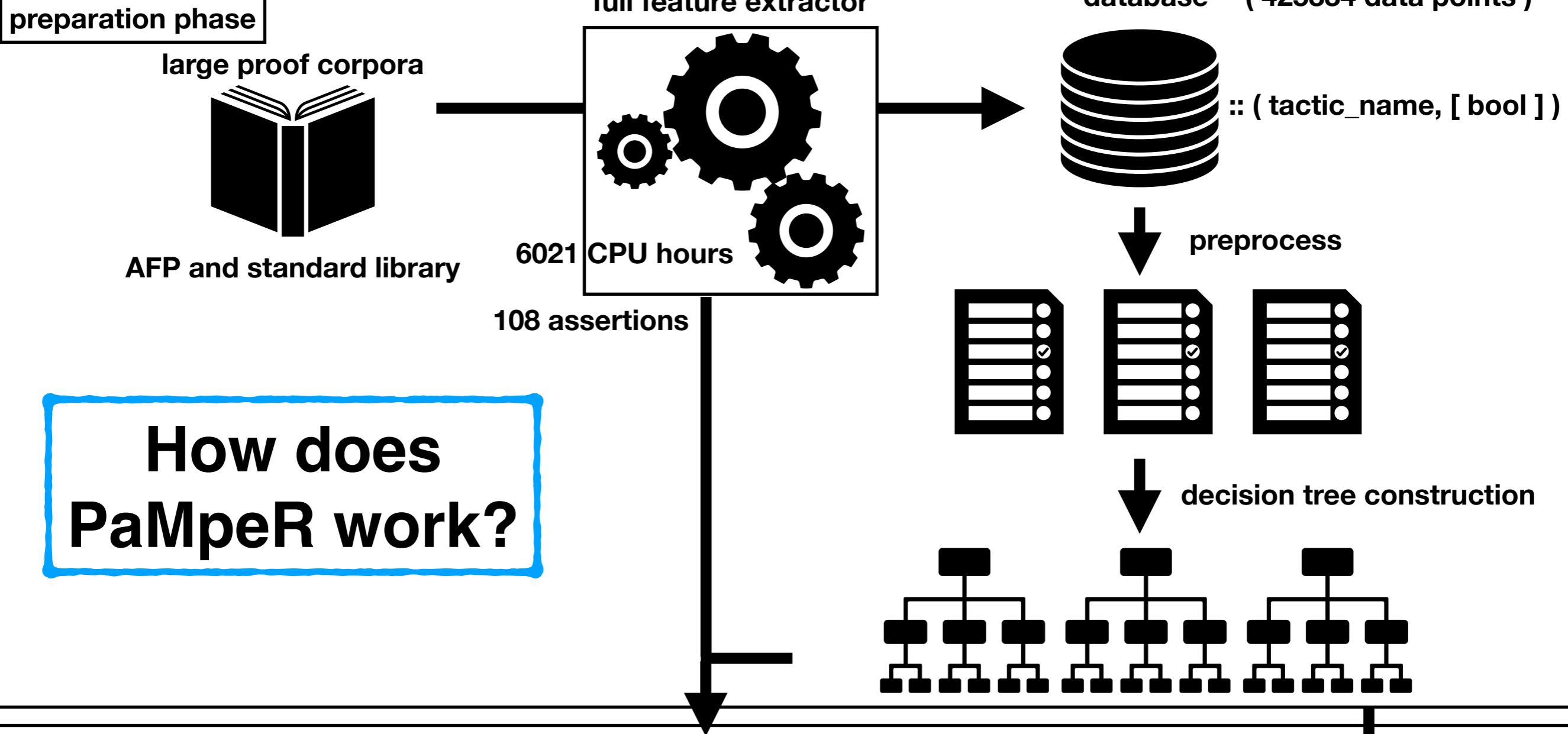


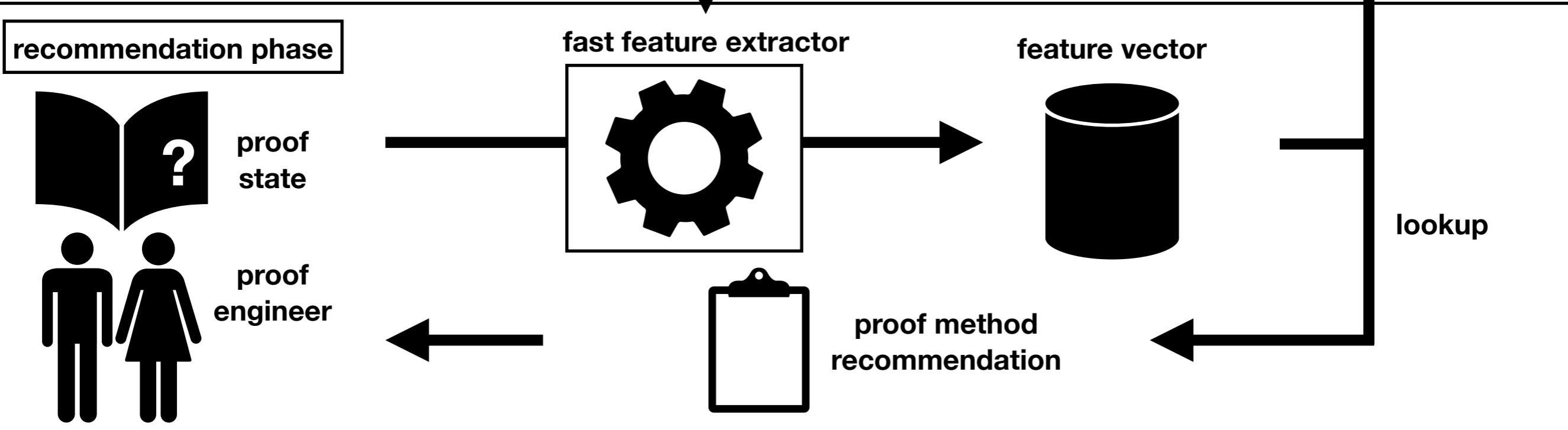
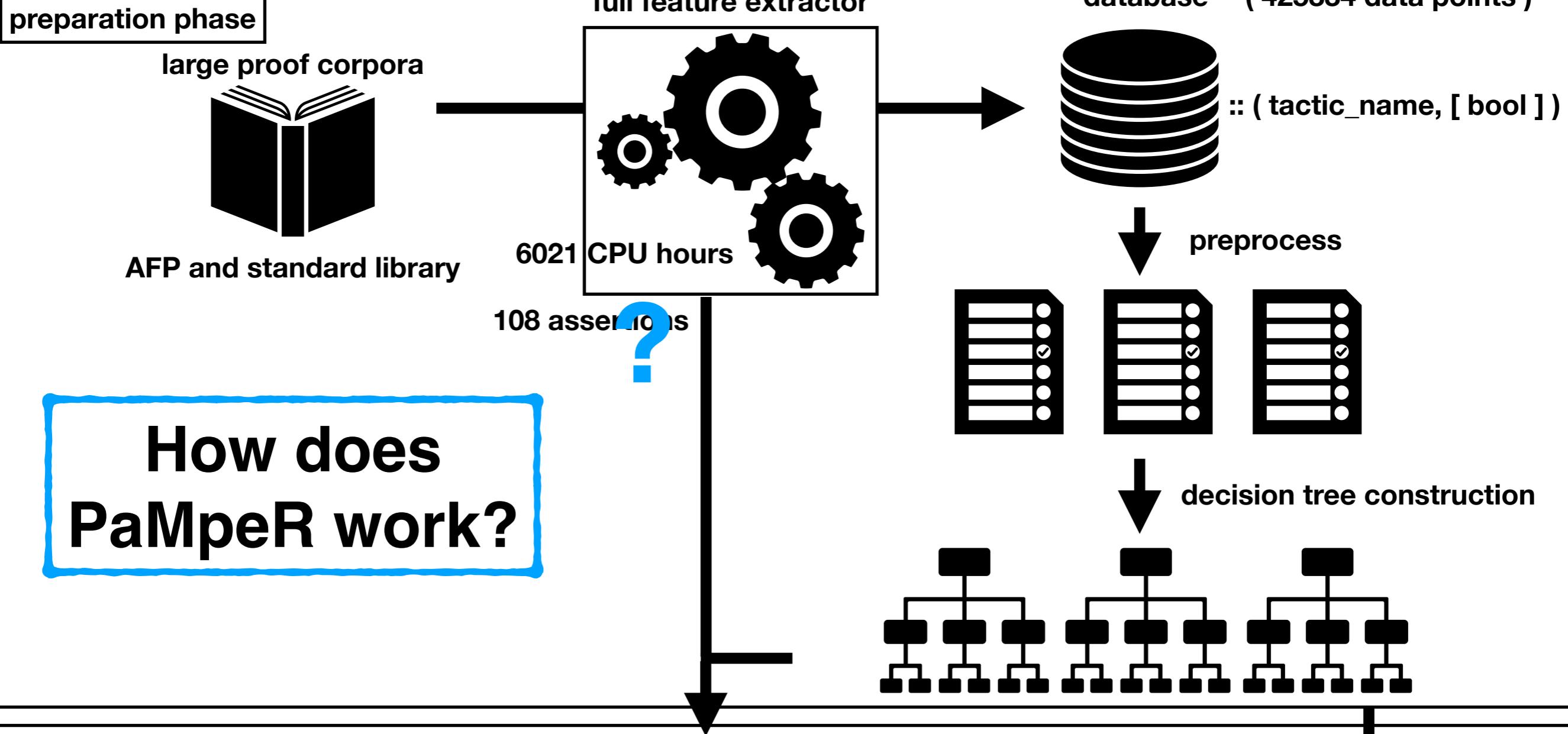
proof engineer

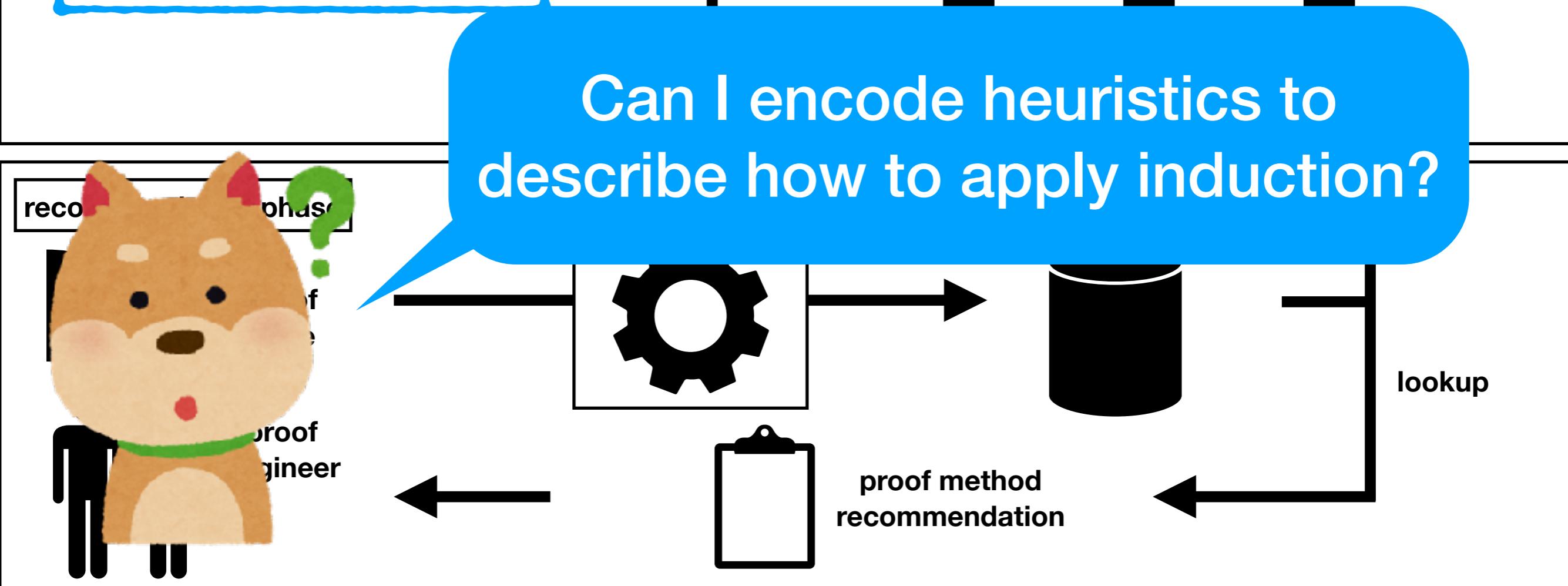
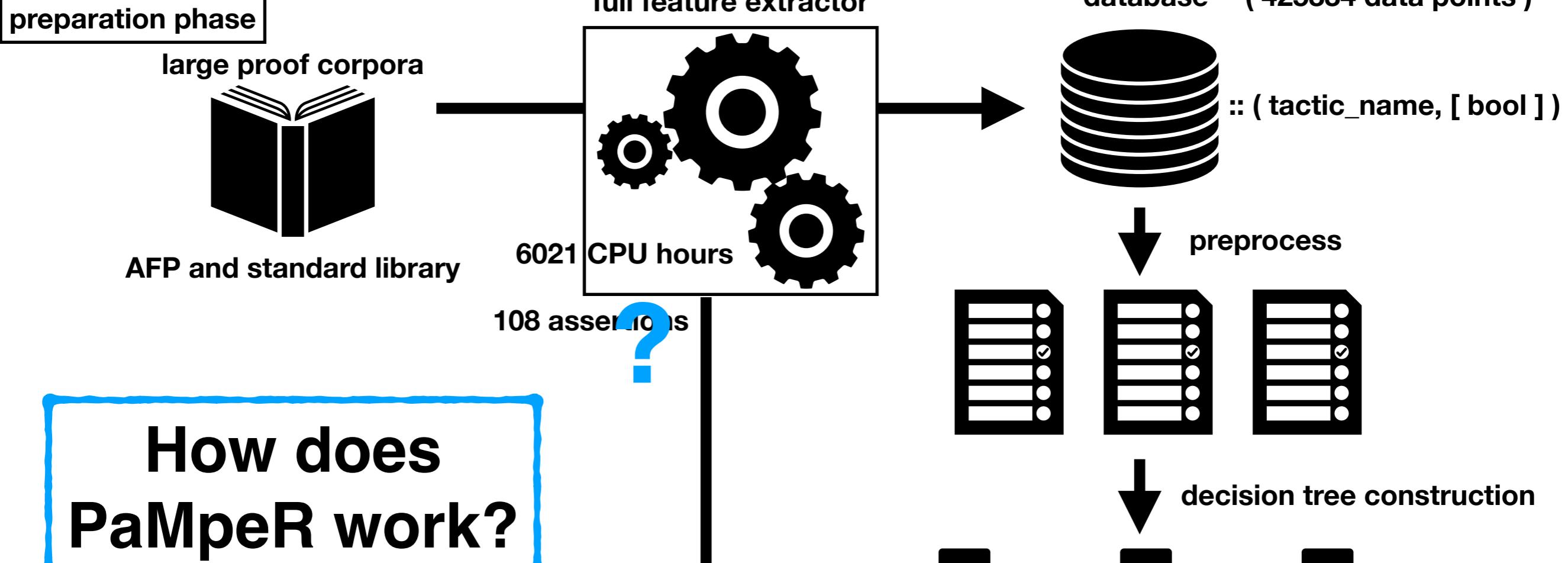


feature vector



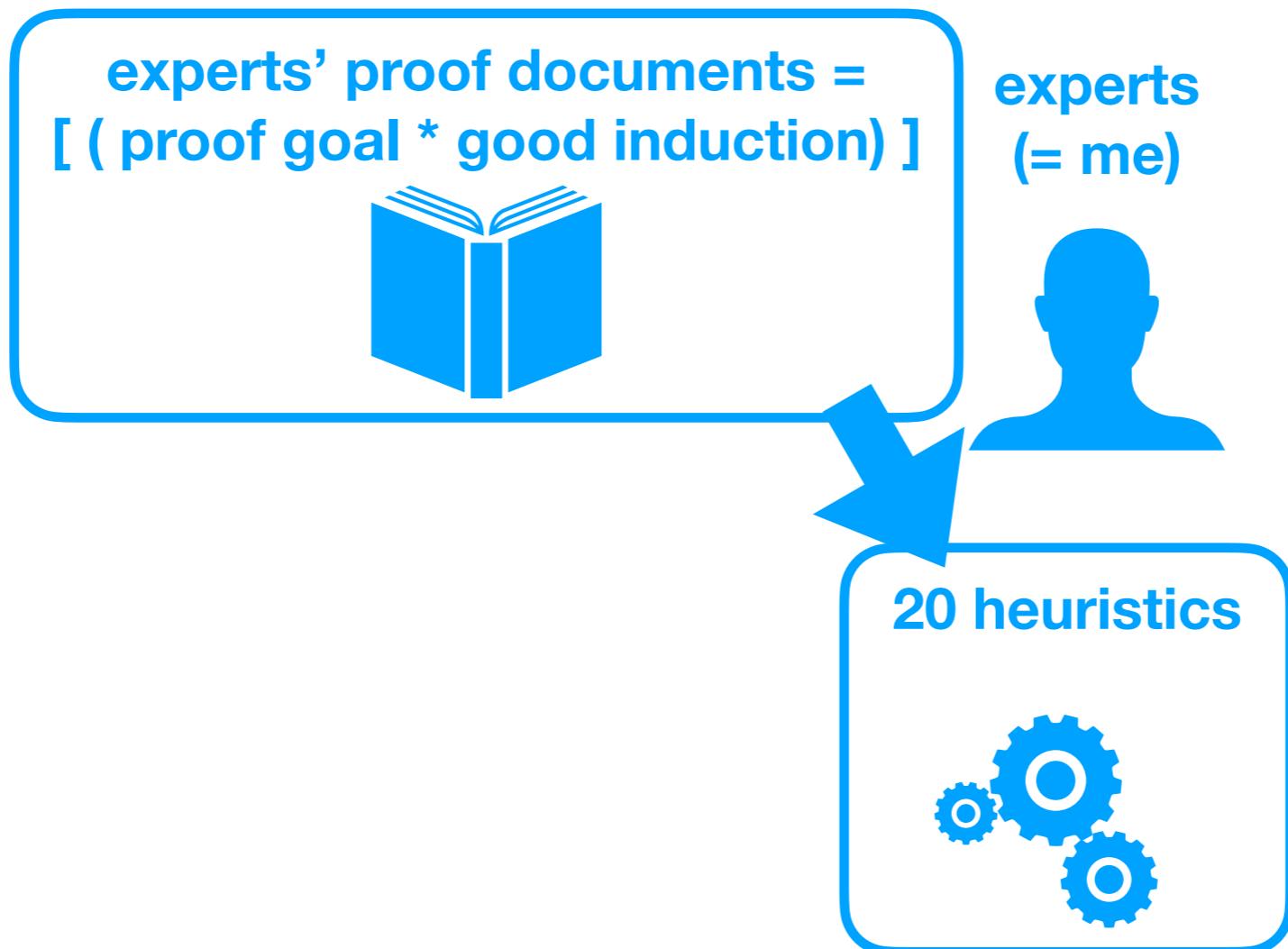






It is difficult to write induction heuristics.

It is difficult to write induction heuristics.



It is difficult to write induction heuristics.

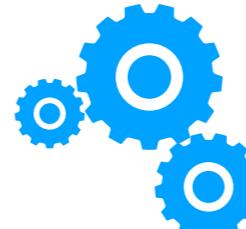
```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

experts' proof documents =
[(proof goal * good induction)]



experts
(= me)

20 heuristics



It is difficult to write induction heuristics.

```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

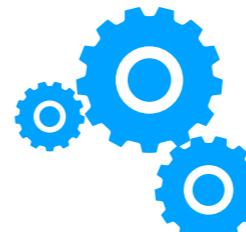
experts' proof documents =
[(proof goal * good induction)]



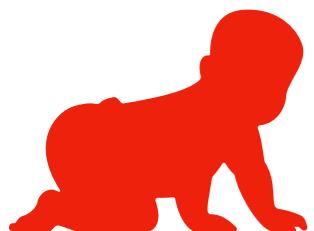
experts
(= me)



20 heuristics



new users



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

It is difficult to write induction heuristics.

```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

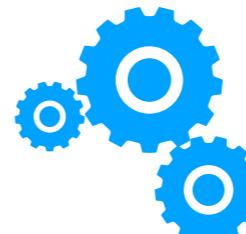
experts' proof documents =
[(proof goal * good induction)]



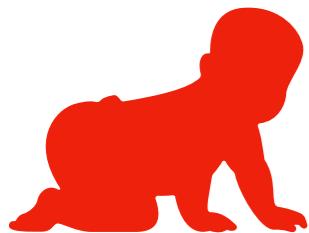
experts
(= me)



20 heuristics



new users



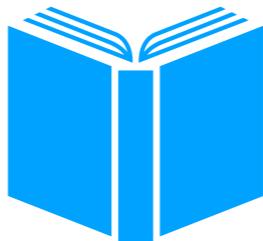
```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

new proof goal consisting of new constants and variables of new types!

It is difficult to write induction heuristics.

```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

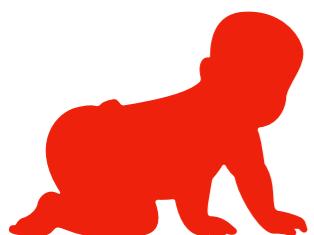
experts' proof documents =
[(proof goal * good induction)]



experts
(= me)

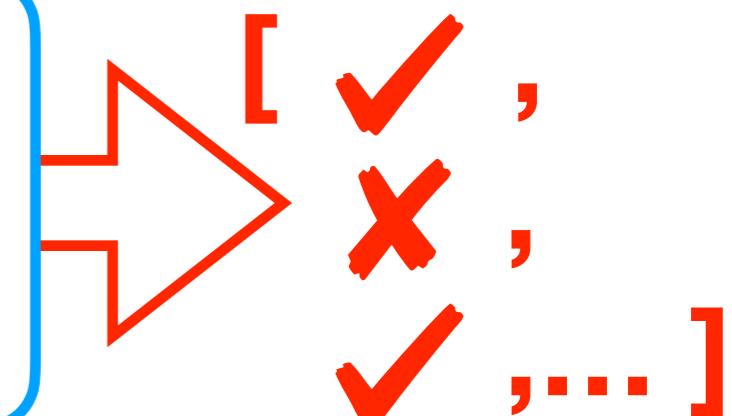
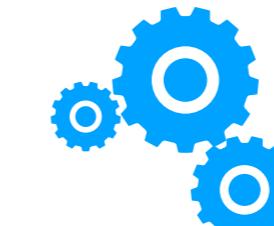


new users



```
proof goal candidate induction  
apply (  
induct is1 s stk  
rule: exec.induct)
```

20 heuristics



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

new proof goal consisting of new constants and variables of new types!

It is difficult to write induction heuristics.

① blue = what I can see before releasing smart_induct

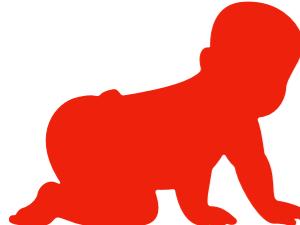
```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

experts' proof documents =
[(proof goal * good induction)]

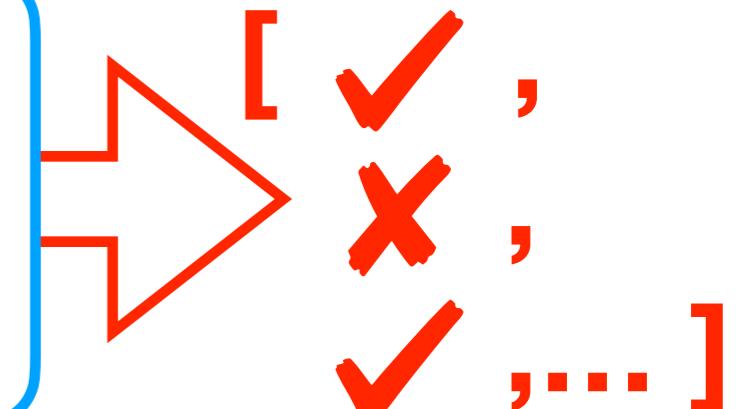
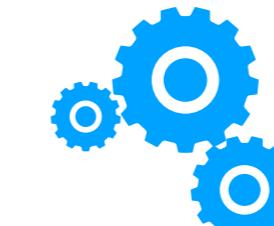


experts
(= me)



new users

proof goal candidate induction
apply (
induct is1 s stk
rule: exec.induct)

20 heuristics



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

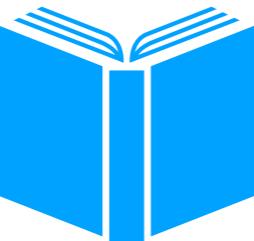
new proof goal consisting of new constants and variables of new types!

It is difficult to write induction heuristics.

① blue = what I can see before releasing smart_induct

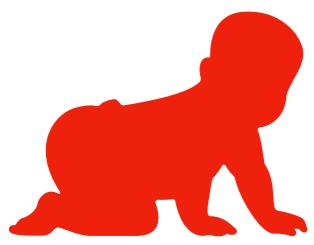
```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

experts' proof documents =
[(proof goal * good induction)]

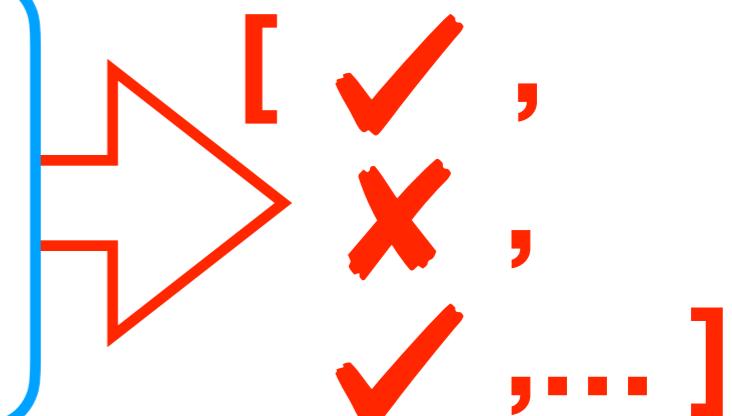
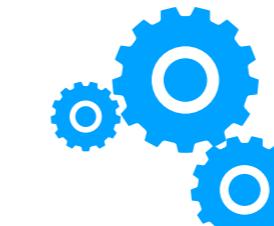


experts
(= me)



new users

proof goal candidate induction
apply (
induct is1 s stk
rule: exec.induct)

20 heuristics



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

new proof goal consisting of new constants and variables of new types!

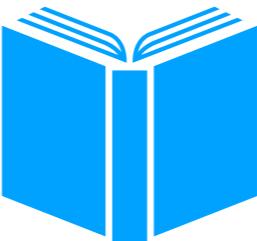
② red = what will be developed after releasing smart_induct

It is difficult to write induction heuristics.

① blue = what I can see before releasing smart_induct

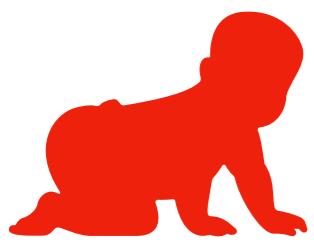
```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

experts' proof documents =
[(proof goal * good induction)]

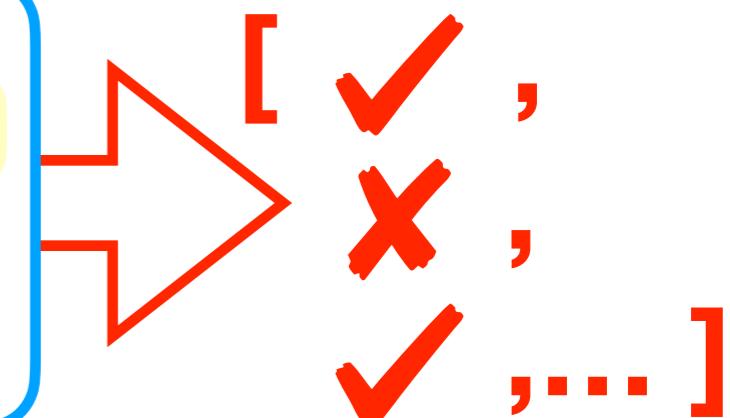


experts
(= me)



new users

proof goal candidate induction
apply (
induct is1 s stk
rule: exec.induct)

20 heuristics
in LiFtEr



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

new proof goal consisting of new constants and variables of new types!

② red = what will be developed after releasing smart_induct

It is difficult to write induction heuristics.

① blue = what I can see before releasing smart_induct

```
lemma "itrev xs ys = rev xs @ ys"  
by(induct xs ys rule:"itrev.induct") auto
```

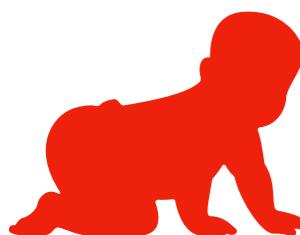
experts' proof documents =
[(proof goal * good induction)]



experts
(= me)

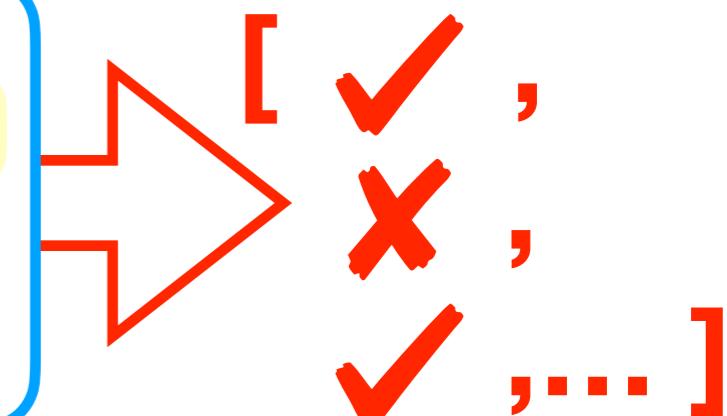
LiFtEr: Logical
Feature Extractor

new users



proof goal candidate induction
apply (
induct is1 s stk
rule: exec.induct)

20 heuristics
in LiFtEr



```
lemma "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

new proof goal consisting of new constants and variables of new types!

② red = what will be developed after releasing smart_induct

Example Heuristic in LiFtEr (in Abstract Syntax)

```
∃ r1 : rule. True
→
∃ r1 : rule.
  ∃ t1 : term.
    ∃ to1 : term_occurrence ∈ t1 : term.
      r1 is_rule_of to1
      ∧
      ∀ t2 : term ∈ induction_term.
        ∃ to2 : term_occurrence ∈ t2 : term.
          ∃ n : number.
            is_nth_argument_of (to2, n, to1)
            ∧
            t2 is_nth_induction_term n
```

Example Heuristic in LiFtEr (in Abstract Syntax)

implication



```
 $\exists r1 : \text{rule}. \text{True}$ 
 $\rightarrow$ 
 $\exists r1 : \text{rule}.$ 
 $\exists t1 : \text{term}.$ 
 $\exists to1 : \text{term\_occurrence} \in t1 : \text{term}.$ 
 $r1 \text{ is\_rule\_of } to1$ 
 $\wedge$ 
 $\forall t2 : \text{term} \in \text{induction\_term}.$ 
 $\exists to2 : \text{term\_occurrence} \in t2 : \text{term}.$ 
 $\exists n : \text{number}.$ 
 $\text{is\_nth\_argument\_of } (to2, n, to1)$ 
 $\wedge$ 
 $t2 \text{ is\_nth\_induction\_term } n$ 
```

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

↓

$\exists r1 : \text{rule}. \text{True}$

→

$\exists r1 : \text{rule}.$
 $\exists t1 : \text{term}.$
 $\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$
 $r1 \text{ is_rule_of } to1$

\wedge ← conjunction

$\forall t2 : \text{term} \in \text{induction_term}.$
 $\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$
 $\exists n : \text{number}.$
 $\text{is_nth_argument_of} (to2, n, to1)$

\wedge

$t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

↓

$\exists r1 : \text{rule}. \text{True}$

→

$\exists r1 : \text{rule}.$ variable for auxiliary lemmas

$\exists t1 : \text{term}.$

$\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$

$r1 \text{ is_rule_of } to1$

 ∧ conjunction

$\forall t2 : \text{term} \in \text{induction_term}.$

$\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$

$\exists n : \text{number}.$

$\text{is_nth_argument_of} (to2, n, to1)$

\wedge

$t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

↓
 $\exists r1 : \text{rule}. \text{True}$ → variable for auxiliary lemmas
 $\exists r1 : \text{rule}.$ ←
 $\exists t1 : \text{term}.$ ← variable for terms
 $\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$
 $r1 \text{ is_rule_of } to1$
 ∧ conjunction
 $\forall t2 : \text{term} \in \text{induction_term}.$
 $\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$
 $\exists n : \text{number}.$
 $\text{is_nth_argument_of } (to2, n, to1)$
 \wedge
 $t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

↓

$$\exists r1 : \text{rule}. \text{True}$$

→ variable for auxiliary lemmas

$$\exists r1 : \text{rule}.$$

$\exists t1 : \text{term}.$ ← variable for terms

$$\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$$

$r1 \text{ is_rule_of } to1$ ← variable for term occurrences

$$\wedge$$
 conjunction
$$\forall t2 : \text{term} \in \text{induction_term}.$$
$$\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$$
$$\exists n : \text{number}.$$
$$\text{is_nth_argument_of } (to2, n, to1)$$
$$\wedge$$
$$t2 \text{ is_nth_induction_term } n$$

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

↓
 $\exists r1 : \text{rule}. \text{True}$ → variable for auxiliary lemmas
 $\exists r1 : \text{rule}.$ ←
 $\exists t1 : \text{term}.$ ← variable for terms
 $\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$ ← variable for term occurrences
 $r1 \text{ is_rule_of } to1$ ←
 \wedge conjunction
 $\forall t2 : \text{term} \in \text{induction_term}.$
 $\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$
 $\exists n : \text{number}.$ ← variable for natural numbers
 $\text{is_nth_argument_of } (to2, n, to1)$
 \wedge
 $t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

implication

$\exists r1 : \text{rule}. \text{True}$

\rightarrow variable for auxiliary lemmas

$\exists r1 : \text{rule}.$

$\exists t1 : \text{term}.$ variable for terms

$\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$

$r1 \text{ is_rule_of } to1$ variable for term occurrences

\wedge conjunction

$\forall t2 : \text{term} \in \text{induction_term}.$

$\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$

$\exists n : \text{number}.$ variable for natural numbers

$\text{is_nth_argument_of } (to2, n, to1)$

\wedge

$t2 \text{ is_nth_induction_term } n$

universal
quantifier

Example Heuristic in LiFtEr (in Abstract Syntax)

implication existential quantifier

$\exists r1 : \text{rule}. \text{True}$ variable for auxiliary lemmas

$\exists r1 : \text{rule}.$

$\exists t1 : \text{term}.$ variable for terms

$\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$ variable for term occurrences

$r1 \text{ is_rule_of } to1$

\wedge conjunction

$\forall t2 : \text{term} \in \text{induction_term}.$

$\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$

$\exists n : \text{number}.$ variable for natural numbers

$\text{is_nth_argument_of} (to2, n, to1)$

\wedge

$t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

LiFtEr heuristic: (proof goal * induction arguments) -> bool

implication existential quantifier
↓ ↓
 $\exists r1 : \text{rule. True}$ variable for auxiliary lemmas
→
 $\exists r1 : \text{rule.}$ ← variable for terms
 $\exists t1 : \text{term.}$ ← variable for term occurrences
 $\exists to1 : \text{term_occurrence} \in t1 : \text{term.}$ ← variable for natural numbers
 $r1 \text{ is_rule_of } to1$ conjunction
 \wedge
 $\forall t2 : \text{term} \in \text{induction_term.}$
 $\exists to2 : \text{term_occurrence} \in t2 : \text{term.}$
 $\exists n : \text{number.}$ ← variable for natural numbers
 $\text{is_nth_argument_of } (to2, n, to1)$
 \wedge
 $t2 \text{ is_nth_induction_term } n$

Example Heuristic in LiFtEr (in Abstract Syntax)

LiFtEr heuristic: (proof goal * induction arguments) -> bool
should be true if induction is good
should be false if induction is bad

implication existential quantifier

$\exists r1 : \text{rule}. \text{True}$ variable for auxiliary lemmas

$\exists r1 : \text{rule}.$

$\exists t1 : \text{term}.$ variable for terms

$\exists to1 : \text{term_occurrence} \in t1 : \text{term}.$ variable for term occurrences

$r1 \text{ is_rule_of } to1$

\wedge conjunction

$\forall t2 : \text{term} \in \text{induction_term}.$

$\exists to2 : \text{term_occurrence} \in t2 : \text{term}.$

$\exists n : \text{number}.$ variable for natural numbers

$\text{is_nth_argument_of} (to2, n, to1)$

\wedge

$t2 \text{ is_nth_induction_term } n$

DEMO3: smart_induct

The screenshot shows the Isabelle/HOL proof assistant interface. The main window displays the following code:

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

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

The code defines a function `sep` that takes a type class `'a` and a list of `'a`s, and returns a list of `'a`s. It uses pattern matching to handle three cases: an empty list, a single-element list, and a list with multiple elements. A lemma states that applying a function `f` to each element of a list separated by `a` is equivalent to separating the results of `f` applied to each element by `a`.

Below the code, the proof state is shown:

```
proof (prove)
goal (1 subgoal):
  1. map f (sep a xs) = sep (f a) (map f xs)
```

The proof state indicates there is one subgoal to prove. The goal is to show that the two sides of the equation are equal.

DEMO3: smart_induct

The screenshot shows the Isabelle IDE interface. The top bar has icons for file operations like Open, Save, and Print. The title bar says "Induction_Demo.thy (~/Workplace/PSL/Smart_Induct/Example)". The left sidebar has sections for File Browser, Documentation, and Sidekick. The main text area contains the following code:

```
1 fun sep :: "'a :: type list ⇒ 'a :: type list" where
2   "sep a [] = []" |
3   "sep a [x] = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"
5
6 lemma "map f (sep a xs) = sep (f a) (map f xs)"
7   smart_induct
```

Proof state: Auto update: Update: Search: 100% ▾

smart_induct started producing combinations of induction arguments.
smart_induct produced 256 combinations of arguments for the induct method.
... out of which 40 of them return some results.
... out of which only 40 of them passes the second screening stage.
LiFtEr assertions are evaluating the first 40 of them.
Try these 10 most promising inductions!
1st candidate is apply (induct a xs rule: Induction_Demo.sep.induct)
(* The score is 27 out of 27. *)
2nd candidate is apply (induct xs arbitrary: a f)
(* The score is 27 out of 27. *)

DEMO3: smart_induct

The screenshot shows the Isabelle/HOL proof assistant interface. The main window displays a file named "Induction_Demo.thy" containing the following code:

```
File Browser Documentation
Induction_Demo.thy (-/Workplace/PSL/Smart_Induct/Example/)

1 fun sep :: "'a :: type list ⇒ 'a :: type list" where
2   "sep a [] = []" |
3   "sep a [x] = [x]" |
4   "sep a (x#y#zs) = x # a # sep a (y#zs)"

5 lemma "map f (sep a xs) = sep (f a) (map f xs)"
6   smart_induct
7   apply(induction a xs rule: sep.induct)
8   apply auto
9   done
```

The cursor is positioned at the end of the "done" command. Below the code editor, the proof state is shown:

```
proof (prove)
goal:
No subgoals!
```

The bottom status bar indicates the current proof state: "33.11 {8 / 4885} (isabelle,isabelle,UIF-8-Isabelle) | n m r o U.. 191/512MB 1:42 AM".

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: **Coincidence Rates** of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: **Coincidence Rates** of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

Table 2: Coincidence Rates of `smart_induct` Based Only on Induction Terms.

theory	total	top_1	top_3	top_5	top_10
Nearest_Neighbors	16	5 (31%)	12 (75%)	15 (94%)	15 (94%)

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

Table 2: Coincidence Rates of `smart_induct` Based Only on Induction Terms.

theory	total	top_1	top_3	top_5	top_10
Nearest_Neighbors	16	5 (31%)	12 (75%)	15 (94%)	15 (94%)

smart_induct... does it work?

Table 1: Coincidence Rates of smart_induct.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

Table 2: Coincidence Rates of smart_induct Based Only on Induction Terms.

theory	total	top_1	top_3	top_5	top_10
Nearest_Neighbors	16	5 (31%)	12 (75%)	15 (94%)	15 (94%)

Overall, the results are good!

But finding out what variables to generalise remains as a challenge!

smart_induct... does it work?

Table 1: Coincidence Rates of `smart_induct`.

theory	total	top_1	top_3	top_5	top_10
DFS	10	6 (60%)	9 (90%)	9 (90%)	9 (90%)
Nearest_Neighbors	16	3 (19%)	4 (25%)	7 (44%)	12 (75%)
RST_RBT	24	24 (100%)	24 (100%)	24 (100%)	24 (100%)
sum	50	33 (65%)	37 (74%)	40 (80%)	45 (90%)

Table 2: Coincidence Rates of `smart_induct` Based Only on Induction Terms.

theory	total	top_1	top_3	top_5	top_10
Nearest_Neighbors	16	5 (31%)	12 (75%)	15 (94%)	15 (94%)

Overall, the results are good!

But finding out what variables to generalise remains as a challenge!

...which will be solved by LiFtEr2.

Cogent

As a contributor.

PSL

As the main developer.

PaMpeR

LiFtEr

smart_induct

Cogent

As a contributor.

PSL

As the main developer.

PaMpeR

LiFtEr

smart_induct

DSL with search & tracing

Cogent

As a contributor.

PSL

As the main developer.

PaMpeR

DSL with search & tracing

LiFtEr

ML to recommend proof tactics

smart_induct

Cogent

As a contributor.

PSL

As the main developer.

PaMpeR

DSL with search & tracing

LiFtEr

ML to recommend proof tactics

smart_induct

DSL to implement feature extractor

Cogent

As a contributor.

PSL

As the main developer.

PaMpeR

DSL with search & tracing

LiFtEr

ML to recommend proof tactics

smart_induct

DSL to implement feature extractor

Recommendation base on LiFtEr

Cogent

As a contributor.

Certifying compiler

PSL

As the main developer.

DSL with search & tracing

PaMpeR

ML to recommend proof tactics

LiFtEr

DSL to implement feature extractor

smart_induct

Recommendation base on LiFtEr

Cogent

As a contributor.

Certifying compiler

PSL

As the main developer.

DSL with search & tracing

PaMpeR

ML to recommend proof tactics

LiFtEr

DSL to implement feature extractor

smart_induct

Recommendation base on LiFtEr



I am a FM / PL / ML / AI practitioner.