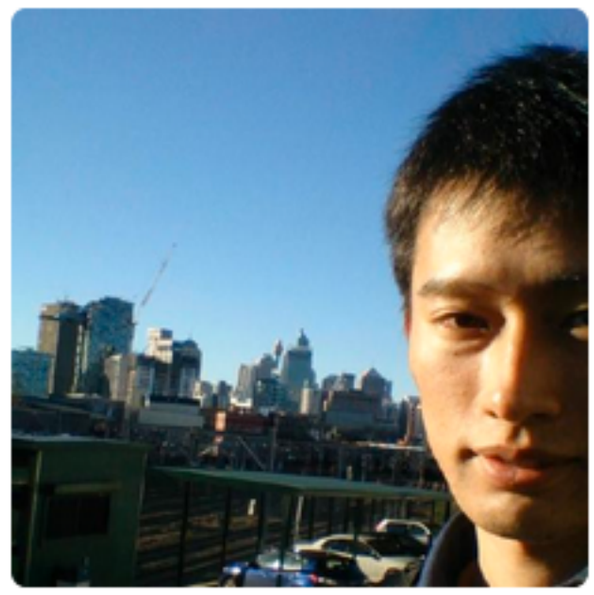
Interactive theorem proving in Isabelle/HOL with AI&ML&DSL

Yutaka Nagashima



Yutaka Ng

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

Yutaka Nagashima

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

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

with Prof. Gerwin Klein

Cogent

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

PSL

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Isabelle HOL

(seL4)

Security. Performance. Proof.

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

Isabelle HOL

PSL

PaMpeR

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

PSL

PaMpeR

2018 ~ 2020/21

http://ai4reason.org/members.html

with Dr. Josef Urban

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

PSL

PaMpeR

2018 ~ 2020/21

http://ai4reason.org/members.html

with Dr. Josef Urban

LiFtEr

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

PSL

PaMpeR

2018 ~ 2020/21

http://ai4reason.org/members.html

with Dr. Josef Urban

LiFtEr

smart_induct

Background

As a contributor.

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

Isabelle HOL

(seL4)

Security. Performance. Proof.

Cogent

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

Isabelle HOL

PSL

PaMpeR

2018 ~ 2020/21

http://ai4reason.org/members.html

with Dr. Josef Urban

Isabelle HOL

LiFtEr

smart_induct

Background

2013 ~ 2017

Intern & Engineer

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

with Prof. Gerwin Klein

As a contributor.

Cogent

As the main developer.

PhD in AI for theorem proving

2017 ~ 2018

http://cl-informatik.uibk.ac.at/users/cek/

with Prof. Cezary Kaliszyk

2018 ~ 2020/21

http://ai4reason.org/members.html

with Dr. Josef Urban

PSL

PaMpeR

LiFtEr

smart_induct

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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)"
```

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

```
consts
  sep :: "'a ⇒ 'a list ⇒ 'a list"
Found termination order: "(λp. length (snd p)) <*mlex*> {}"
```

Output | Query | Sledgehammer | Symbols

10,42 (221/935)

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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]"
```

Proof state Auto update Update Search: 100%

```
"[0, 1, 0, 1, 0]"
  :: "int list"
```

Output Query Sledgehammer Symbols

12,29 (251/963) (isabelle,isabelle,UTF-8-Isabelle) 1:17 AM

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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]"



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

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

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

Output | Query | Sledgehammer | Symbols

17,44 (304/1012) | Input/output complete | (isabelle,isabelle,UTF-8-Isabelle) | 241/535MB 1:17 AM

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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)"
```

my language (PSL)

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

Output | Query | Sledgehammer | Symbols

14,58 (310/1067)

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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 state | Auto update | Update | Search: | 100%

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

Output | Query | Sledgehammer | Symbols

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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 state | Auto update | Update | Search: | 100%

```
Number of lines of commands: 3
apply (induct xs rule: Example.sep.induct)
apply auto
done
```

Output | Query | Sledgehammer | Symbols

18,18 (379/1084) (isabelle,isabelle,UTF-8-Isabelle) 251/535MB 1:05 AM

DEMO1: PSL

Example.thy (~/Workplace/PSL/PGT/)

```
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
apply (induct xs rule: Example.sep.induct)
apply auto
done
```

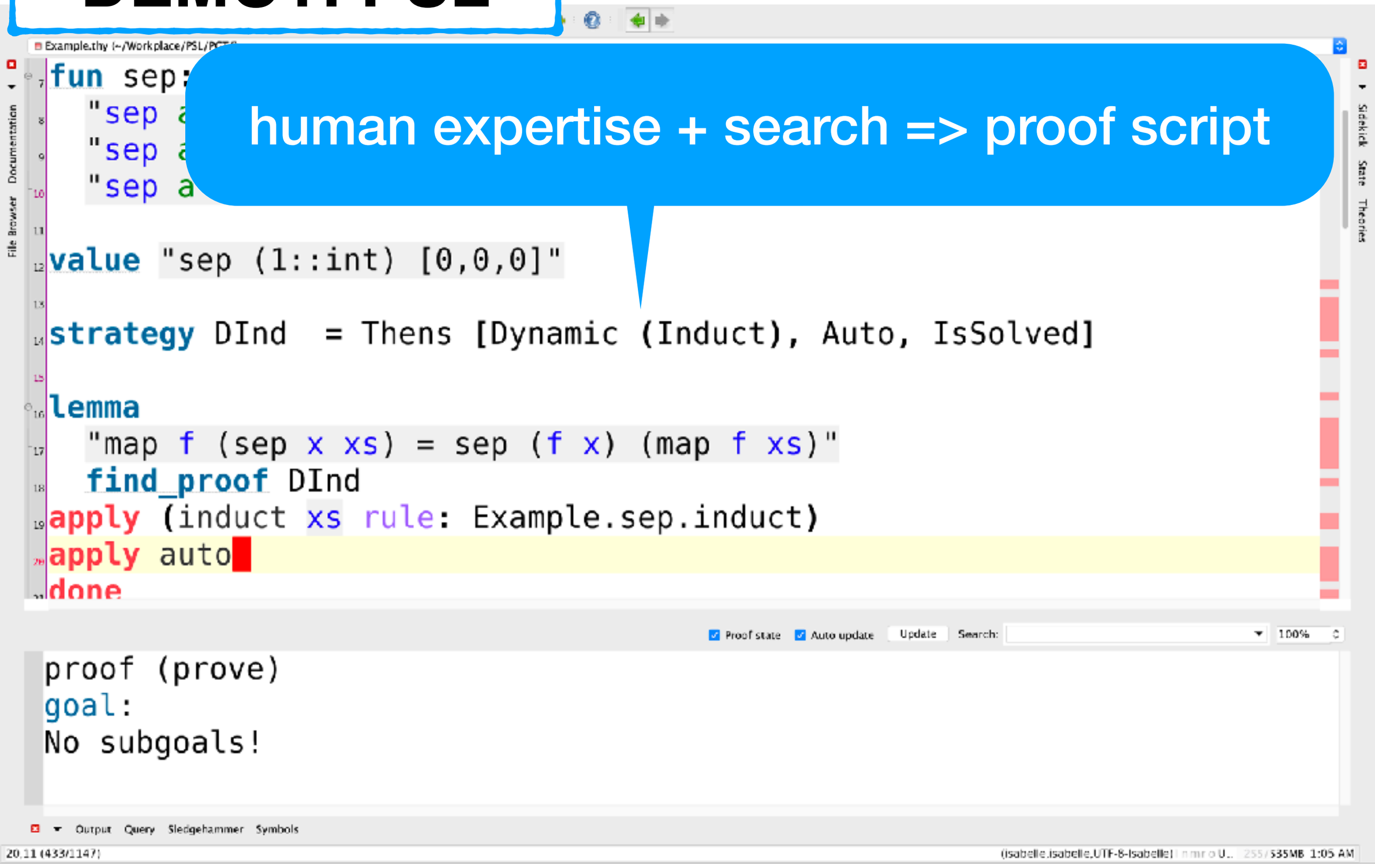
Proof state | Auto update | Update | Search: | 100%

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

Output | Query | Sledgehammer | Symbols

20,11 (433/1147) (isabelle,isabelle,UTF-8-Isabelle) 255/535MB 1:05 AM

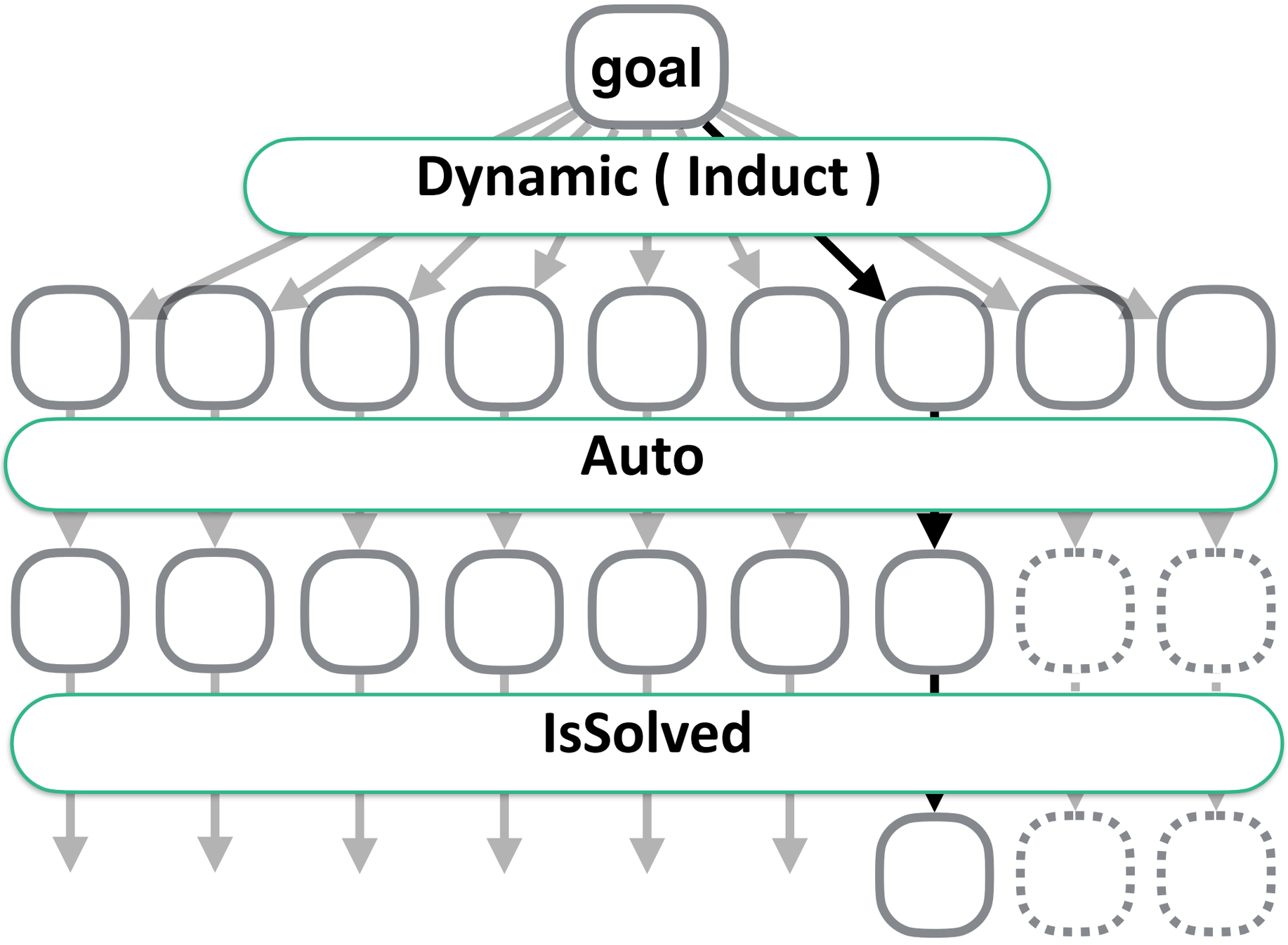
DEMO1: PSL


human expertise + search => proof script
Example.thy (~/Workplace/PSL/PCT
fun sep:
"sep a
"sep a
"sep a
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
apply (induct xs rule: Example.sep.induct)
apply auto
done
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal:
No subgoals!
Output
Query
Sledgehammer
Symbols
20,11 (433/1147)
(isabelle,isabelle,UTF-8-Isabelle)
535MB 1:05 AM

DEMO1

PSL: Proof Strategy Language

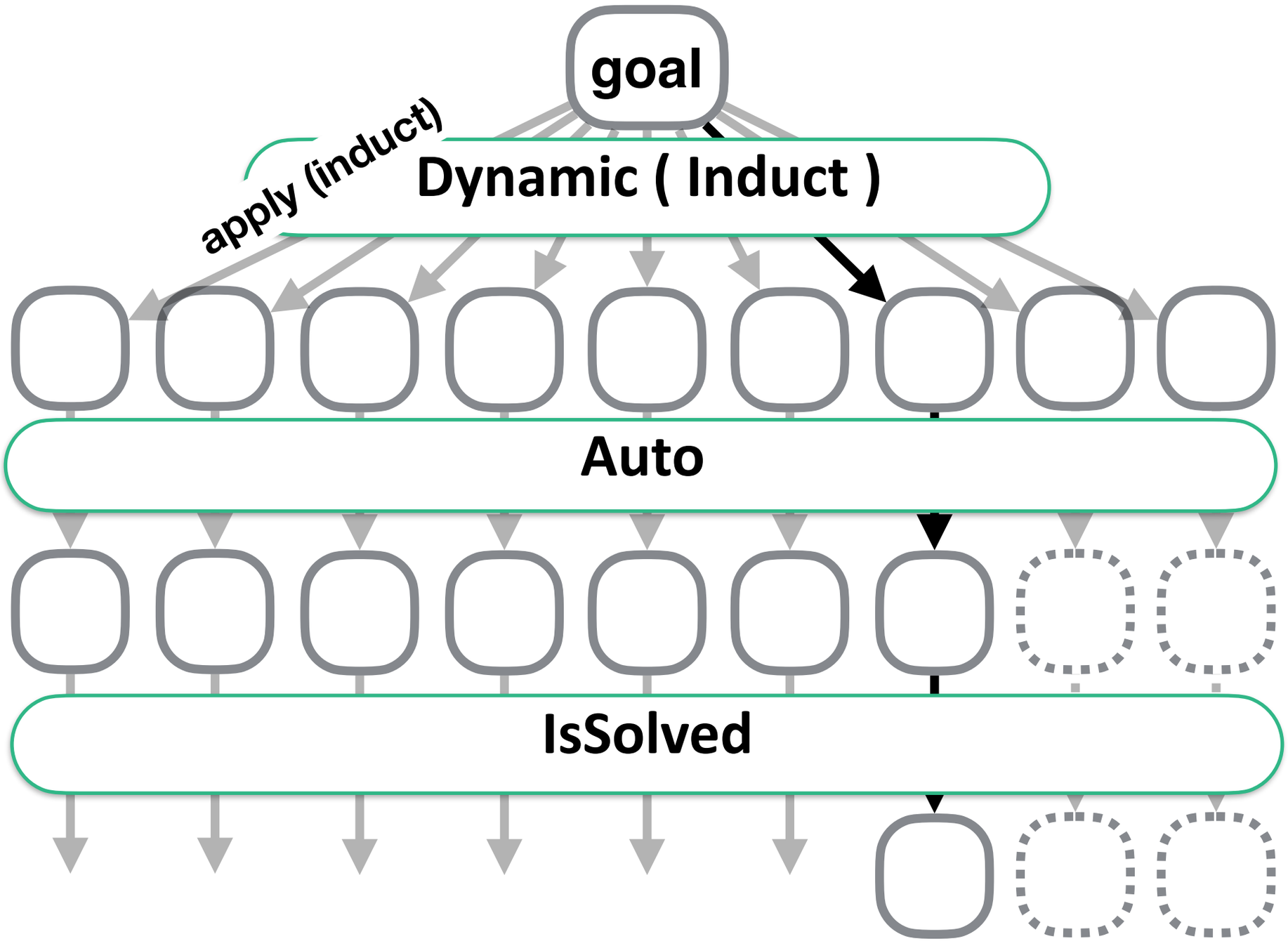
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

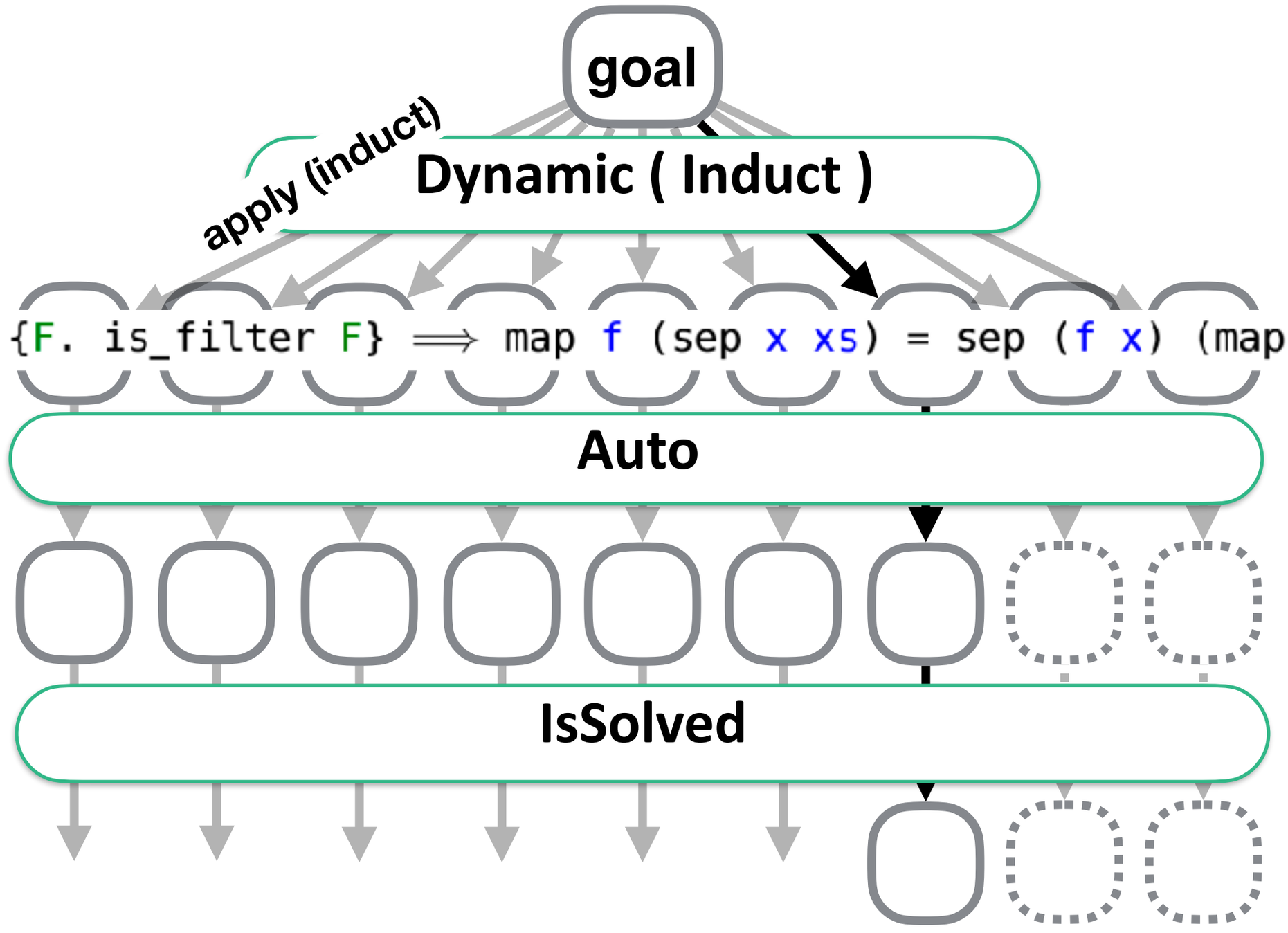
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



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]*)
```

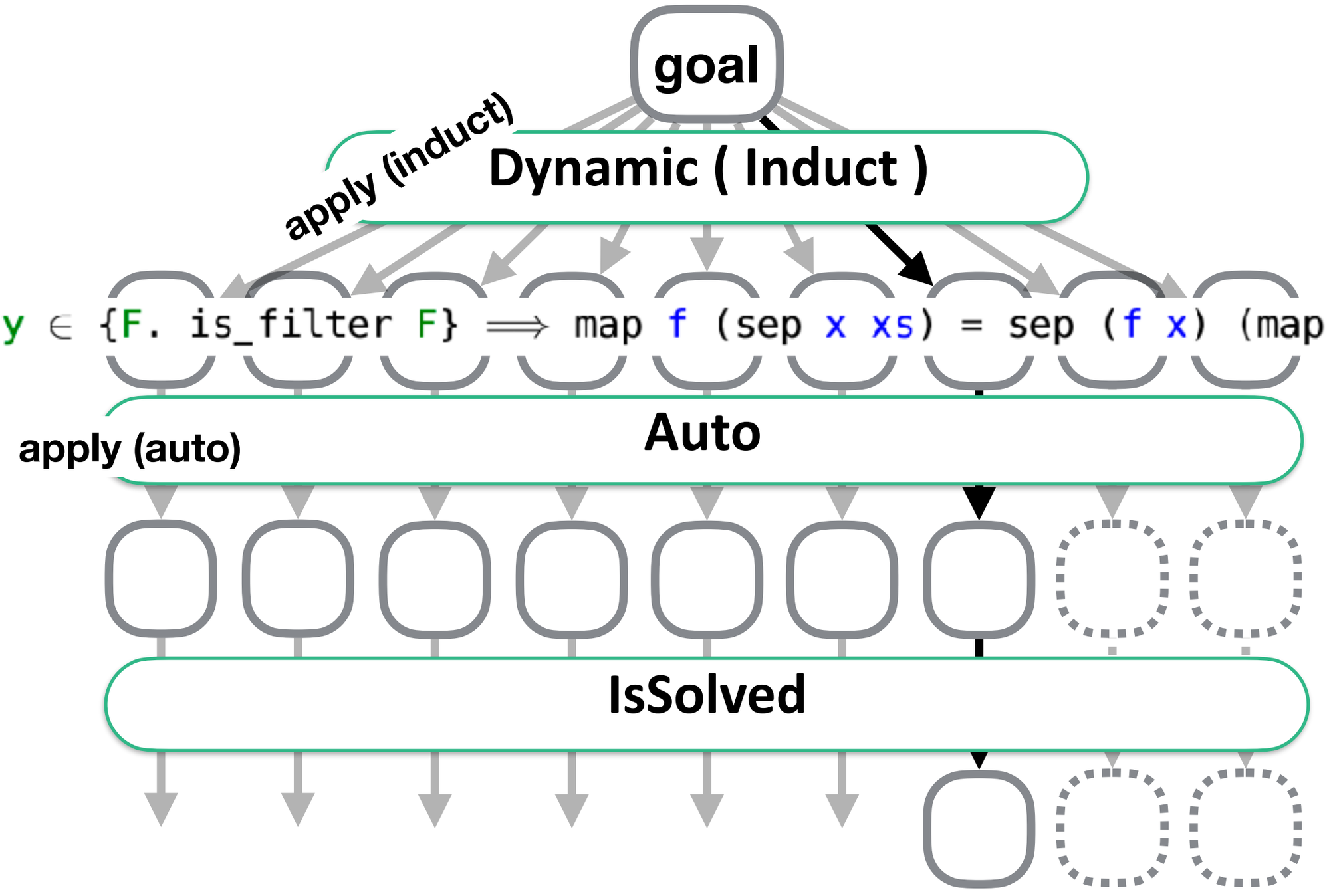



```
⋀y. y ∈ {F. is_filter F} ⟹ map f (sep x xs) = sep (f x) (map f xs)
```

DEMO1

PSL: Proof Strategy Language

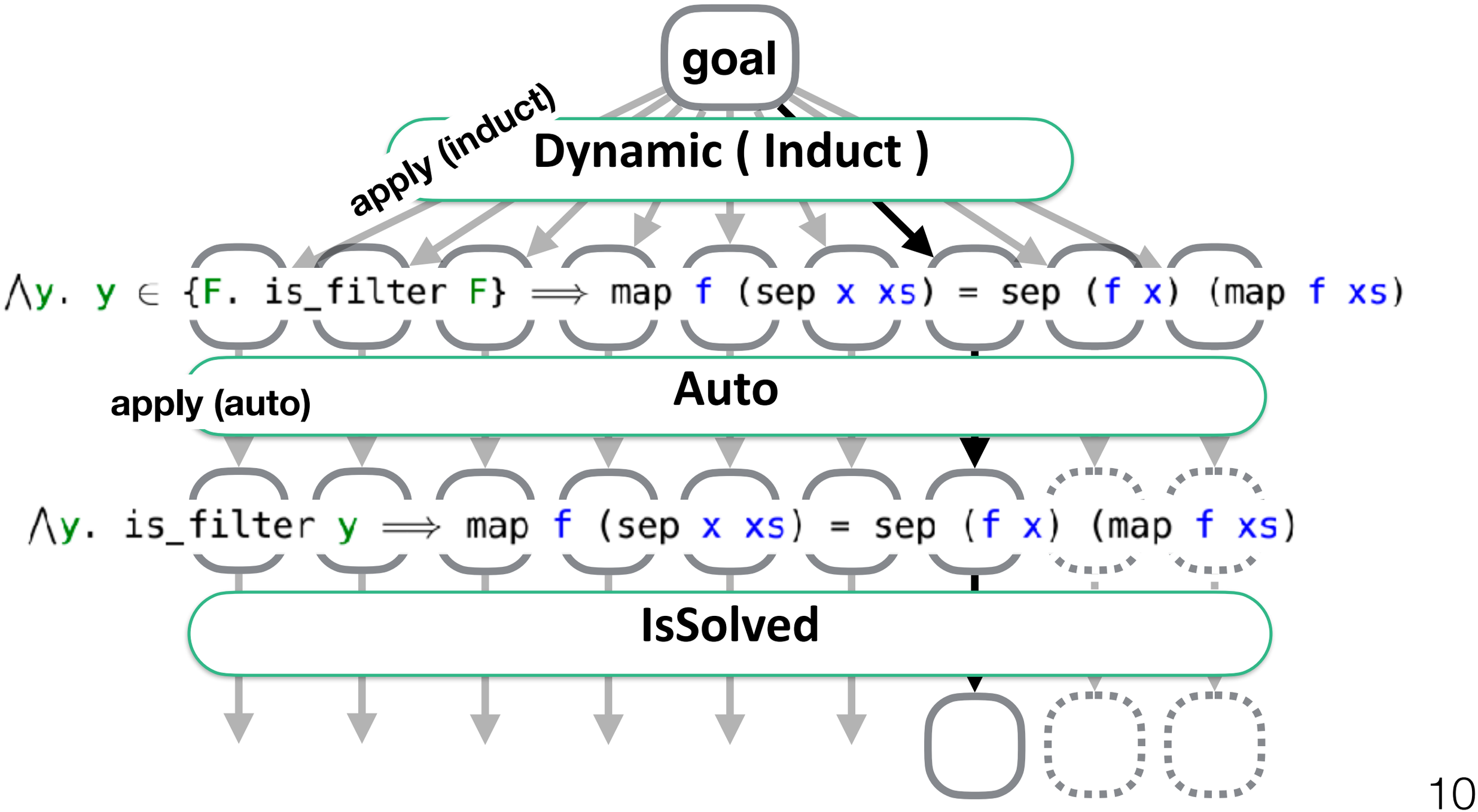
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

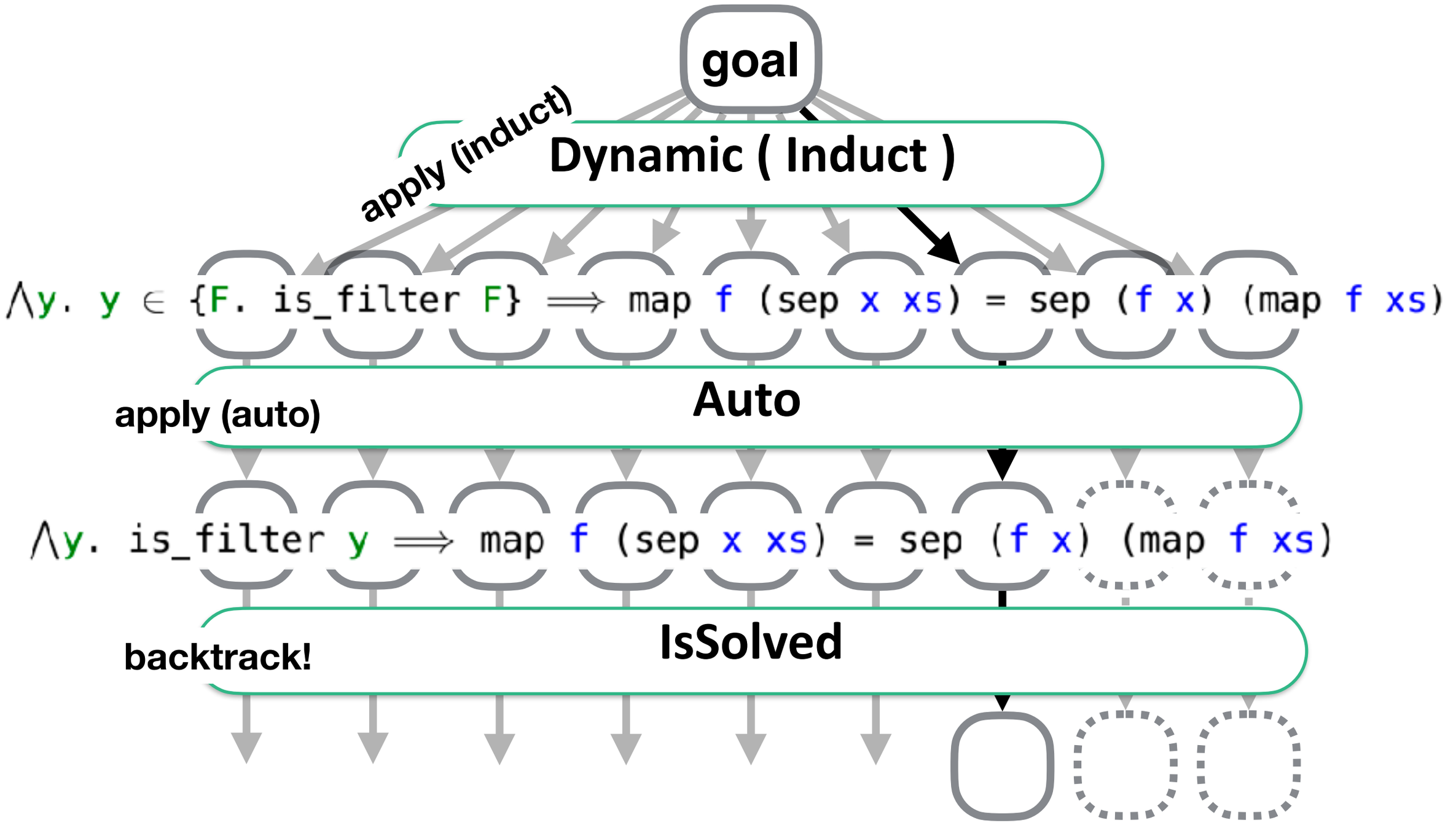
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

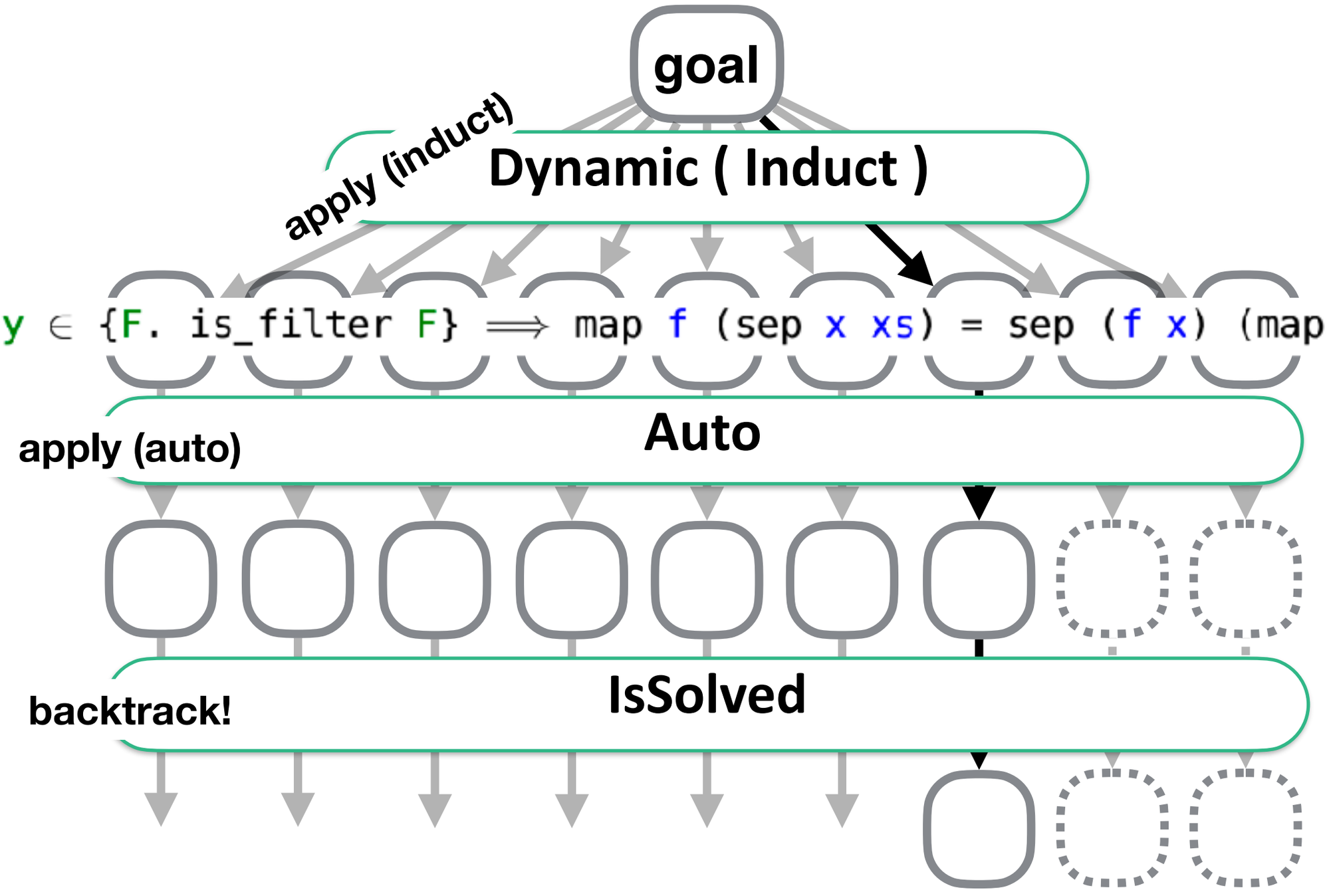
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

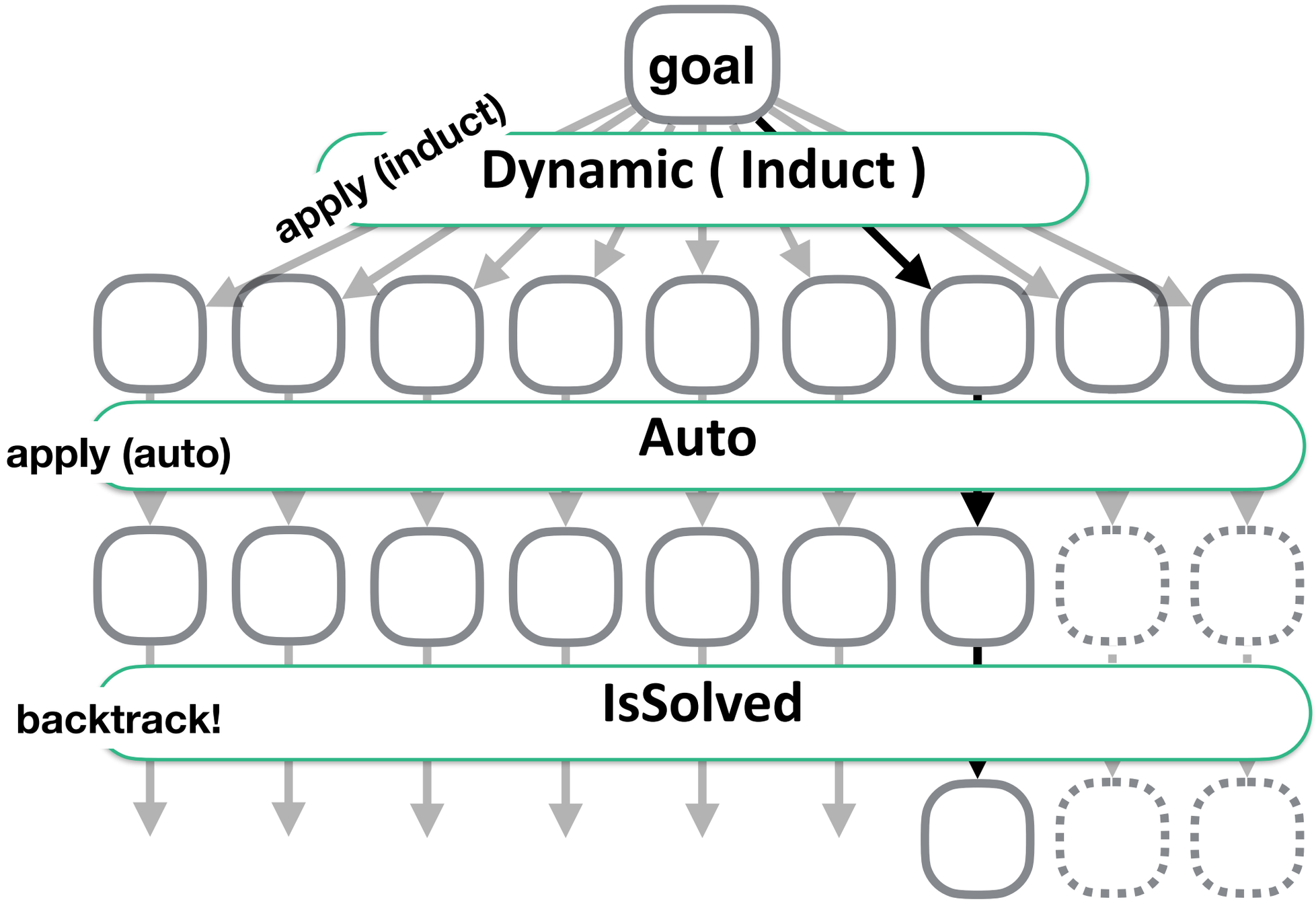
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

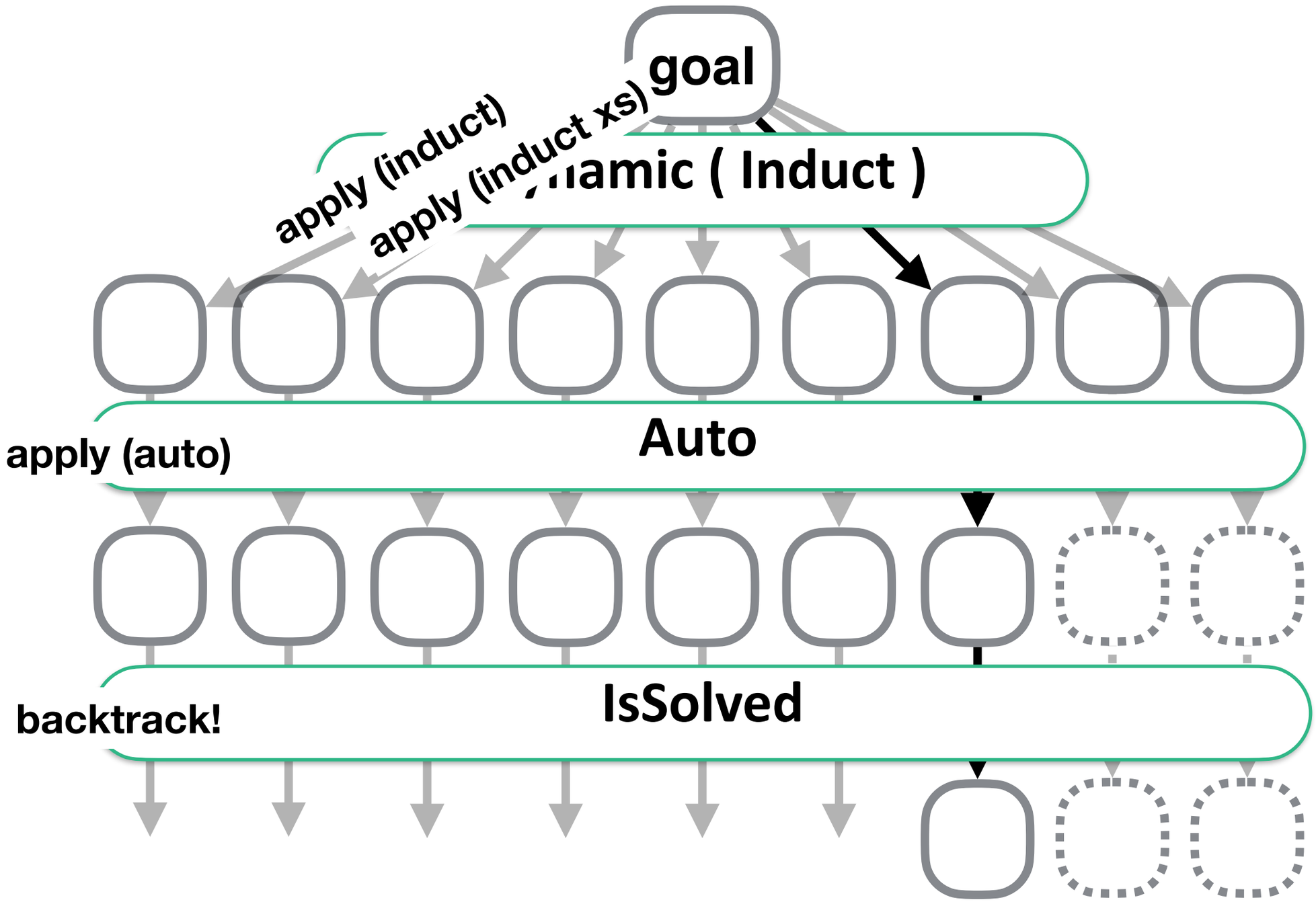
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

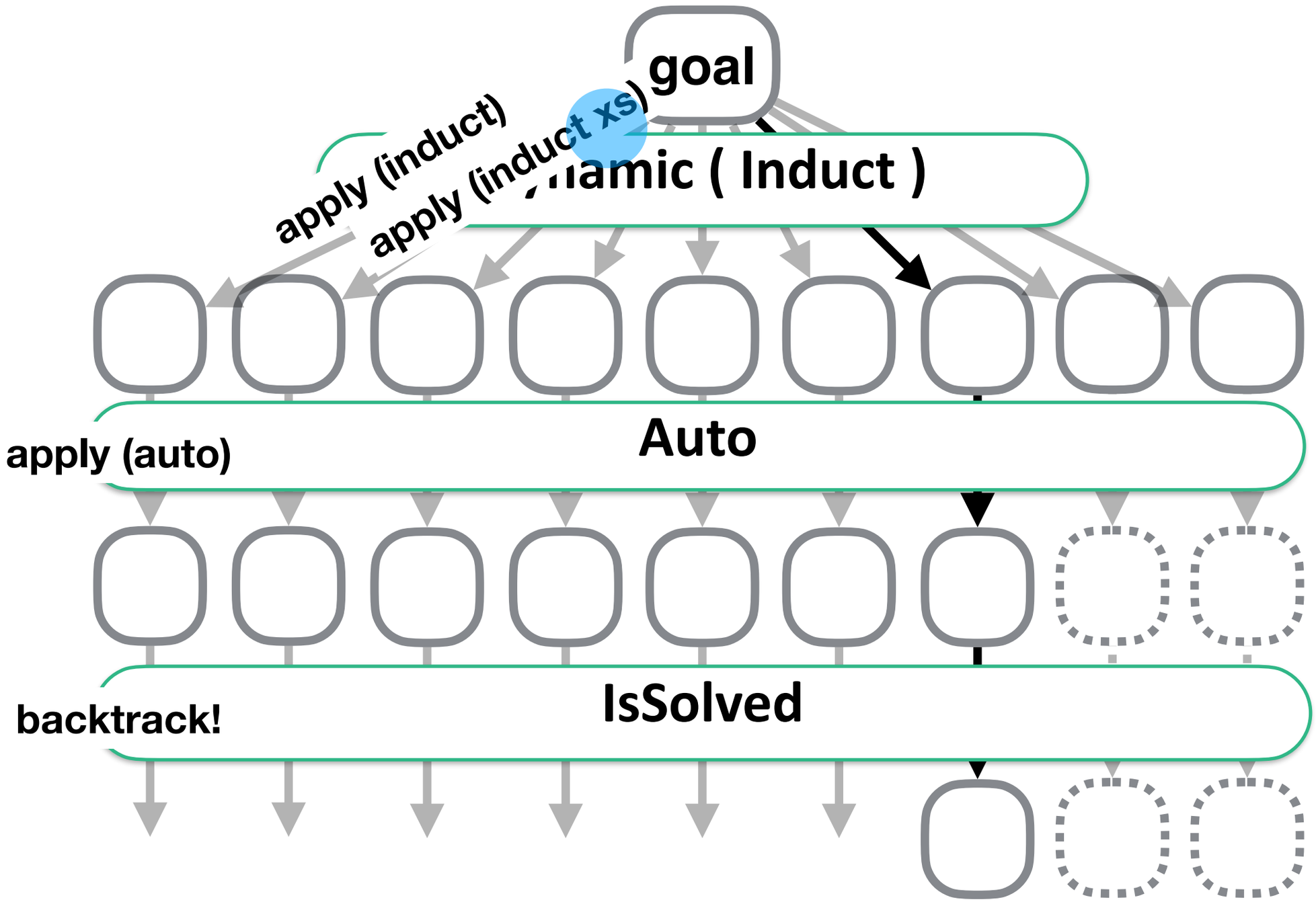
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

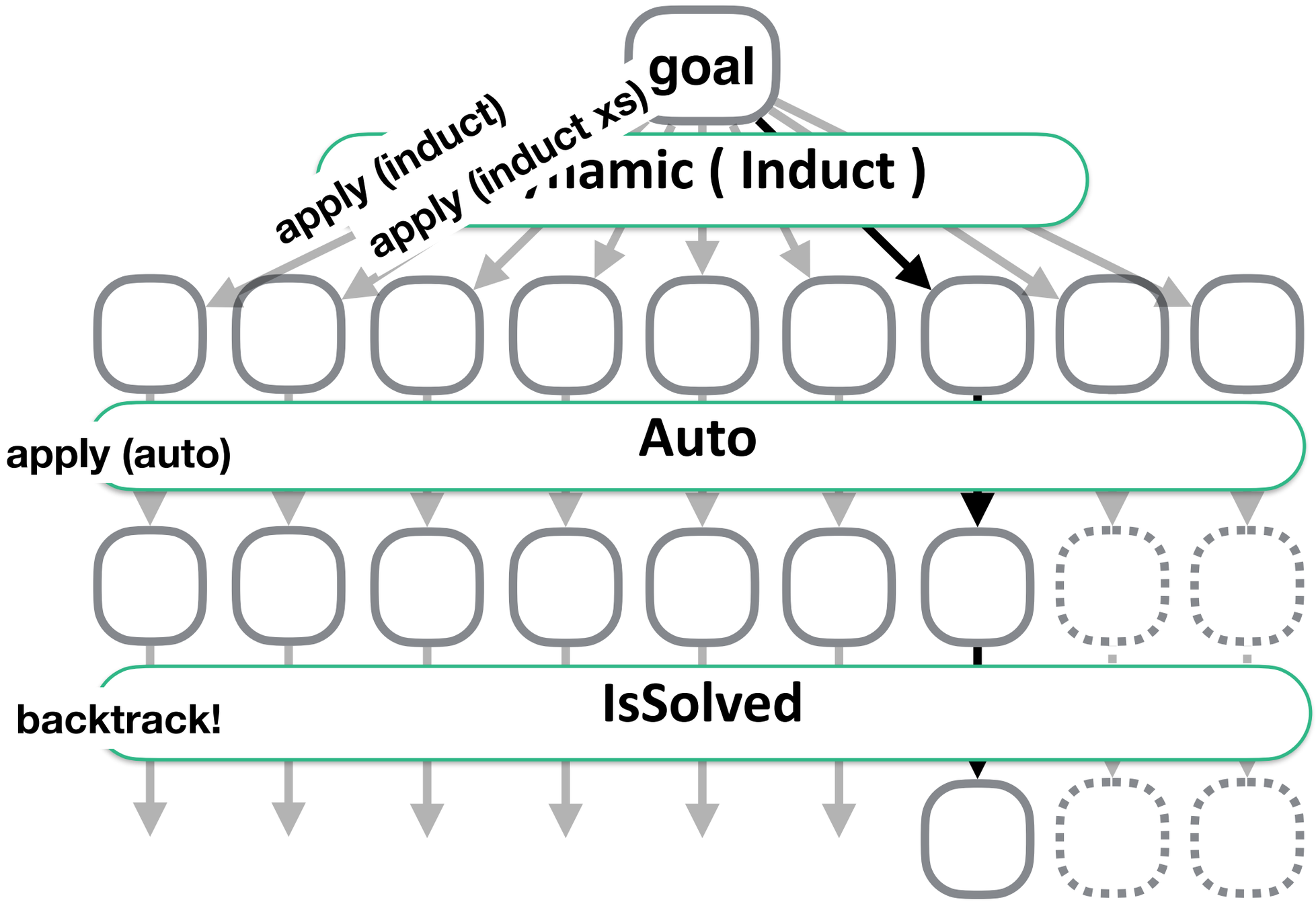
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

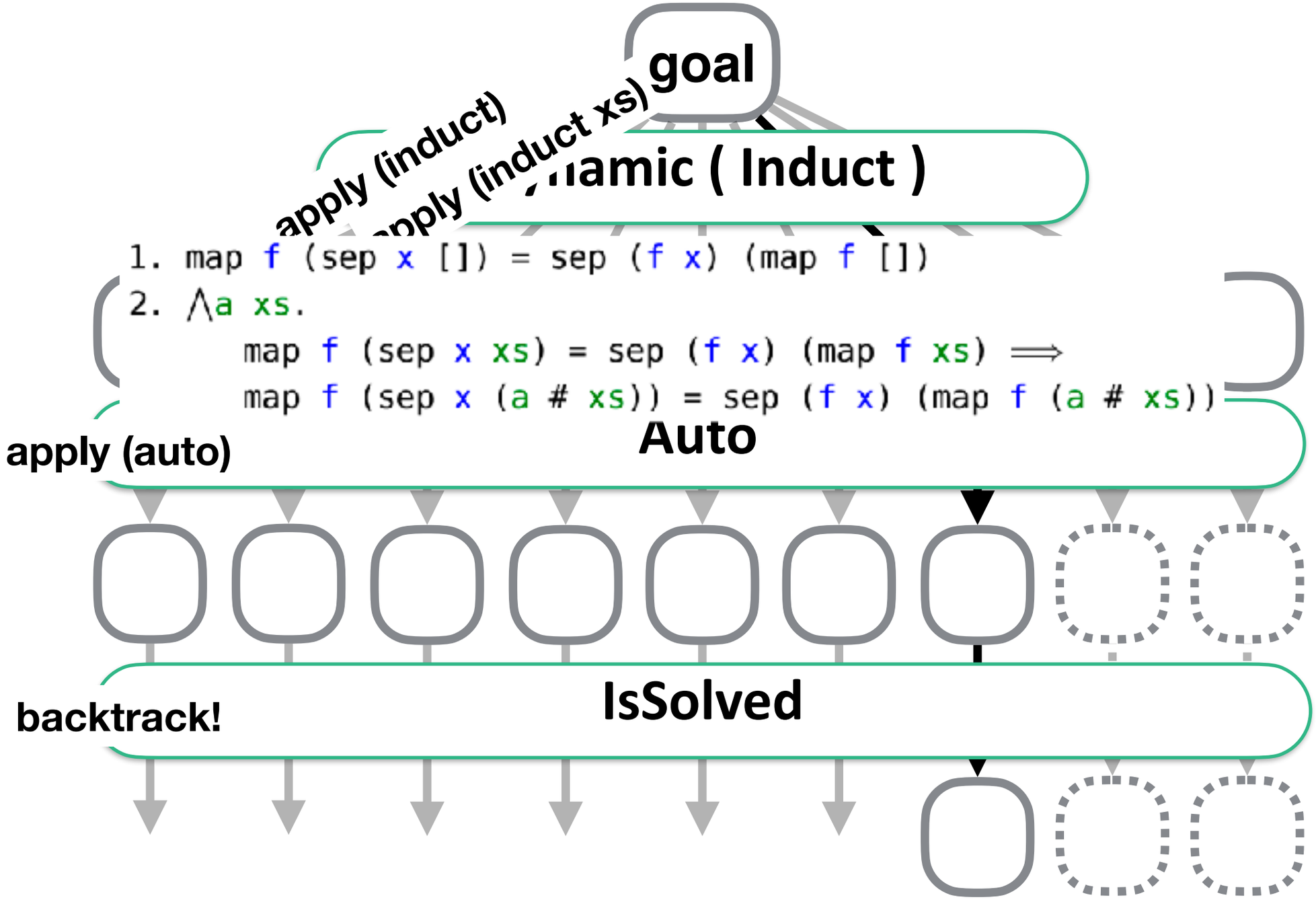
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

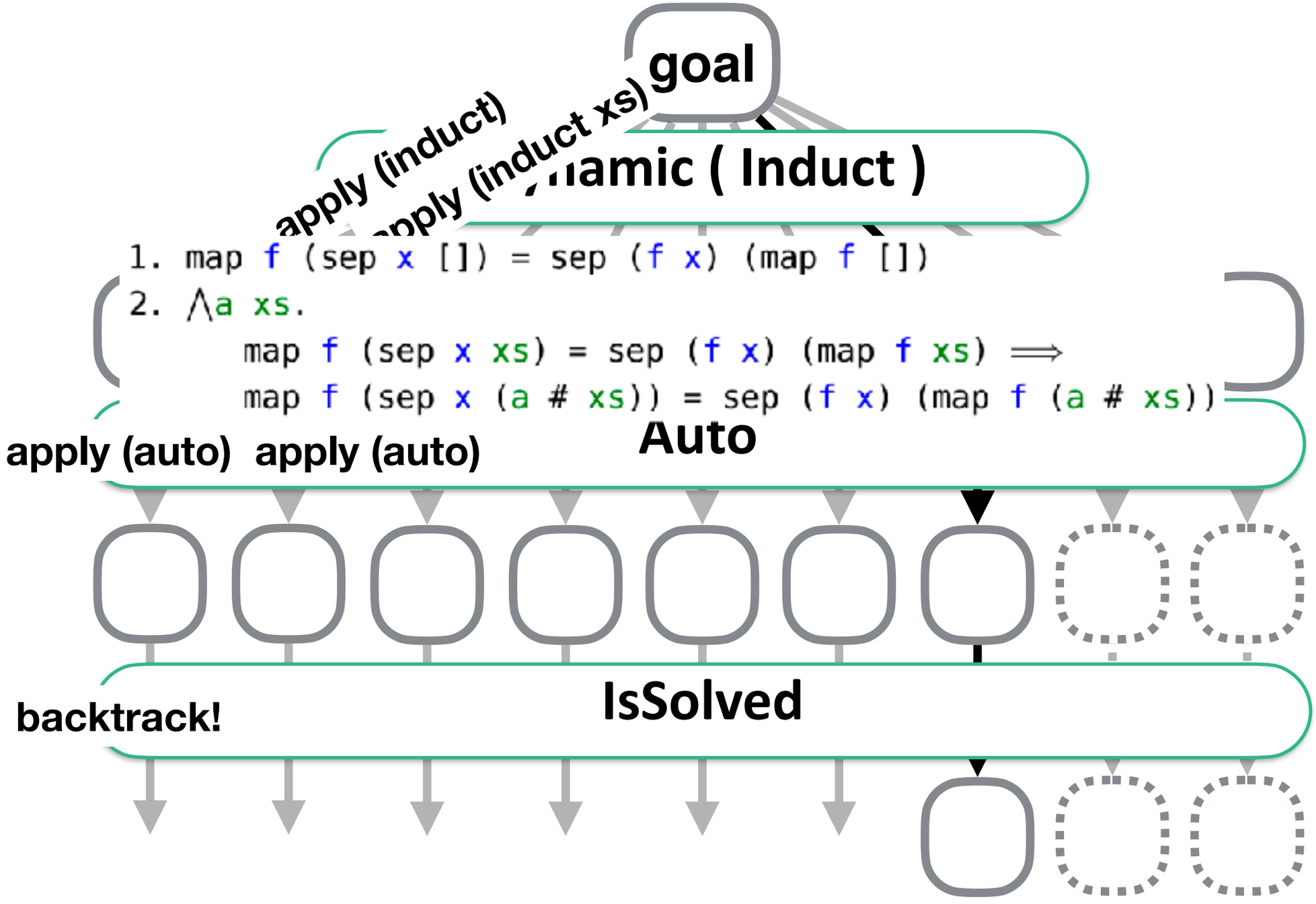
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

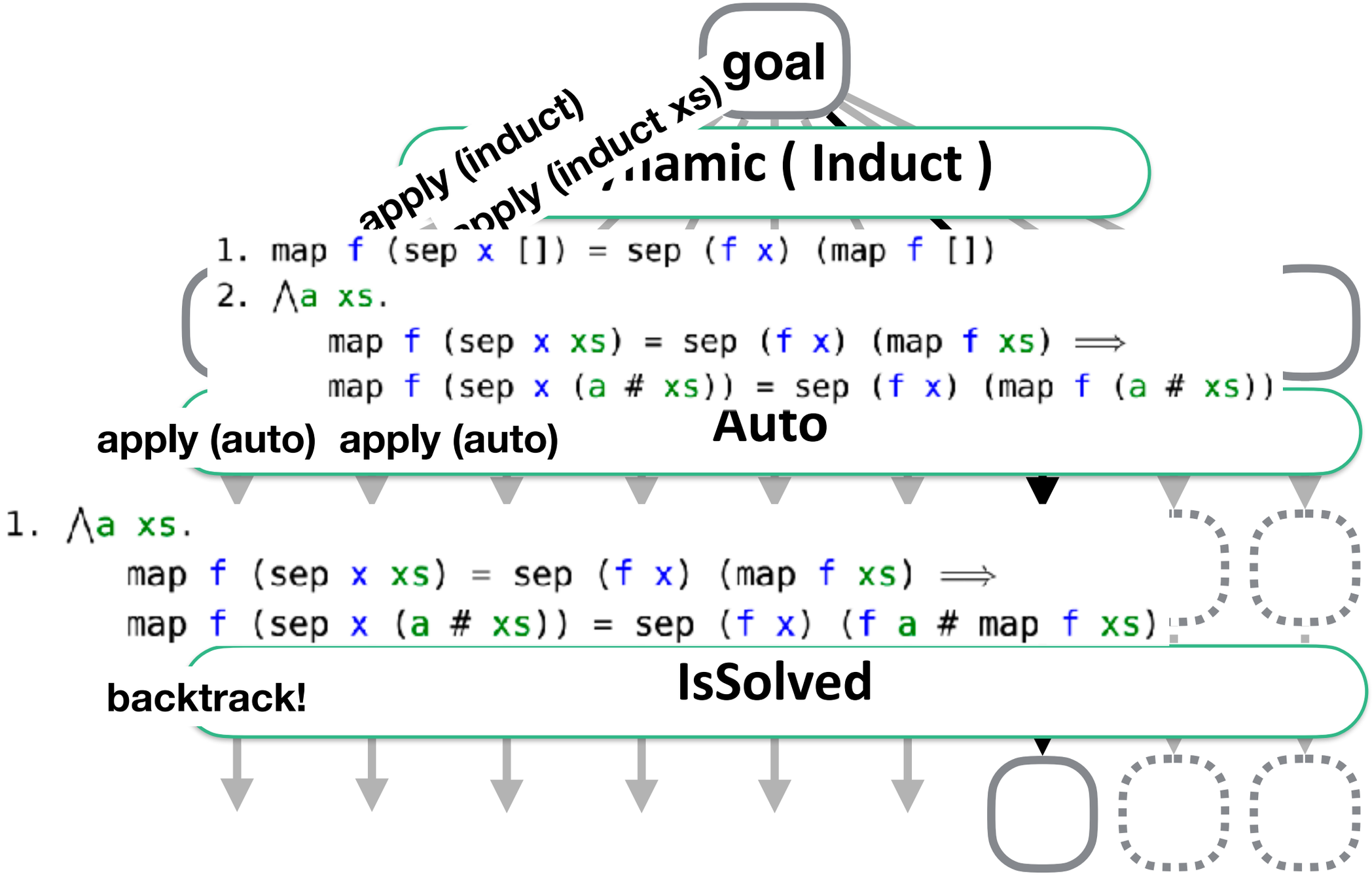
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

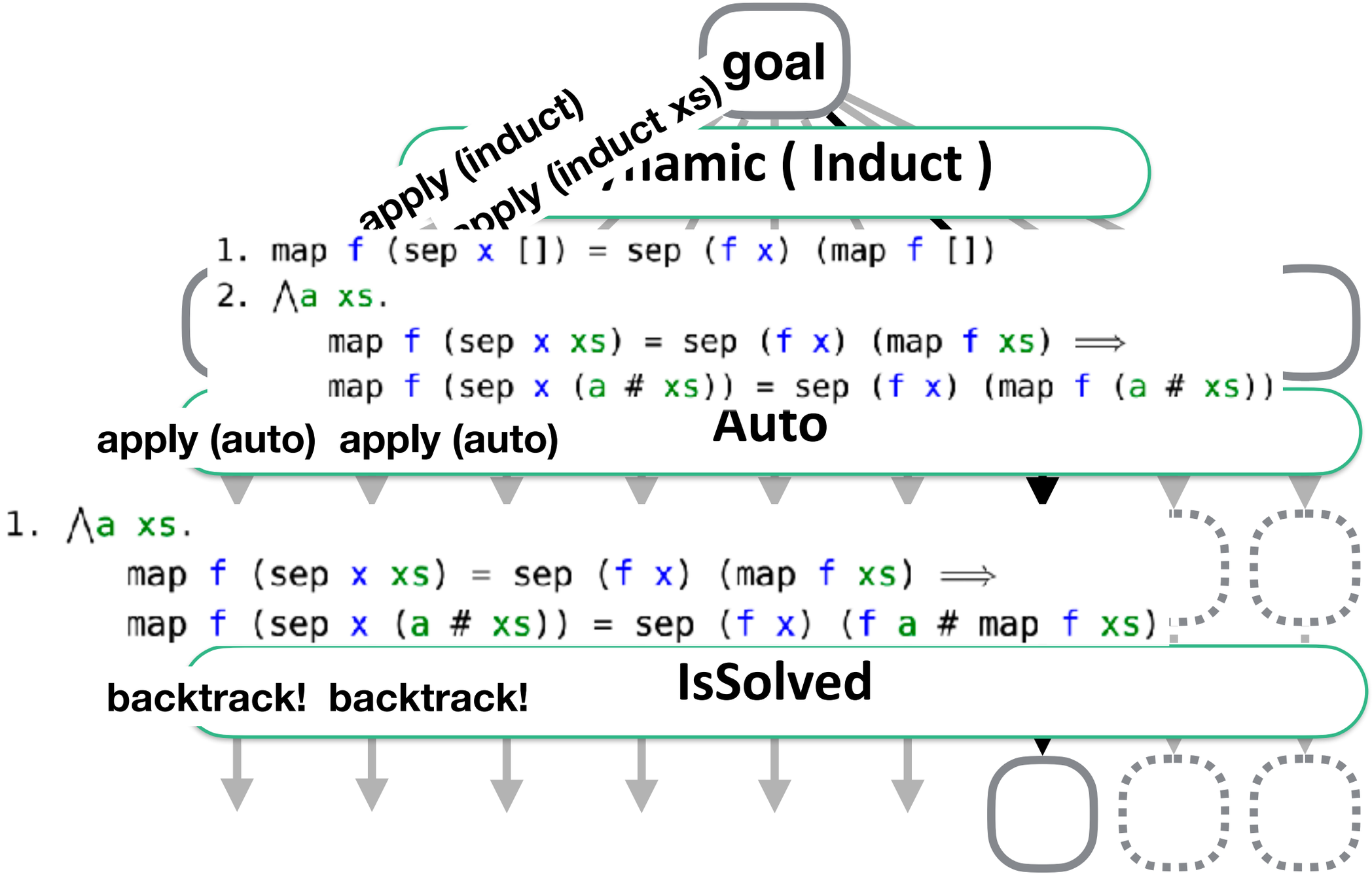
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

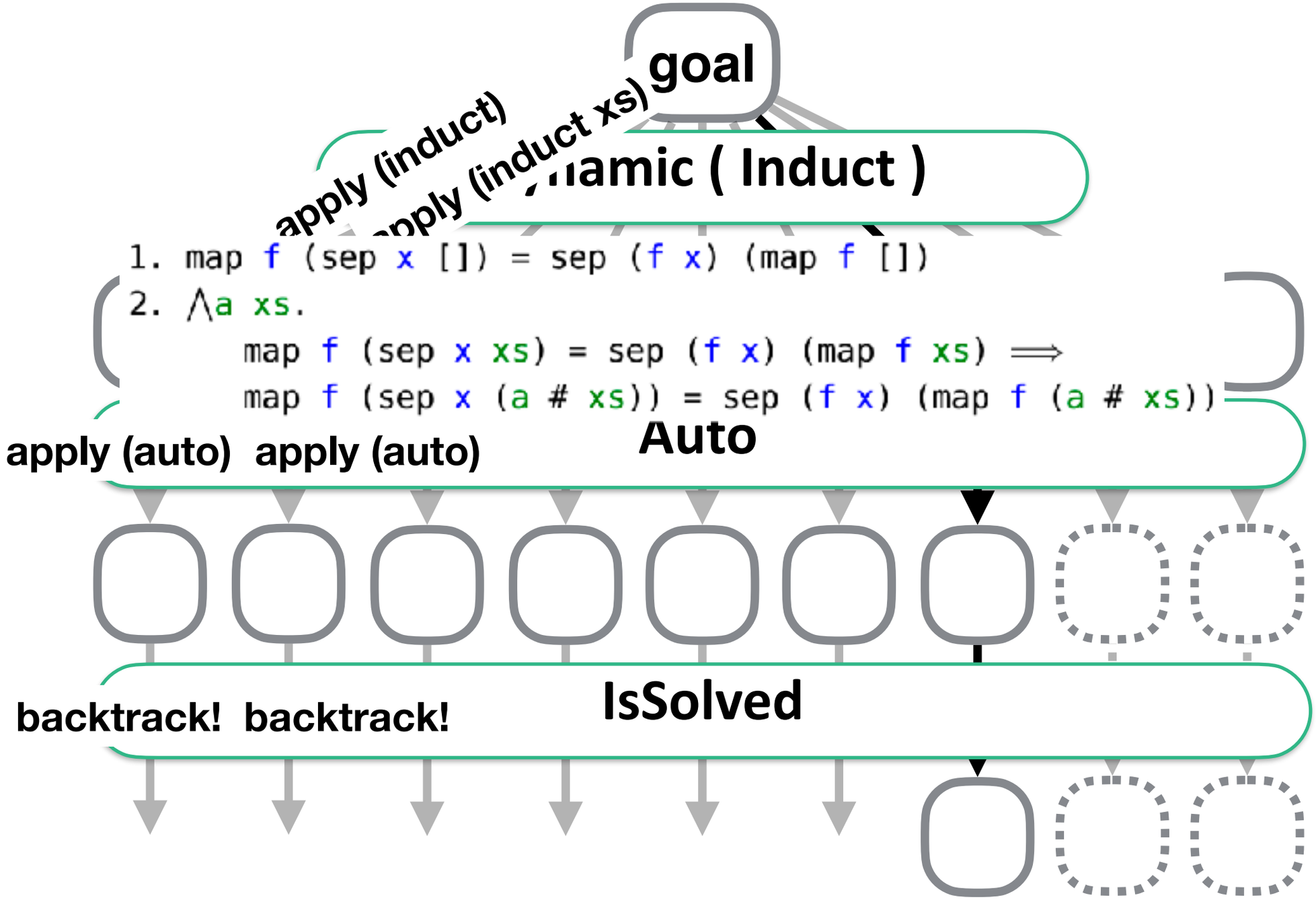
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

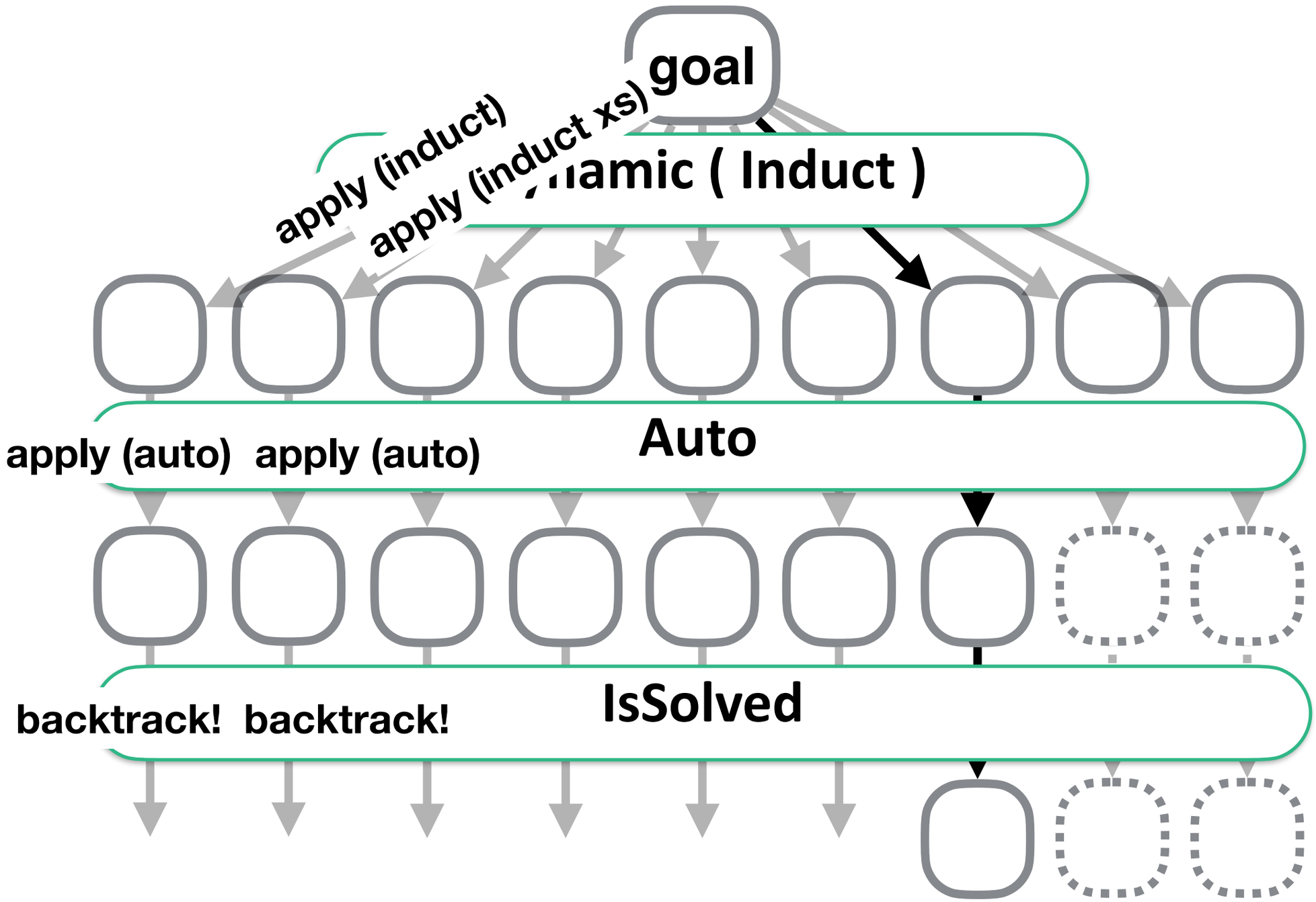
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

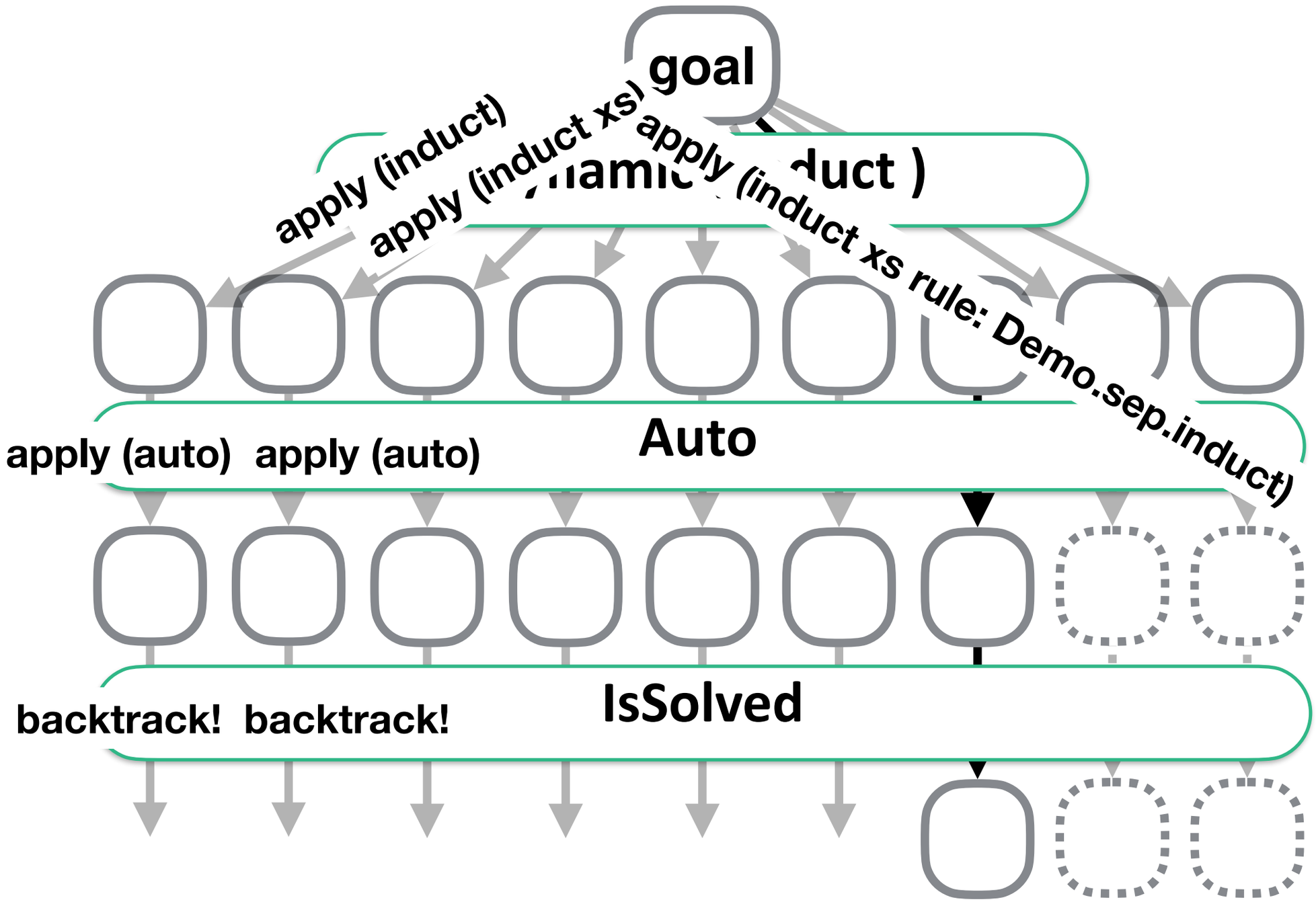
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

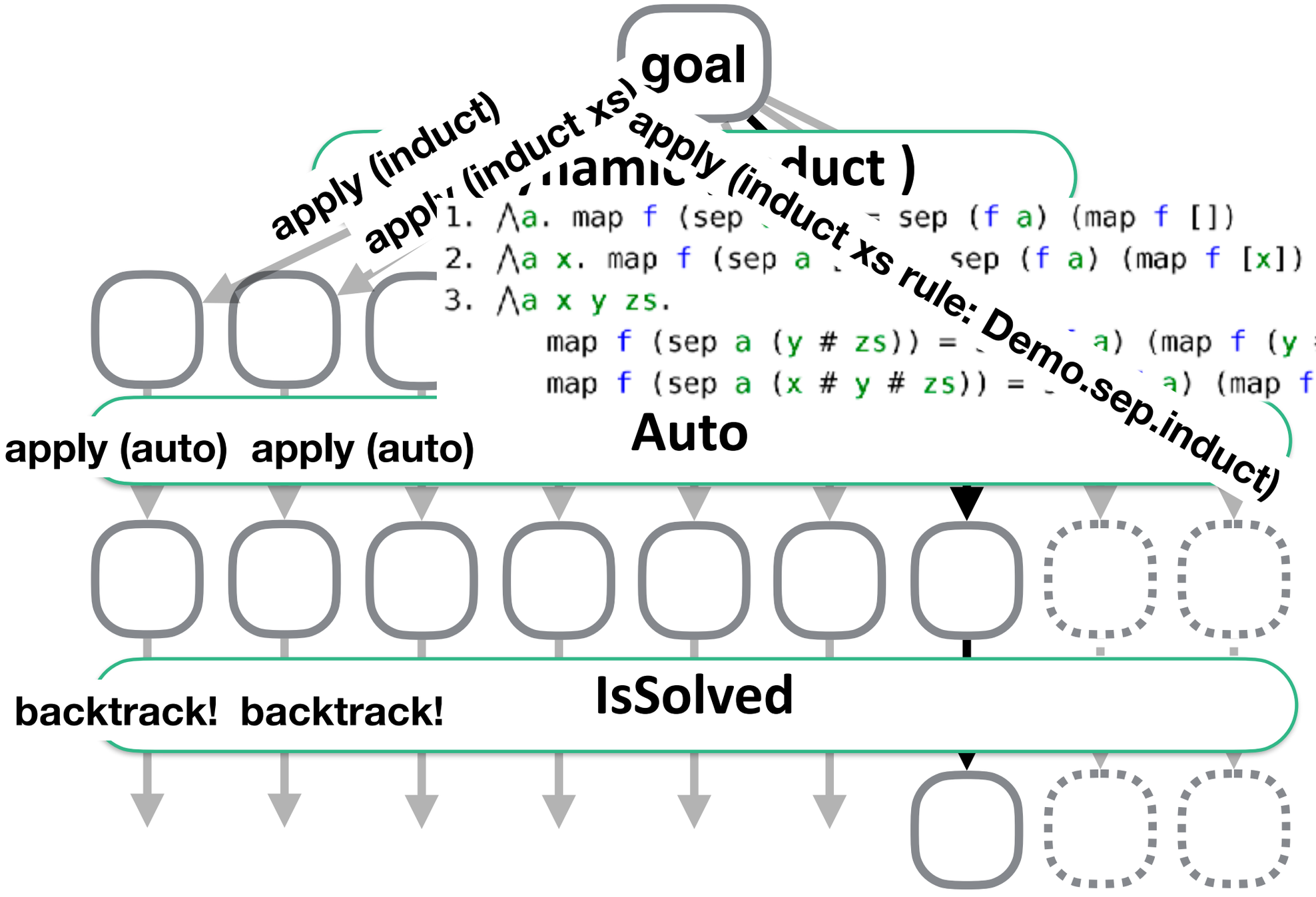
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

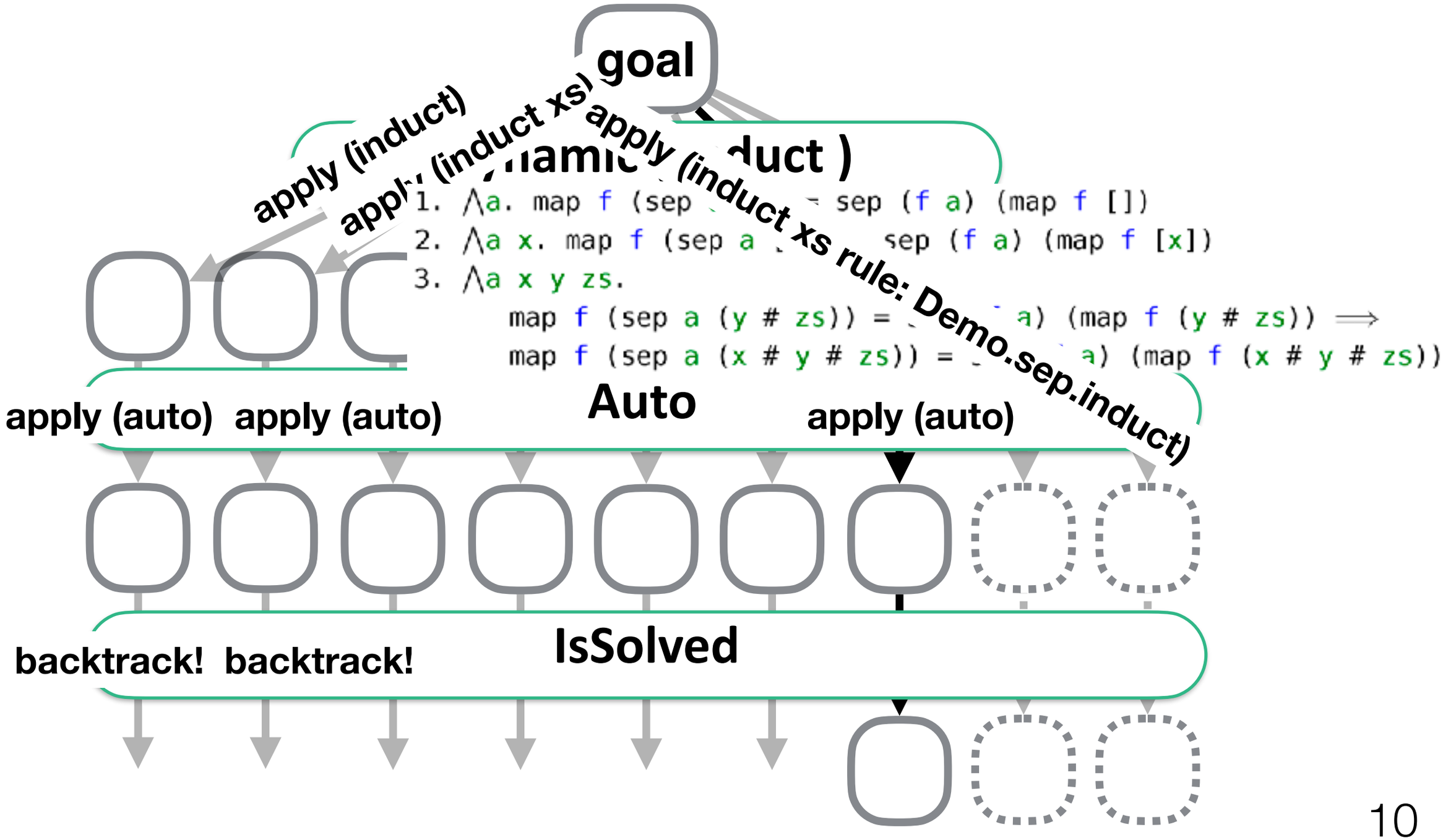
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

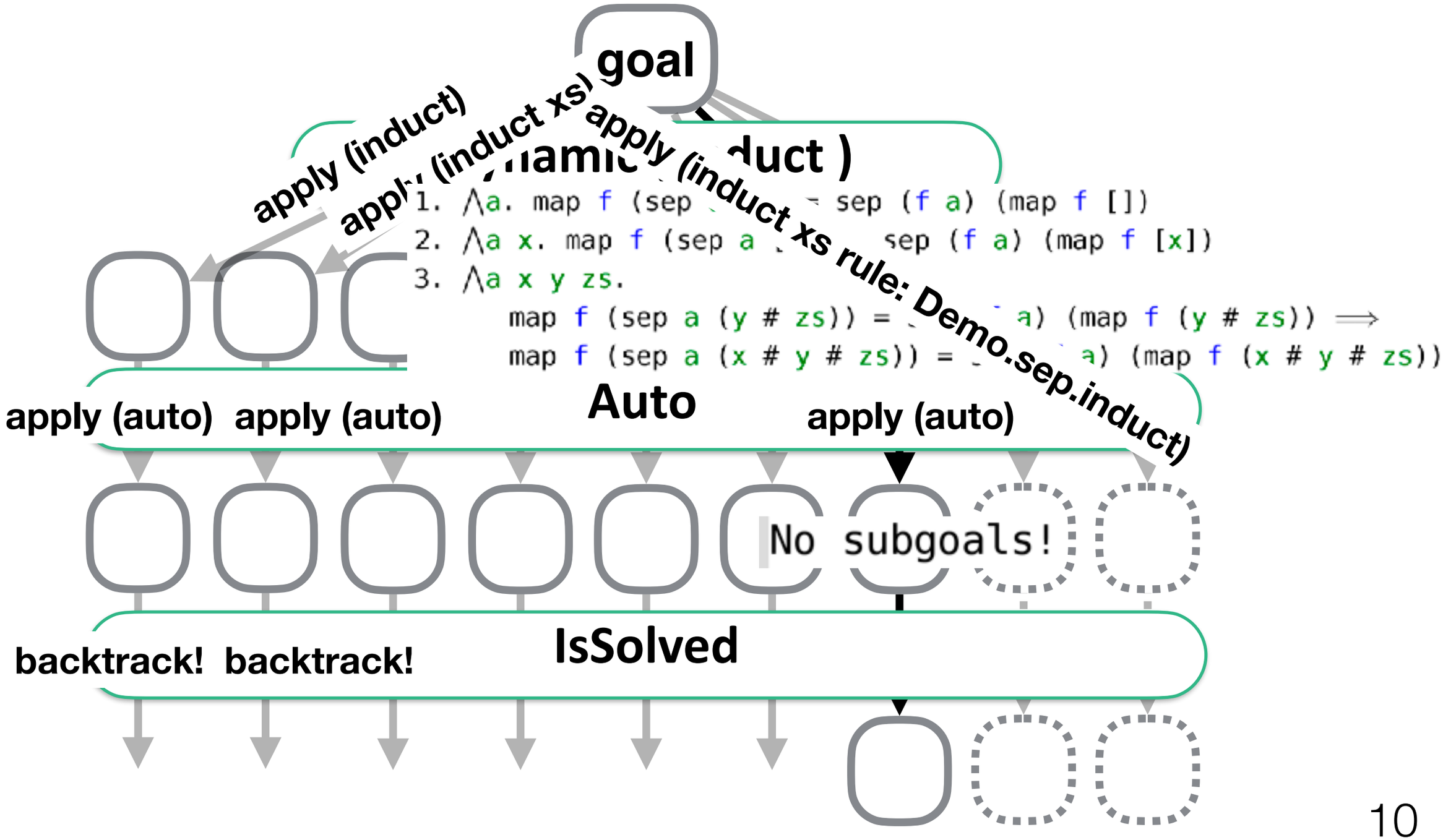
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

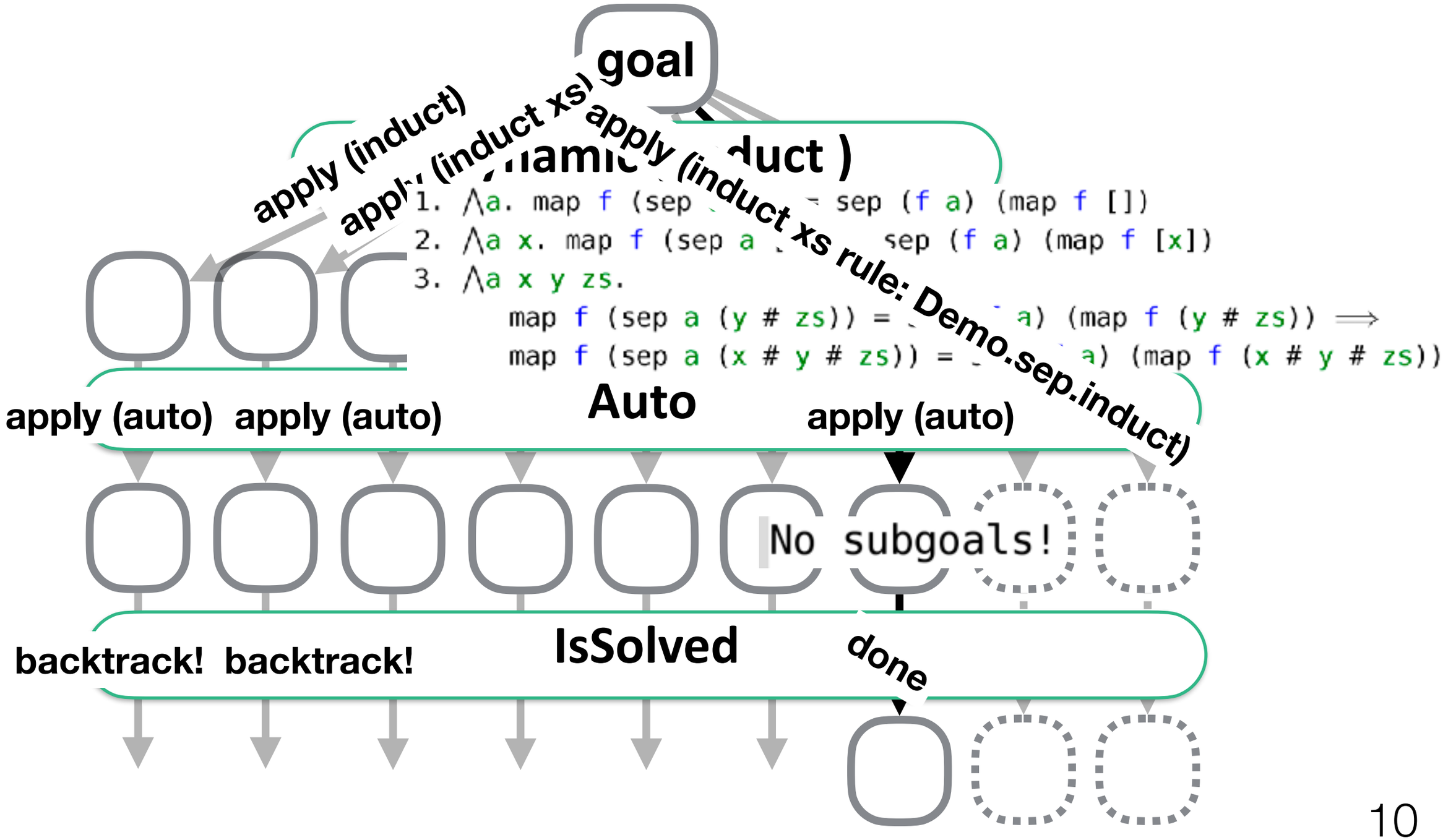
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

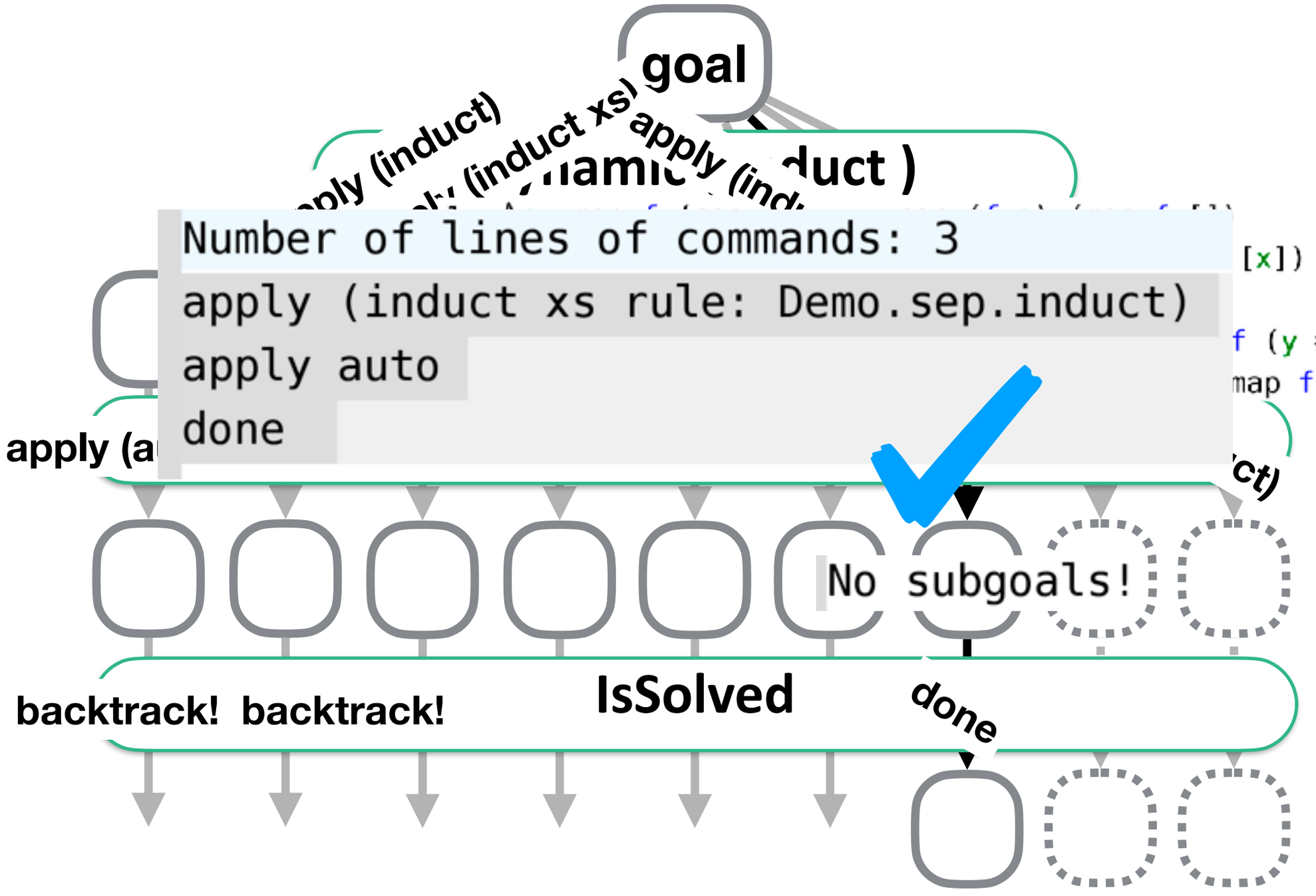
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



DEMO1

PSL: Proof Strategy Language

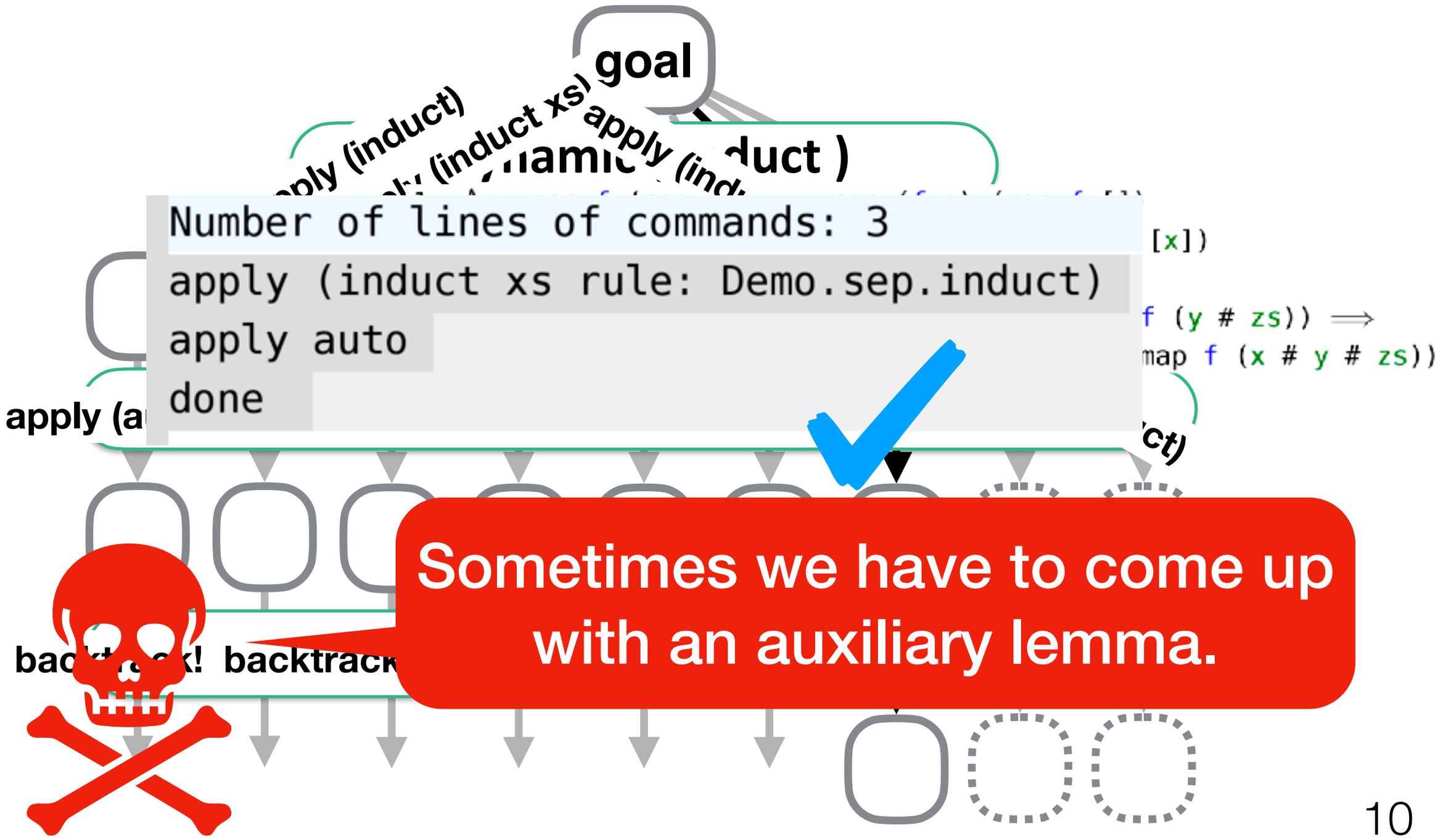
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



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

```
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)"
```

Proof state Auto update Update Search: 100%

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

Output Query Sledgehammer Symbols

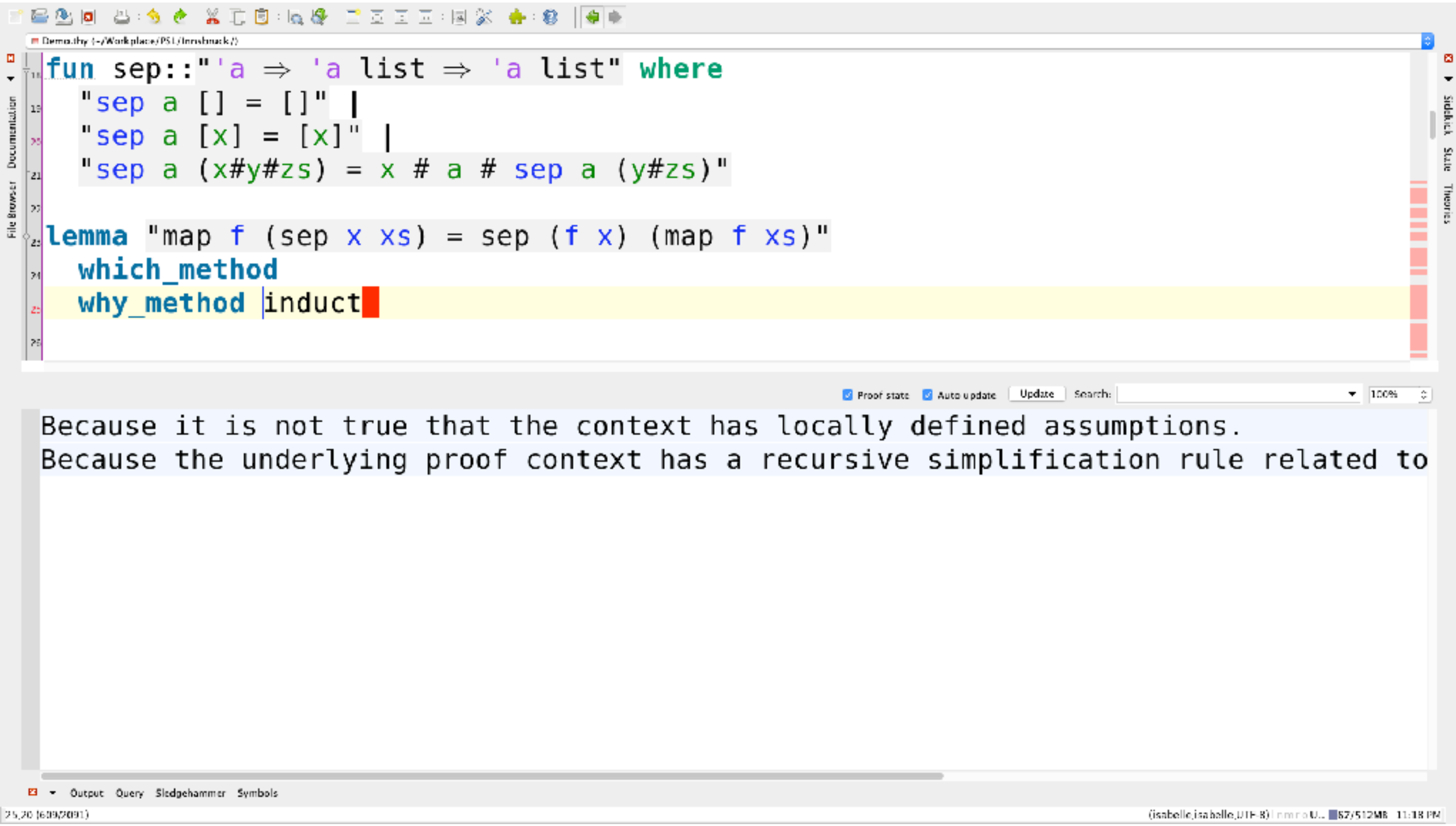
DEMO2: PaMpeR

```
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
```

```
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
```

DEMO2: PaMpeR



DEMO2: PaMpeR

```
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
  rank_method coinduction
```

```
coinduction 123 out of 239
```

preparation phase

How does PaMpeR work?

recommendation phase

preparation phase

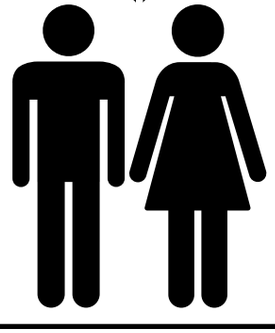
large proof corpora

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

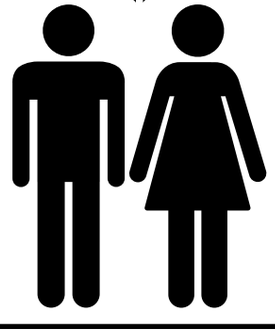
large proof corpora

AFP and standard library

How does PaMpeR work?

recommendation phase

proof state

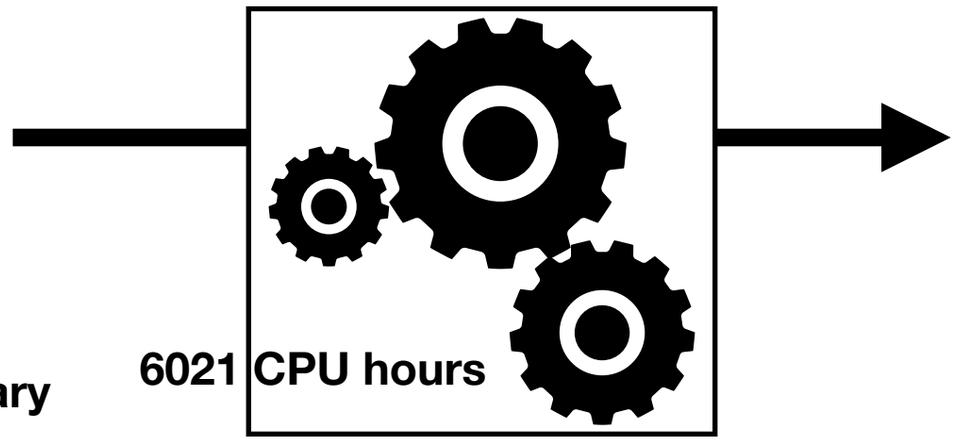


proof engineer

preparation phase

large proof corpora

AFP and standard library

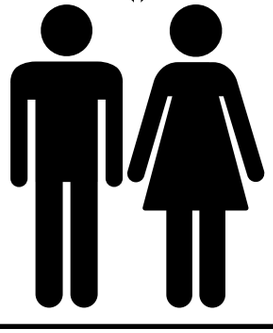


108 assertions

How does PaMpeR work?

recommendation phase

proof state



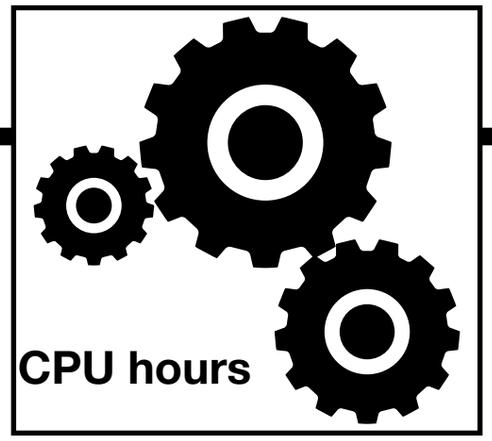
proof engineer

preparation phase

large proof corpora

AFP and standard library

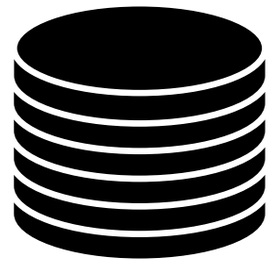
full feature extractor



6021 CPU hours

108 assertions

database (425334 data points)

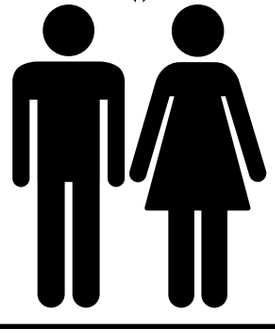


:: (tactic_name, [bool])

How does PaMpeR work?

recommendation phase

proof state



proof engineer

preparation phase

large proof corpora

AFP and standard library

full feature extractor

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

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

preparation phase

large proof corpora

AFP and standard library

full feature extractor

6021 CPU hours

database (425334 data points)

:: (tactic_name, [bool])

preprocess

decision tree construction

108 assertions

How does PaMpeR work?

recommendation phase

proof state

proof engineer

fast feature extractor

feature vector

How does PaMpeR work?

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

recommendation phase

proof state

proof engineer

fast feature extractor

feature vector

lookup

proof method recommendation

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

preparation phase

large proof corpora

AFP and standard library

full feature extractor

6021 CPU hours

108 assertions

database (425334 data points)

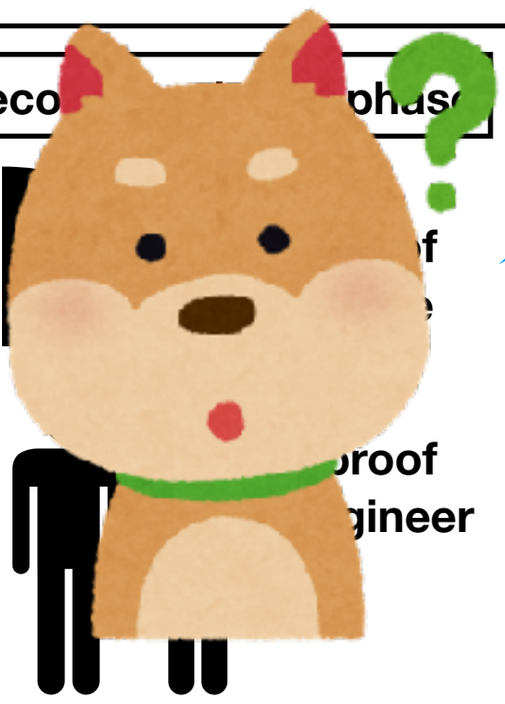
:: (tactic_name, [bool])

preprocess

decision tree construction

How does PaMpeR work?

Can I encode heuristics to describe how to apply induction?

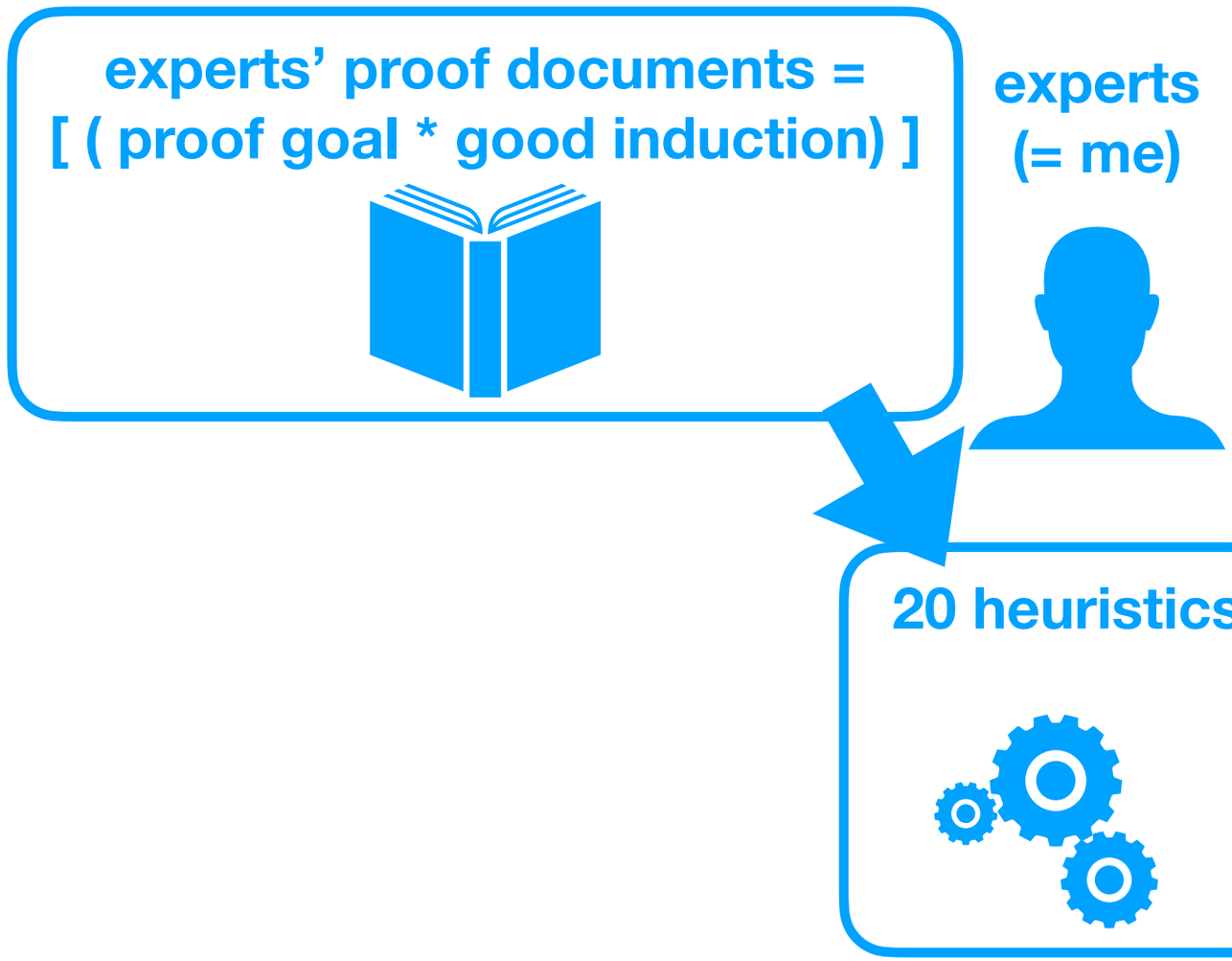


lookup

proof method recommendation

It is difficult to write induction heuristics.

It is difficult to write induction heuristics.



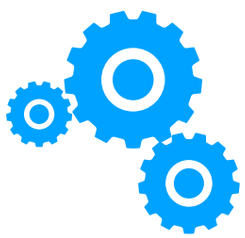
It is difficult to write induction heuristics.

```
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



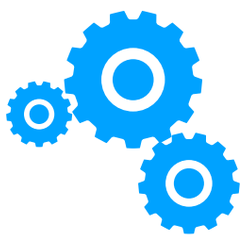
It is difficult to write induction heuristics.

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

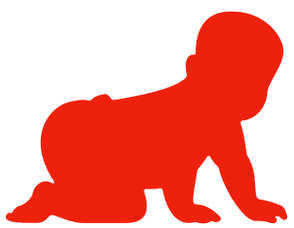
experts' proof documents =
[(proof goal * good induction)]

experts
(= me)

20 heuristics



new users



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

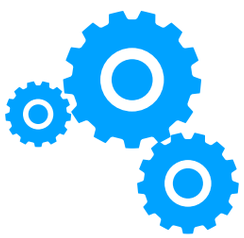
It is difficult to write induction heuristics.

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

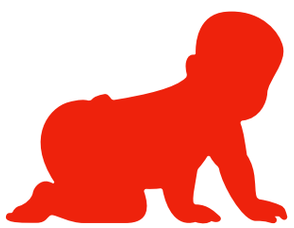
experts' proof documents =
[(proof goal * good induction)]

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!

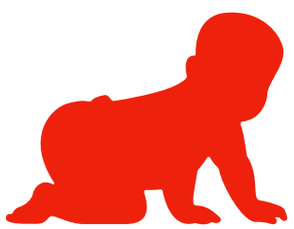
It is difficult to write induction heuristics.

```
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

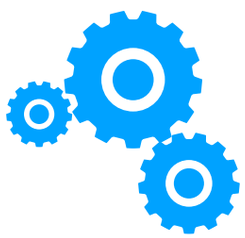


proof goal

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.

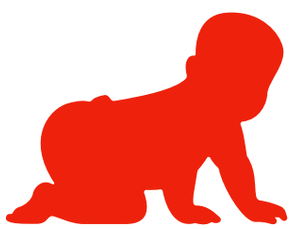
① blue = what I can see before releasing smart_induct

```
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

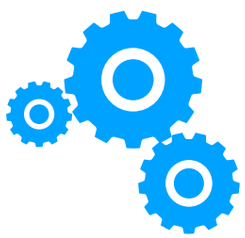


proof goal

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.

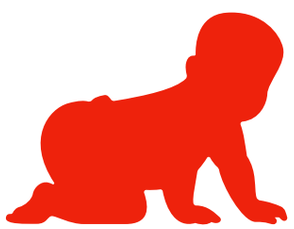
① blue = what I can see before releasing smart_induct

```
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

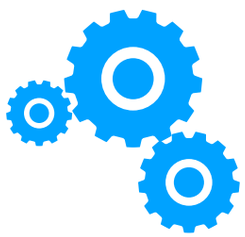


proof goal

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

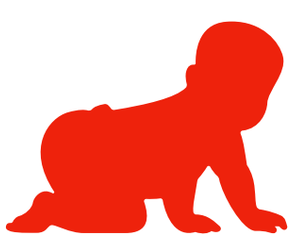
It is difficult to write induction heuristics.

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)



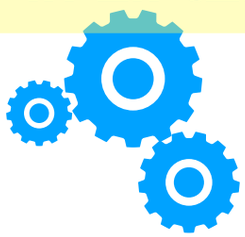
new users

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

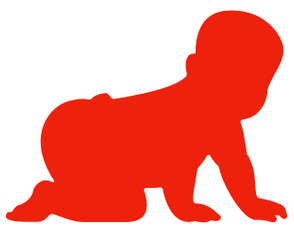
```
lemma "itrev xs ys = rev xs @ ys"
by(induct xs ys rule:"itrev.induct") auto
```

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

experts
(= me)

LiFtEr: Logical
Feature Extractor

new users

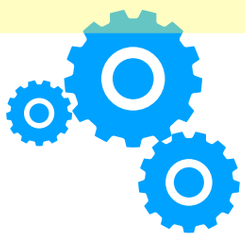


proof goal

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

$\rightarrow$

```
∃ 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

```
∃ r1 : rule. True
→
  ∃ r1 : rule.
    ∃ t1 : term.
      ∃ to1 : term_occurrence ∈ t1 : term.
          r1 is_rule_of to1
        ∧                          conjunction
          ∀ 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

variable for auxiliary lemmas

conjunction

```
∃ 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

```
∃ 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
```

variable for auxiliary lemmas

variable for terms

conjunction

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
```

implication

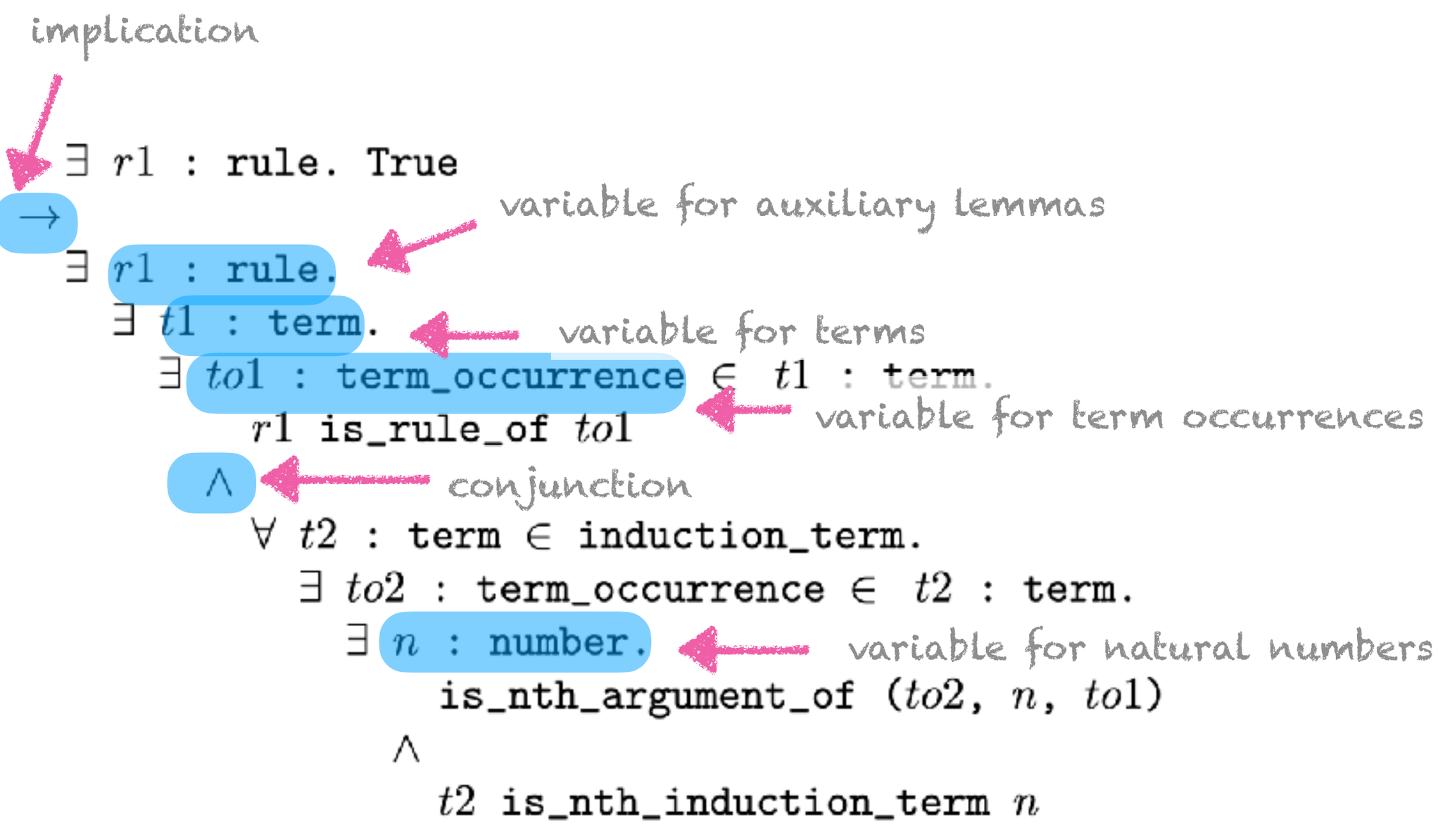
variable for auxiliary lemmas

variable for terms

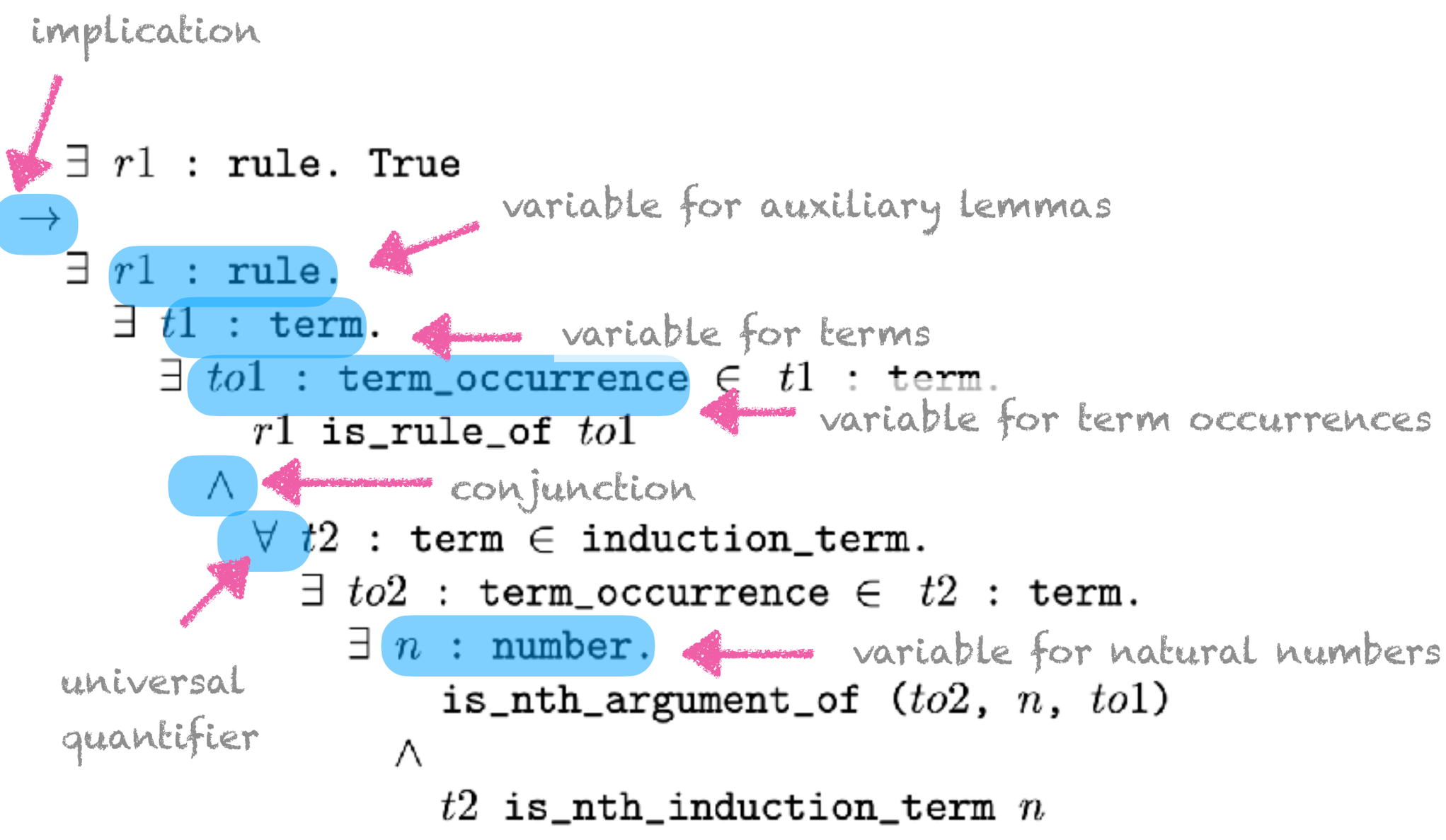
variable for term occurrences

conjunction

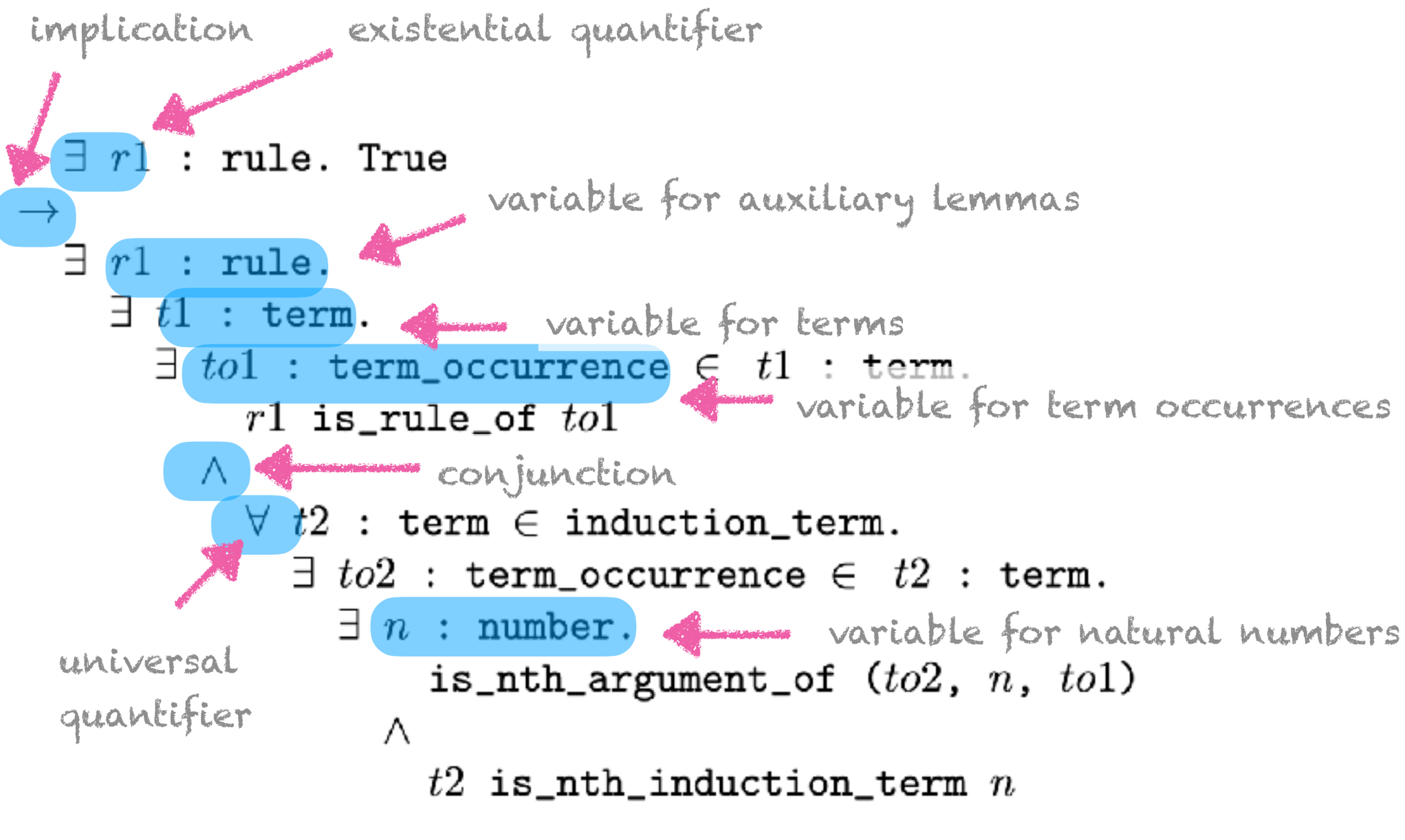
Example Heuristic in LiFtEr (in Abstract Syntax)



Example Heuristic in LiFtEr (in Abstract Syntax)

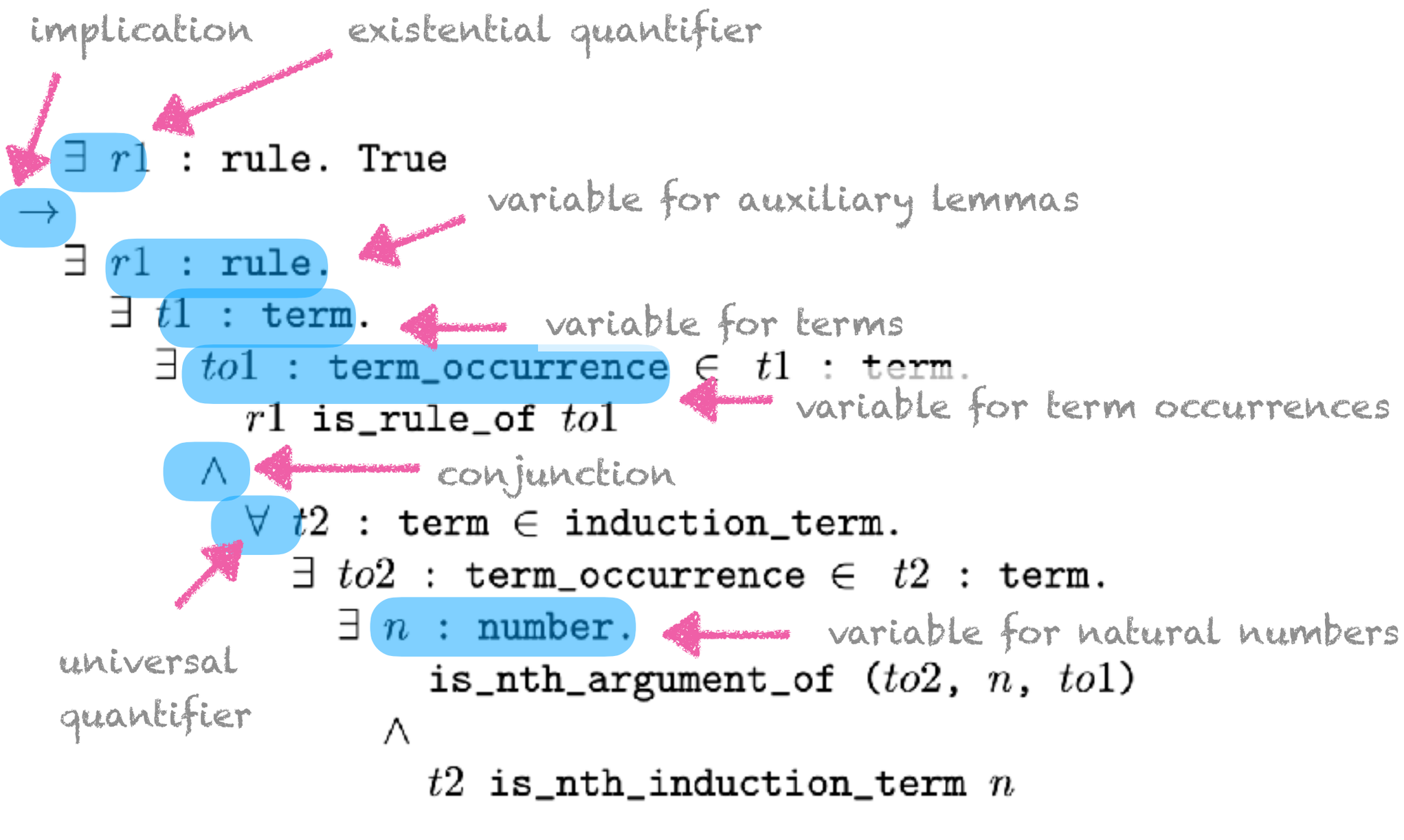


Example Heuristic in LiFtEr (in Abstract Syntax)



Example Heuristic in LiFtEr (in Abstract Syntax)

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



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

```
∃ 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
```

variable for auxiliary lemmas

variable for terms

variable for term occurrences

conjunction

universal quantifier

variable for natural numbers

DEMO3: smart_induct

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

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

DEMO3: smart_induct

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

```
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

```
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 a xs) = sep (f a) (map f xs)"
  smart_induct
apply(induction a xs rule: sep.induct)
apply auto
done
```

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

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

PaMpeR

LiFtEr

smart_induct

As the main developer.

Cogent

As a contributor.

PSL

PaMpeR

LiFtEr

smart_induct

As the main developer.

DSL with search & tracing

Cogent

As a contributor.

PSL

As the main developer.

DSL with search & tracing

PaMpeR

ML to recommend proof tactics

LiFtEr

smart_induct

As a contributor.

Cogent

As the main developer.

PSL — DSL with search & tracing

PaMpeR — ML to recommend proof tactics

LiFtEr — DSL to implement feature extractor

smart_induct

Cogent

As a contributor.

As the main developer.

PSL

DSL with search & tracing

PaMpeR

ML to recommend proof tactics

LiFtEr

DSL to implement feature extractor

smart_induct

Recommendation base on LiFtEr

As a contributor.

Cogent — **Certifying compiler**

As the main developer.

PSL — **DSL with search & tracing**

PaMpeR — **ML to recommend proof tactics**

LiFtEr — **DSL to implement feature extractor**

smart_induct — **Recommendation base on LiFtEr**

As a contributor.

Cogent — **Certifying compiler**

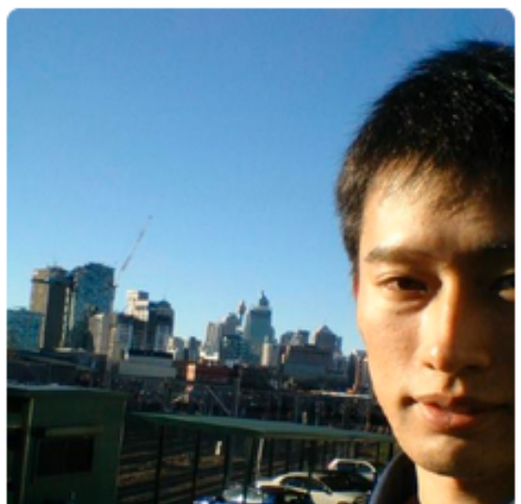
As the main developer.

PSL — **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.