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

Towards Machine Learning Induction for Isabelle/HOL

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

Yutaka Nagashima
University of Innsbruck
Czech Technical University

universität innsbruck

Yutaka Ng
yutakang

Block or report user

CVUT, CTU, CIIRC

CZECH INSTITUTE OF INFORMATICS ROBOTICS AND CYBERNETICS CTU IN PRAGUE

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

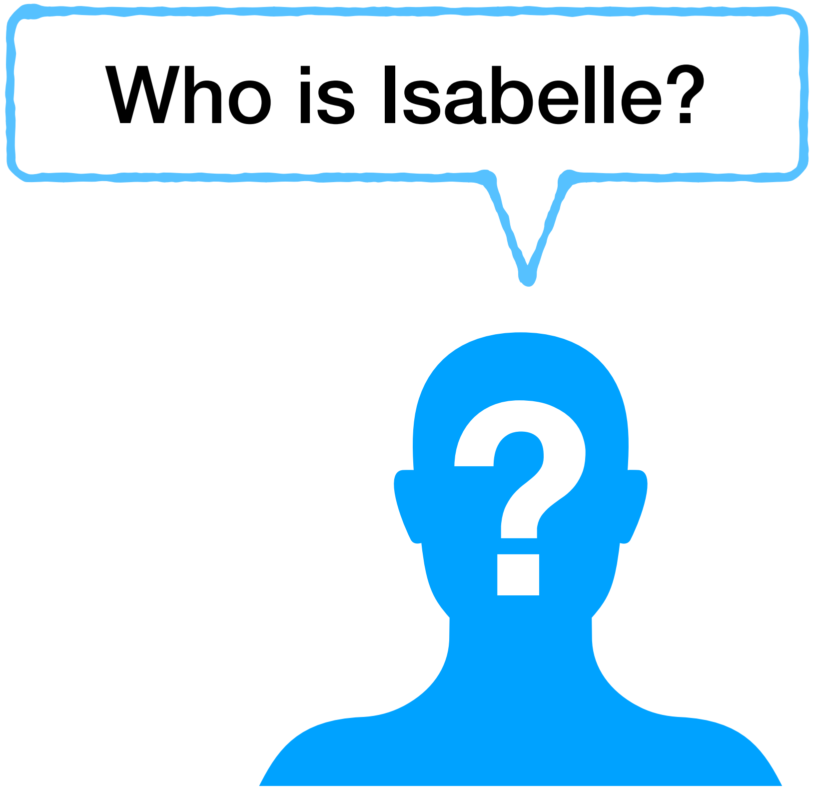
Towards Machine Learning Induction for Isabelle/HOL

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

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

Towards Machine Learning Induction for Isabelle/HOL

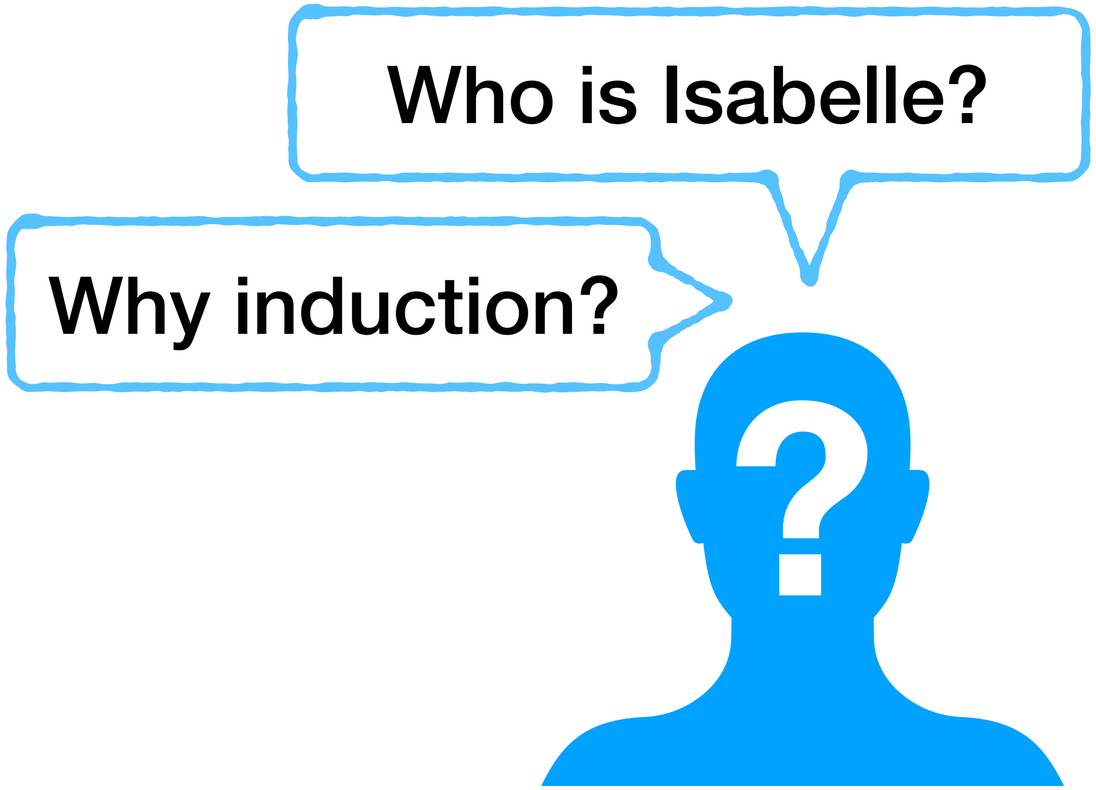
This work was supported by the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).



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

Towards Machine Learning Induction for Isabelle/HOL

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



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

Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving) problems are at the heart of many verification and reasoning tasks in

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

Who is Isabelle?

Why induction?

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

Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving) problems are at the heart of many verification and reasoning tasks in

Who is Isabelle?

Why induction?

we are convinced that substantial progress in ITP will take time.

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

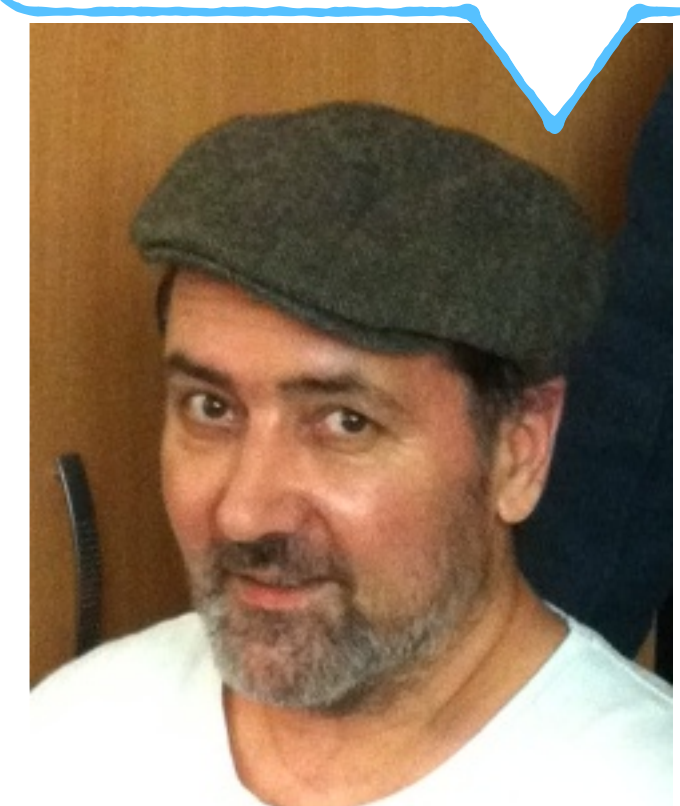
Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving) problems are at the heart of many verification and reasoning tasks in

Who is Isabelle?

Why induction?



we are convinced that substantial progress in ITP will take time.

spectacular breakthroughs are unrealistic, in view of the enormous problems and the inherent difficulty of inductive theorem proving.

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

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

Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving) problems are at the heart verification and reasoning tasks in

Challenge accepted!

?

Why in

we are convinced that subst progress in ITP will take ti

spectacular breakt ***unrealistic***, in view of problems and the inhe inductive theore

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

Yutaka Ng
yutakang
Block or report user
CVUT, CTU, CIIRC

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

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

Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving)
problems are at the heart
verificati

Challenge accepted!

?

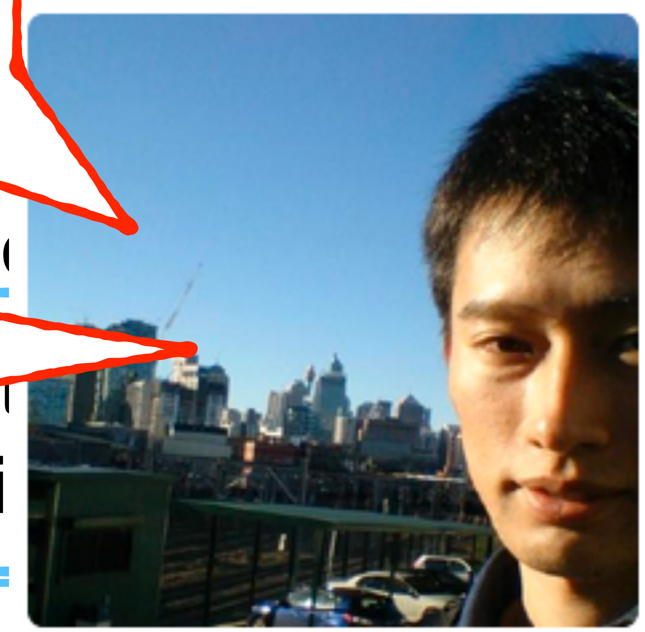
The time has come!

y in

ke ti

spectacular breakt
unrealistic, in view of
problems and the inhe
inductive theore

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/



Yutaka Ng
yutakang
Block or report user
CVUT, CTU, CIIRC

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

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

Towards Machine Learning Induction for Isabelle/HOL

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

ITP (Inductive Theorem Proving)
problems are at the heart
verificati

Challenge accepted!

?

y in

The time has come!

absi

ke ti

...or is coming soon.

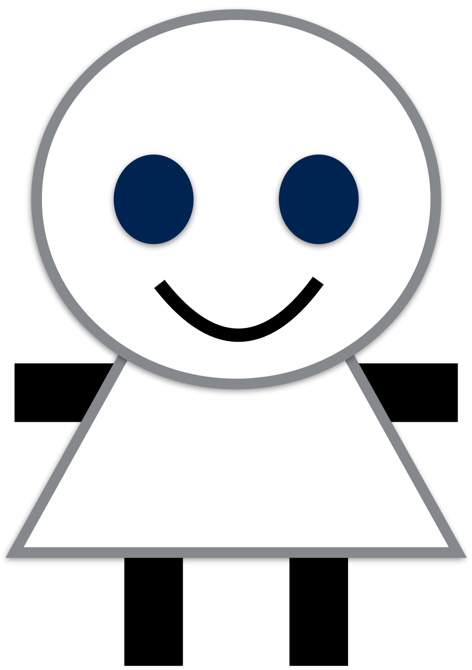
sp
unrealistic, in view of
problems and the inhe
inductive theorei

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

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

Interactive theorem proving with Isabelle/HOL



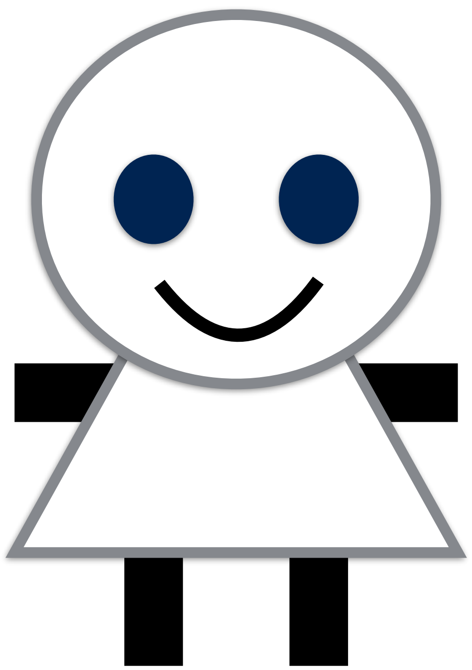
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL

proof goal

context

tactic / proof method



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

Interactive theorem proving with Isabelle/HOL

proof goal

context

tactic / proof method

error-message

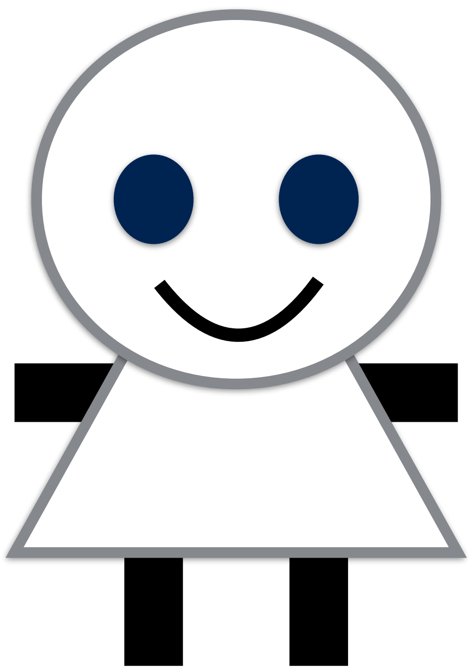
subgoals

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

Interactive theorem proving with Isabelle/HOL

proof goal **context**

tactic / proof method



error-message

subgoals

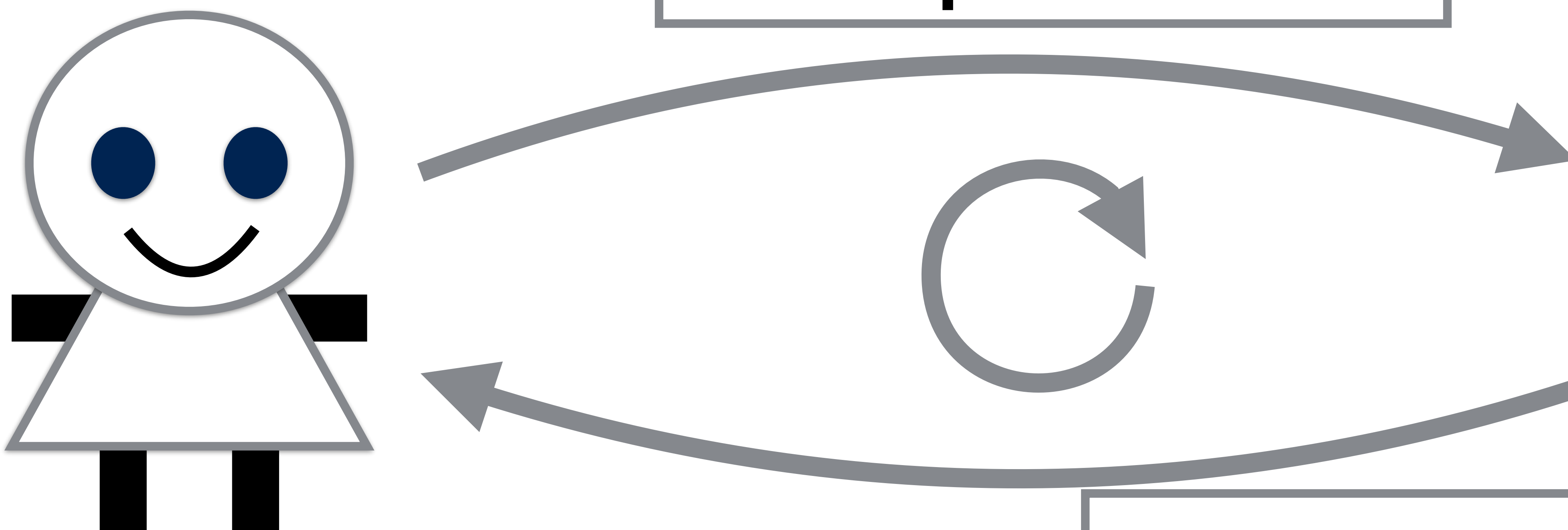
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL

proof goal

context

tactic / proof method



error-message

subgoals

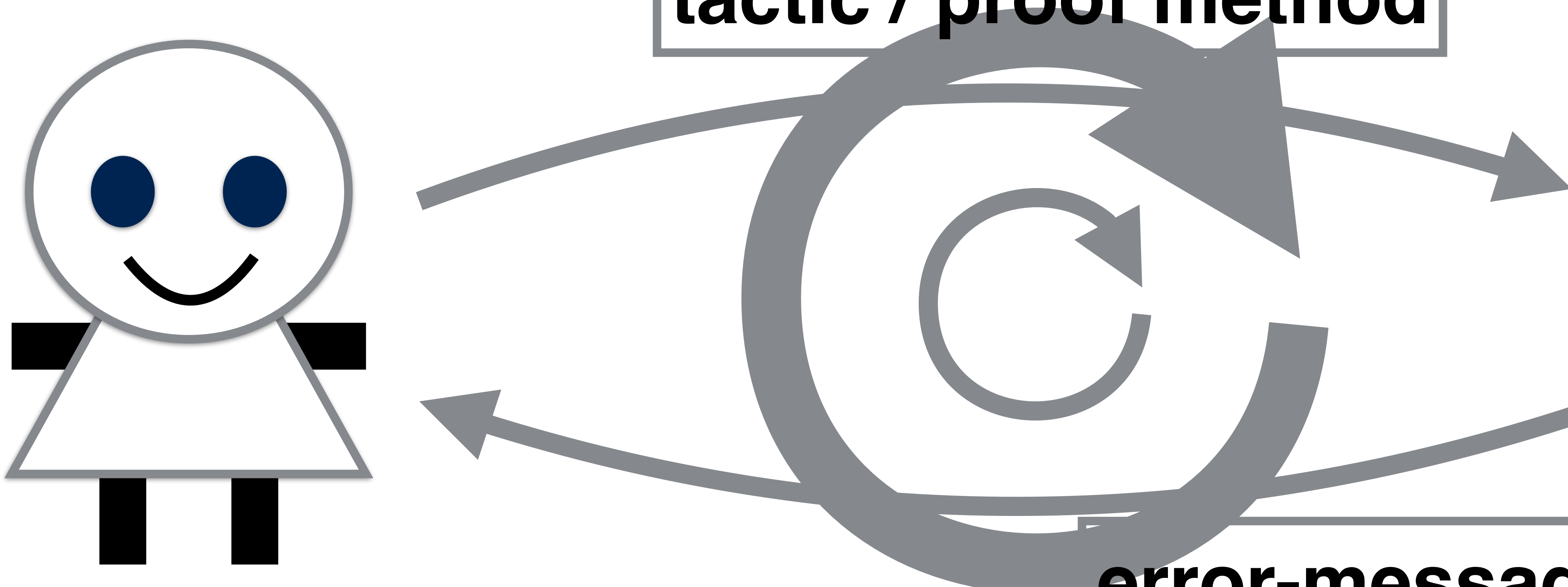
no sub-goal!

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

Interactive theorem proving with Isabelle/HOL

proof goal **context**

tactic / proof method



error-message

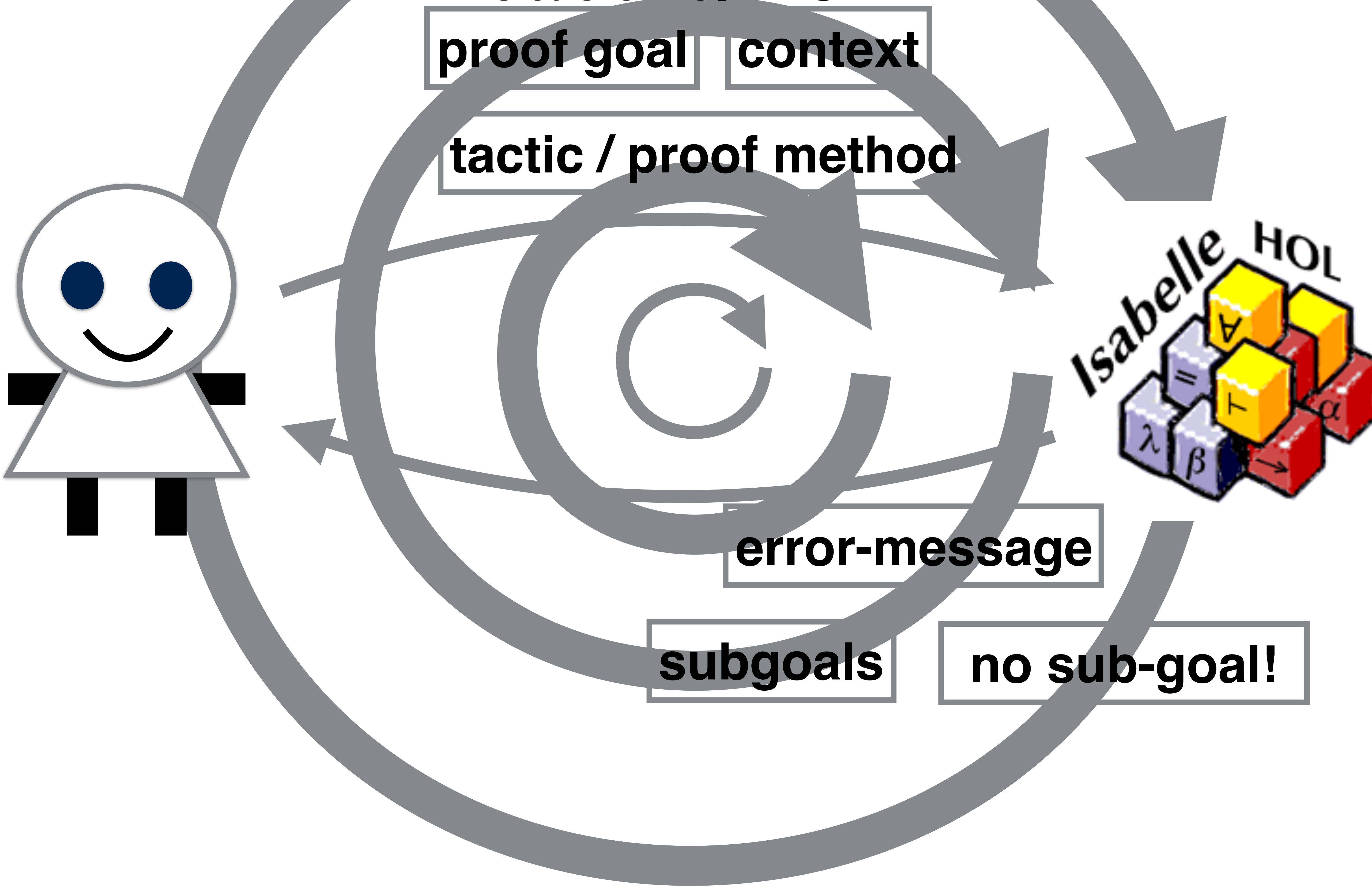
subgoals **no sub-goal!**

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

Interactive theorem proving with Isabelle/HOL

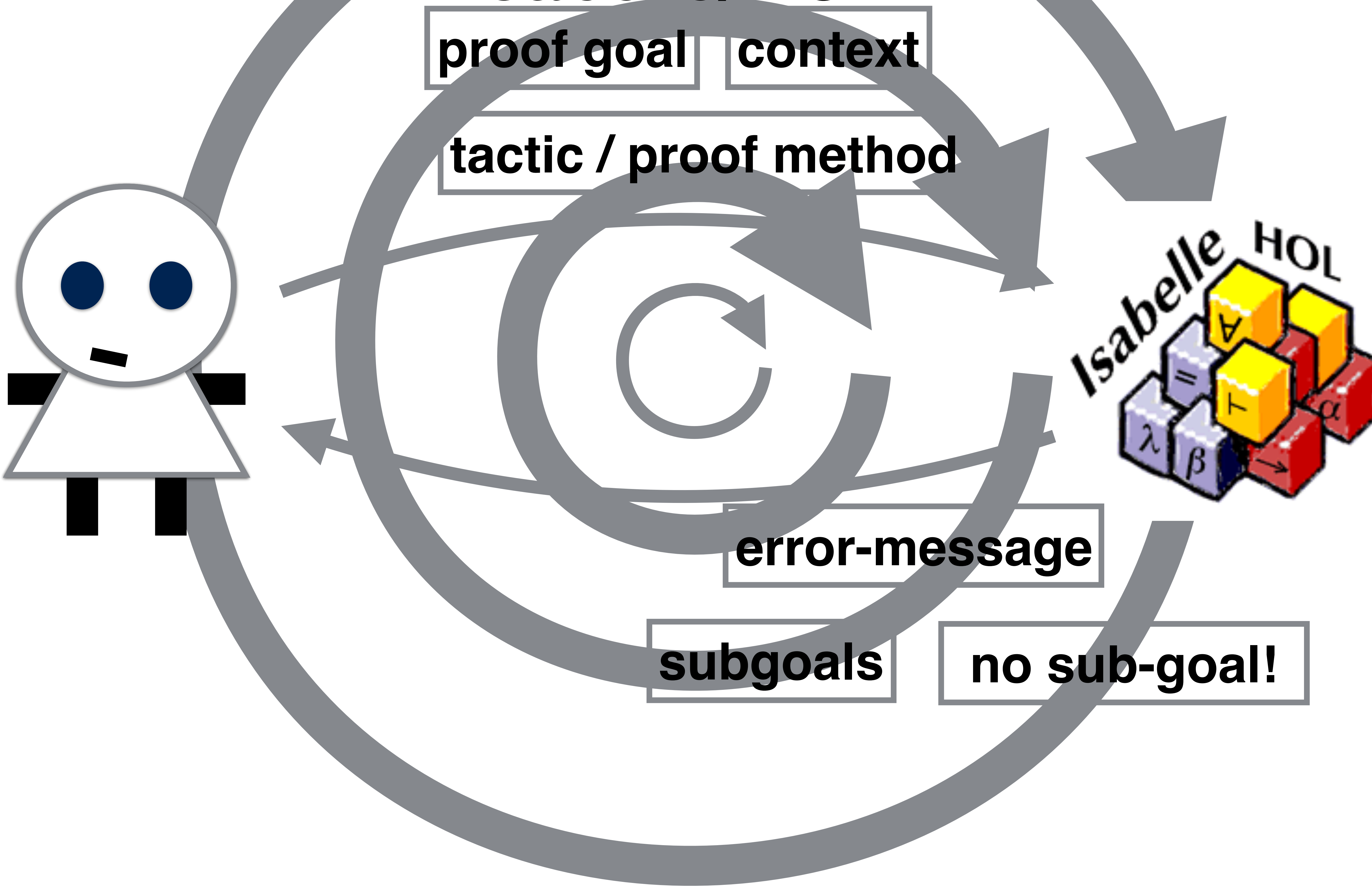
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL



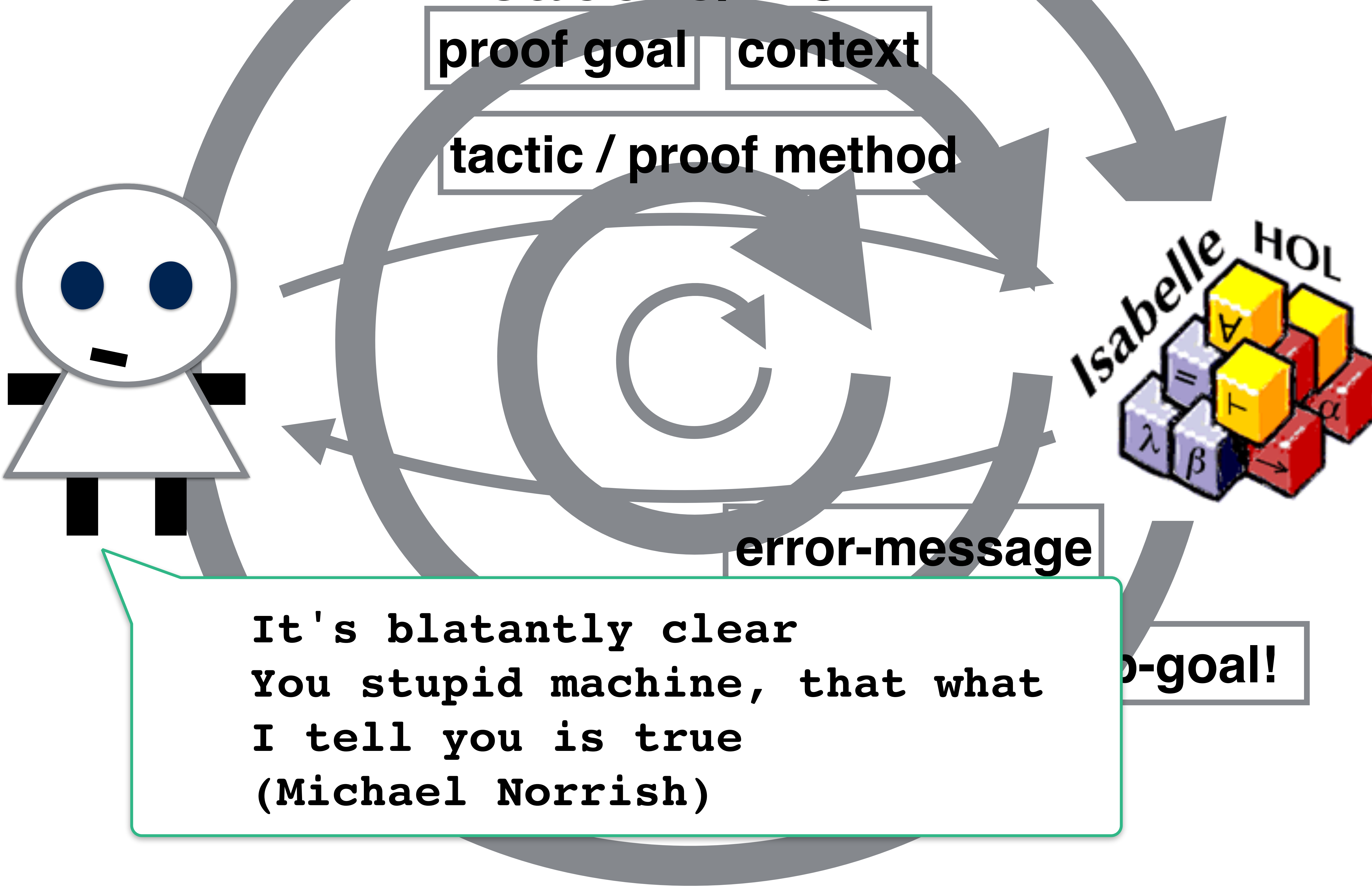
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL



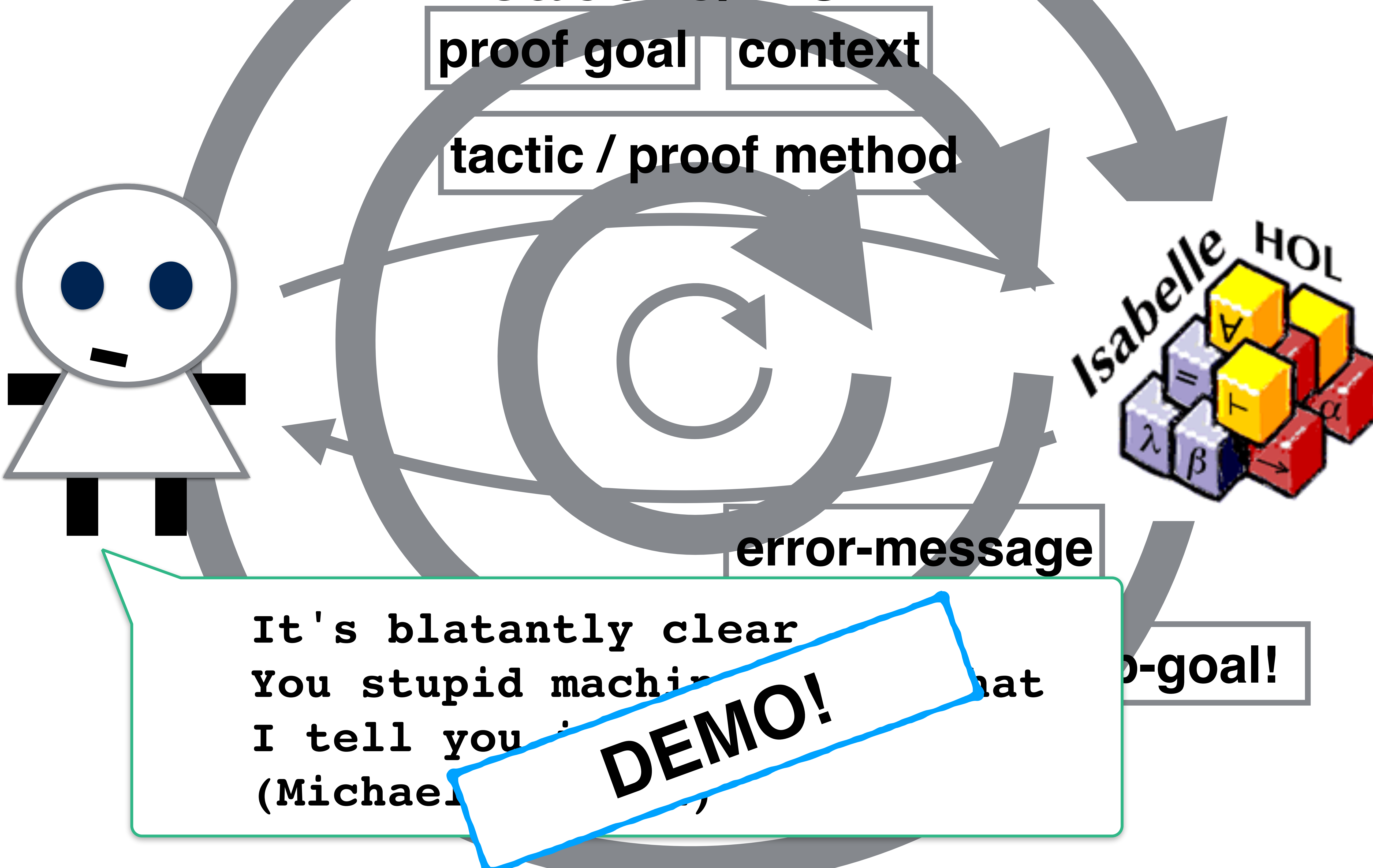
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL



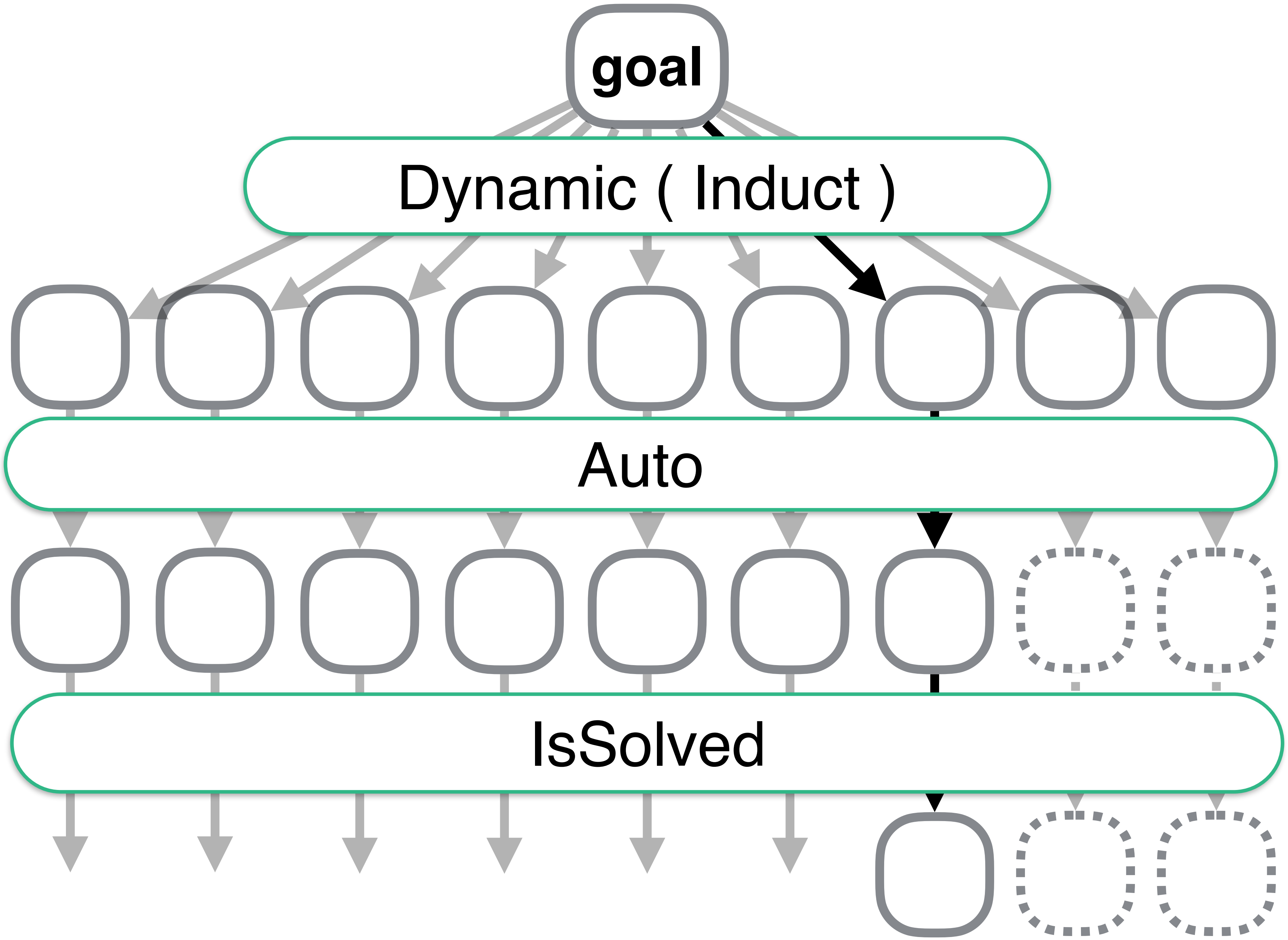
git clone https://github.com/data61/PSL

Interactive theorem proving with Isabelle/HOL



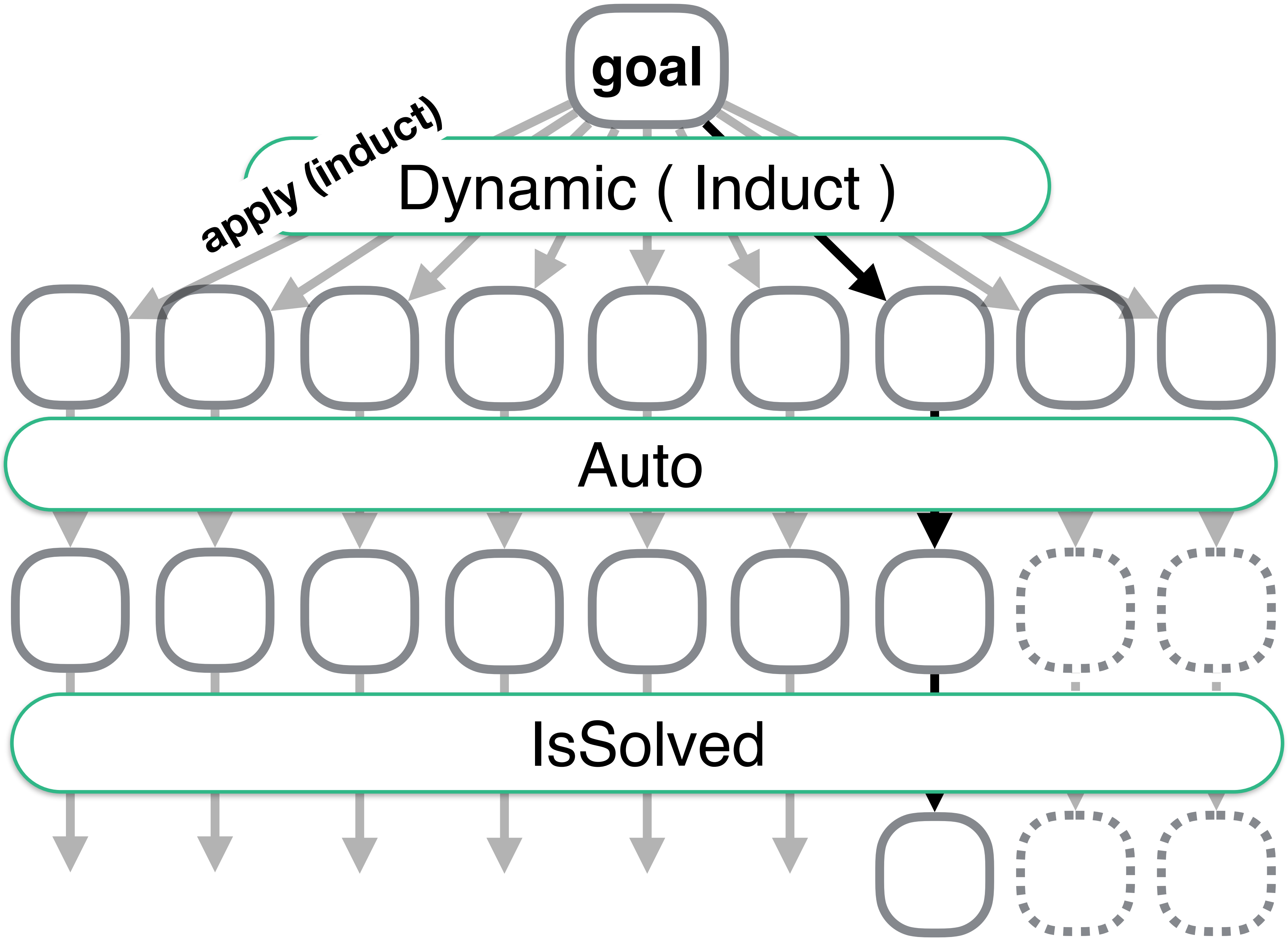
git clone https://github.com/data61/PSL

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



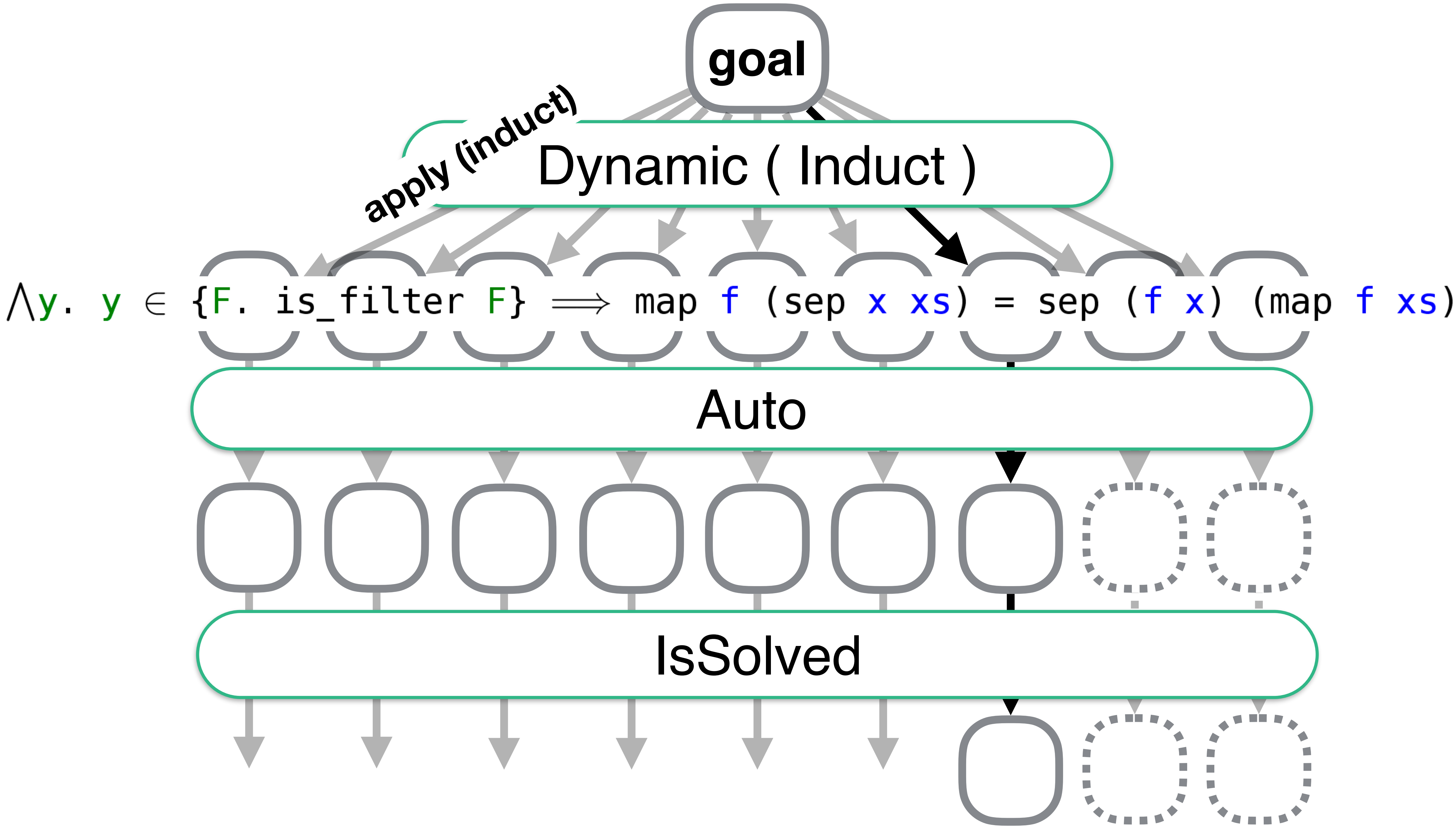
git clone https://github.com/data61/PSL

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



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

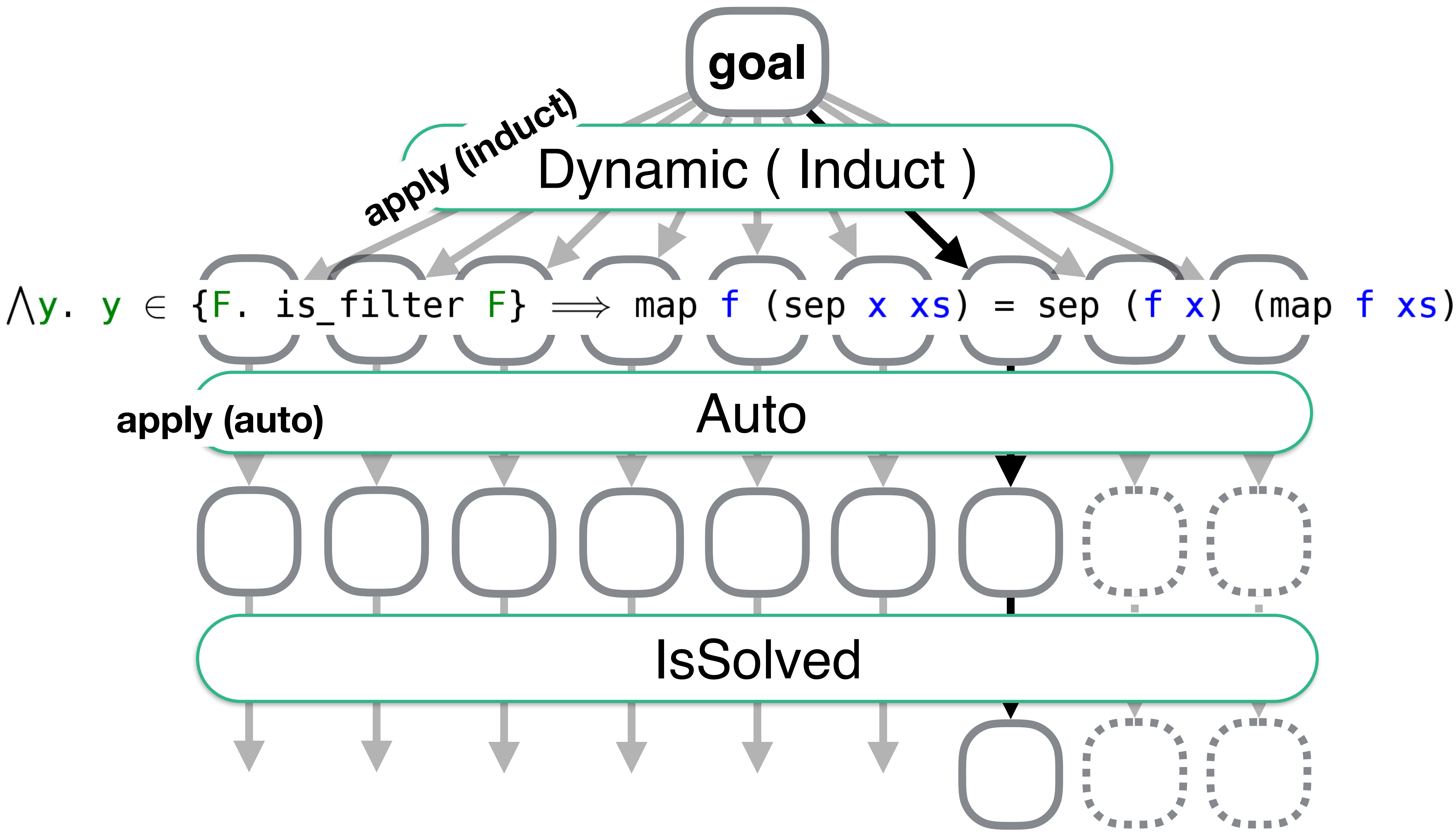
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"

find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



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

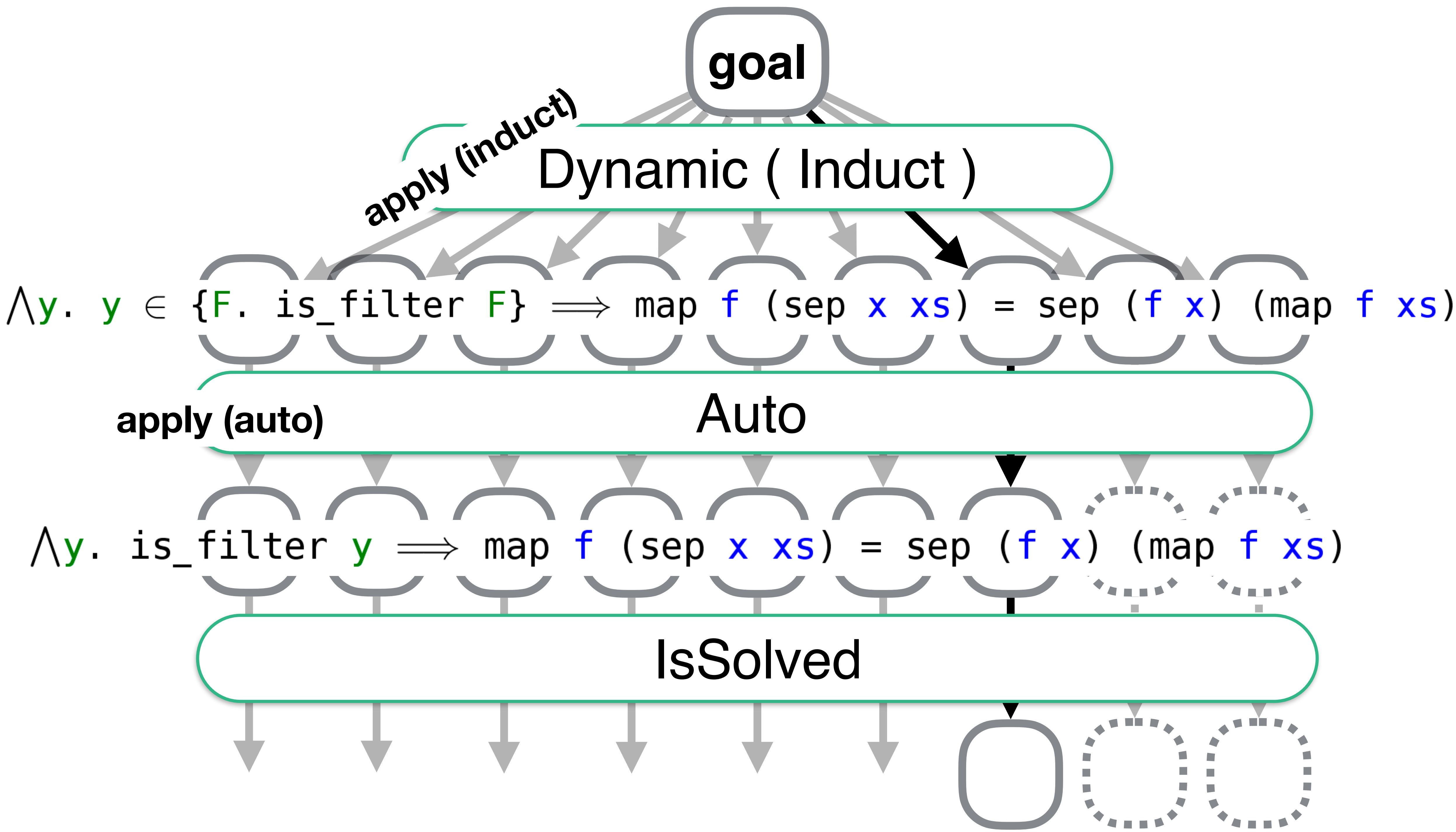
```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
```

```
find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



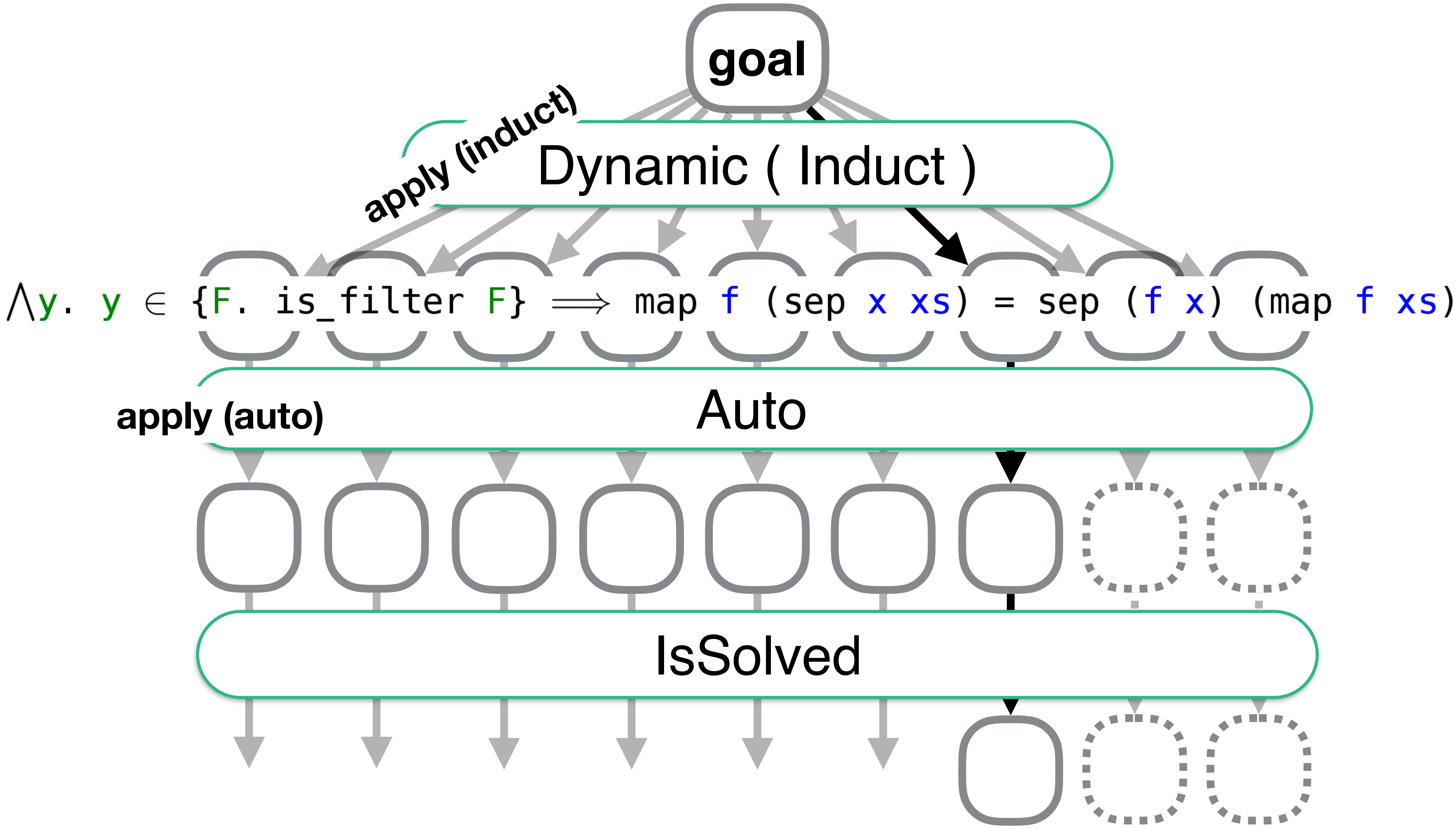
git clone https://github.com/data61/PSL

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



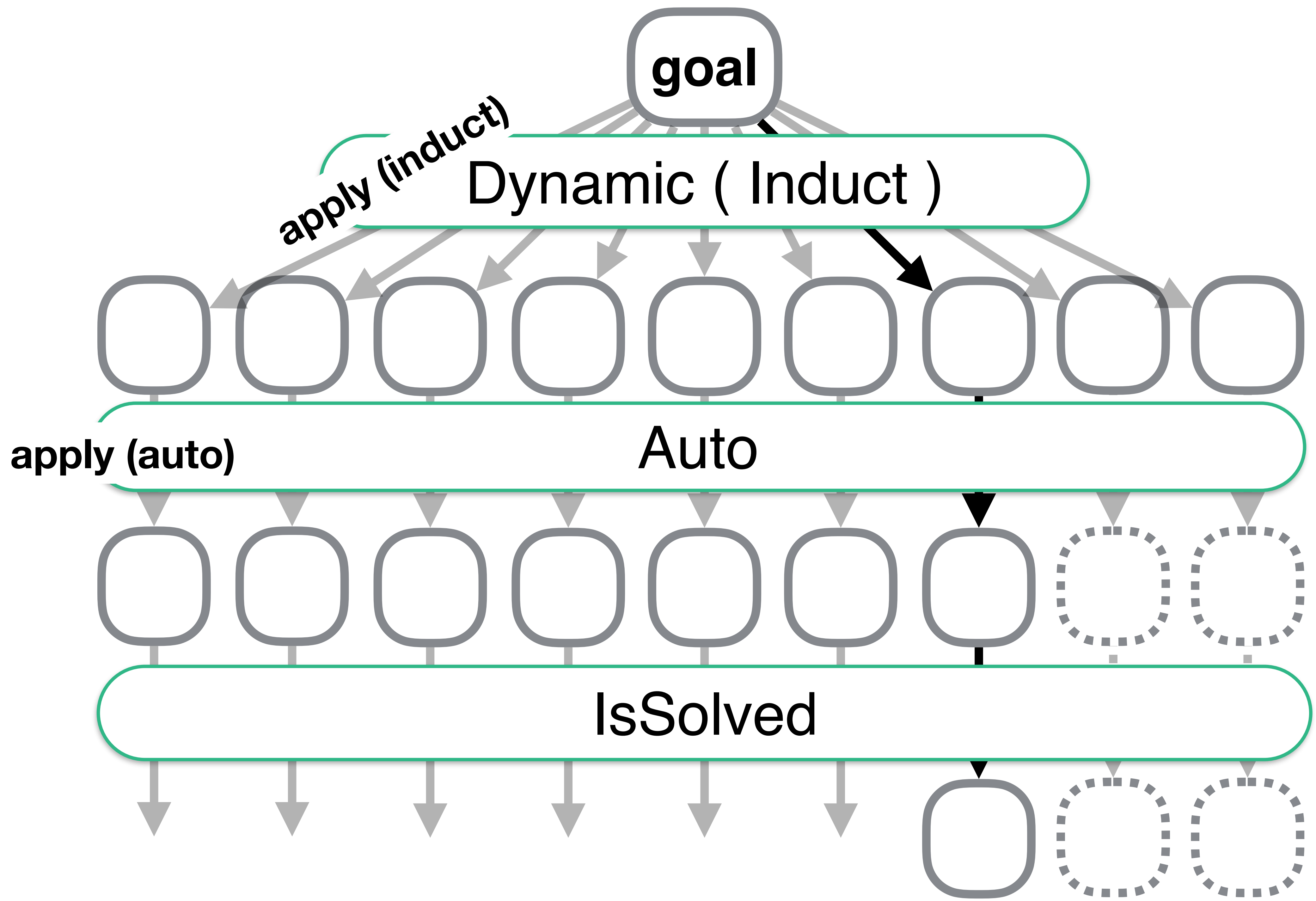
git clone https://github.com/data61/PSL

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



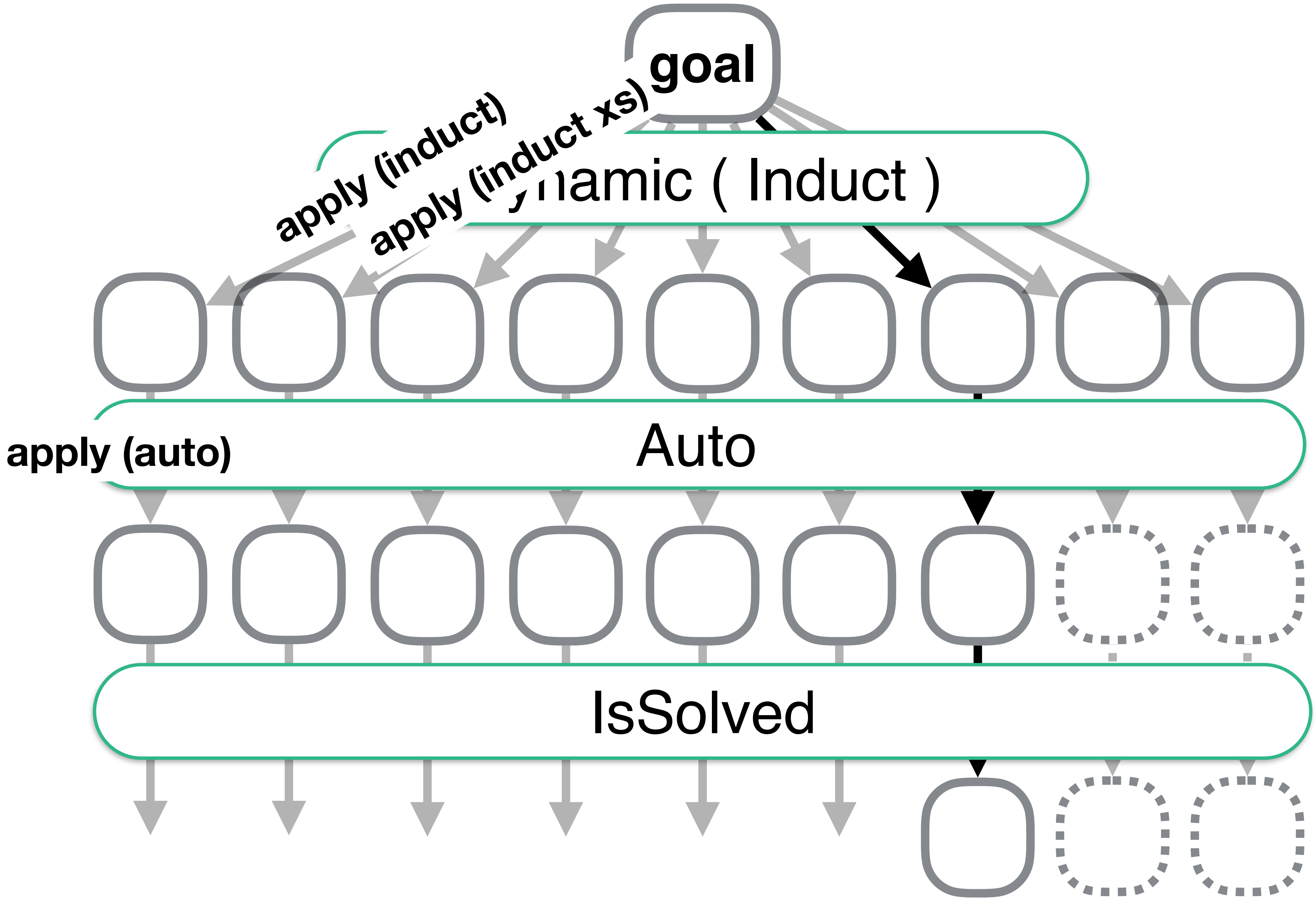
git clone https://github.com/data61/PSL

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



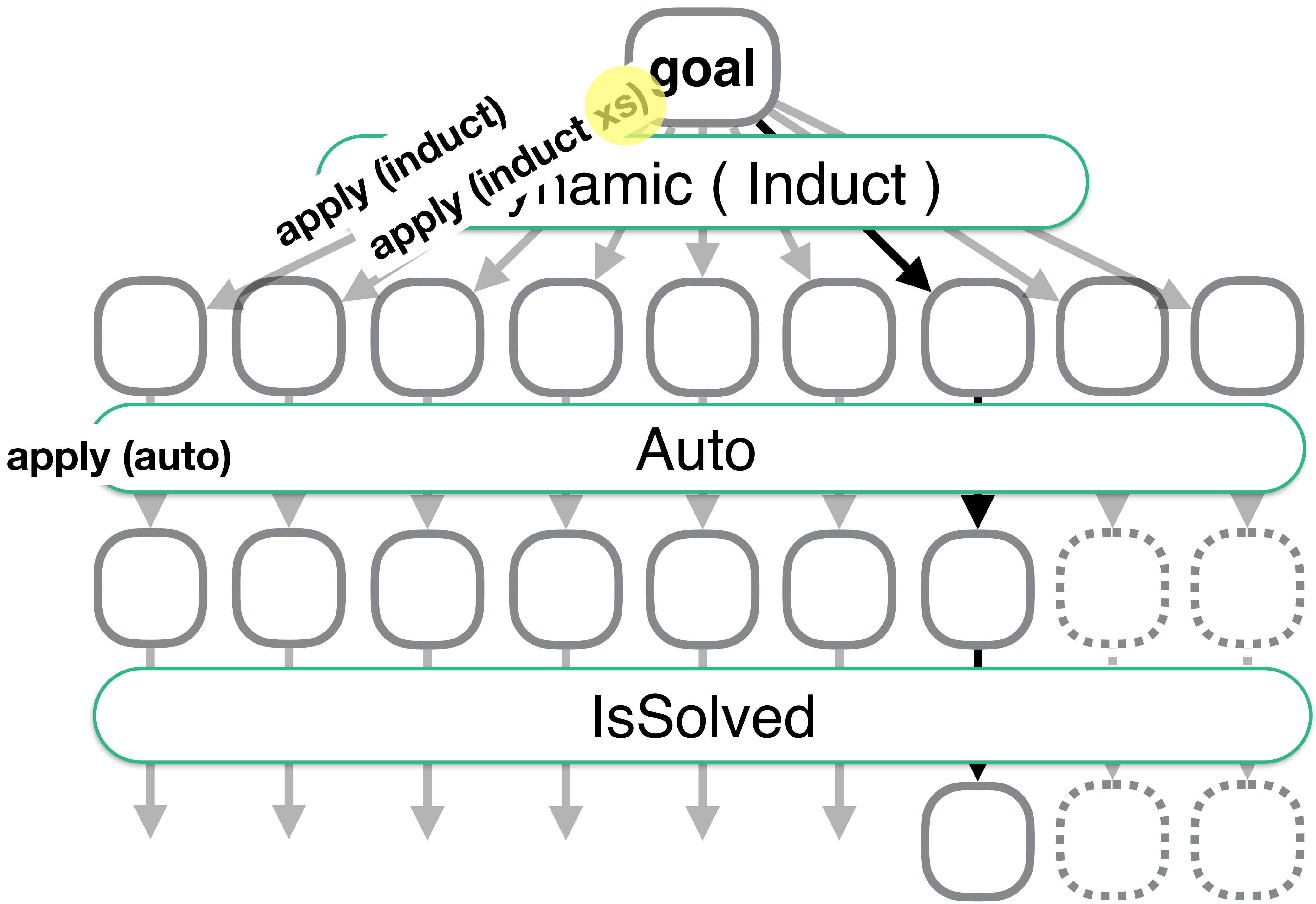
git clone https://github.com/data61/PSL

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



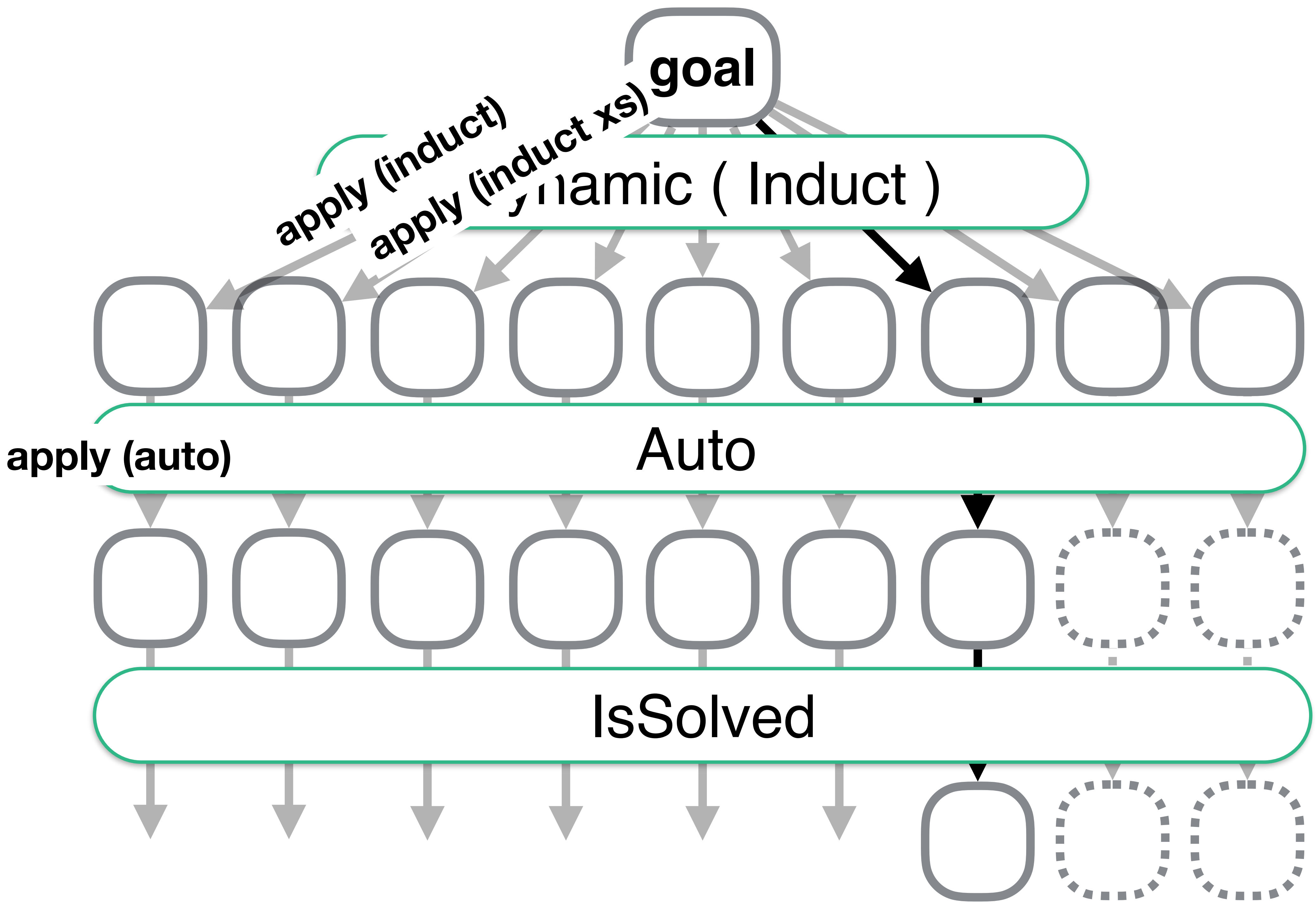
git clone https://github.com/data61/PSL

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



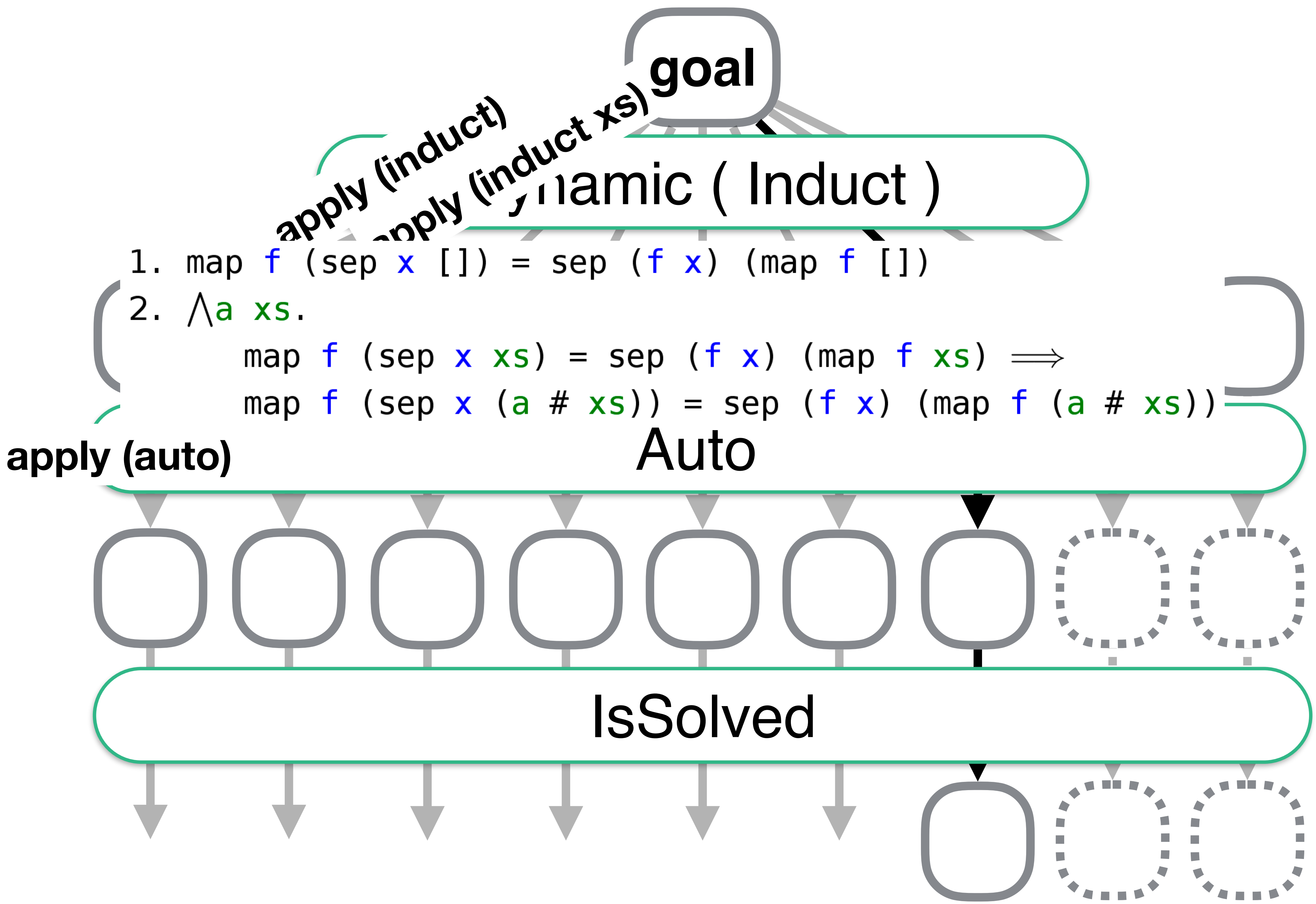
git clone https://github.com/data61/PSL

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



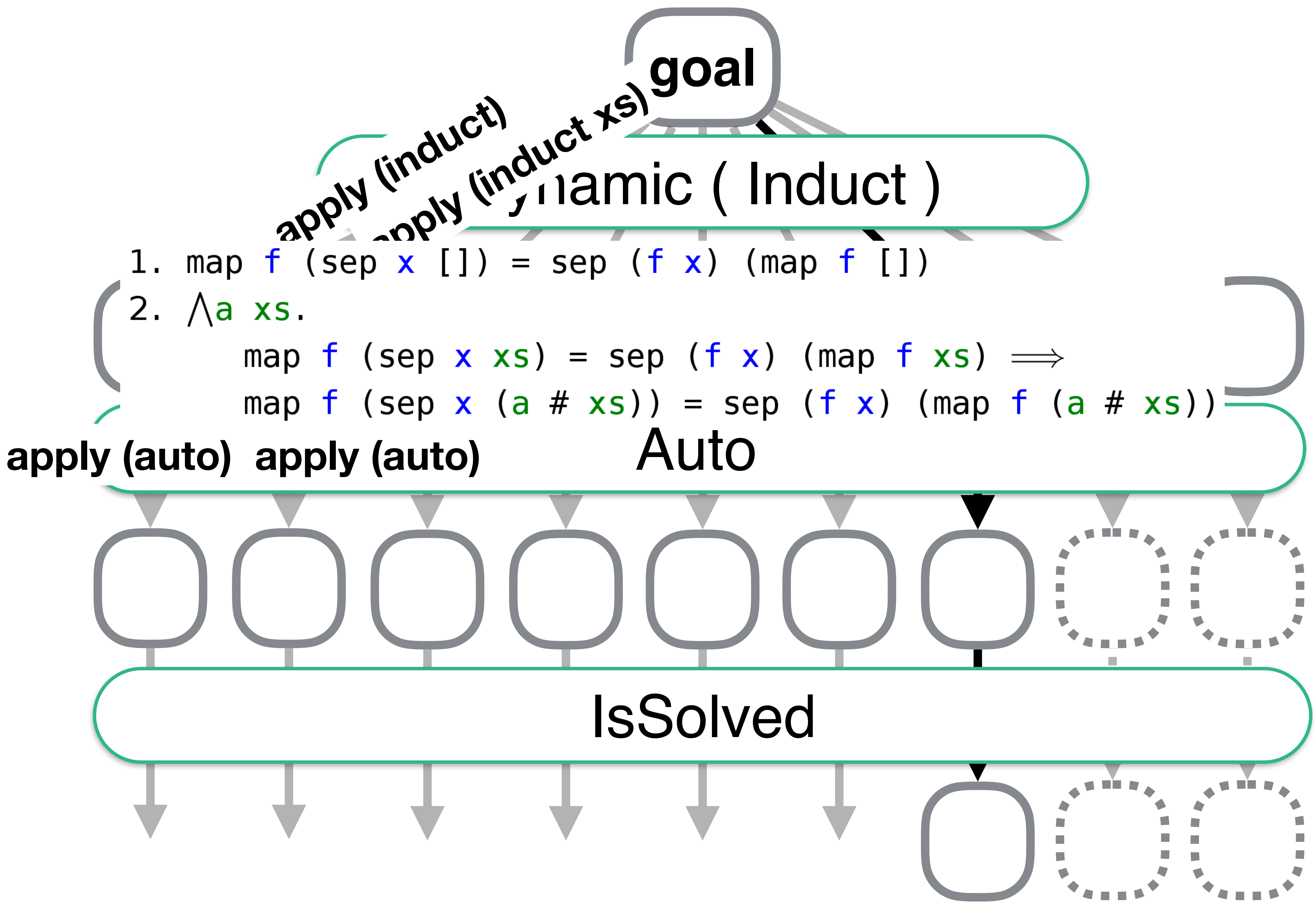
git clone https://github.com/data61/PSL

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



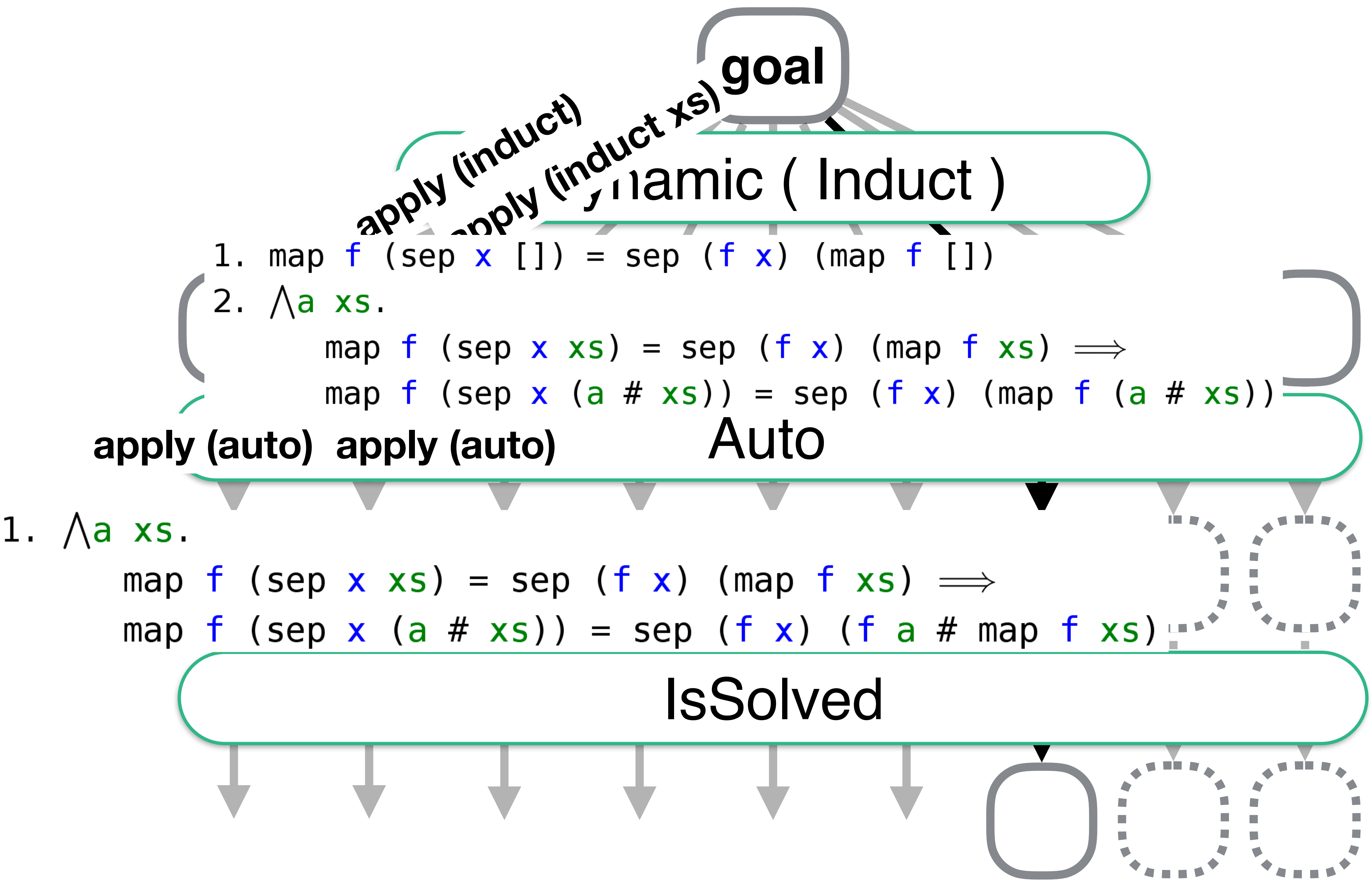
git clone https://github.com/data61/PSL

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



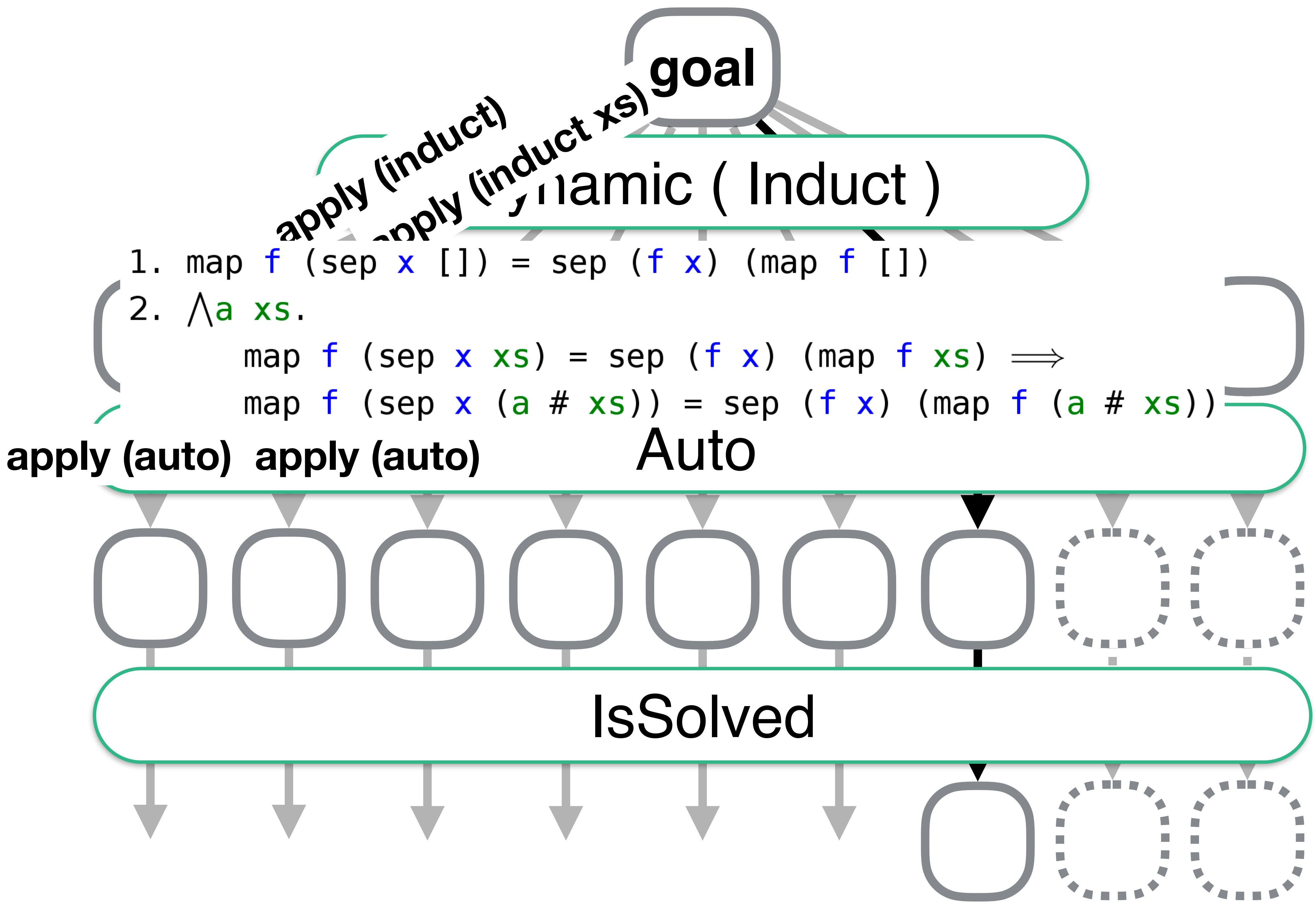
git clone https://github.com/data61/PSL

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



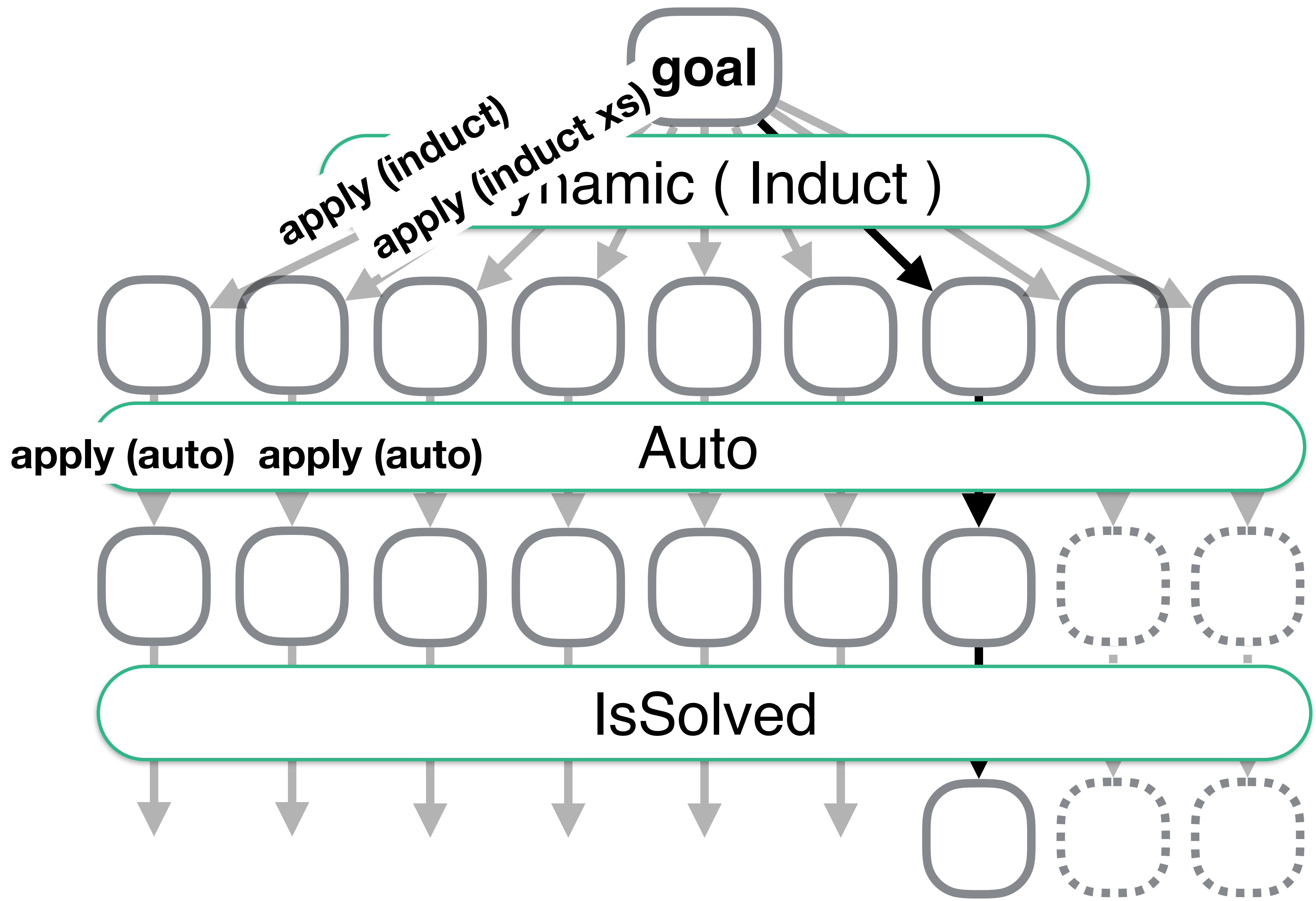
git clone https://github.com/data61/PSL

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



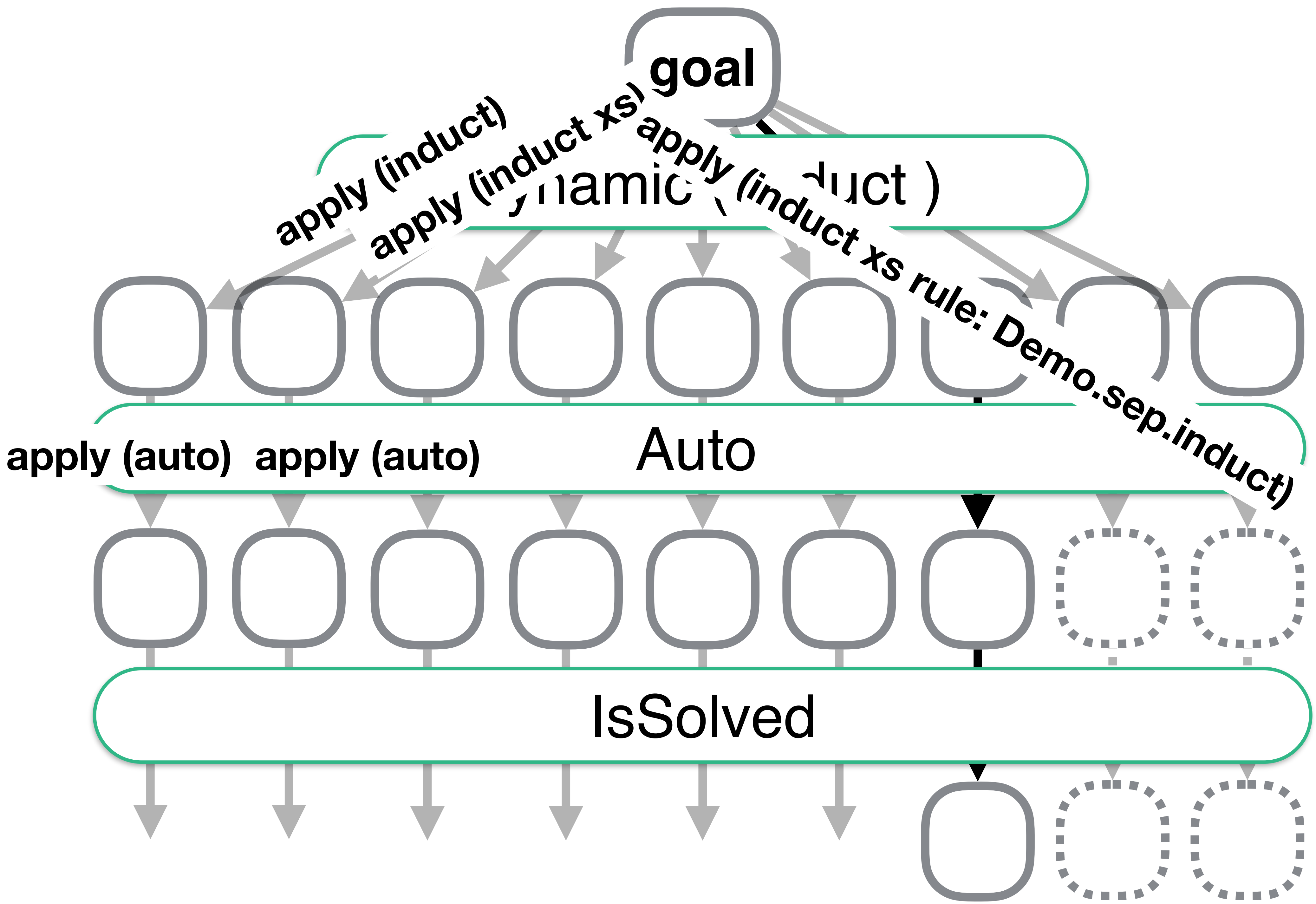
git clone https://github.com/data61/PSL

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



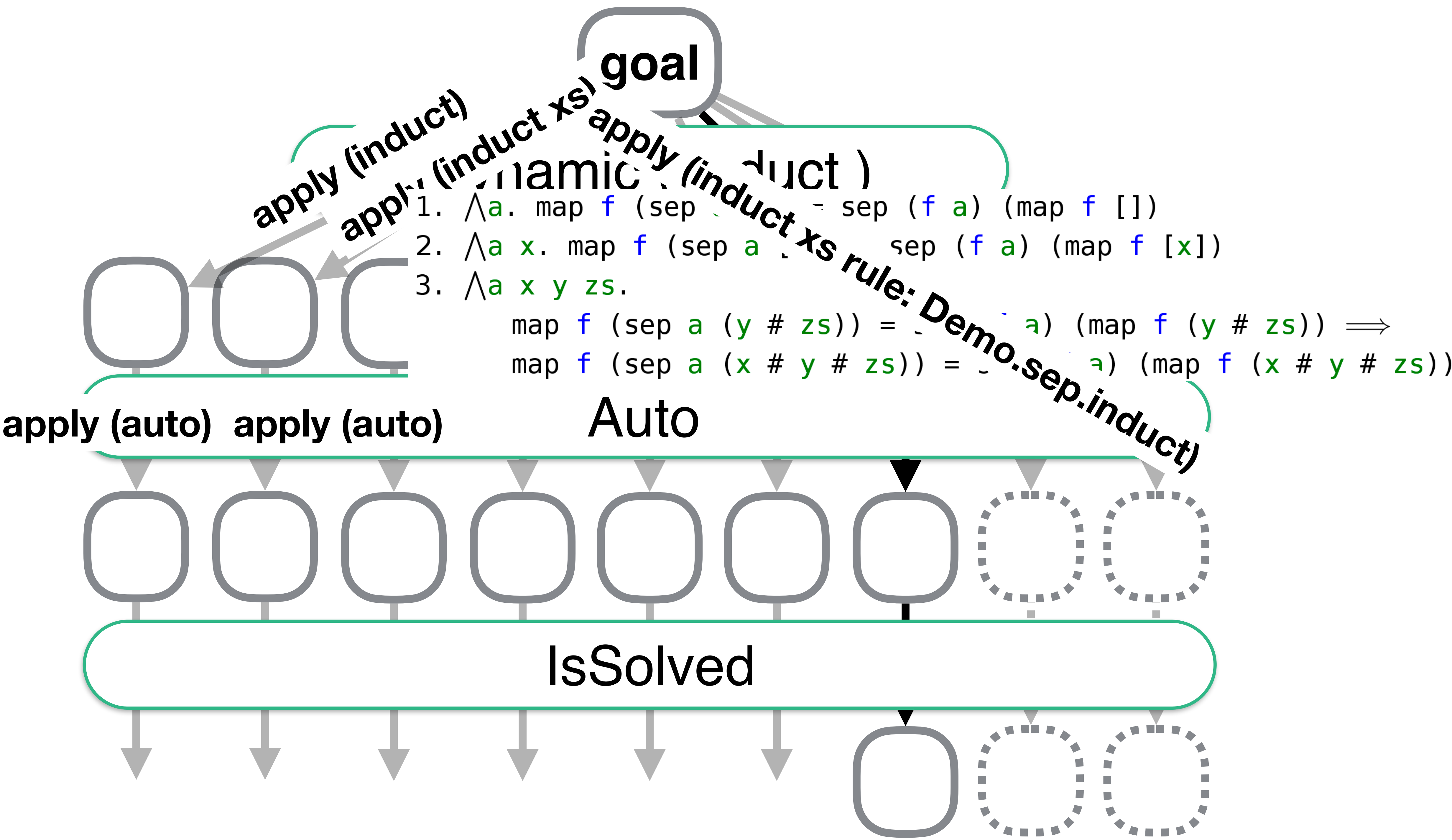
git clone https://github.com/data61/PSL

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



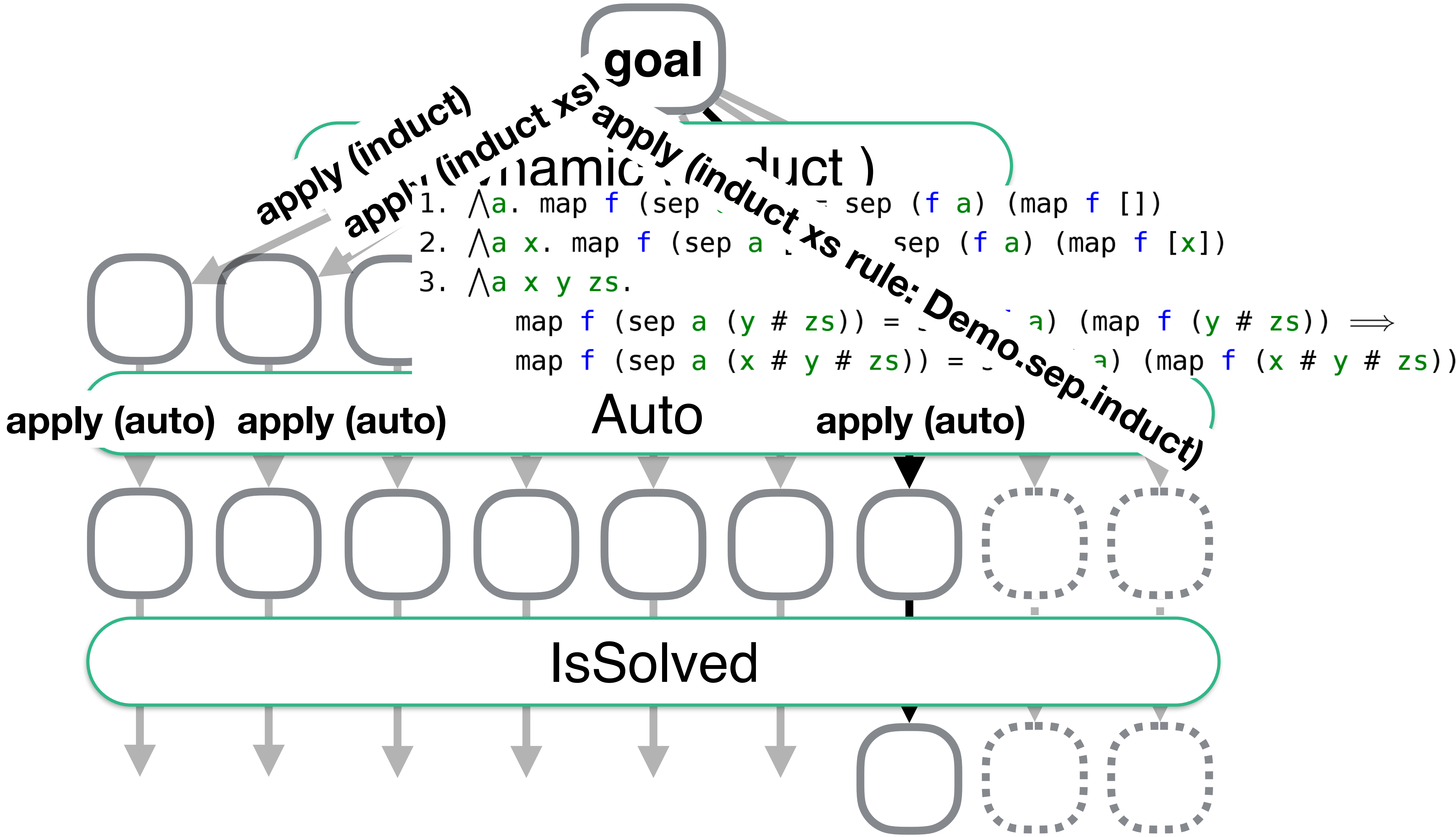
git clone https://github.com/data61/PSL

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



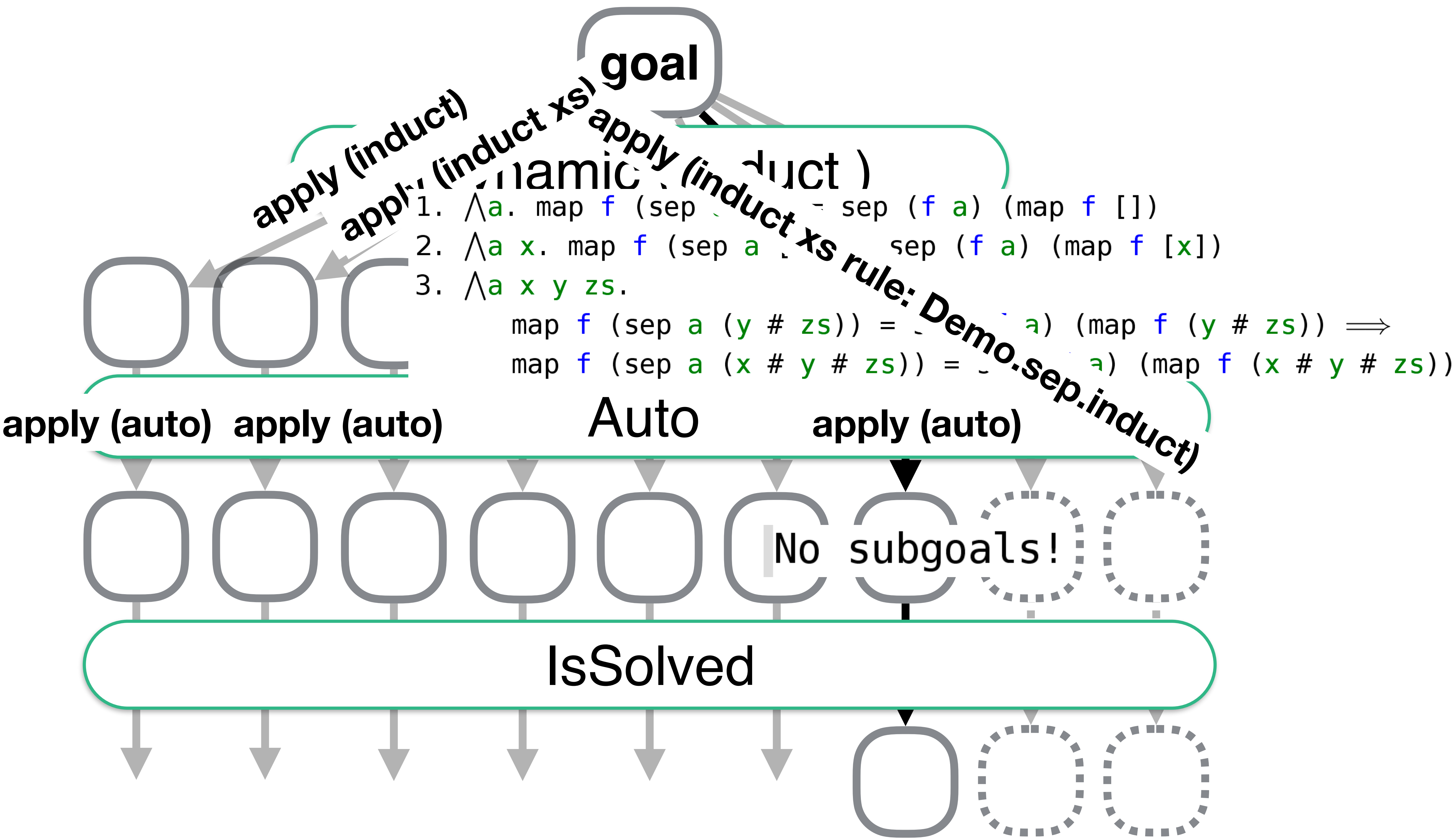
git clone https://github.com/data61/PSL

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

find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



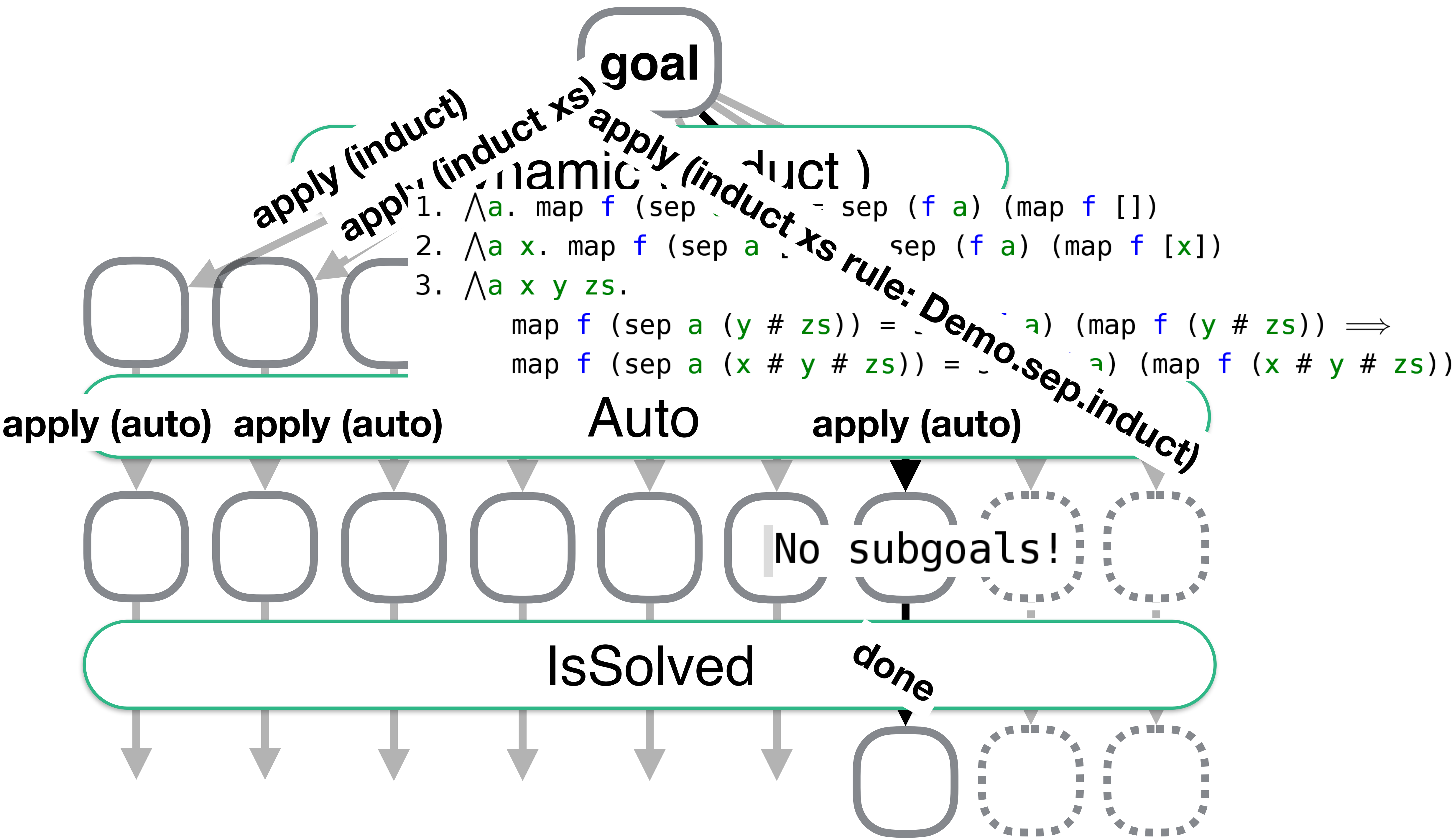
git clone https://github.com/data61/PSL

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

find_proof DInd(*= Thens [Dynamic (Induct), Auto, IsSolved]*)
```



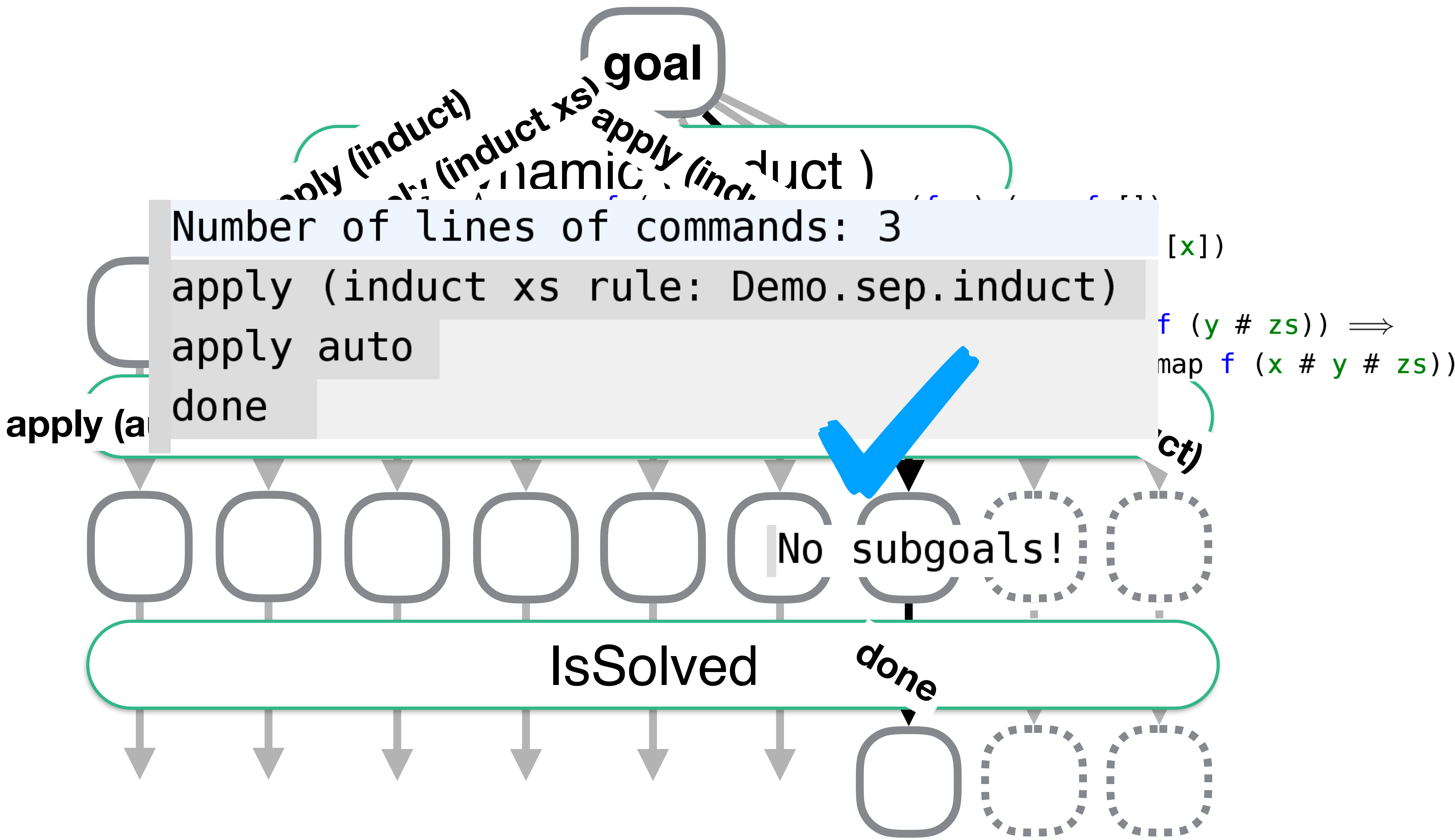
git clone https://github.com/data61/PSL

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



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

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



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

Try_Hard: the default strategy

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

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

16 percentage point performance improvement compared to sledgehammer

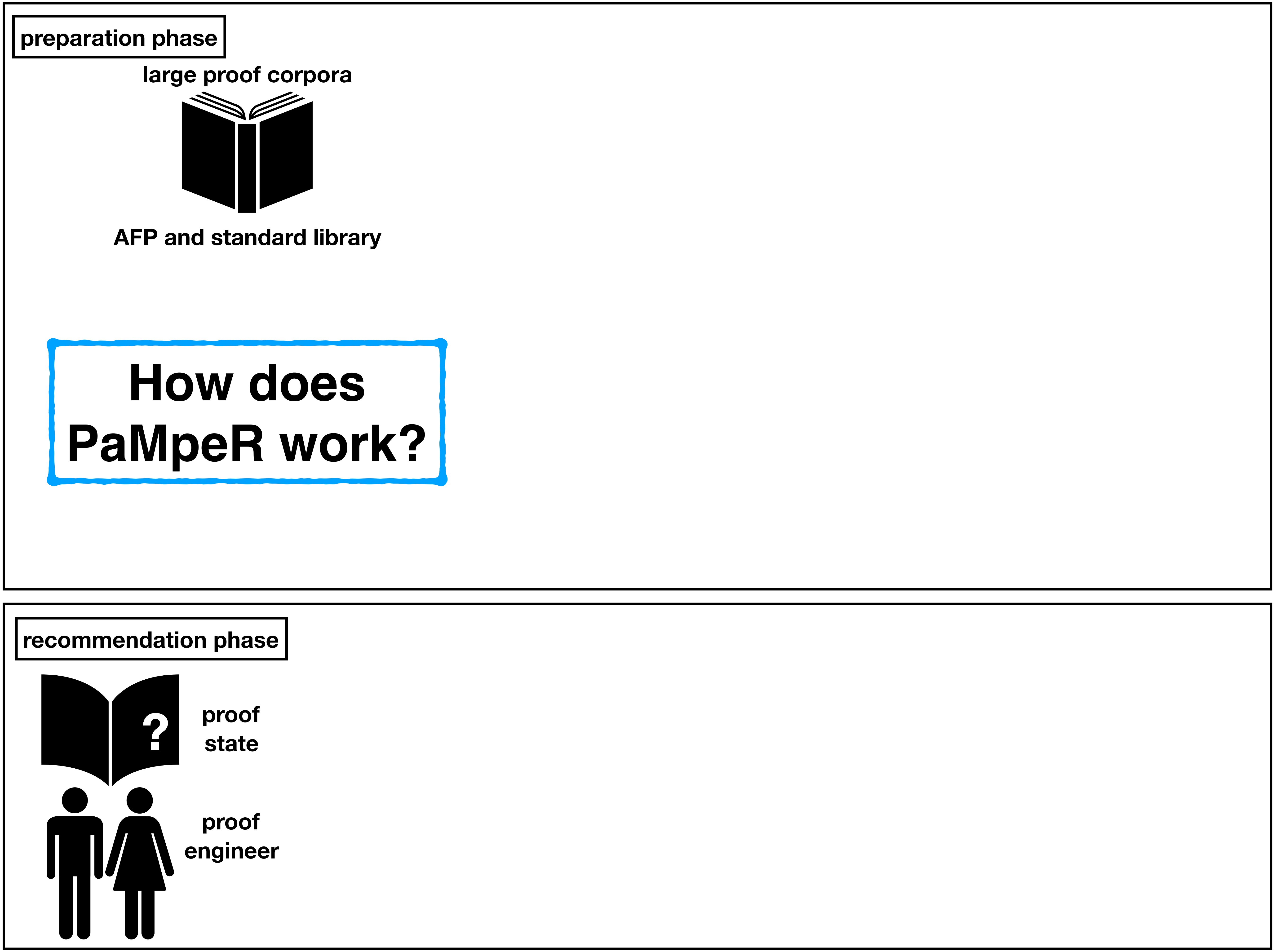
but the search space explodes

PaMpeR: Proof Method Recommendation

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

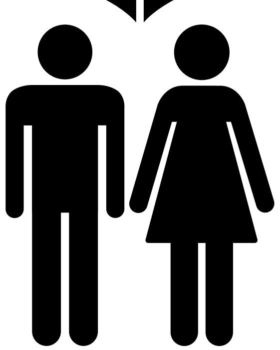
large proof corpora

AFP and standard library

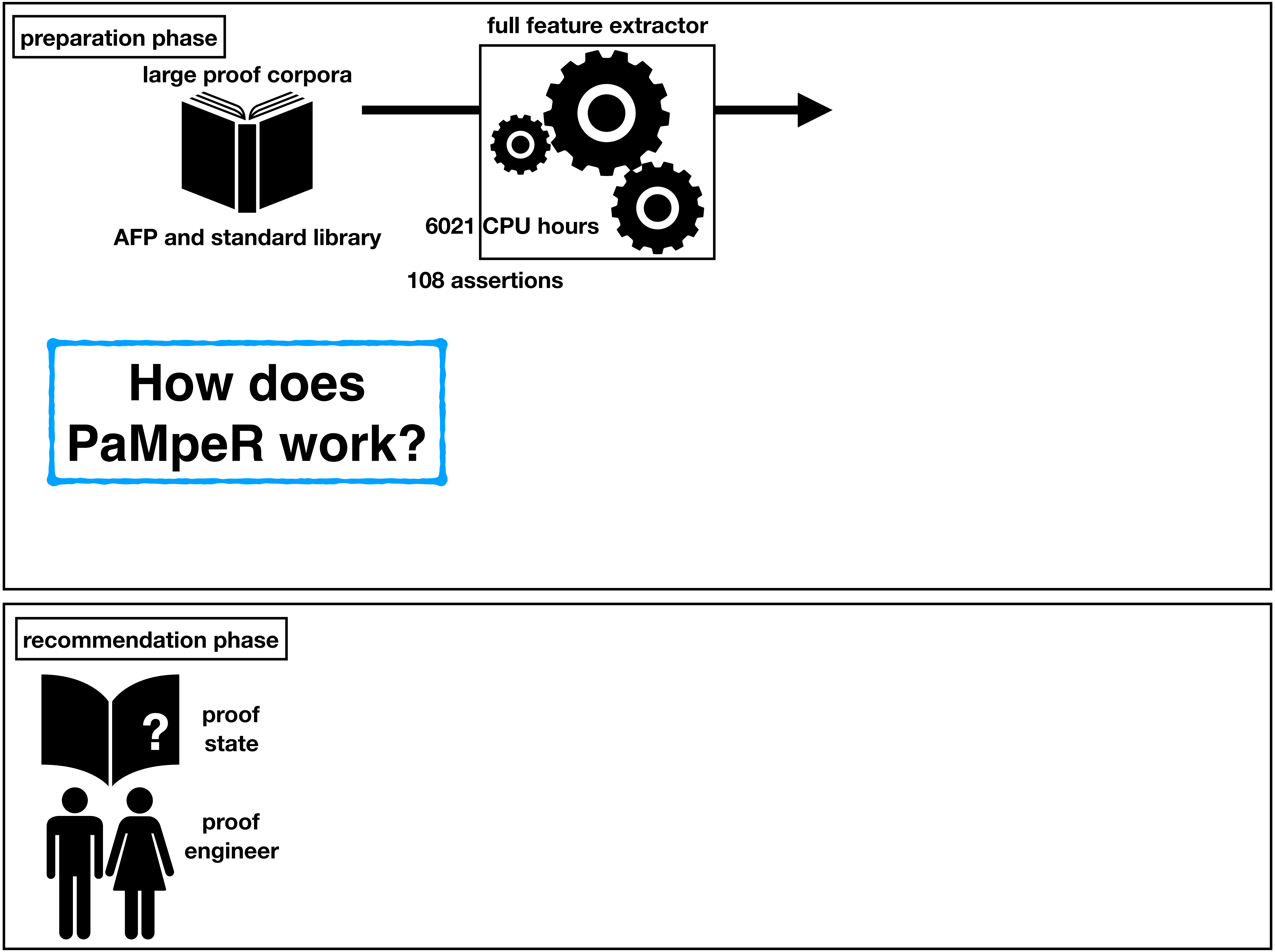
How does PaMpeR work?

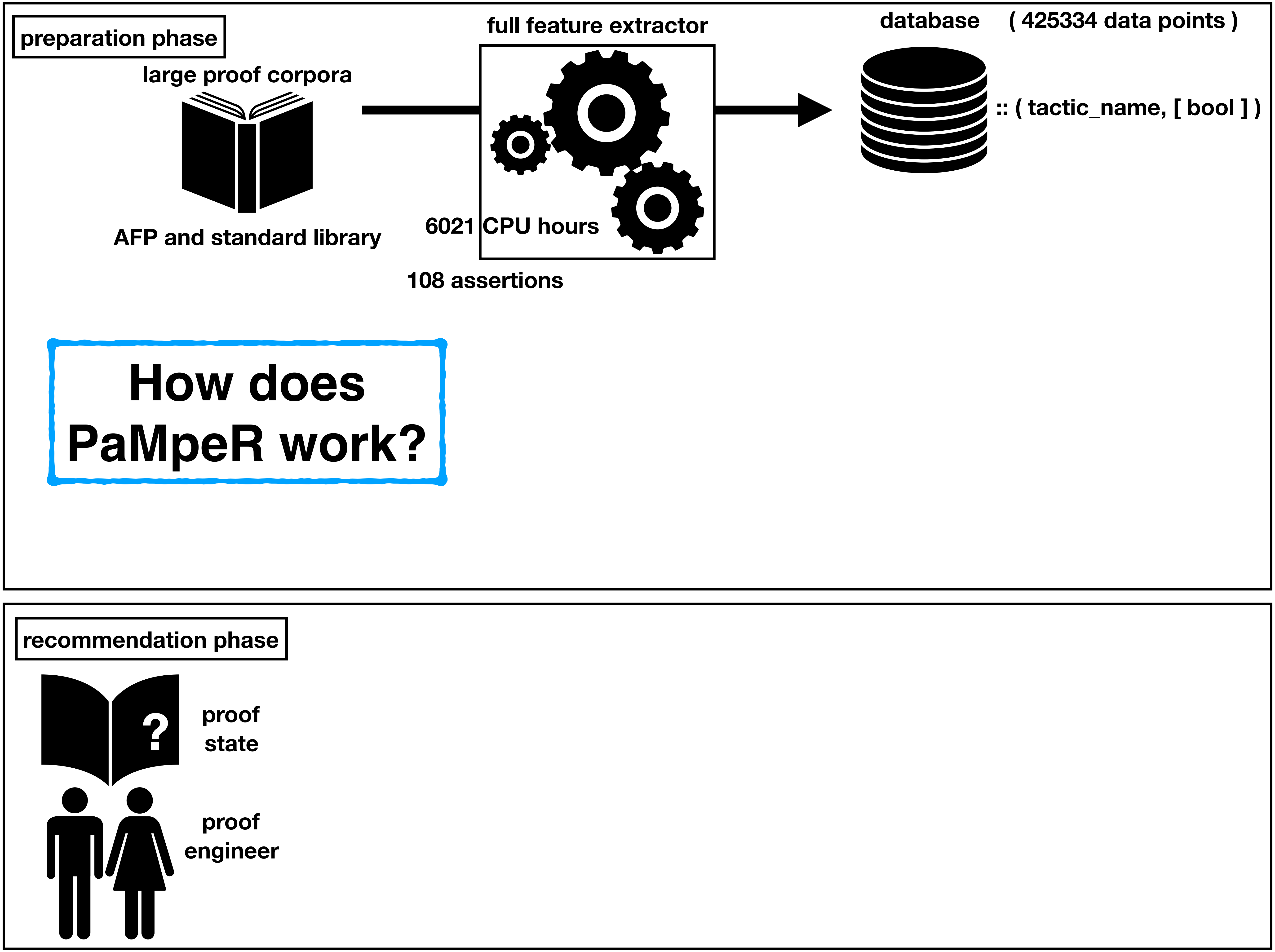
recommendation phase

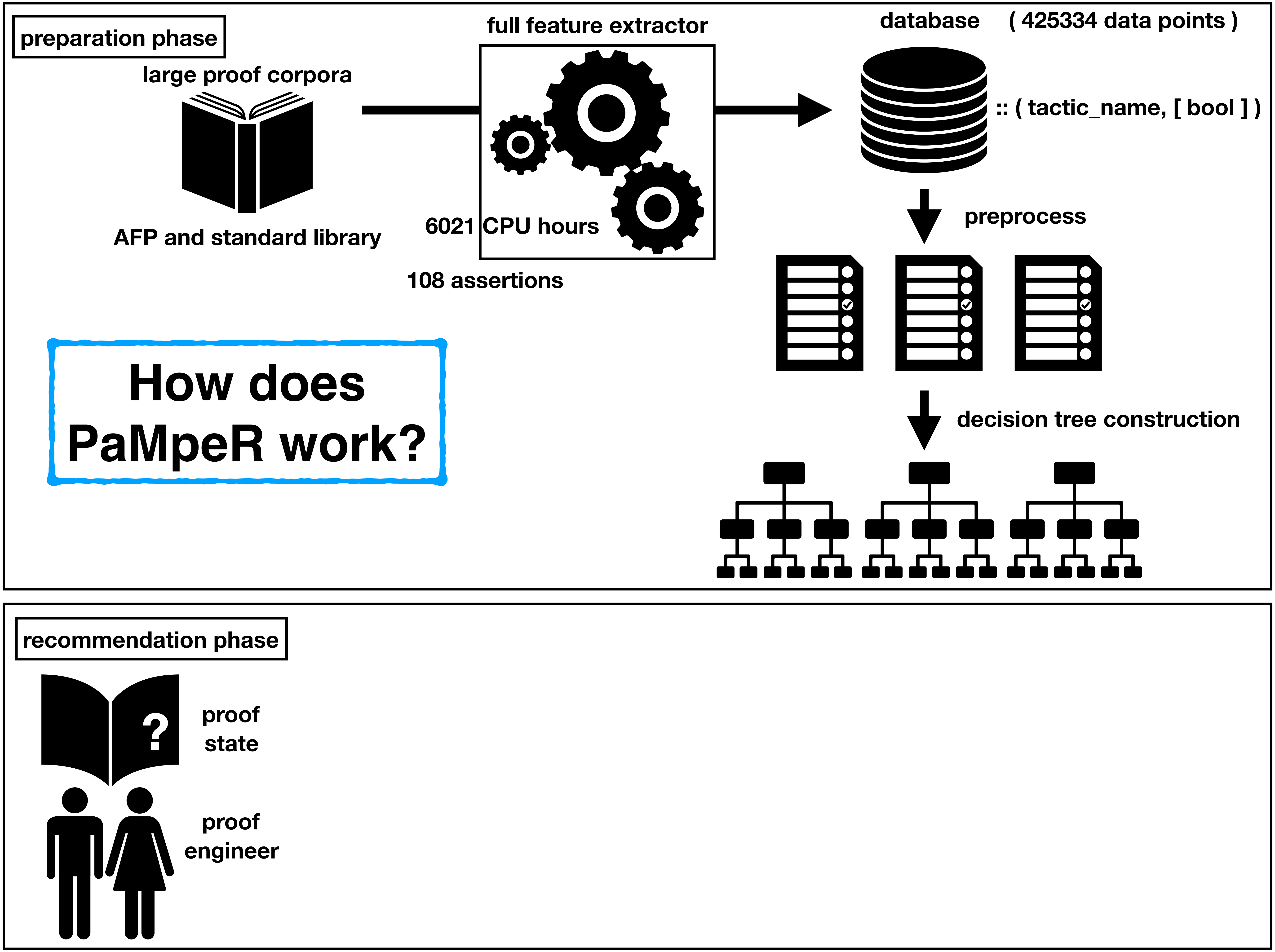
proof state

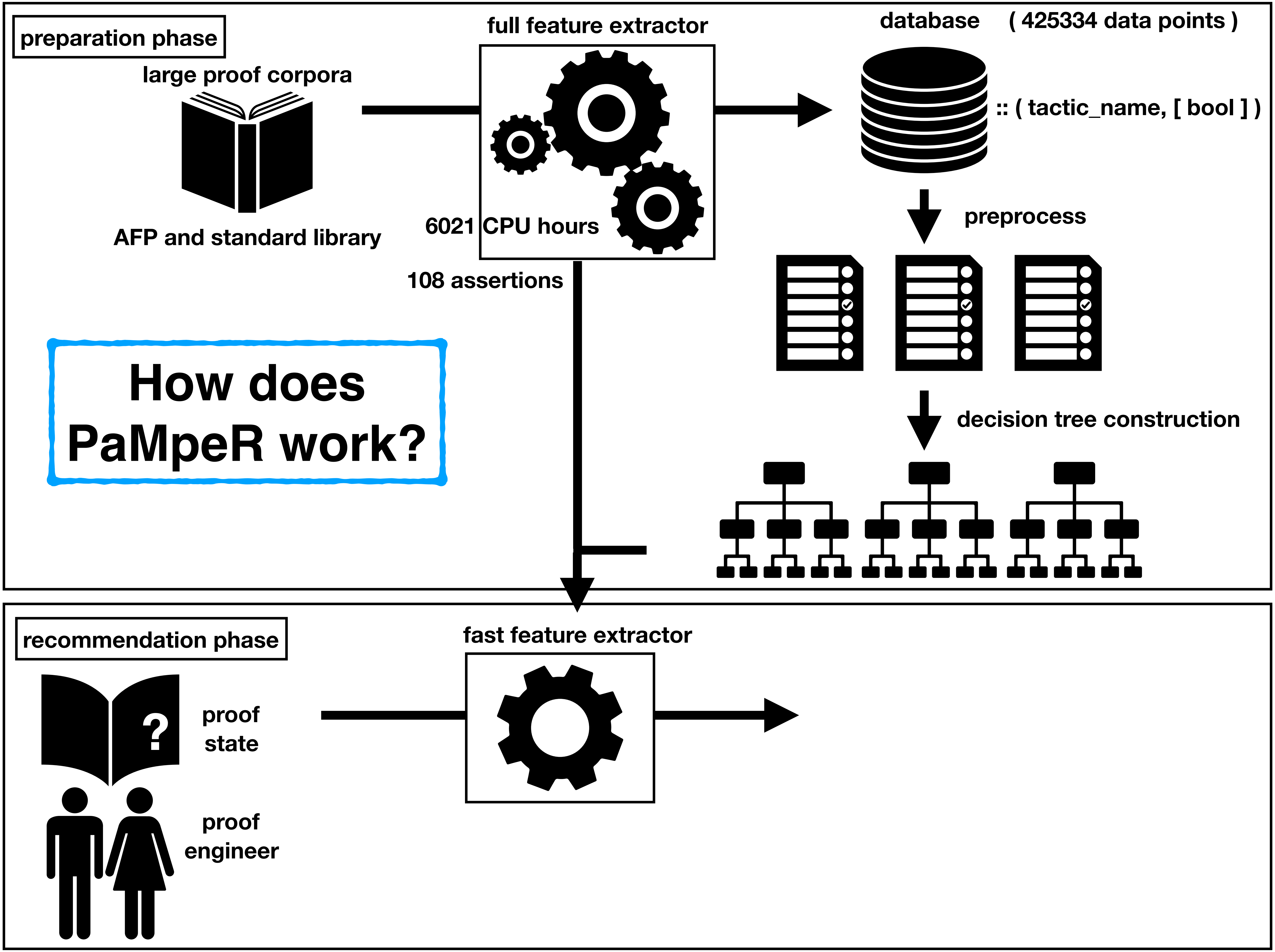


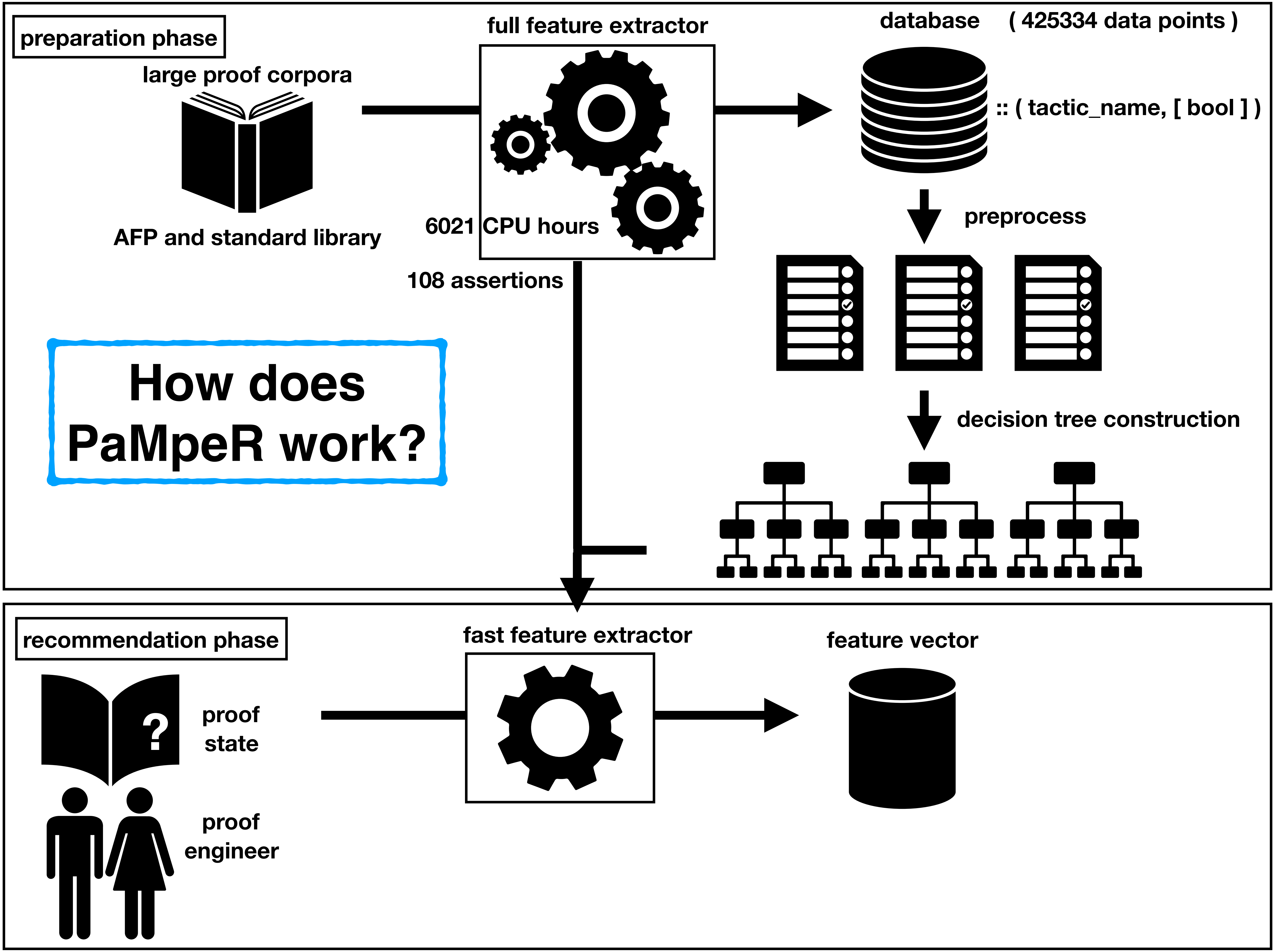
proof engineer

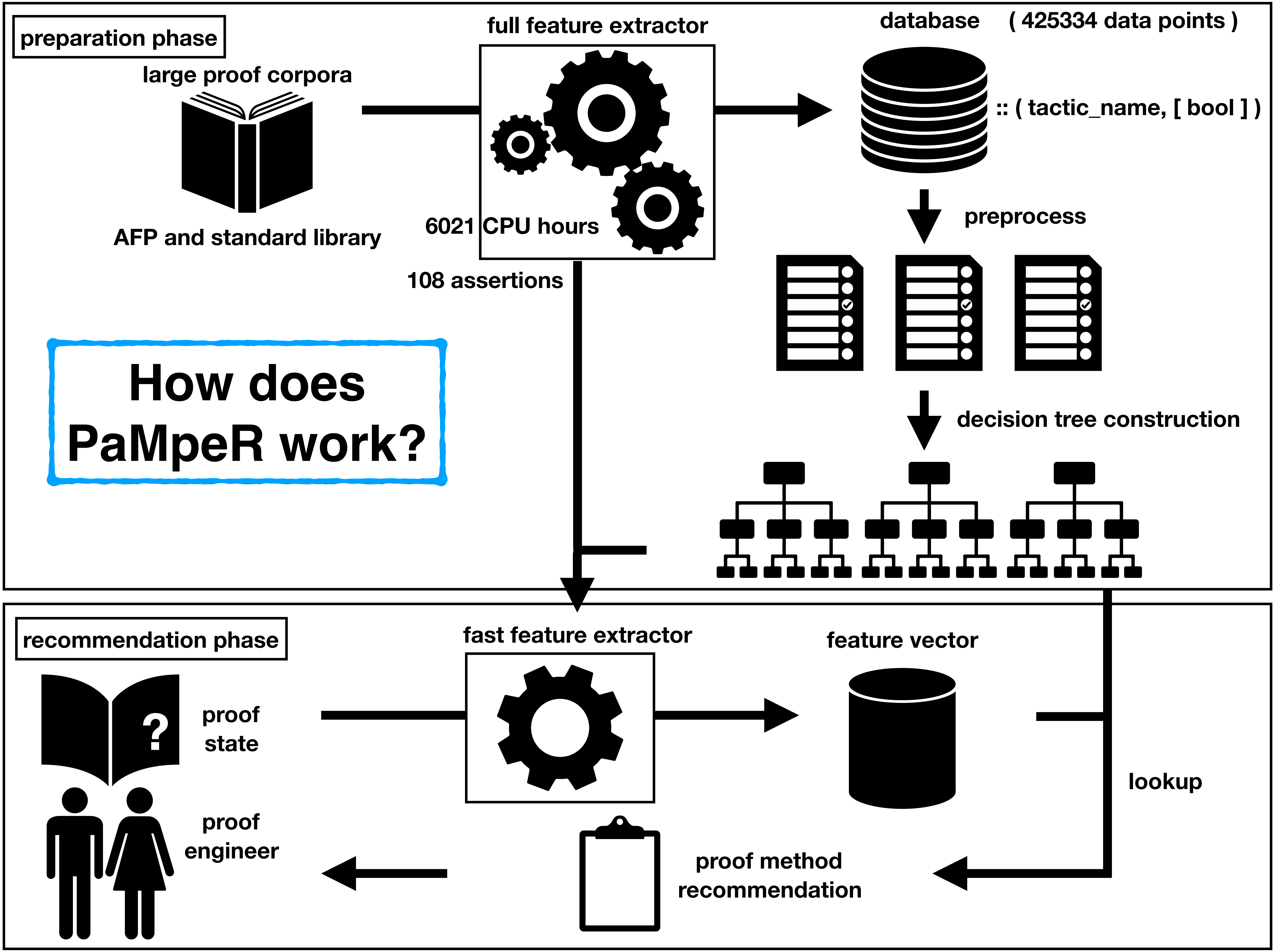

preparation phase
large proof corpora
AFP and standard library
full feature extractor
6021 CPU hours
108 assertions
How does PaMpeR work?
recommendation phase
proof state
proof engineer

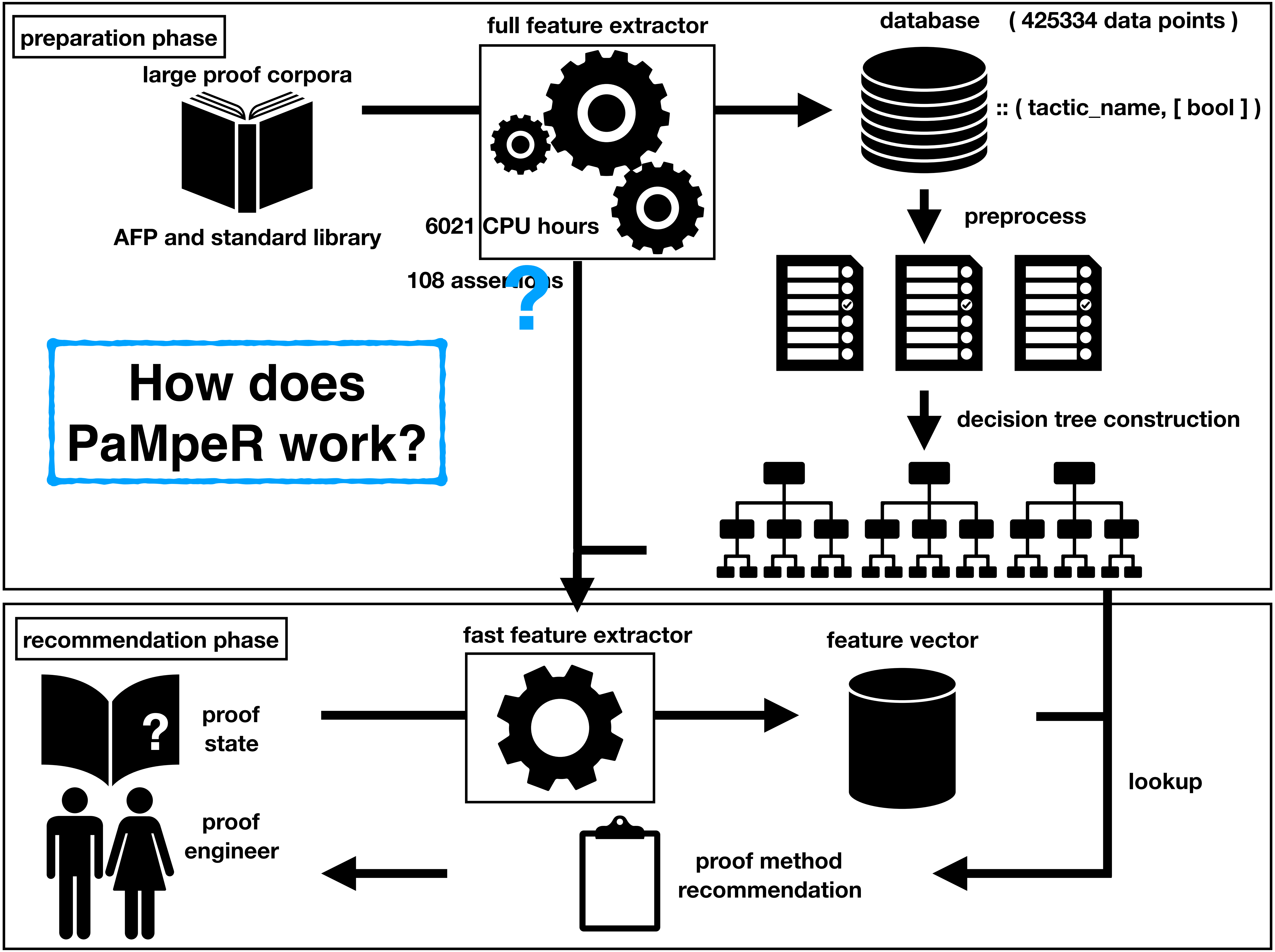

preparation phase
large proof corpora
AFP and standard library
full feature extractor
6021 CPU hours
108 assertions
database (425334 data points)
:: (tactic_name, [bool])
How does PaMpeR work?
recommendation phase
proof state
proof engineer


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
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

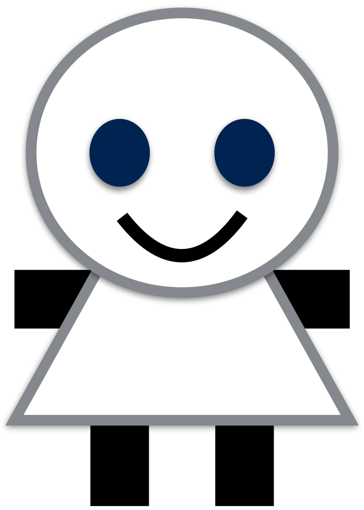

preparation phase
large proof corpora
AFP and standard library
full feature extractor
6021 CPU hours
108 assertions
database (425334 data points)
:: (tactic_name, [bool])
preprocess
decision tree construction
How does PaMpeR work?
recommendation phase
proof state
proof engineer
fast feature extractor
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?
recommendation phase
proof state
proof engineer
fast feature extractor
feature vector
lookup
proof method recommendation

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

AITP2018 review

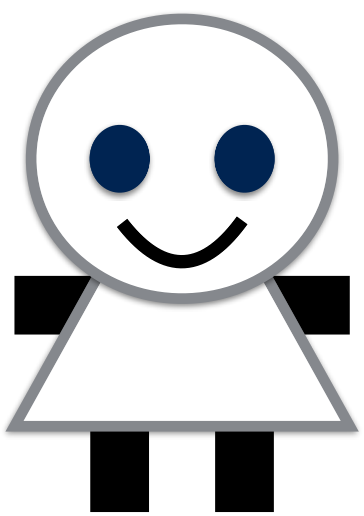


anonymous reviewer

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

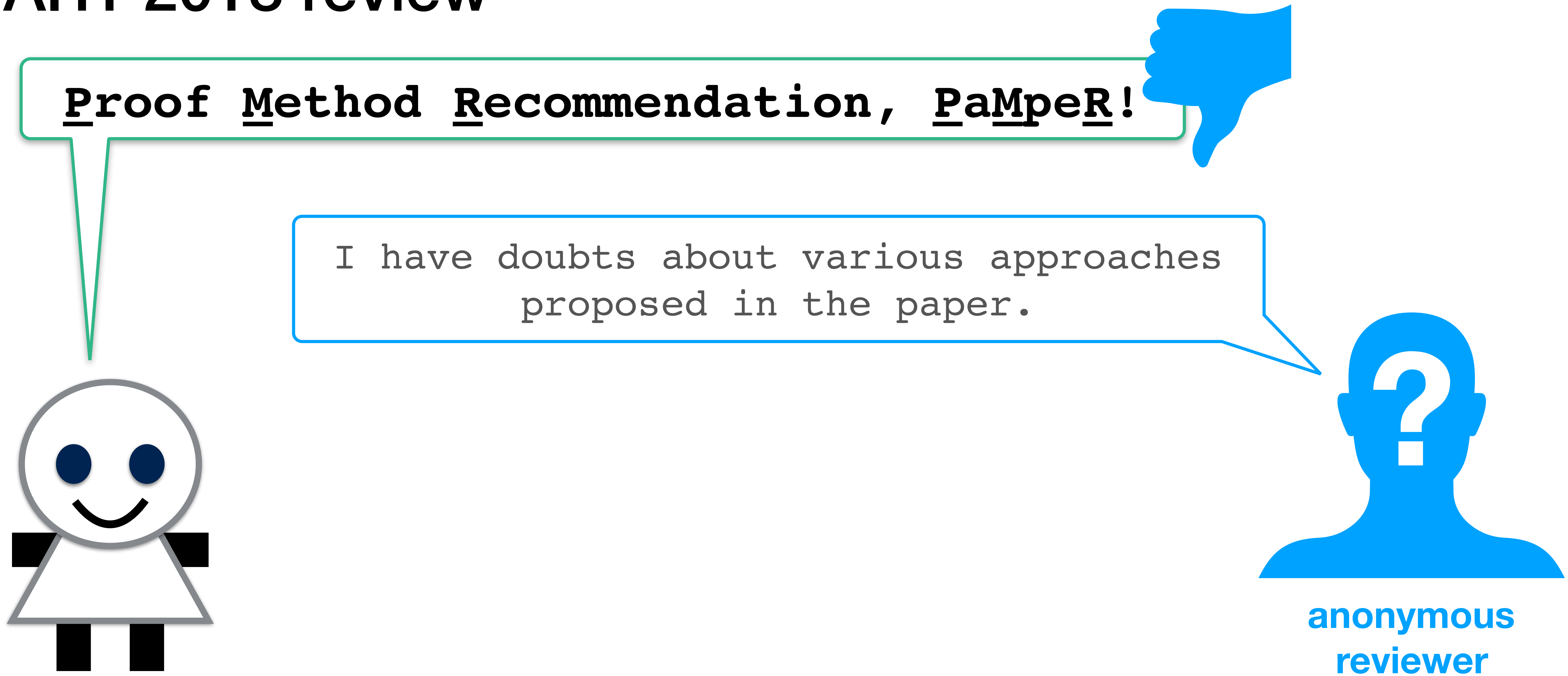
AITP2018 review

Proof Method Recommendation, PaMpeR!



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

AITP2018 review

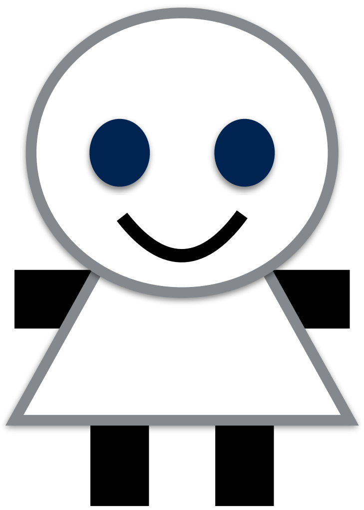


~~A~~ITP2018 review

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

Proof Method Recommendation, PaMpeR!

I have doubts about various approaches proposed in the paper.

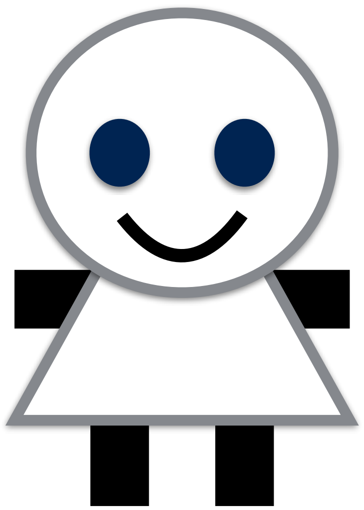


anonymous reviewer

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

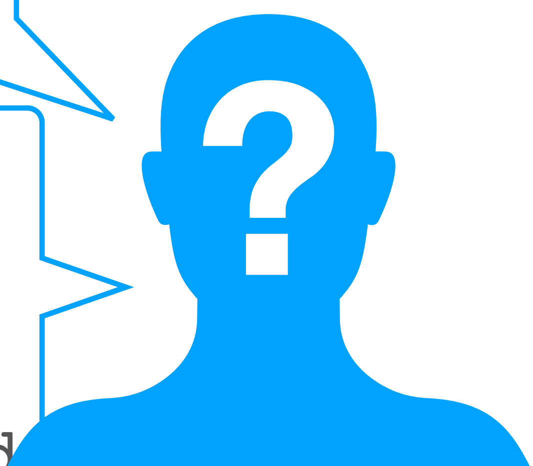
~~A~~ITP2018 review

Proof Method Recommendation, PaMpeR!



I have doubts about various approaches proposed in the paper.

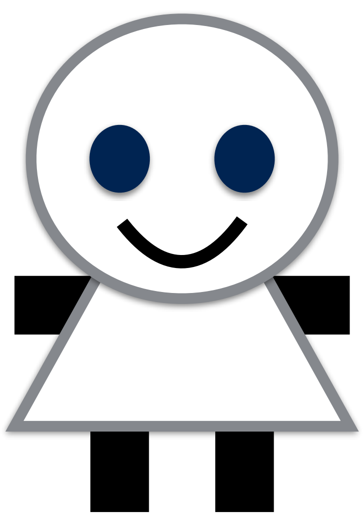
New users of Isabelle are facing many challenges from
- writing their first definitions,
- stating suitable theorem statements, and
- producing properly structured proofs.



~~A~~ITP2018 review

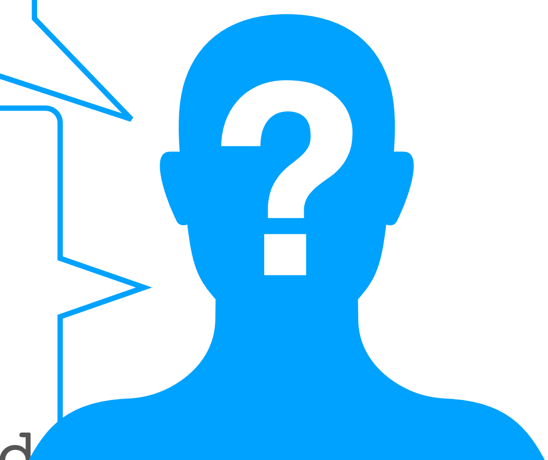
git clone https://github.com/data61/PSL

Proof Method Recommendation, PaMpeR!



I have doubts about various approaches proposed in the paper.

New users of Isabelle are facing many challenges from
- writing their first definitions,
- stating suitable theorem statements, and
- producing properly structured proofs.



anonymous reviewer

Proof methods are merely the bits at the bottom of that.

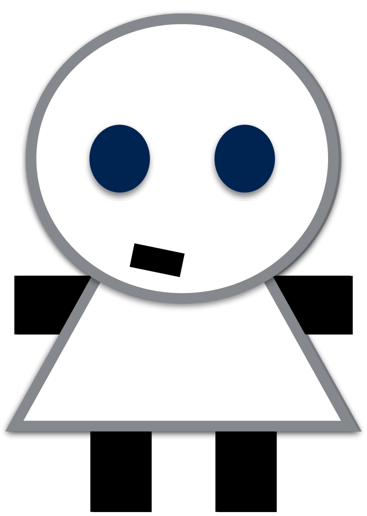
git clone https://github.com/data61/PSL

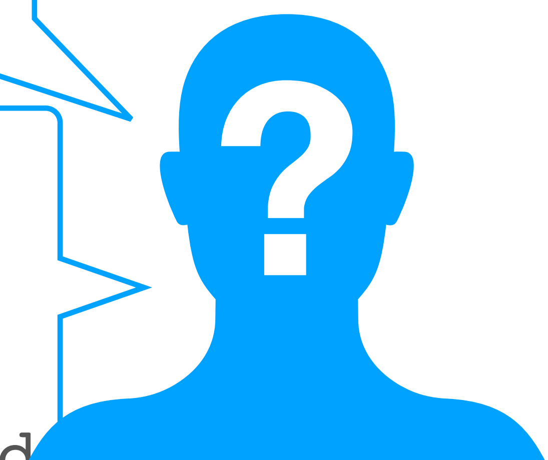
~~A~~ITP2018 review

Proof Method Recommendation, PaMpeR!

I have doubts about various approaches proposed in the paper.

New users of Isabelle are facing many challenges from
- writing their first definitions,
- stating suitable theorem statements, and
- producing properly structured proofs.





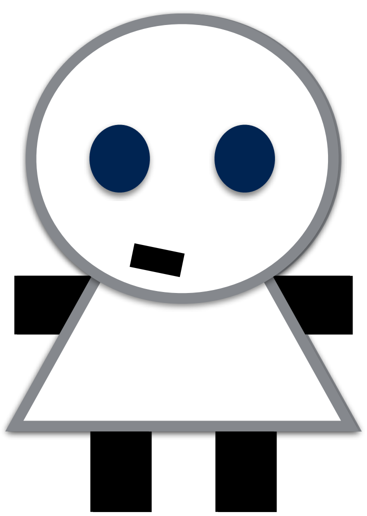
Proof methods are merely the bits at the bottom of that.

I was writing how to prove not how to specify!

~~A~~ITP2018 review

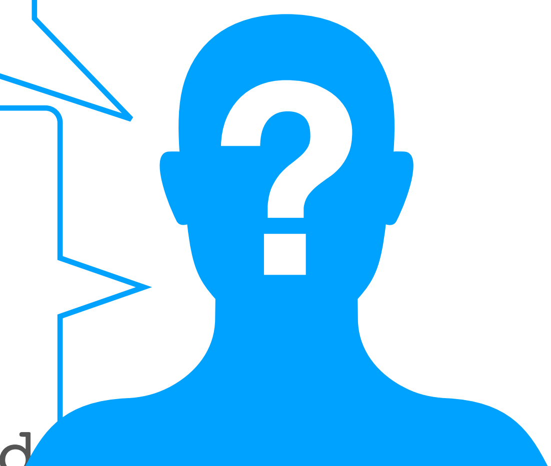
git clone https://github.com/data61/PSL

Proof Method Recommendation, PaMpeR!



I have doubts about various approaches proposed in the paper.

New users of Isabelle are facing many challenges from
- writing their first definitions,
- stating suitable theorem statements, and
- producing properly structured proofs.



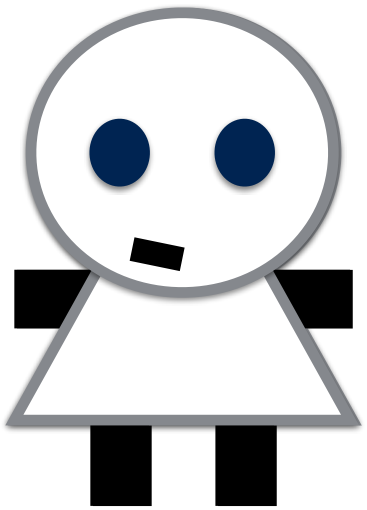
Proof methods are merely the bits at the bottom of that.

I was writing how to prove not how to specify!

ITP2018 review

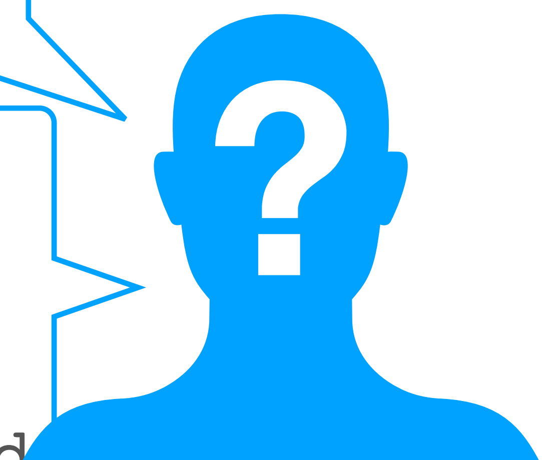
git clone https://github.com/data61/PSL

Proof Method Recommendation, PaMpeR!



I have doubts about various approaches proposed in the paper.

New users of Isabelle are facing many challenges from
- writing their first definitions,
- stating suitable theorem statements, and
- producing properly structured proofs.



Proof methods are merely the bits at the bottom of that.

I was writing how to prove not how to specify!

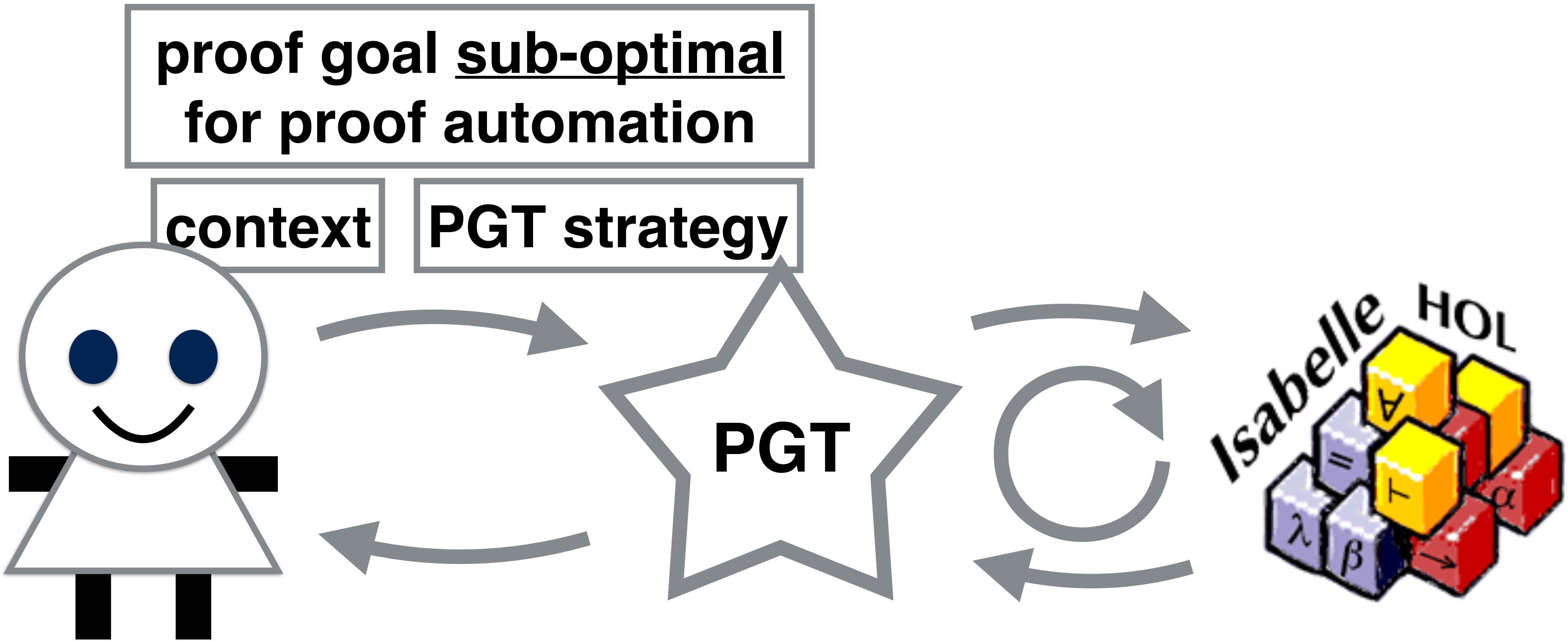
Proof Goal Transformer, PGT!

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

PSL with PGT

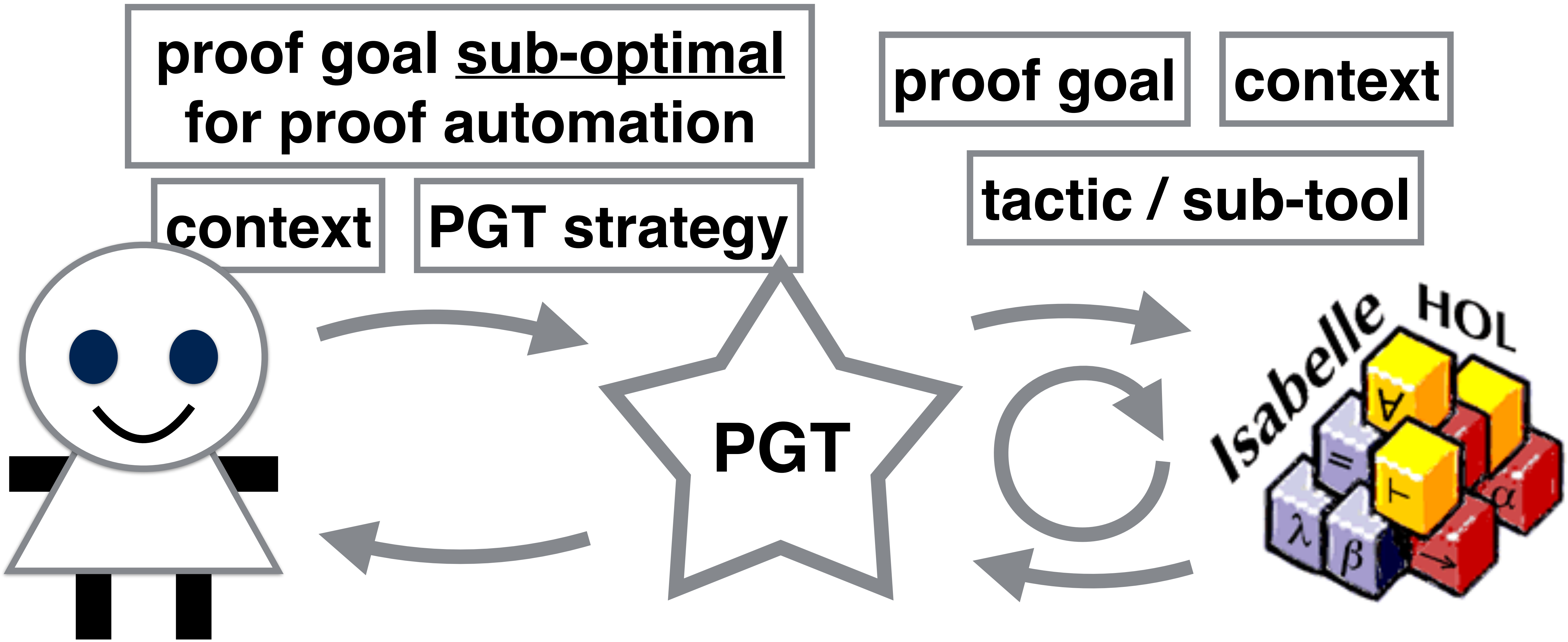
git clone https://github.com/data61/PSL

PSL with PGT



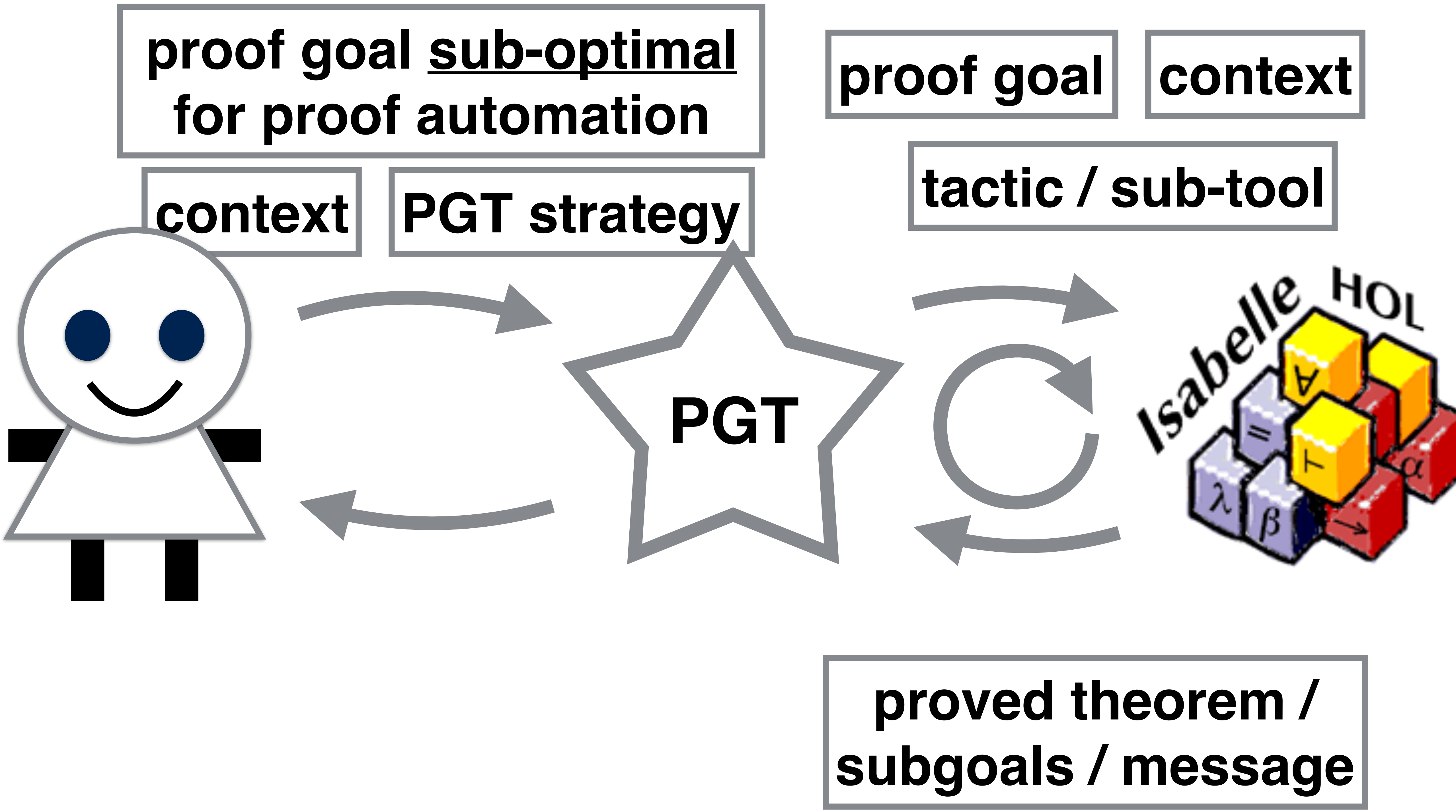
git clone https://github.com/data61/PSL

PSL with PGT



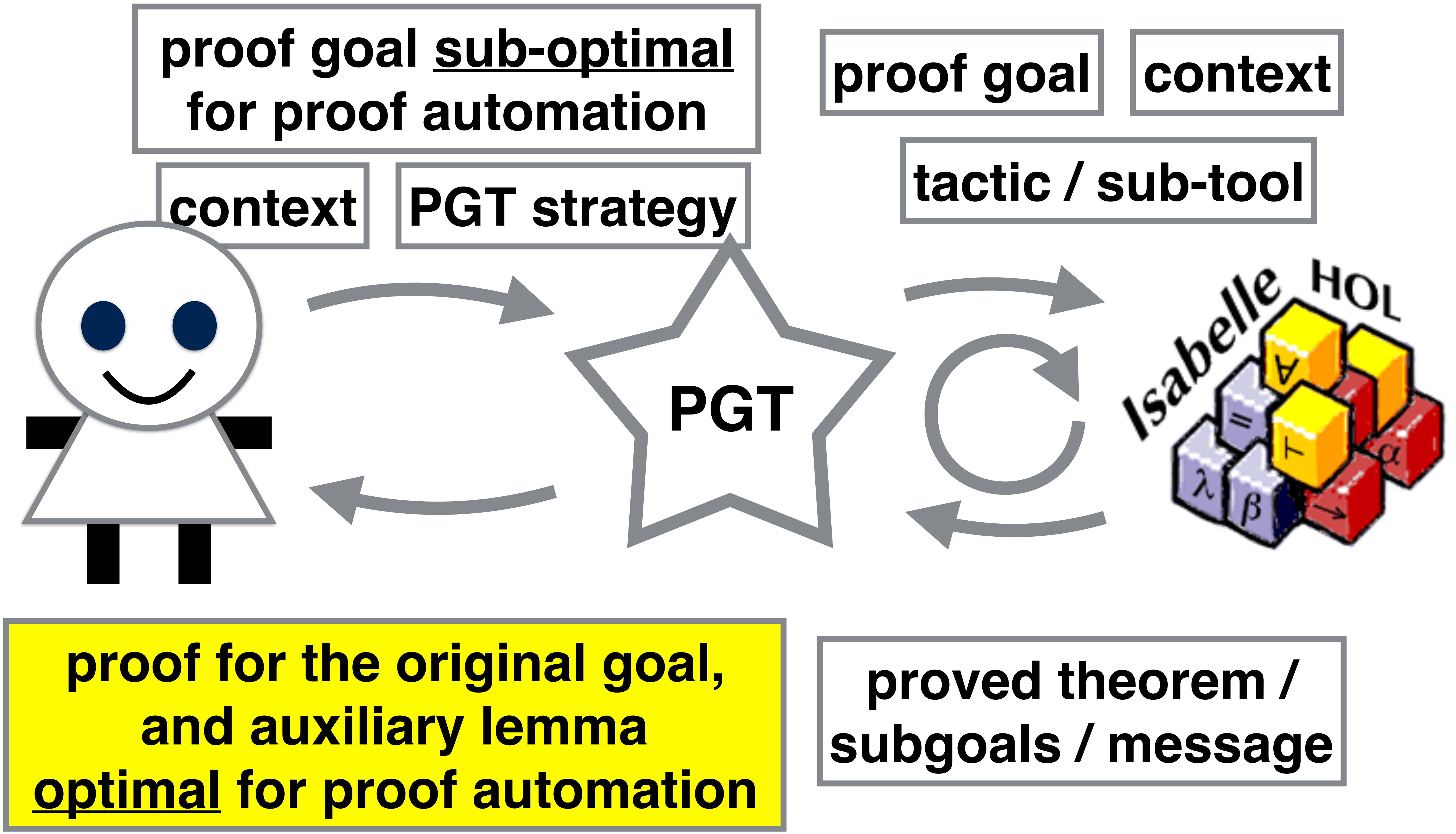
git clone https://github.com/data61/PSL

PSL with PGT

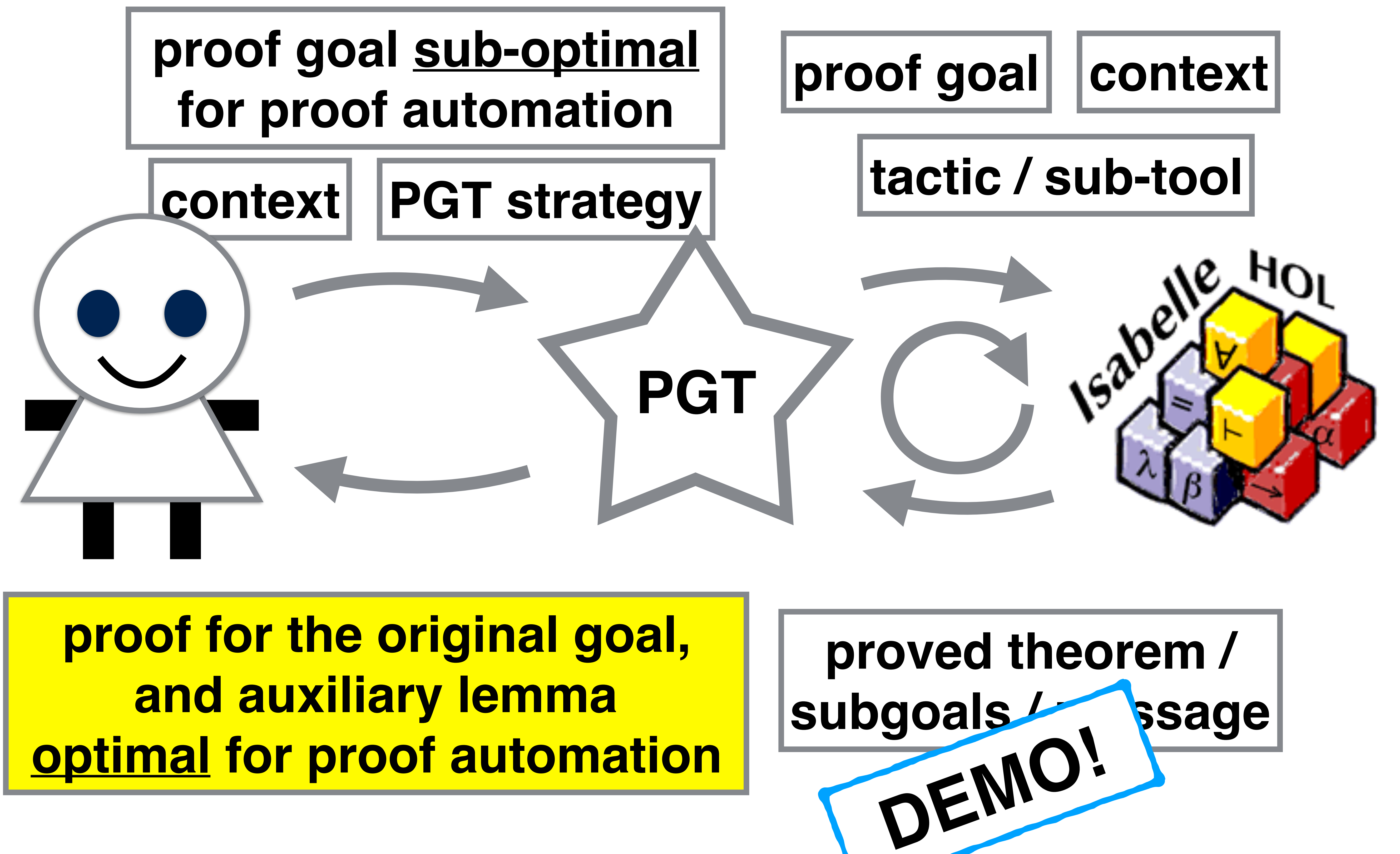


git clone https://github.com/data61/PSL

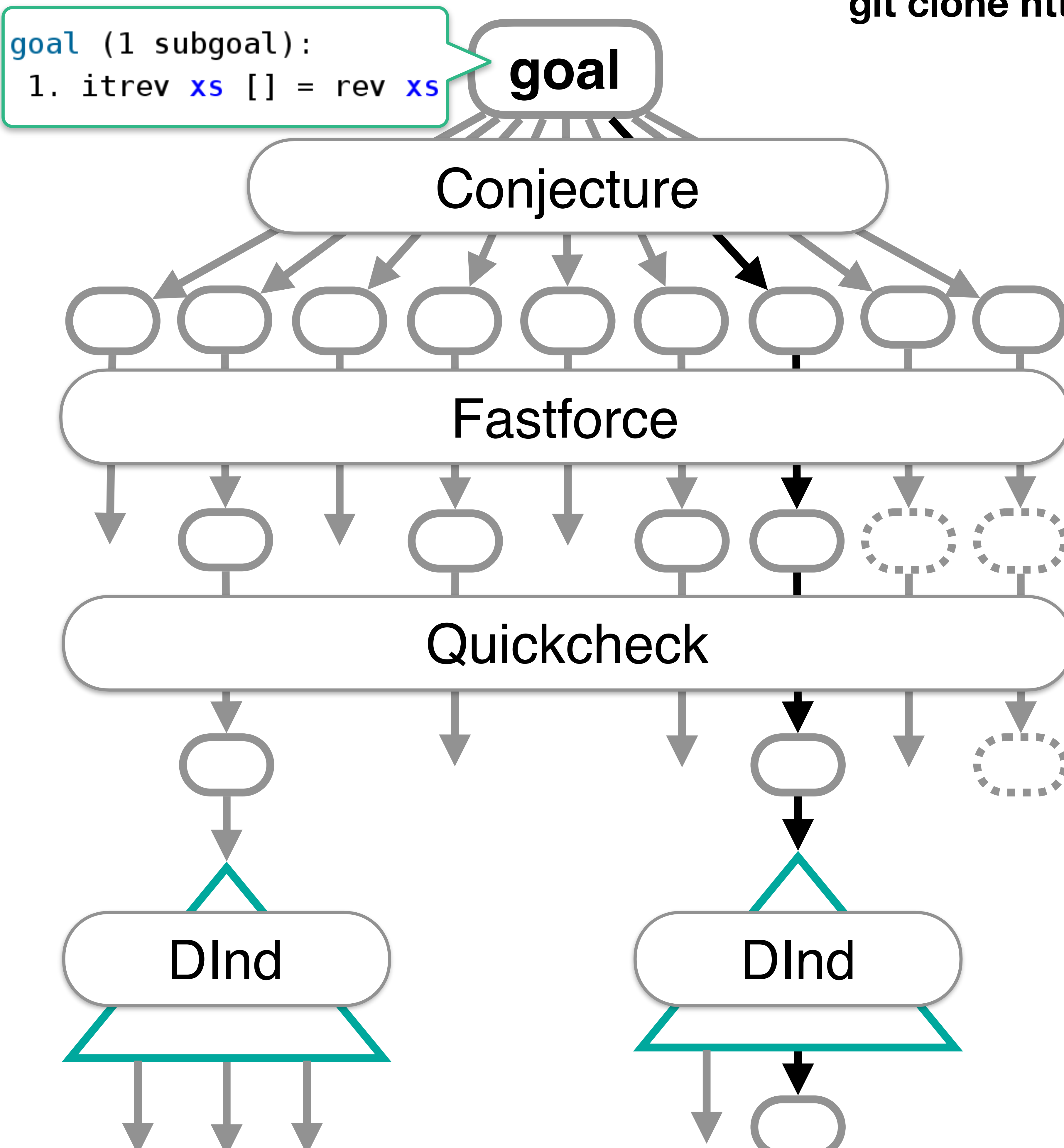
PSL with PGT

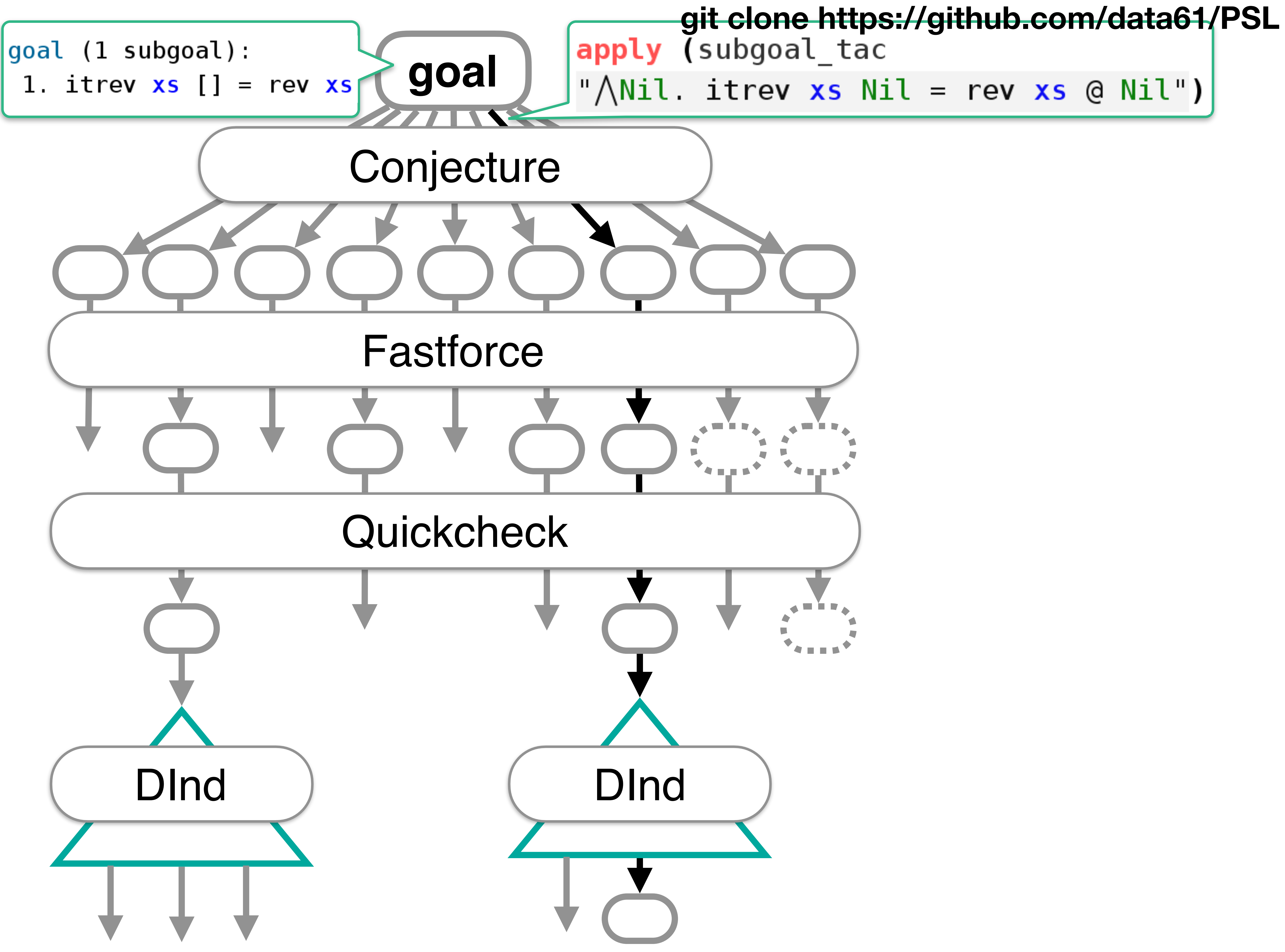


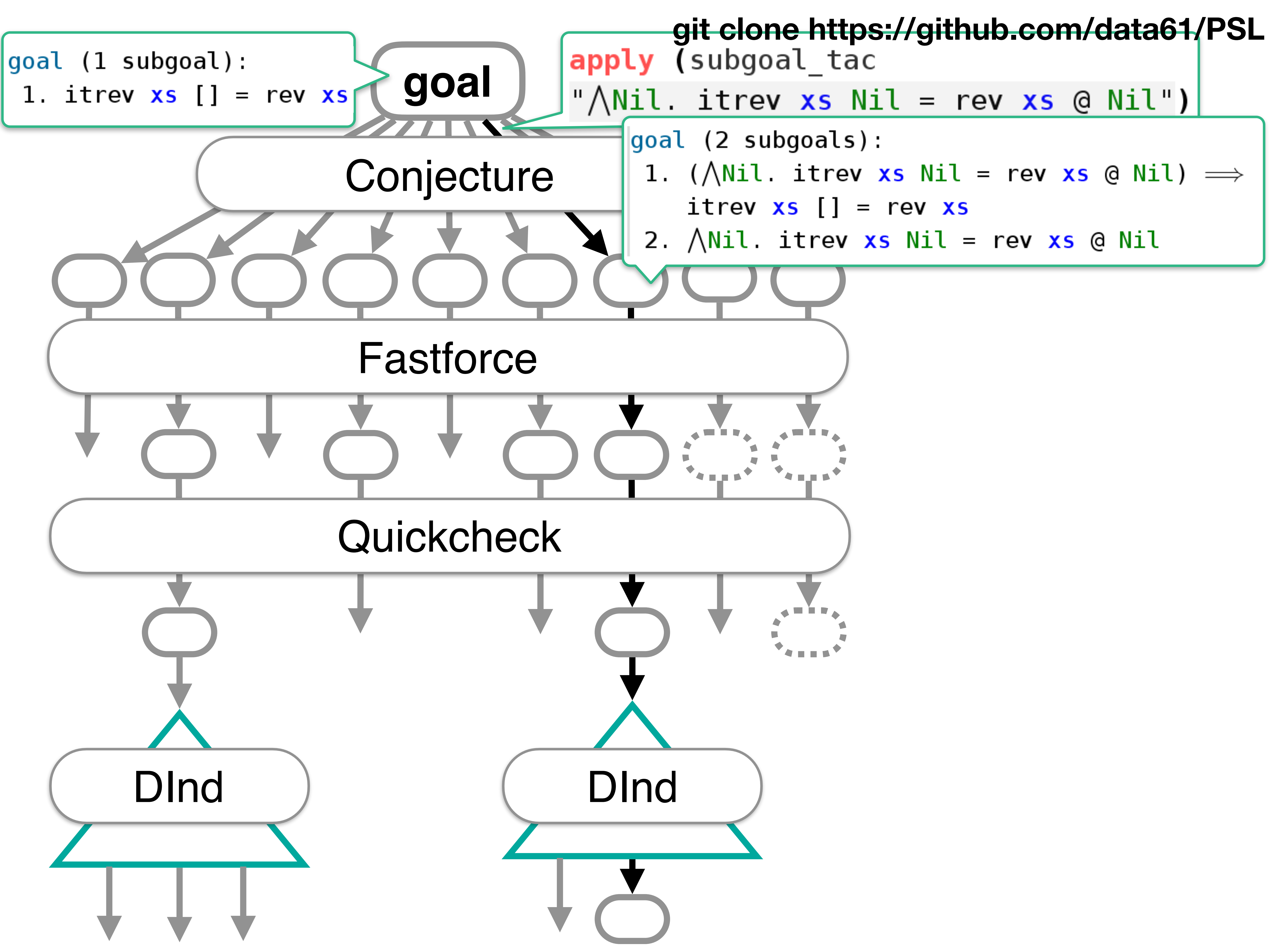
PSL with PGT


proof goal sub-optimal
for proof automation
context
PGT strategy
PGT
proof goal
context
tactic / sub-tool
Isabelle
HOL
proof for the original goal,
and auxiliary lemma
optimal for proof automation
proved theorem /
subgoals /
ssage
DEMO!

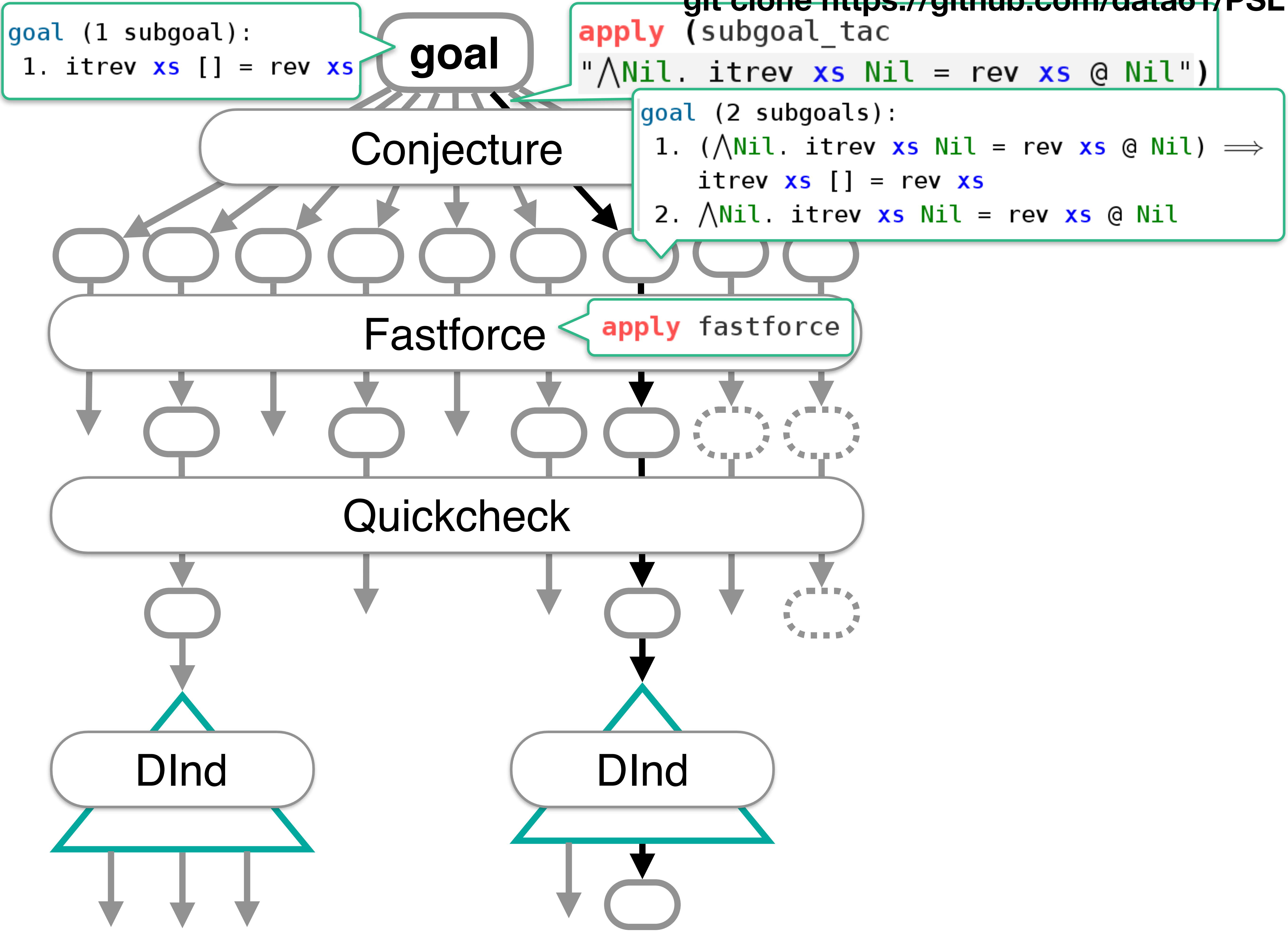
git clone https://github.com/data61/PSL

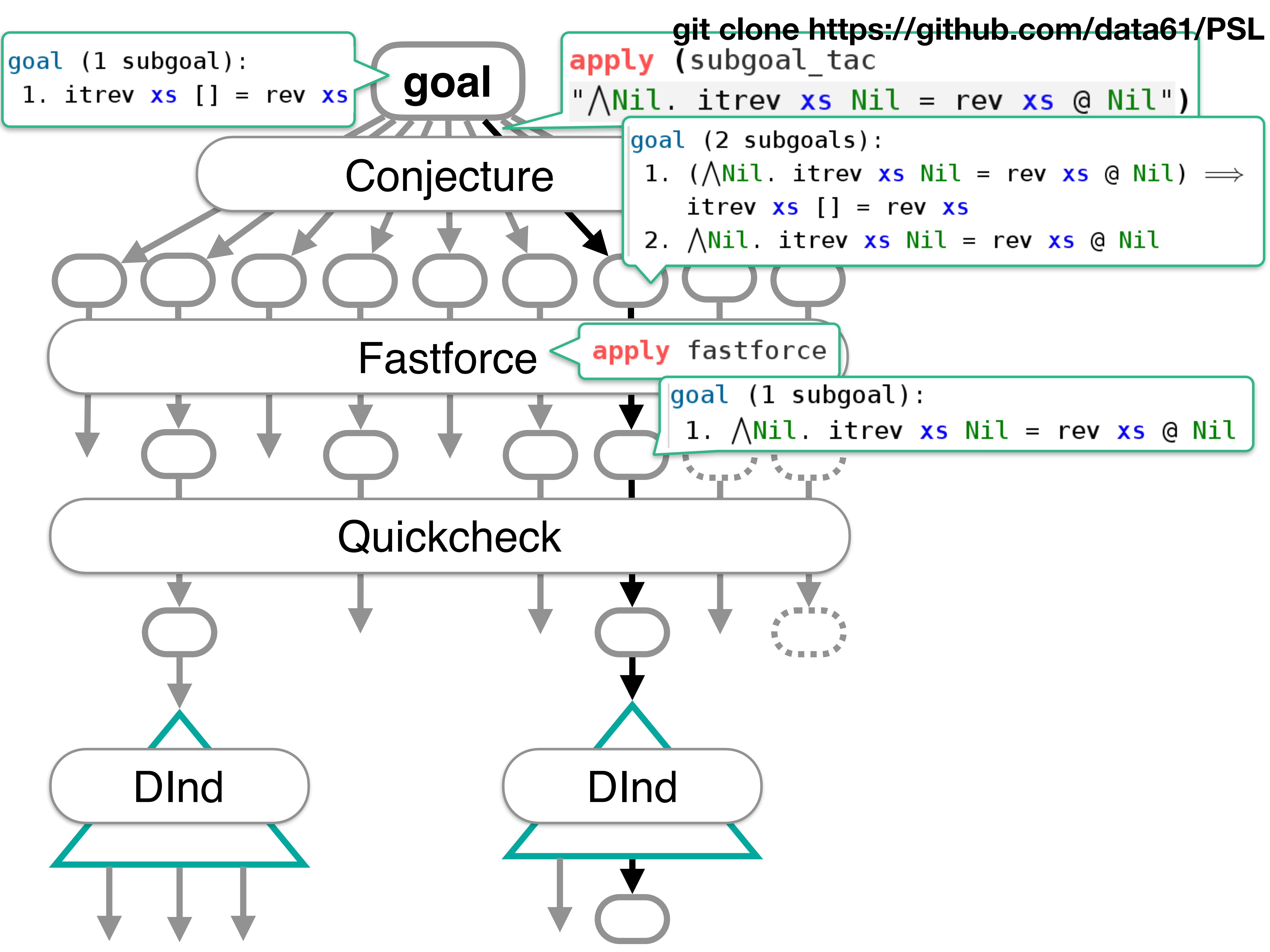


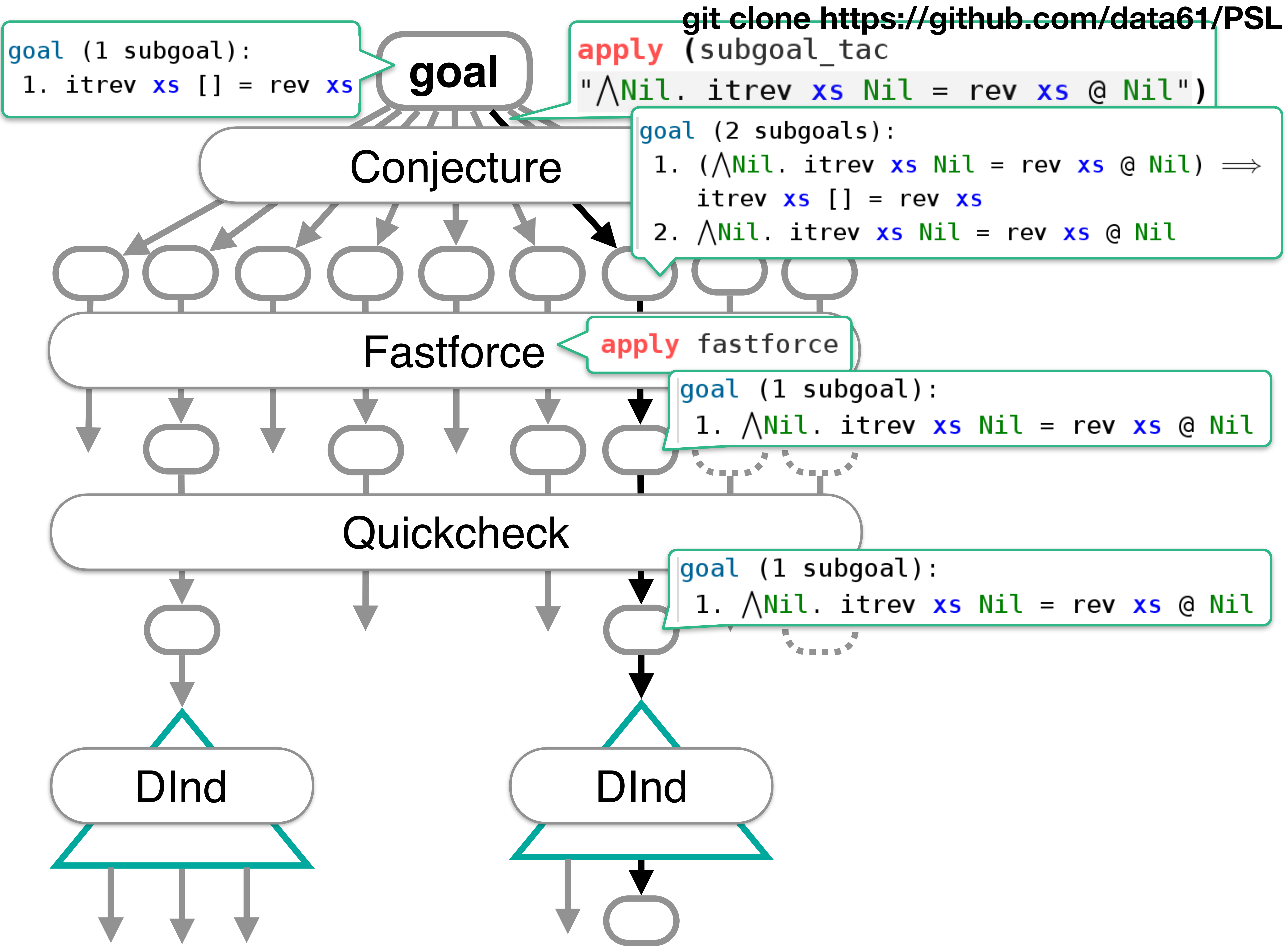

git clone https://github.com/data61/PSL
goal (1 subgoal):
1. itrev xs [] = rev xs
goal
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
Conjecture
Fastforce
Quickcheck
DInd
DInd

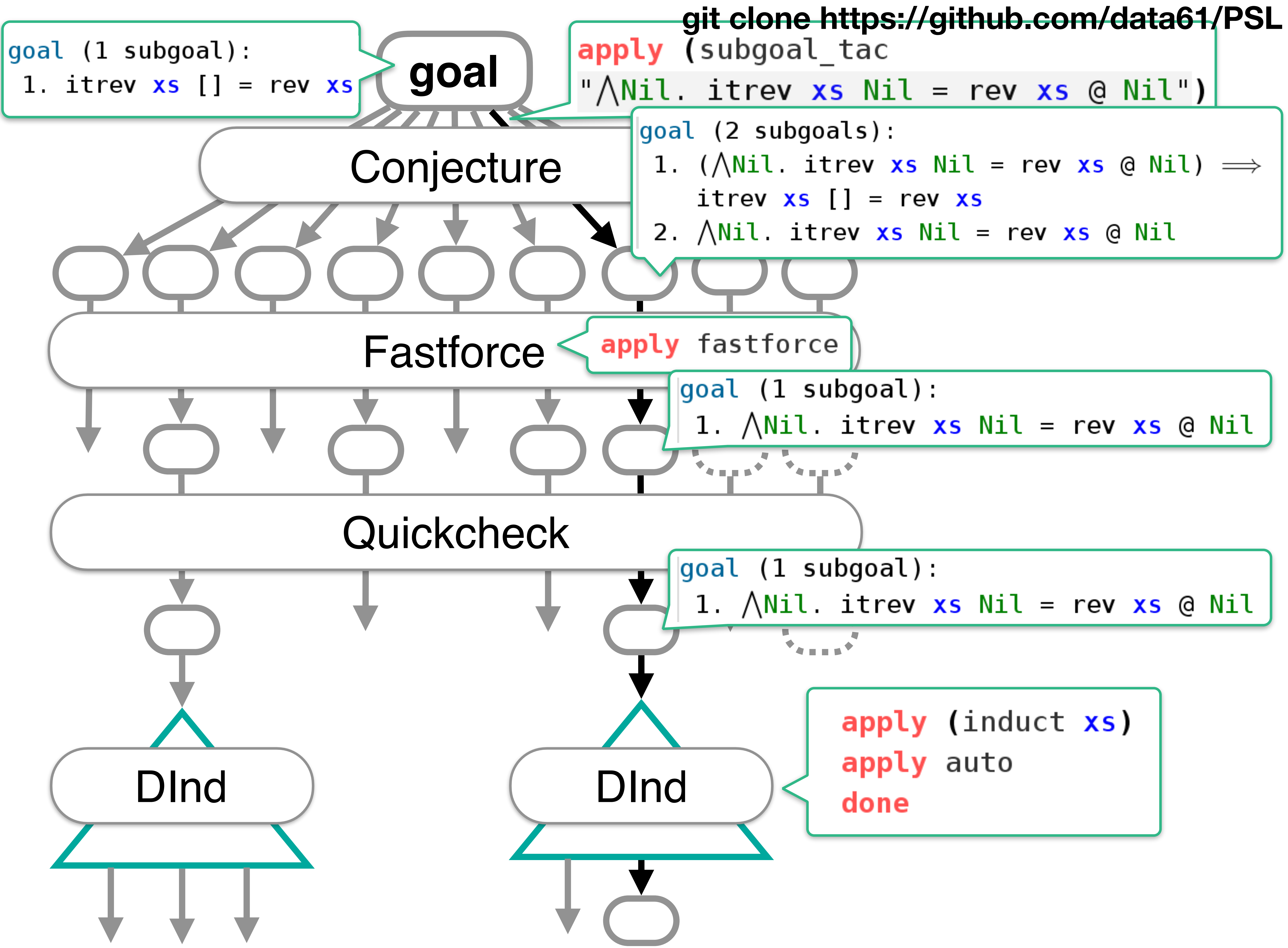

git clone https://github.com/data61/PSL
goal (1 subgoal):
1. itrev xs [] = rev xs
goal
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
goal (2 subgoals):
1. (⋀Nil. itrev xs Nil = rev xs @ Nil) ⟹
itrev xs [] = rev xs
2. ⋀Nil. itrev xs Nil = rev xs @ Nil
Conjecture
Fastforce
Quickcheck
DInd
DInd

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



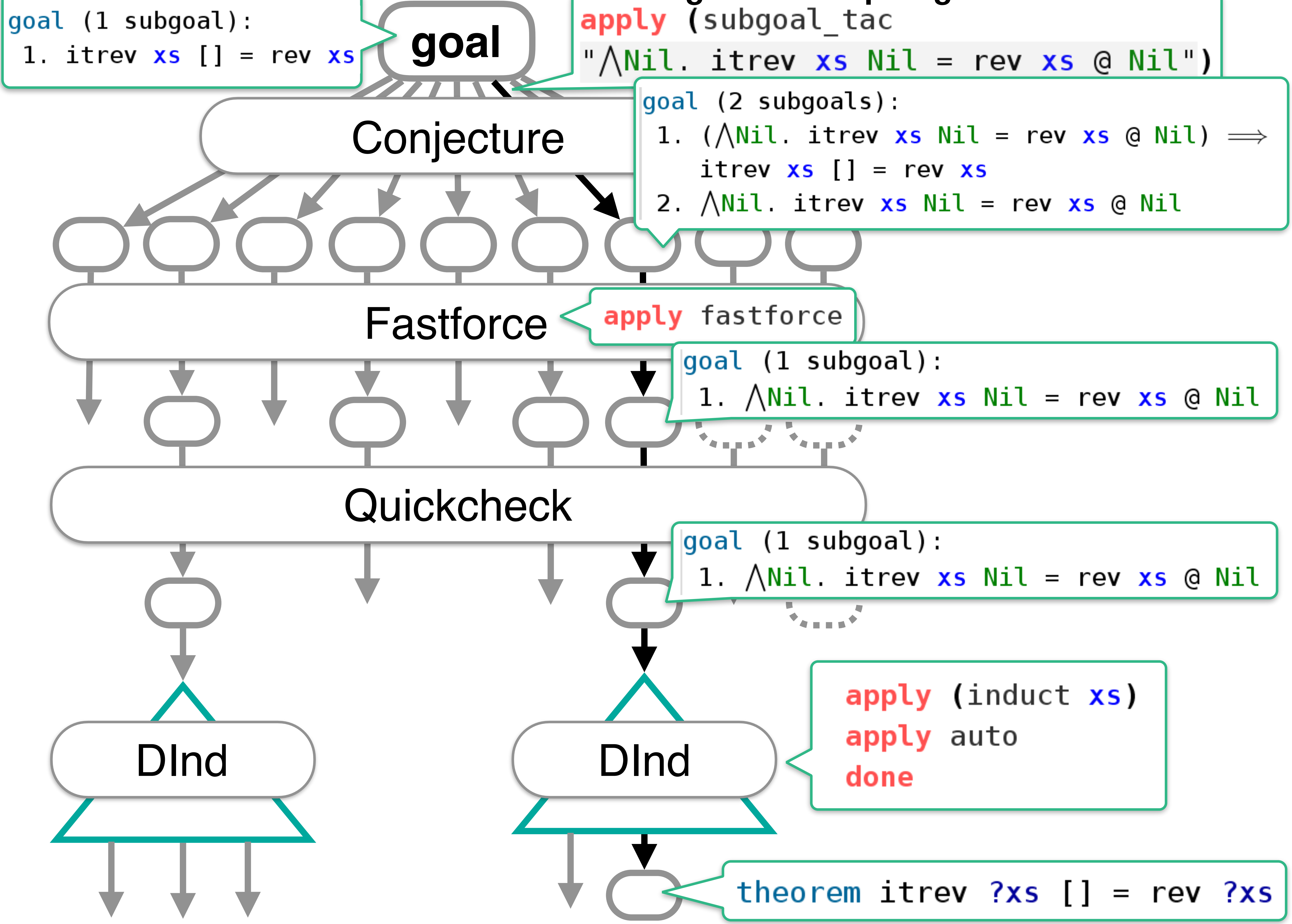

git clone https://github.com/data61/PSL
goal (1 subgoal):
1. itrev xs [] = rev xs
goal
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
goal (2 subgoals):
1. (⋀Nil. itrev xs Nil = rev xs @ Nil) ⟹
itrev xs [] = rev xs
2. ⋀Nil. itrev xs Nil = rev xs @ Nil
Conjecture
Fastforce
apply fastforce
goal (1 subgoal):
1. ⋀Nil. itrev xs Nil = rev xs @ Nil
Quickcheck
DInd
DInd


git clone https://github.com/data61/PSL
goal
goal (1 subgoal):
1. itrev xs [] = rev xs
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
Conjecture
goal (2 subgoals):
1. (⋀Nil. itrev xs Nil = rev xs @ Nil) ⟹
itrev xs [] = rev xs
2. ⋀Nil. itrev xs Nil = rev xs @ Nil
Fastforce
apply fastforce
goal (1 subgoal):
1. ⋀Nil. itrev xs Nil = rev xs @ Nil
Quickcheck
goal (1 subgoal):
1. ⋀Nil. itrev xs Nil = rev xs @ Nil
DInd
DInd

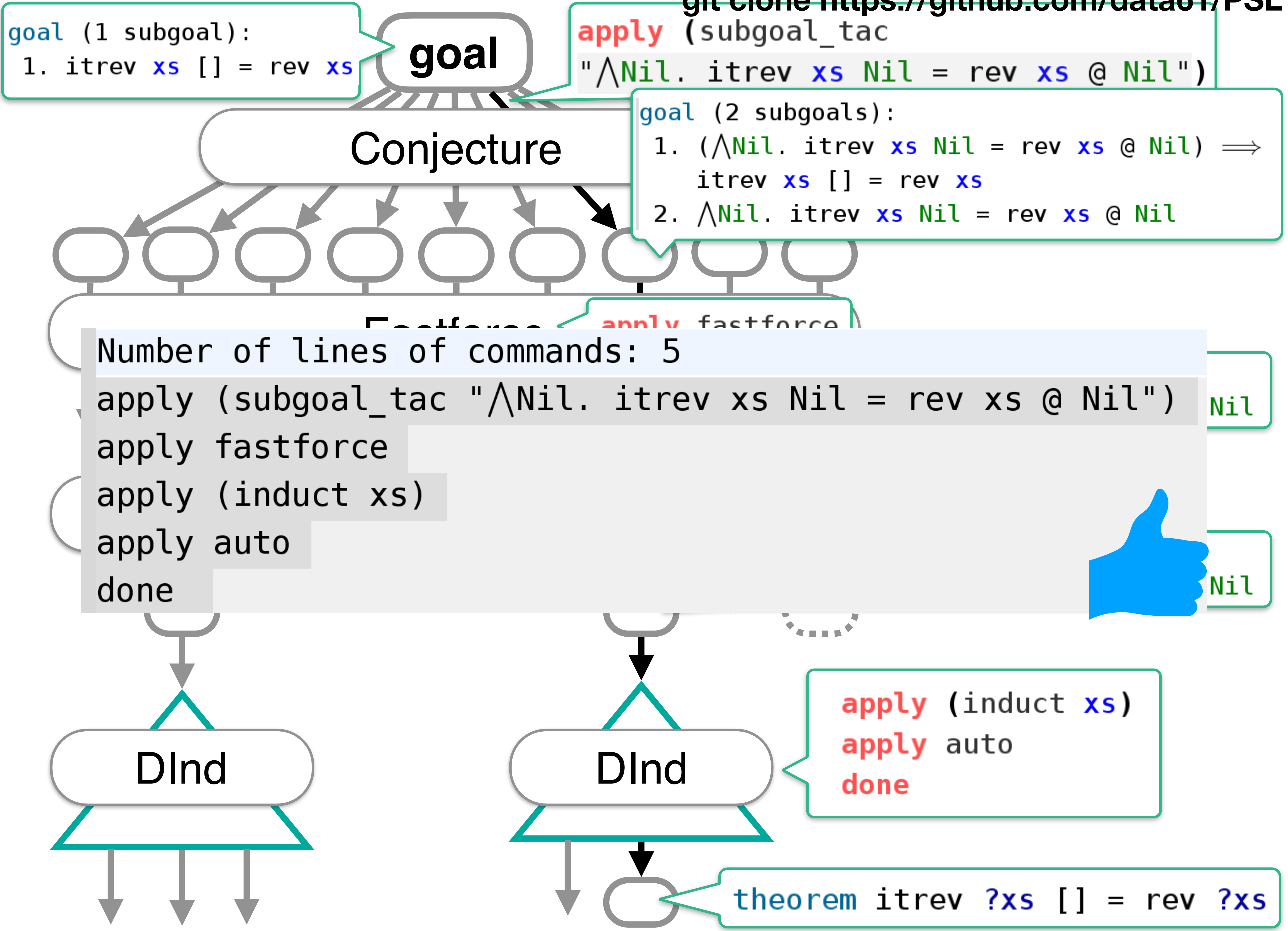


git clone https://github.com/data61/PSL
goal (1 subgoal):
1. itrev xs [] = rev xs
goal
apply (subgoal_tac
"⋀Nil. itrev xs Nil = rev xs @ Nil")
goal (2 subgoals):
1. (⋀Nil. itrev xs Nil = rev xs @ Nil) ⟹
itrev xs [] = rev xs
2. ⋀Nil. itrev xs Nil = rev xs @ Nil
Conjecture
Fastforce
apply fastforce
goal (1 subgoal):
1. ⋀Nil. itrev xs Nil = rev xs @ Nil
Quickcheck
goal (1 subgoal):
1. ⋀Nil. itrev xs Nil = rev xs @ Nil
DInd
DInd
apply (induct xs)
apply auto
done

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



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



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

Success story

PSL can find how to apply induction for easy problems.

Success story

PSL can find how to apply induction for easy problems.

PaMpeR recommends which proof methods to use.

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

Success story

PSL can find how to apply induction for easy problems.

PaMpeR recommends which proof methods to use.

PGT produces useful auxiliary lemmas.

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

Success story

PSL can find how to apply induction for easy problems.

CADE2017

PaMpeR recommends which proof methods to use.

PGT produces useful auxiliary lemmas.

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

Success story

PSL can find how to apply induction for easy problems.

PaMpeR recommends which proof methods to use.

PGT produces useful auxiliary lemmas.

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

Success story

PSL can find how to apply induction for easy problems.

CADE2017

PaMpeR recommends which proof methods to use.

ASE2018

PGT produces useful auxiliary lemmas.

CICM2018 (best system award)

Too good to be true?

PSL can find how to apply induction for easy problems.

PaMpeR recommends which proof methods to use.

PGT produces useful auxiliary lemmas.

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

Too good to be true?

PSL can find how to apply induction for easy problems.

only if PSL completes a proof search

PaMpeR recommends which proof methods to use.

PGT produces useful auxiliary lemmas.

only if PSL with PGT completes a proof search

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

Too good to be true?

PSL can find how to apply induction for easy problems.

only if PSL completes a proof search

PaMpeR recommends which proof methods to use.

but PaMpeR does not recommend arguments for proof methods

PGT produces useful auxiliary lemmas.

only if PSL with PGT completes a proof search

Too good to be true?

PSL can find how to apply induction for easy problems

only if PSL completes a proof search

PaMpeR recommends which proof methods to use.

but PaMpeR does not recommend arguments for proof methods

PGT produces useful auxiliary lemmas.

only if PSL with PGT compl

proof search

Recommend how to apply induction without completing a proof.

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

Too good to be true?

PSL can find how to apply induction for easy problems

only if PSL completes a proof search

PaMpeR recommends which proof methods to use.

but PaMpeR does not recommend arguments for proof methods

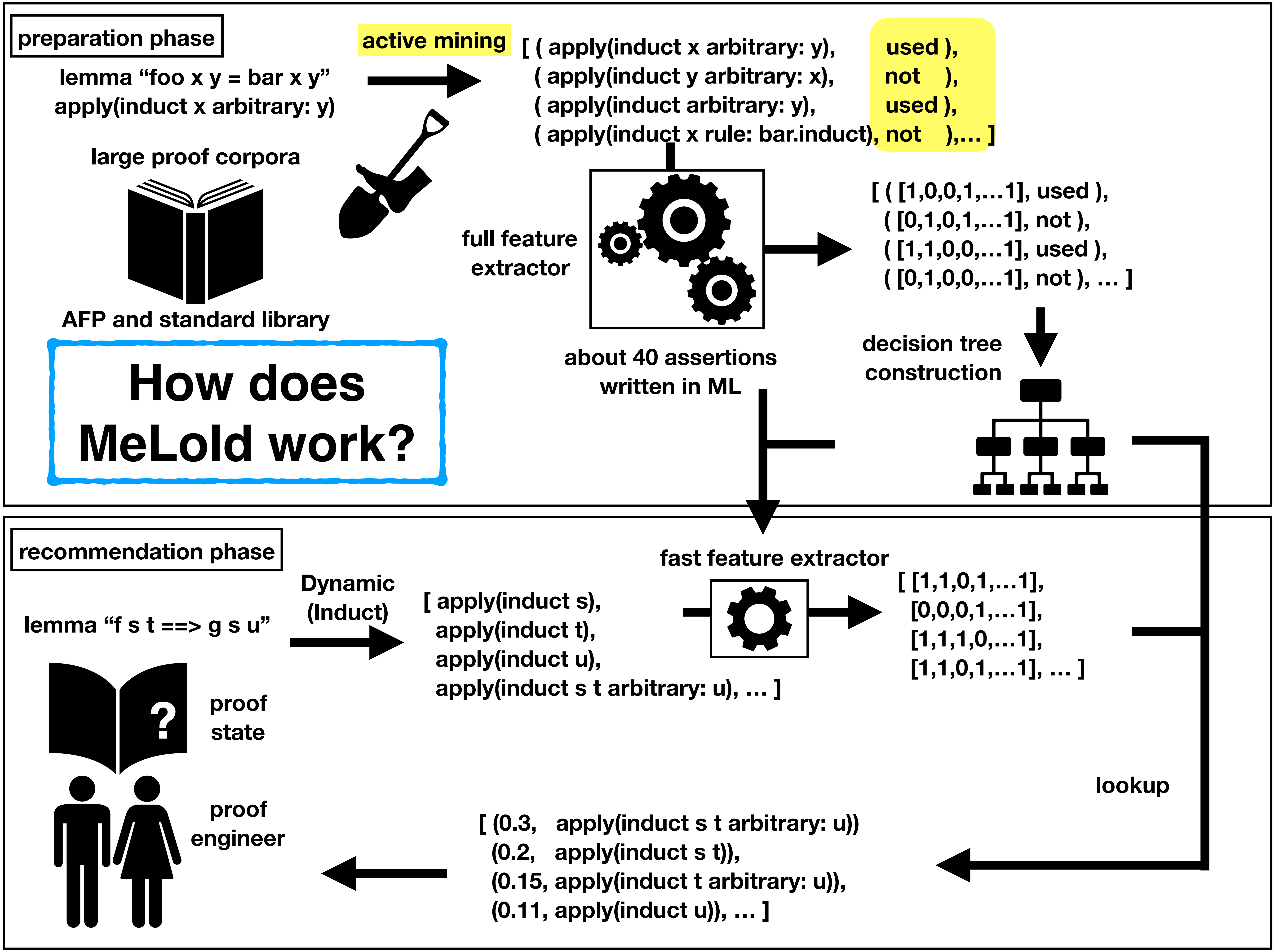
PGT produces useful auxiliary lemmas.

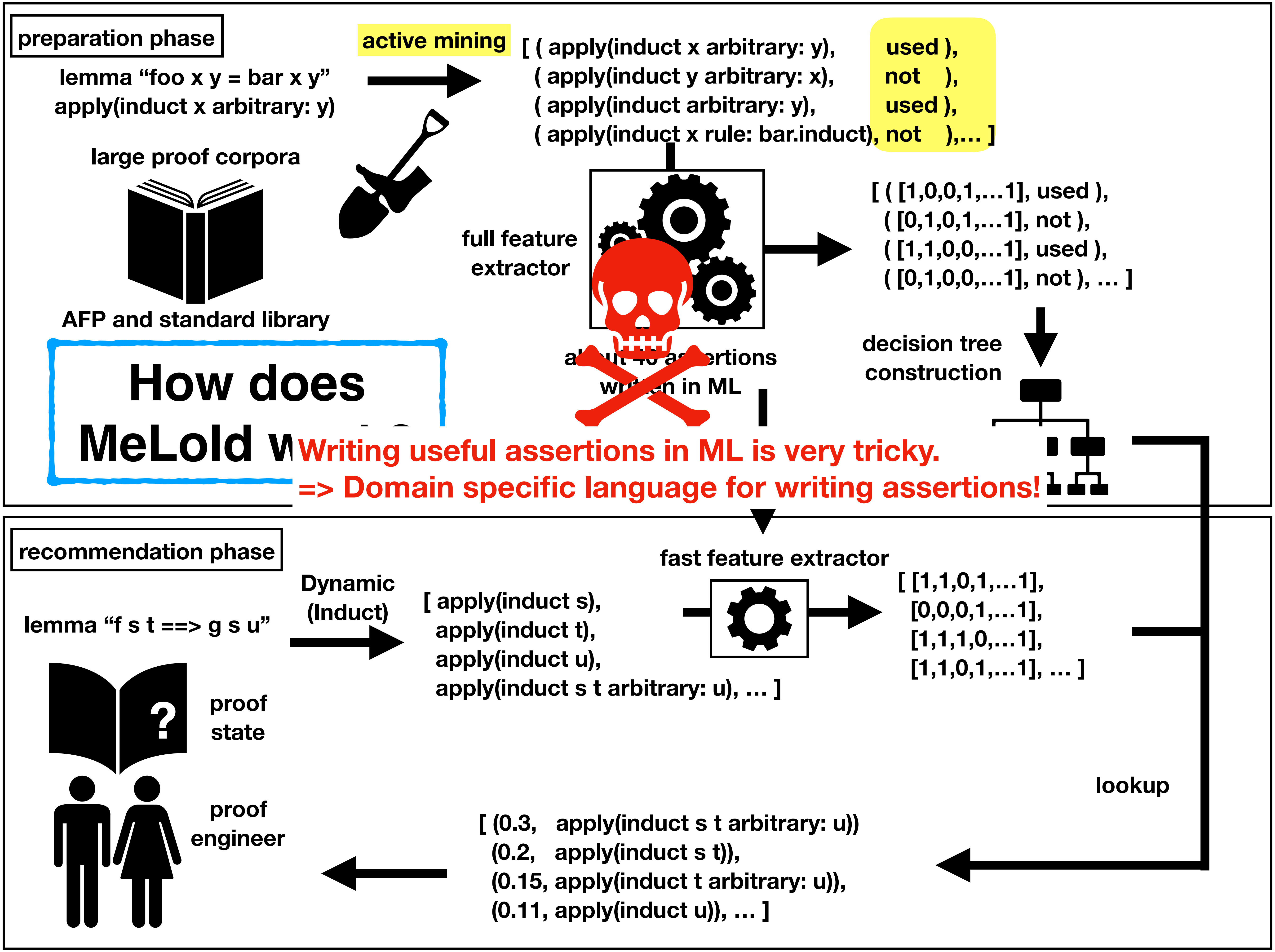
only if PSL with PGT compl

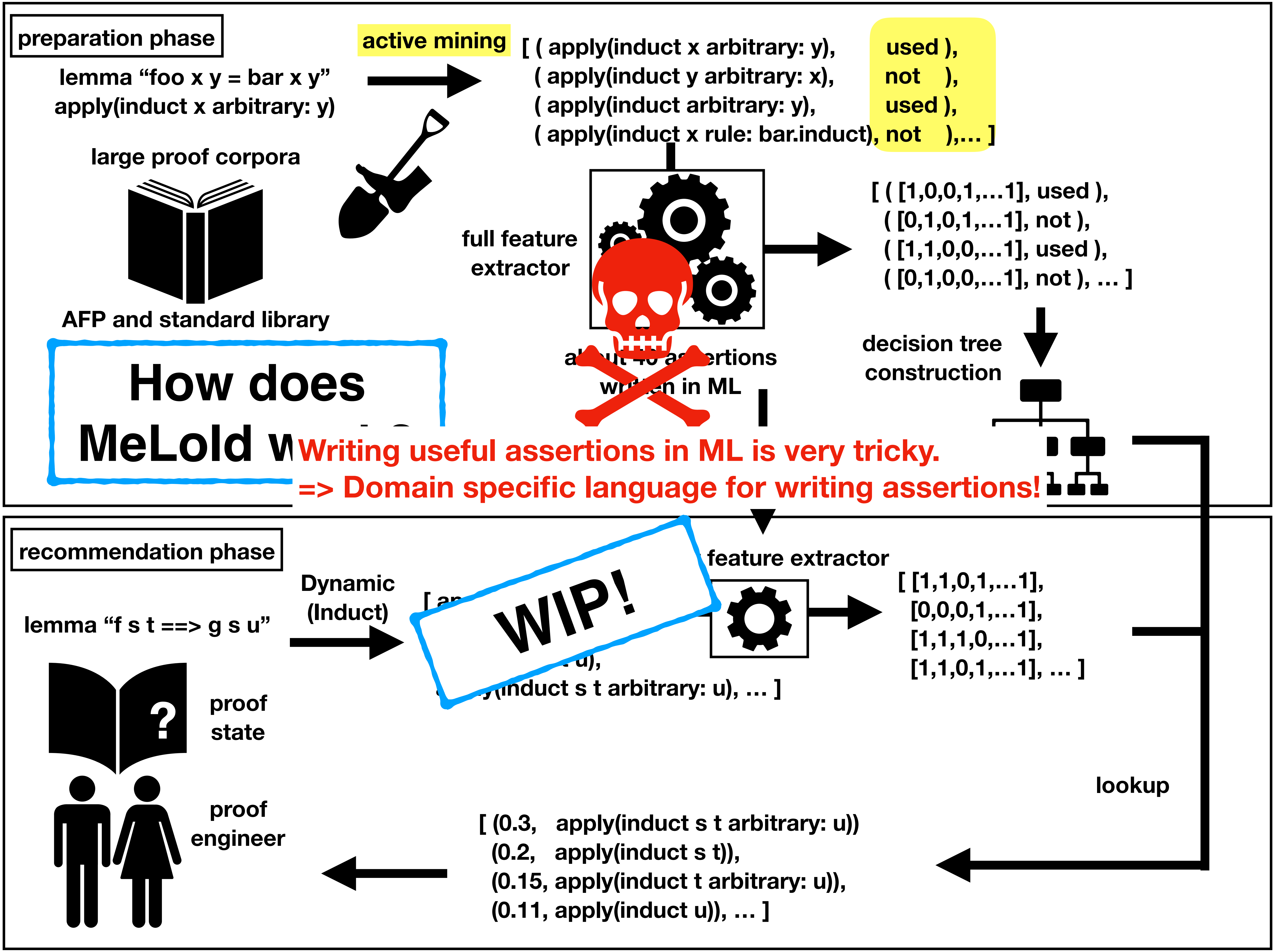
proof search

Recommend how to apply induction without completing a proof.

MeLoId: Machine Learning Induction


preparation phase
active mining
lemma “foo x y = bar x y”
apply(induct x arbitrary: y)
[(apply(induct x arbitrary: y), used),
(apply(induct y arbitrary: x), not),
(apply(induct arbitrary: y), used),
(apply(induct x rule: bar.induct), not),...]
large proof corpora
AFP and standard library
full feature extractor
about 40 assertions written in ML
[([1,0,0,1,...1], used),
([0,1,0,1,...1], not),
([1,1,0,0,...1], used),
([0,1,0,0,...1], not), ...]
decision tree construction
How does MeLold work?
recommendation phase
Dynamic (Induct)
lemma “f s t ==> g s u”
[apply(induct s),
apply(induct t),
apply(induct u),
apply(induct s t arbitrary: u), ...]
fast feature extractor
[[1,1,0,1,...1],
[0,0,0,1,...1],
[1,1,1,0,...1],
[1,1,0,1,...1], ...]
proof state
proof engineer
lookup
[(0.3, apply(induct s t arbitrary: u))
(0.2, apply(induct s t)),
(0.15, apply(induct t arbitrary: u)),
(0.11, apply(induct u)), ...]


preparation phase
active mining
lemma “foo x y = bar x y”
apply(induct x arbitrary: y)
[(apply(induct x arbitrary: y), used),
(apply(induct y arbitrary: x), not),
(apply(induct arbitrary: y), used),
(apply(induct x rule: bar.induct), not),...]
large proof corpora
AFP and standard library
full feature extractor
[([1,0,0,1,...1], used),
([0,1,0,1,...1], not),
([1,1,0,0,...1], used),
([0,1,0,0,...1], not), ...]
decision tree construction
How does
MeLold w
Writing useful assertions in ML is very tricky.
=> Domain specific language for writing assertions!
recommendation phase
Dynamic (Induct)
lemma “f s t ==> g s u”
[apply(induct s),
apply(induct t),
apply(induct u),
apply(induct s t arbitrary: u), ...]
fast feature extractor
[[1,1,0,1,...1],
[0,0,0,1,...1],
[1,1,1,0,...1],
[1,1,0,1,...1], ...]
proof state
proof engineer
lookup
[(0.3, apply(induct s t arbitrary: u))
(0.2, apply(induct s t)),
(0.15, apply(induct t arbitrary: u)),
(0.11, apply(induct u)), ...]


preparation phase
active mining
lemma “foo x y = bar x y”
apply(induct x arbitrary: y)
[(apply(induct x arbitrary: y), used),
(apply(induct y arbitrary: x), not),
(apply(induct arbitrary: y), used),
(apply(induct x rule: bar.induct), not),...]
large proof corpora
AFP and standard library
full feature extractor
[([1,0,0,1,...1], used),
([0,1,0,1,...1], not),
([1,1,0,0,...1], used),
([0,1,0,0,...1], not), ...]
decision tree construction
in ML
How does
MeLold w
Writing useful assertions in ML is very tricky.
=> Domain specific language for writing assertions!
recommendation phase
WIP!
feature extractor
lemma “f s t ==> g s u”
Dynamic (Induct)
u),
(induct s t arbitrary: u), ...]
[[1,1,0,1,...1],
[0,0,0,1,...1],
[1,1,1,0,...1],
[1,1,0,1,...1], ...]
proof state
proof engineer
lookup
[(0.3, apply(induct s t arbitrary: u))
(0.2, apply(induct s t)),
(0.15, apply(induct t arbitrary: u)),
(0.11, apply(induct u)), ...]

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

Thank you!

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

Thank you!

Leave a star at GitHub for PSL!

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

Thank you!

Leave a star at GitHub for PSL!

Let's write a review paper "AITP deserves High-Performance Computing, Too!"

Thank you!

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

Leave a star at GitHub for PSL!

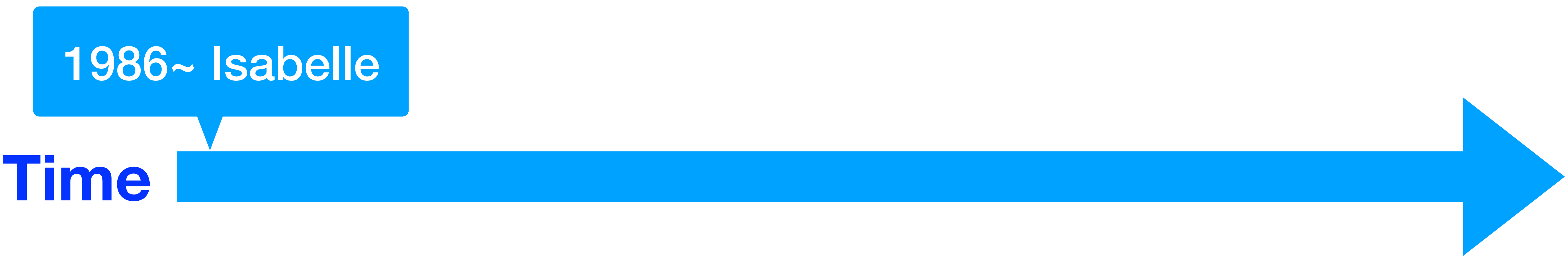
Let's write a review paper
"AITP deserves High-Performance Computing, Too!"

PaMpeR's feature extractor?

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

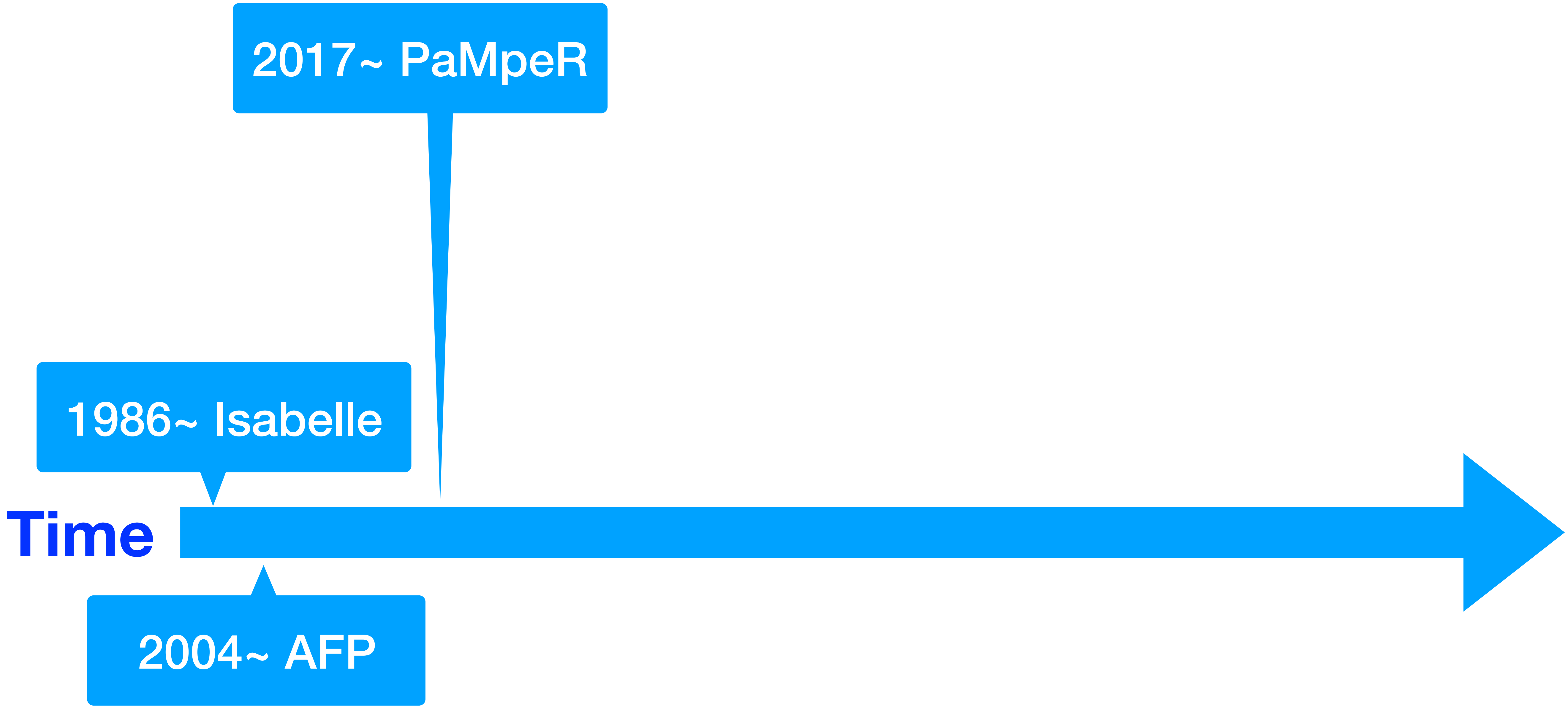
Time

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

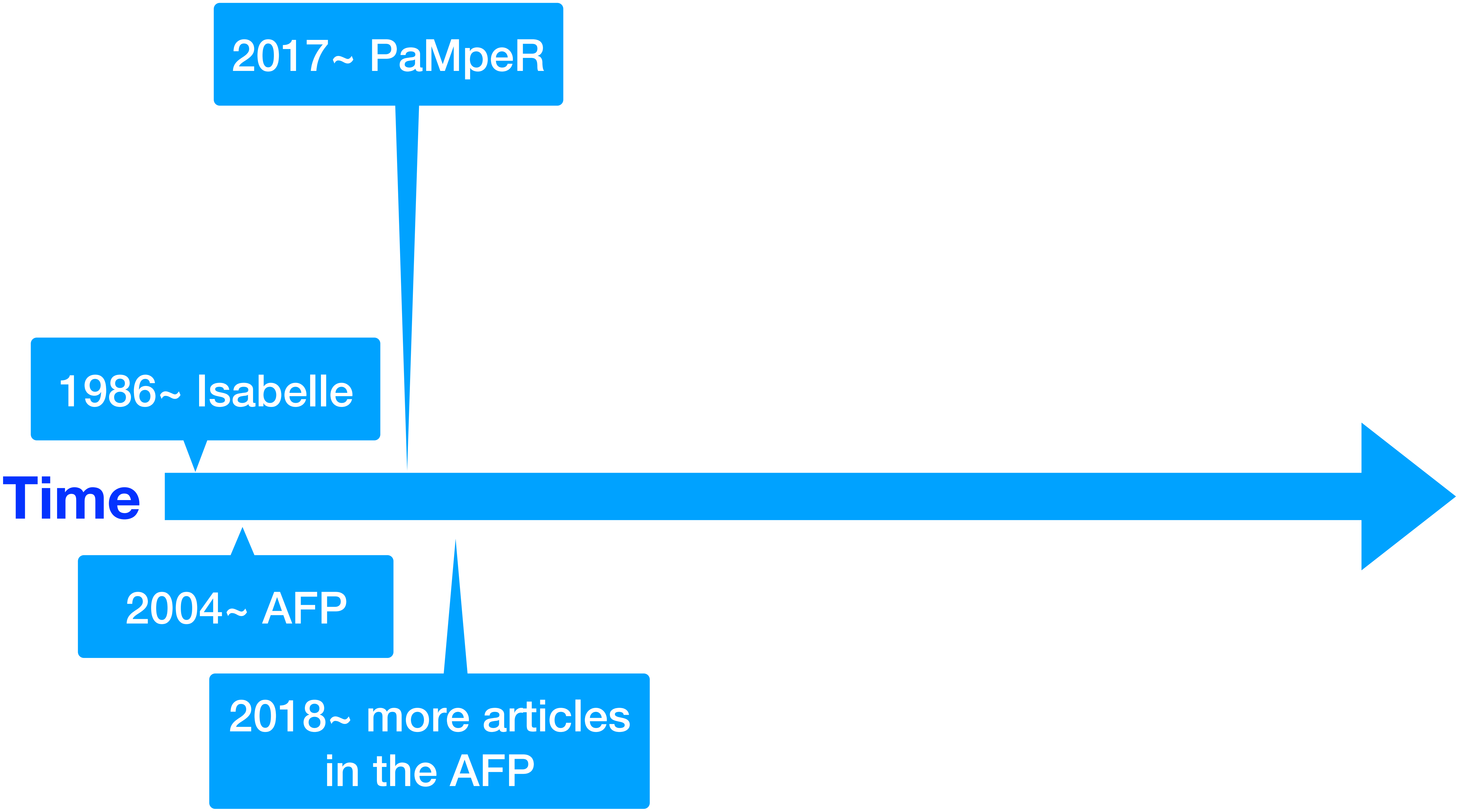


git clone https://github.com/data61/PSL

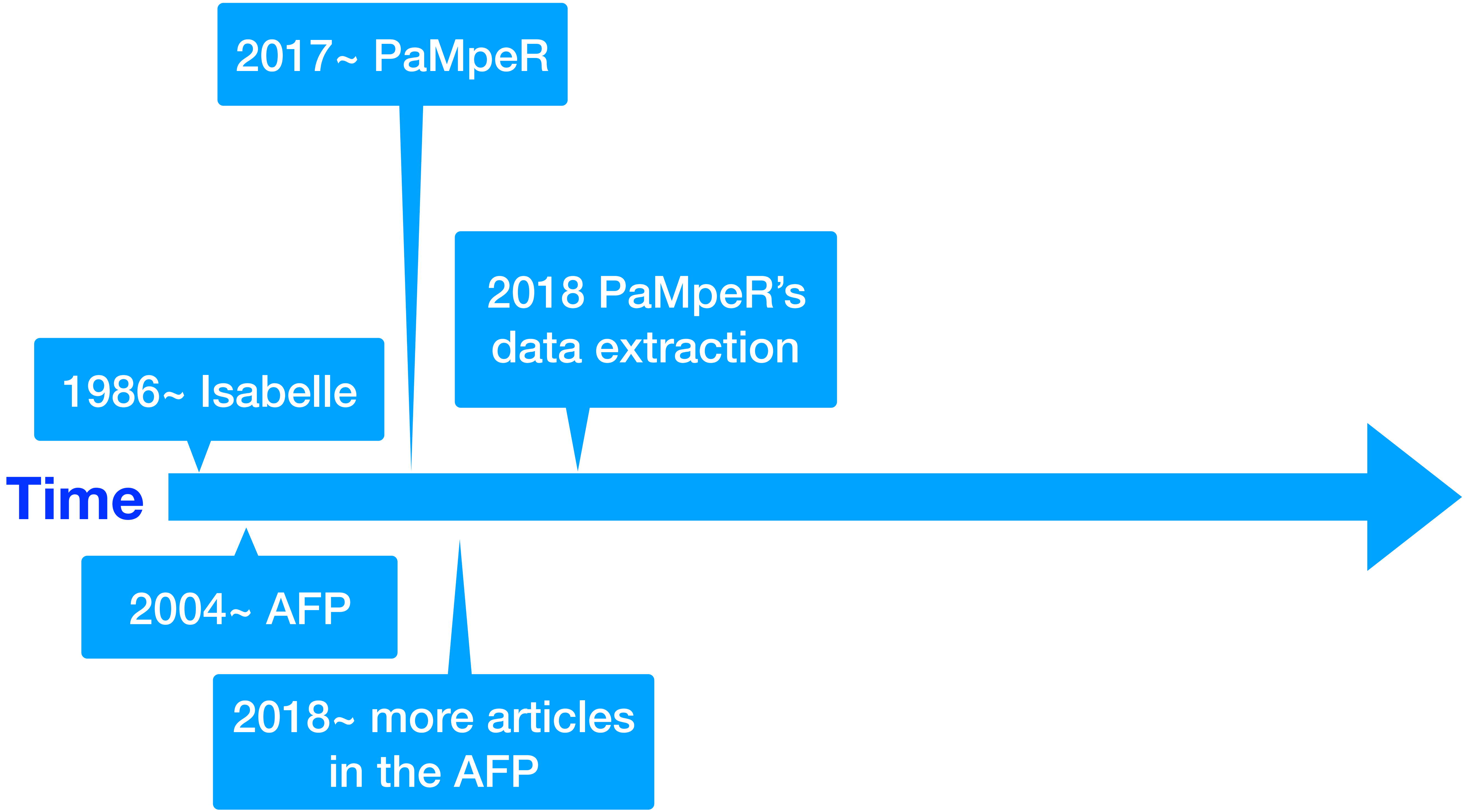
git clone https://github.com/data61/PSL



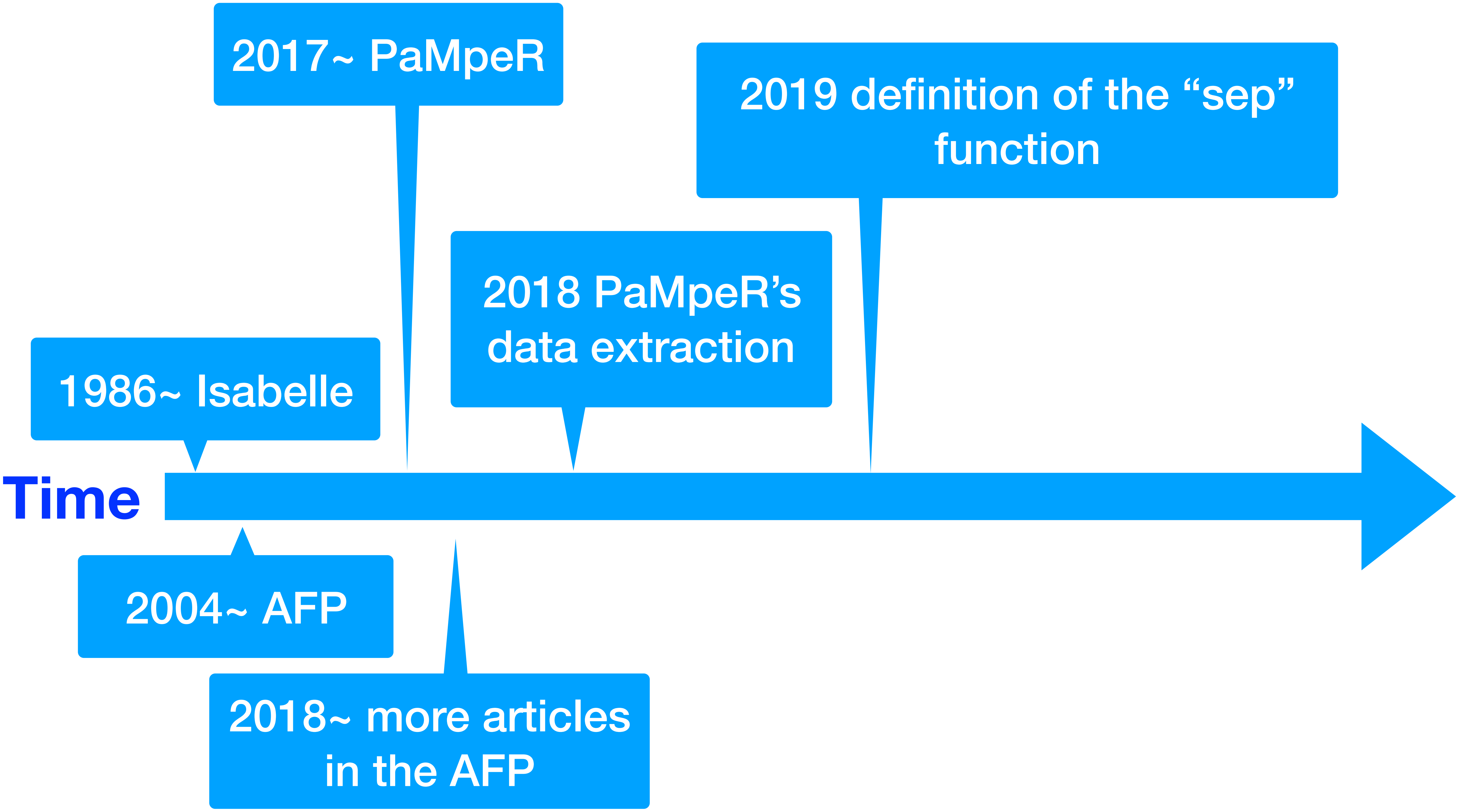
git clone https://github.com/data61/PSL



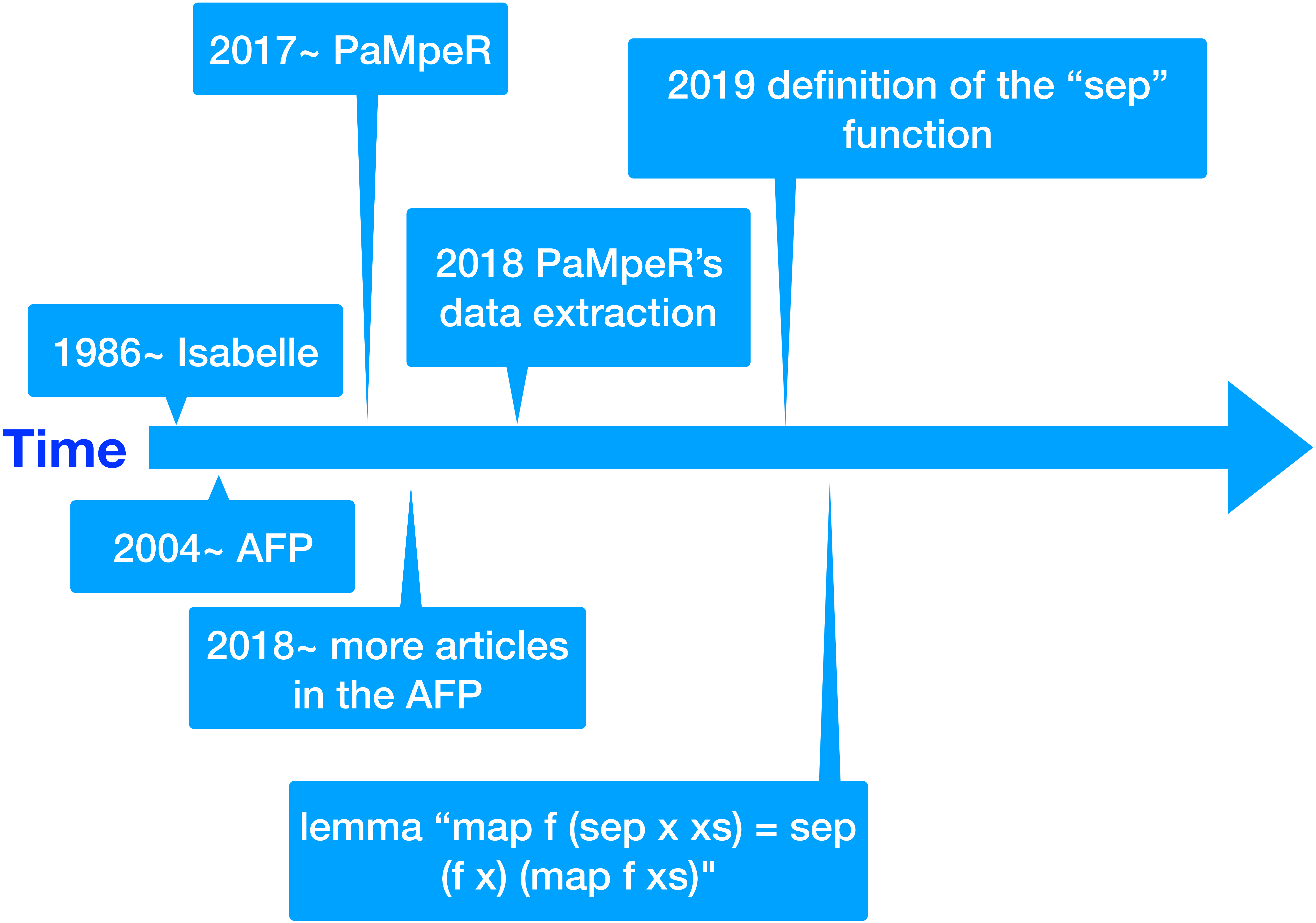
git clone https://github.com/data61/PSL



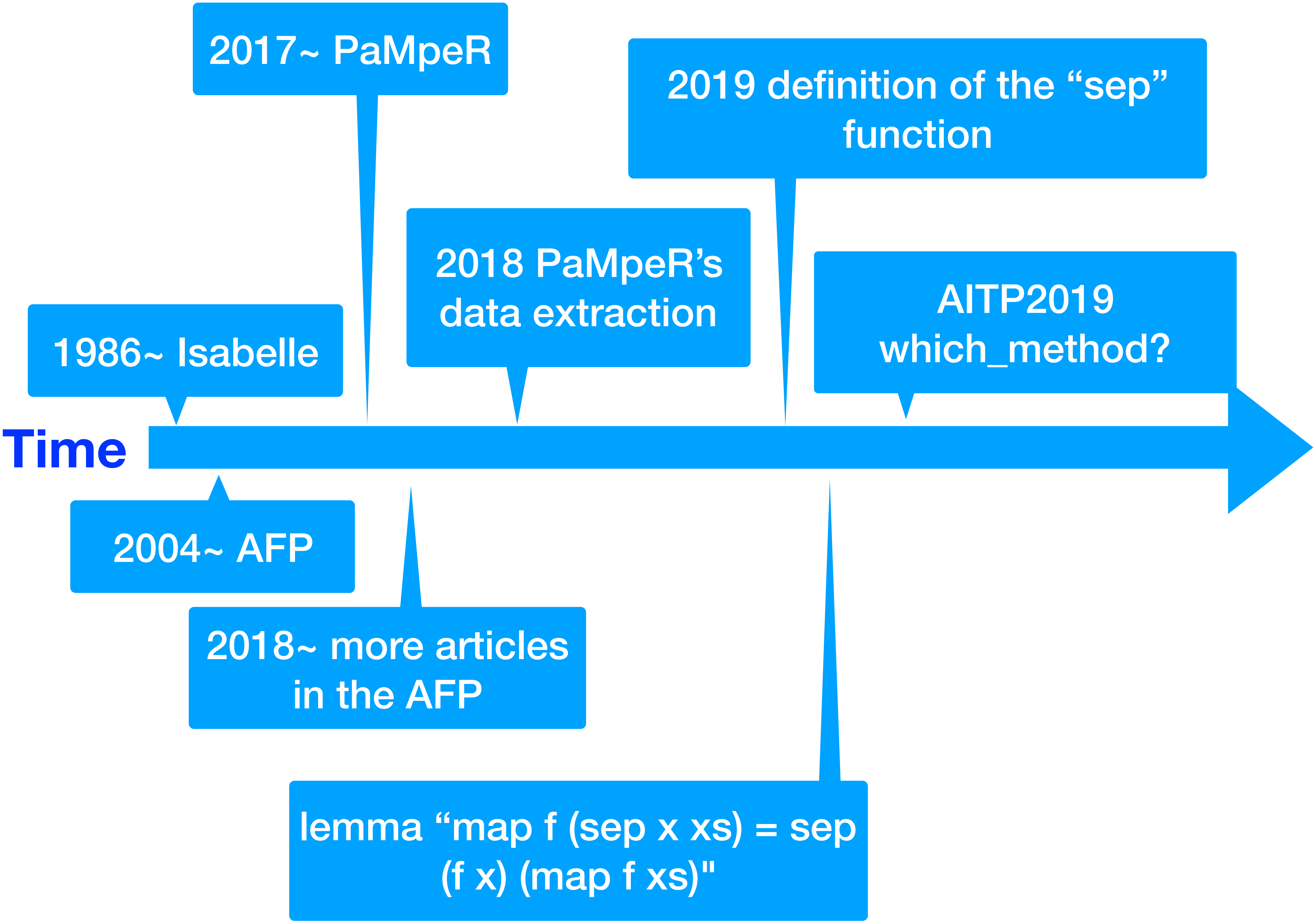
git clone https://github.com/data61/PSL



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



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



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

2017~ PaMpeR

2019 definition of the “sep” function

2018 PaMpeR’s data extraction

AITP2019

19

Tim

PaMpeR’s feature extractor has to be able to analyze things (e.g. “sep”) that do not exist yet!

2018~ more articles in the AFP

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

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

2017~ PaMpeR

2019 definition of the "sep" function

2018 PaMpeR's data extraction

AITP2019

PaMpeR's feature extractor has to be able to analyze things (e.g. "sep") that do not exist yet!

2018~ more articles in the AFP

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

DEMO!

Feature extractor?

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

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

Feature extractor?

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

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

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

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

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

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

Feature extractor?

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

automatically proves and saves many auxiliary lemmas in the context

sep.simps, sep.induct, sep.elims, etc.

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

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

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

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

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

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

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

10th 27th 32nd 58th 93rd

What assertions I wanted to write / wrote…

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
 apply (induct x xs rule: sep.induct)
```

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"

lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                      apply auto
```

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"
```

```
lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                      apply auto
```

Assertion 02:

Do induction on argument number i if the function is defined by recursion in argument number i?

What assertions I wanted to write / wrote…

```
lemma "map f (sep x xs) = sep (f x) (map f xs)"
apply (induct x xs rule: sep.induct)
```

Assertion 01:

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"

lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                       apply auto
```

Assertion 02:

Do induction on argument number i if the function is defined by recursion in argument number i?

definition of constants!

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"

lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                       apply auto
```

Assertion 02:

Do induction on argument number i if the function is defined by recursion in argument number i?

definition of constants!

Assertion03:

Are induction variables appear at the deepest level in the syntax tree?

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"
```

```
lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                       apply auto
```

Assertion 02:

Do induction on argument number i if the function is defined by recursion in argument number i?

definition of constants!

Assertion03:

Are induction variables appear at the deepest level in the syntax tree?

depth?

un-currying!

What assertions I wanted to write / wrote…

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

Assertion 01:

```
apply (induct x xs rule: sep.induct)
```

If the induct method uses an auxiliary lemma (sep.induct) …

check if the induction variables (x and xs) are arguments of the constant (sep) that has an auxiliary lemma (sep.induct).

position of arguments relative to certain constants!

Induction variables (x and xs) appear multiple times in the goal!

```
primrec my_append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@@" 65
append_Nil: "[] @@ ys = ys" |
append_Cons: "(x#xs) @@ ys = x # xs @@ ys"

lemma "(x @@ y) @@ z = x @@ (y @@ z)" apply (induct x)
                                      apply auto
```

Assertion 02:

Do induction on argument number i if the function is defined by recursion in argument number i?

definition of constants!

Assertion03:

Are induction variables appear at the deepest level in the syntax tree?

depth?

un-currying!

P x y ==> Q y z ==> R z w

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

Time

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

2017: PSL

Time

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

2017: PSL

2019: define the “sep” function

Time

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

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

2017: PSL

2019: define the “sep” function

2019: define the “DInd” strategy

2019:

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

Time

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

At the time of development (2017), PSL does not know about

- user defined constants (e.g. “sep”) or
- user defined proof strategies (e.g. DInd).