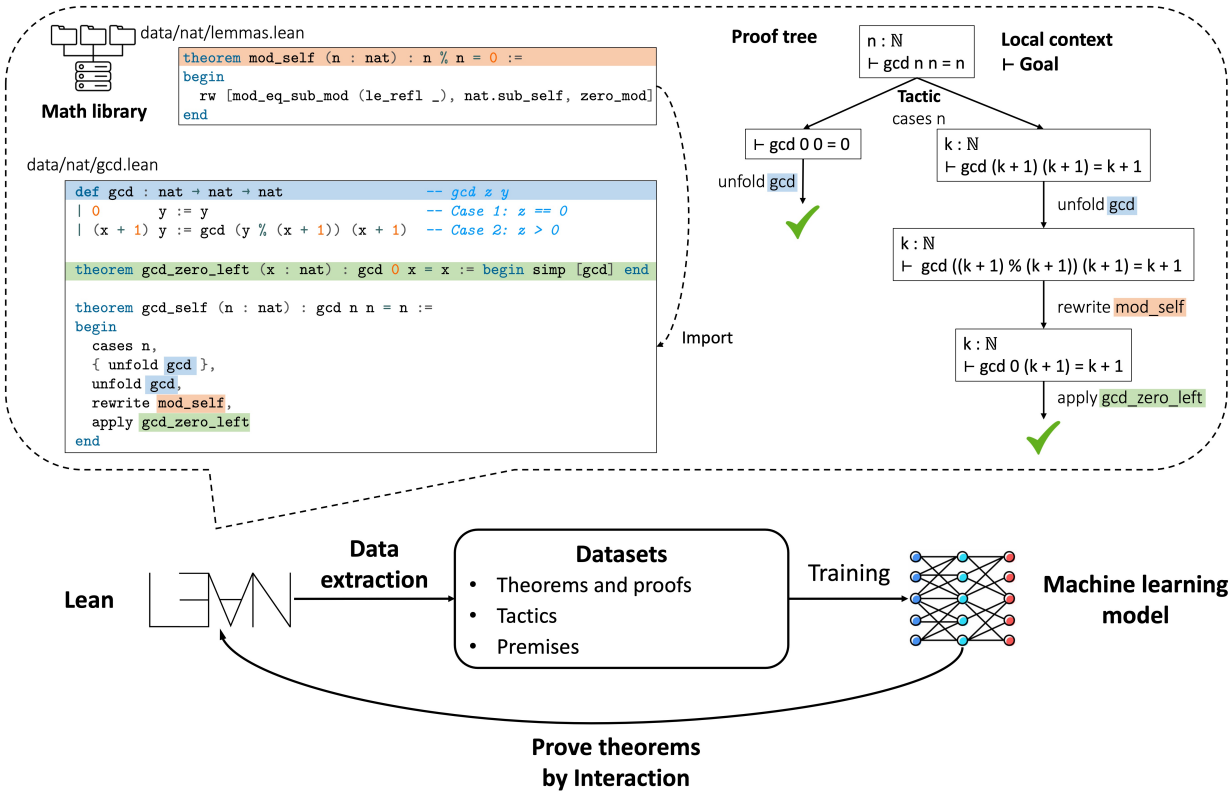
LeanDojo

Release 1.8.2

Kaiyu Yang

Apr 11, 2024

CONTENTS



LeanDojo is a Python library for learning–based theorem provers in Lean, supporting both Lean 3 and Lean 4. It provides two main features:

- Extracting data (proof states, tactics, premises, etc.) from Lean repos.
- Interacting with Lean programmatically.

CHAPTER

ONE

RELATED LINKS

- LeanDojo Website: The official website of LeanDojo.
- LeanDojo Benchmark: The Lean 3 dataset used in our paper, consisting of theorems and proofs extracted from mathlib by generate-benchmark-lean3.ipynb.
- LeanDojo Benchmark 4: The Lean 4 version of LeanDojo Benchmark, consisting of theorems and proofs extracted from mathlib4 by generate-benchmark-lean4.ipynb.
- ReProver: The ReProver (Retrieval-Augmented Prover) model in our paper.
- LeanInfer: Native neural network inference for running ReProver directly in Lean 4.

CHAPTER

TWO

CITATION

```bibtex
@inproceedings{yang2023leandojo,
  title={{LeanDojo}: Theorem Proving with Retrieval-Augmented Language Models},
  author={Yang, Kaiyu and Swope, Aidan and Gu, Alex and Chalamala, Rahul and Song, Peiyang and Yu, Shixing and Godil, Saad and Prenger, Ryan and Anandkumar, Anima},
  booktitle={Neural Information Processing Systems (NeurIPS)},
  year={2023}
}
```

CHAPTER

THREE

CONTENTS

3.1 Getting Started

This tutorial walks you through a simple example of using LeanDojo to extract data and interact with Lean.

3.1.1 Requirements

- Supported platforms: Linux, Windows (WSL), and macOS
- Git >= 2.25
- 3.9 <= Python < 3.11
- wget
- elan
- Generate a GitHub personal access token and set the environment variable `GITHUB_ACCESS_TOKEN` to it

3.1.2 Installation

LeanDojo is available on PyPI and can be installed via `pip install lean-dojo`. Alternatively, you can install the most recent version by running `pip install .` locally in the root directory of LeanDojo's GitHub repo.

3.1.3 Extracting Data from Lean 4

LeanDojo can also extract data from Lean 4 repos. We use lean4-example as a simple example, which has a single Lean file with the theorem:

Listing 1: Lean4Example.lean

```
open Nat (add_assoc add_comm)

theorem hello_world (a b c : Nat)
  : a + b + c = a + c + b := by
  rw [add_assoc, add_comm b, ←add_assoc]

theorem foo (a : Nat) : a + 1 = Nat.succ a := by rfl
```

We use LeanDojo to trace the repo in Python by specifying its URL and a commit hash:

```
from lean_dojo import LeanGitRepo, trace

repo = LeanGitRepo("https://github.com/yangky11/lean4-example",
↪"04e29174a45eefaccb49b835a372aa762321194e")
trace(repo, dst_dir="traced_lean4-example")
```

After a few minutes, it generates a `traced_lean4-example` directory with the subdirectories below. Please check out *Troubleshooting* if you encounter any issue.

```
traced_lean4-example
└─lean4-example
  ├─.lake
  │ ├─packages
  │ │ └─lean4
  │ └─build
  │   ├─ir
  │   │ ├─Lean4Example.dep_paths
  │   │ ├─Lean4Example.ast.json
  │   │ └─Lean4Example.trace.xml
  │   └─lib
  │     └─Lean4Example.olean
  ├─Lean4Example.lean
  └─...
```

`lean4` is the traced Lean 4 repo, and `lean-example` is the traced example repo. We call them "traced" because each `*.lean` file is accompanied by the following files:

- `*.olean`: Lean's compiled object file.
- `*.dep_paths`: Paths of dependencies imported by the current file.
- `*.ast.json`: ASTs exported by ExtractData.lean.
- `*.trace.xml`: Syntactic and semantic information extracted from Lean.

The most important one is `*.trace.xml`. For example, below is `traced_lean4-example/lean4-example/.lake/build/ir/Lean4Example.trace.xml`:

Listing 2: Lean4Example.trace.xml

```
<TracedFile path="Lean4Example.lean" md5="f8eb6563cd78c62389ff6cf40f485a1e">
  <FileNode start="(1, 1)" end="(7, 53)">
    <ModuleHeaderNode>
      <NullNode/>
      <NullNode/>
    </ModuleHeaderNode>
    <CommandOpenNode start="(1, 1)" end="(1, 30)">
      <AtomNode start="(1, 1)" end="(1, 5)" leading="" trailing=" " val="open"/>
      <CommandOpenonlyNode start="(1, 6)" end="(1, 30)">
        <IdentNode start="(1, 6)" end="(1, 9)" leading="" trailing=" " raw_val="Nat" val=
↪"Nat"/>
        <AtomNode start="(1, 10)" end="(1, 11)" leading="" trailing="" val="("/>
        <NullNode start="(1, 11)" end="(1, 29)">
          <IdentNode start="(1, 11)" end="(1, 20)" leading="" trailing=" " raw_val="add_
↪assoc" val="add_assoc" full_name="Nat.add_assoc" mod_name="Init.Data.Nat.Basic" def_
↪path=".lake/packages/lean4/src/lean/Init/Data/Nat/Basic.lean" def_start="(138, 19)"␣
```

(continues on next page)

(continued from previous page)

```
↪def_end="(138, 28)"/>
          <IdentNode start="(1, 21)" end="(1, 29)" leading="" trailing="" raw_val="add_
↪comm" val="add_comm" full_name="Nat.add_comm" mod_name="Init.Data.Nat.Basic" def_path=
↪".lake/packages/lean4/src/lean/Init/Data/Nat/Basic.lean" def_start="(131, 19)" def_end=
↪"(131, 27)"/>
        </NullNode>
        <AtomNode start="(1, 29)" end="(1, 30)" leading="" trailing="&#10;&#10;" val=")"/
↪>
      </CommandOpenonlyNode>
    </CommandOpenNode>
    <CommandDeclarationNode start="(3, 1)" end="(5, 41)" name="hello_world" full_name=
↪"hello_world">
      <CommandDeclmodifiersNode>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
      </CommandDeclmodifiersNode>
      <CommandTheoremNode start="(3, 1)" end="(5, 41)" name="hello_world" full_name=
↪"hello_world" _is_private_decl="False">
        <AtomNode start="(3, 1)" end="(3, 8)" leading="" trailing=" " val="theorem"/>
        <CommandDeclidNode start="(3, 9)" end="(3, 20)">
          <IdentNode start="(3, 9)" end="(3, 20)" leading="" trailing=" " raw_val="hello_
↪world" val="hello_world"/>
          <NullNode/>
        </CommandDeclidNode>
        <CommandDeclsigNode start="(3, 21)" end="(4, 26)">
          <NullNode start="(3, 21)" end="(3, 34)">
            <TermExplicitbinderNode start="(3, 21)" end="(3, 34)">
              <AtomNode start="(3, 21)" end="(3, 22)" leading="" trailing="" val="("/>
              <NullNode start="(3, 22)" end="(3, 27)">
                <IdentNode start="(3, 22)" end="(3, 23)" leading="" trailing=" " raw_val=
↪"a" val="a"/>
                <IdentNode start="(3, 24)" end="(3, 25)" leading="" trailing=" " raw_val=
↪"b" val="b"/>
                <IdentNode start="(3, 26)" end="(3, 27)" leading="" trailing=" " raw_val=
↪"c" val="c"/>
              </NullNode>
              <NullNode start="(3, 28)" end="(3, 33)">
                <AtomNode start="(3, 28)" end="(3, 29)" leading="" trailing=" " val=":"/>
                <IdentNode start="(3, 30)" end="(3, 33)" leading="" trailing="" raw_val=
↪"Nat" val="Nat" full_name="Nat" mod_name="Init.Prelude" def_path=".lake/packages/lean4/
↪src/lean/Init/Prelude.lean" def_start="(1059, 11)" def_end="(1059, 14)"/>
              </NullNode>
              <NullNode/>
              <AtomNode start="(3, 33)" end="(3, 34)" leading="" trailing="&#10;  " val=
↪")"/>
            </TermExplicitbinderNode>
          </NullNode>
          <TermTypespecNode start="(4, 3)" end="(4, 26)">
```

(continues on next page)

(continued from previous page)

```
            <AtomNode start="(4, 3)" end="(4, 4)" leading="" trailing=" " val=":"/>
            <OtherNode start="(4, 5)" end="(4, 26)" kind="«term_=_»">
              <OtherNode start="(4, 5)" end="(4, 14)" kind="«term_+_»">
                <OtherNode start="(4, 5)" end="(4, 10)" kind="«term_+_»">
                  <IdentNode start="(4, 5)" end="(4, 6)" leading="" trailing=" " raw_val=
↪"a" val="a"/>
                  <AtomNode start="(4, 7)" end="(4, 8)" leading="" trailing=" " val="+"/>
                  <IdentNode start="(4, 9)" end="(4, 10)" leading="" trailing=" " raw_
↪val="b" val="b"/>
                </OtherNode>
                <AtomNode start="(4, 11)" end="(4, 12)" leading="" trailing=" " val="+"/>
                <IdentNode start="(4, 13)" end="(4, 14)" leading="" trailing=" " raw_val=
↪"c" val="c"/>
              </OtherNode>
              <AtomNode start="(4, 15)" end="(4, 16)" leading="" trailing=" " val="="/>
              <OtherNode start="(4, 17)" end="(4, 26)" kind="«term_+_»">
                <OtherNode start="(4, 17)" end="(4, 22)" kind="«term_+_»">
                  <IdentNode start="(4, 17)" end="(4, 18)" leading="" trailing=" " raw_
↪val="a" val="a"/>
                  <AtomNode start="(4, 19)" end="(4, 20)" leading="" trailing=" " val="+
↪"/>
                  <IdentNode start="(4, 21)" end="(4, 22)" leading="" trailing=" " raw_
↪val="c" val="c"/>
                </OtherNode>
                <AtomNode start="(4, 23)" end="(4, 24)" leading="" trailing=" " val="+"/>
                <IdentNode start="(4, 25)" end="(4, 26)" leading="" trailing=" " raw_val=
↪"b" val="b"/>
              </OtherNode>
            </OtherNode>
          </TermTypespecNode>
        </CommandDeclsigNode>
        <CommandDeclvalsimpleNode start="(4, 27)" end="(5, 41)">
          <AtomNode start="(4, 27)" end="(4, 29)" leading="" trailing=" " val=":="/>
          <TermBytacticNode start="(4, 30)" end="(5, 41)">
            <AtomNode start="(4, 30)" end="(4, 32)" leading="" trailing="&#10;  " val="by
↪"/>
            <TacticTacticseqNode start="(5, 3)" end="(5, 41)">
              <TacticTacticseq1IndentedNode start="(5, 3)" end="(5, 41)">
                <NullNode start="(5, 3)" end="(5, 41)">
                  <OtherNode start="(5, 3)" end="(5, 41)" kind="Lean.Parser.Tactic.rwSeq
↪" state_before="a b c : Nat&#10; a + b + c = a + c + b" state_after="no goals" tactic=
↪"rw [add_assoc, add_comm b, ←add_assoc]">
                    <AtomNode start="(5, 3)" end="(5, 5)" leading="" trailing=" " val="rw
↪"/>
                    <NullNode/>
                    <OtherNode start="(5, 6)" end="(5, 41)" kind="Lean.Parser.Tactic.
↪rwRuleSeq">
                      <AtomNode start="(5, 6)" end="(5, 7)" leading="" trailing="" val="[
↪"/>
                      <NullNode start="(5, 7)" end="(5, 40)">
                        <OtherNode start="(5, 7)" end="(5, 16)" kind="Lean.Parser.Tactic.
↪rwRule">
```

(continues on next page)

(continued from previous page)

```
                        <NullNode/>
                        <IdentNode start="(5, 7)" end="(5, 16)" leading="" trailing="" raw_val="add_assoc" val="add_assoc" full_name="Nat.add_assoc" mod_name="Init.Data.Nat.Basic" def_path=".lake/packages/lean4/src/lean/Init/Data/Nat/Basic.lean" def_start="(138, 19)" def_end="(138, 28)"/>
                      </OtherNode>
                      <AtomNode start="(5, 16)" end="(5, 17)" leading="" trailing=" " val=","/>
                      <OtherNode start="(5, 18)" end="(5, 28)" kind="Lean.Parser.Tactic.rwRule">
                        <NullNode/>
                        <OtherNode start="(5, 18)" end="(5, 28)" kind="Lean.Parser.Term.app">
                          <IdentNode start="(5, 18)" end="(5, 26)" leading="" trailing=" " raw_val="add_comm" val="add_comm" full_name="Nat.add_comm" mod_name="Init.Data.Nat.Basic" def_path=".lake/packages/lean4/src/lean/Init/Data/Nat/Basic.lean" def_start="(131, 19)" def_end="(131, 27)"/>
                          <NullNode start="(5, 27)" end="(5, 28)">
                            <IdentNode start="(5, 27)" end="(5, 28)" leading="" trailing="" raw_val="b" val="b"/>
                          </NullNode>
                        </OtherNode>
                      </OtherNode>
                      <AtomNode start="(5, 28)" end="(5, 29)" leading="" trailing=" " val=","/>
                      <OtherNode start="(5, 30)" end="(5, 40)" kind="Lean.Parser.Tactic.rwRule">
                        <NullNode start="(5, 30)" end="(5, 31)">
                          <OtherNode start="(5, 30)" end="(5, 31)" kind="patternIgnore">
                            <OtherNode start="(5, 30)" end="(5, 31)" kind="token.«← »">
                              <AtomNode start="(5, 30)" end="(5, 31)" leading="" trailing="" val="←"/>
                            </OtherNode>
                          </OtherNode>
                        </NullNode>
                        <IdentNode start="(5, 31)" end="(5, 40)" leading="" trailing=" " raw_val="add_assoc" val="add_assoc" full_name="Nat.add_assoc" mod_name="Init.Data.Nat.Basic" def_path=".lake/packages/lean4/src/lean/Init/Data/Nat/Basic.lean" def_start="(138, 19)" def_end="(138, 28)"/>
                      </OtherNode>
                    </NullNode>
                    <AtomNode start="(5, 40)" end="(5, 41)" leading="" trailing="&#10;" val="]"/>
                  </OtherNode>
                  <NullNode/>
                </OtherNode>
              </NullNode>
            </TacticTacticseq1IndentedNode>
          </TacticTacticseqNode>
        </TermBytacticNode>
        <OtherNode kind="Lean.Parser.Termination.suffix">
```

(continues on next page)

(continued from previous page)

```
            <NullNode/>
            <NullNode/>
          </OtherNode>
          <NullNode/>
        </CommandDeclvalsimpleNode>
      </CommandTheoremNode>
    </CommandDeclarationNode>
    <CommandDeclarationNode start="(7, 1)" end="(7, 53)" name="foo" full_name="foo">
      <CommandDeclmodifiersNode>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
        <NullNode/>
      </CommandDeclmodifiersNode>
      <CommandTheoremNode start="(7, 1)" end="(7, 53)" name="foo" full_name="foo" _is_
↪private_decl="False">
        <AtomNode start="(7, 1)" end="(7, 8)" leading="" trailing=" " val="theorem"/>
        <CommandDeclidNode start="(7, 9)" end="(7, 12)">
          <IdentNode start="(7, 9)" end="(7, 12)" leading="" trailing=" " raw_val="foo"␣
↪val="foo"/>
          <NullNode/>
        </CommandDeclidNode>
        <CommandDeclsigNode start="(7, 13)" end="(7, 43)">
          <NullNode start="(7, 13)" end="(7, 22)">
            <TermExplicitbinderNode start="(7, 13)" end="(7, 22)">
              <AtomNode start="(7, 13)" end="(7, 14)" leading="" trailing="" val="("/>
              <NullNode start="(7, 14)" end="(7, 15)">
                <IdentNode start="(7, 14)" end="(7, 15)" leading="" trailing=" " raw_val=
↪"a" val="a"/>
              </NullNode>
              <NullNode start="(7, 16)" end="(7, 21)">
                <AtomNode start="(7, 16)" end="(7, 17)" leading="" trailing=" " val=":"/>
                <IdentNode start="(7, 18)" end="(7, 21)" leading="" trailing="" raw_val=
↪"Nat" val="Nat" full_name="Nat" mod_name="Init.Prelude" def_path=".lake/packages/lean4/
↪src/lean/Init/Prelude.lean" def_start="(1059, 11)" def_end="(1059, 14)"/>
              </NullNode>
              <NullNode/>
              <AtomNode start="(7, 21)" end="(7, 22)" leading="" trailing=" " val=")"/>
            </TermExplicitbinderNode>
          </NullNode>
          <TermTypespecNode start="(7, 23)" end="(7, 43)">
            <AtomNode start="(7, 23)" end="(7, 24)" leading="" trailing=" " val=":"/>
            <OtherNode start="(7, 25)" end="(7, 43)" kind="«term_=_»">
              <OtherNode start="(7, 25)" end="(7, 30)" kind="«term_+_»">
                <IdentNode start="(7, 25)" end="(7, 26)" leading="" trailing=" " raw_val=
↪"a" val="a"/>
                <AtomNode start="(7, 27)" end="(7, 28)" leading="" trailing=" " val="+"/>
                <OtherNode start="(7, 29)" end="(7, 30)" kind="num">
                  <AtomNode start="(7, 29)" end="(7, 30)" leading="" trailing=" " val="1
↪"/>
```

(continues on next page)

(continued from previous page)

```
              </OtherNode>
            </OtherNode>
            <AtomNode start="(7, 31)" end="(7, 32)" leading="" trailing=" " val="="/>
            <OtherNode start="(7, 33)" end="(7, 43)" kind="Lean.Parser.Term.app">
              <IdentNode start="(7, 33)" end="(7, 41)" leading="" trailing=" " raw_val=
↪"Nat.succ" val="Nat.succ" full_name="Nat.succ" mod_name="Init.Prelude" def_path=".lake/
↪packages/lean4/src/lean/Init/Prelude.lean" def_start="(1065, 5)" def_end="(1065, 9)"/>
              <NullNode start="(7, 42)" end="(7, 43)">
                <IdentNode start="(7, 42)" end="(7, 43)" leading="" trailing=" " raw_
↪val="a" val="a"/>
              </NullNode>
            </OtherNode>
          </OtherNode>
        </TermTypespecNode>
      </CommandDeclsigNode>
      <CommandDeclvalsimpleNode start="(7, 44)" end="(7, 53)">
        <AtomNode start="(7, 44)" end="(7, 46)" leading="" trailing=" " val=":="/>
        <TermBytacticNode start="(7, 47)" end="(7, 53)">
          <AtomNode start="(7, 47)" end="(7, 49)" leading="" trailing=" " val="by"/>
          <TacticTacticseqNode start="(7, 50)" end="(7, 53)">
            <TacticTacticseq1IndentedNode start="(7, 50)" end="(7, 53)">
              <NullNode start="(7, 50)" end="(7, 53)">
                <OtherNode start="(7, 50)" end="(7, 53)" kind="Lean.Parser.Tactic.
↪tacticRfl" state_before="a : Nat&#10; a + 1 = Nat.succ a" state_after="no goals"␣
↪tactic="rfl">
                  <AtomNode start="(7, 50)" end="(7, 53)" leading="" trailing="&#10;"␣
↪val="rfl"/>
                </OtherNode>
              </NullNode>
            </TacticTacticseq1IndentedNode>
          </TacticTacticseqNode>
        </TermBytacticNode>
        <OtherNode kind="Lean.Parser.Termination.suffix">
          <NullNode/>
          <NullNode/>
        </OtherNode>
        <NullNode/>
      </CommandDeclvalsimpleNode>
    </CommandTheoremNode>
  </CommandDeclarationNode>
 </FileNode>
 <Comments/>
</TracedFile>
```

3.1.4 Interacting with Lean 4

LeanDojo can also interact with Lean 4. Below we prove the `hello_world` theorem in the previous example. Note that the lean4-example repo has to be traced before interacting with any theorem in it. So the code below will first take some time to trace the repo if you haven't followed the steps in *Extracting Data from Lean 4*. The tracing has to be done only once, and the traced repo will be cached for future use. Some repos do not need to be traced locally and can be downloaded from our AWS S3 (see *Caching* for details).

```
from lean_dojo import *

repo = LeanGitRepo("https://github.com/yangky11/lean4-example",
↪"fd14c4c8b29cc74a082e5ae6f64c2fb25b28e15e")
theorem = Theorem(repo, "Lean4Example.lean", "hello_world")

with Dojo(theorem) as (dojo, init_state):
  print(init_state)
  result = dojo.run_tac(init_state, "rw [add_assoc, add_comm b, ←add_assoc]")
  assert isinstance(result, ProofFinished)
  print(result)
```

Listing 3: Expected output:

```
TacticState(pp='a b c : Nat\n a + b + c = a + c + b', id=0, message=None)
ProofFinished(tactic_state_id=1, message='')
```

3.1.5 Next Steps

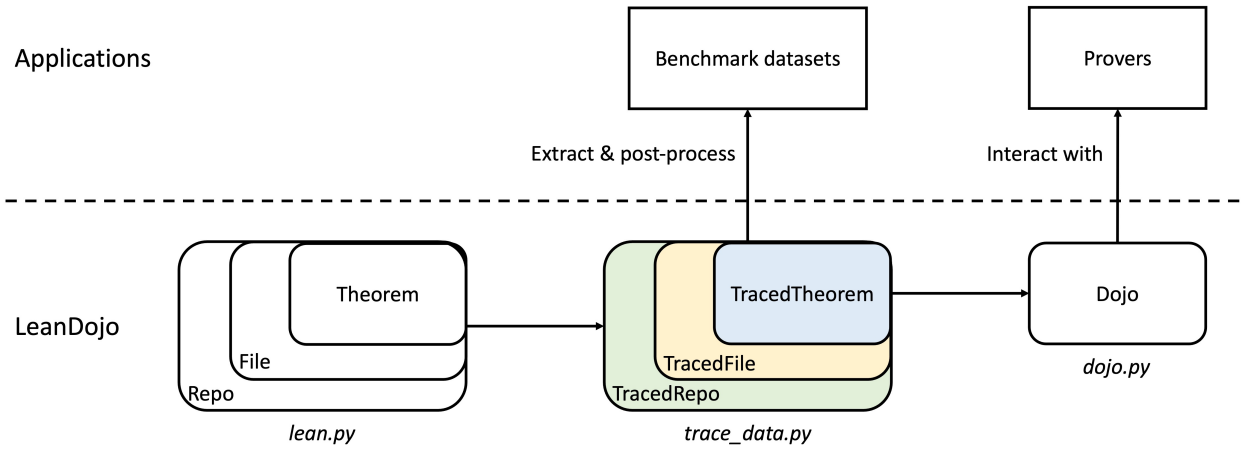
This example is just a glimpse of what LeanDojo can do. Please continue to the demos (Lean 3, Lean 4) and the *User Guide*.

3.2 User Guide

3.2.1 Overview

Below is a high-level overview of LeanDojo. A Lean project is a Git repo consisting of multiple source files (`*.lean`). Each file has multiple theorems (and their proofs). Given a Lean repo, LeanDojo first traces it. The results are traced repo, files, and theorems with rich syntactic/semantic information not readily available in source code but useful for downstream tasks, including but not limited to tactic states, tactics, and premises in tactics. Once we have traced repo/files/theorems, constructing datasets is simply a matter of post-processing.

Once a theorem is traced, LeanDojo enables us to interact with it by replacing the original, human-written proof with a special `repl` tactic. Unlike regular tactics transforming the goal in predefined ways, the `repl` tactic reads tactics from an external prover, executes them in Lean, and reports the results back. We name the resulting artifact Dojo (, the Japanese word for a gym for practicing martial arts). Any external prover (potentially based on machine learning) can interact with Dojo to practice the art of theorem proving.



Next, we explain each concept in mode detail.

3.2.2 Lean Repos and Files

We use lean-liquid in Lean 3 as an example of Lean repos. It is a formalization of Peter Scholze's Liquid Tensor Experiments. Every Lean 3 repo has a config file `leanpkg.toml` at its root.[1] For example,

Listing 4: leanpkg.toml

```
[package]
name = "lean-liquid"
version = "0.1"
lean_version = "leanprover-community/lean:3.48.0"
path = "src"

[dependencies]
mathlib = {git = "https://github.com/leanprover-community/mathlib", rev =
↪"5947fb69cc1fdfebaba1e1b1f0a04f26f0f612bf"}
```

In the config file, `lean_version` specifies the version of Lean this repo requires. `path` specifies the directory for Lean source files. And `[dependencies]` tells us the repo depends on a specific commit of mathlib. The repo can be compiled by running `leanpkg build`, which performs two steps under the hood:

1. Run `lean configure` to pull all dependencies (only mathlib here) into the `_target/deps/` directory.
2. Run `lean --make src` to compile the Lean source files (`src` is configurable in `leanpkg.toml`). Think of it as Lean's analogy of GNU Make, which is smart enough to figure out dependencies.

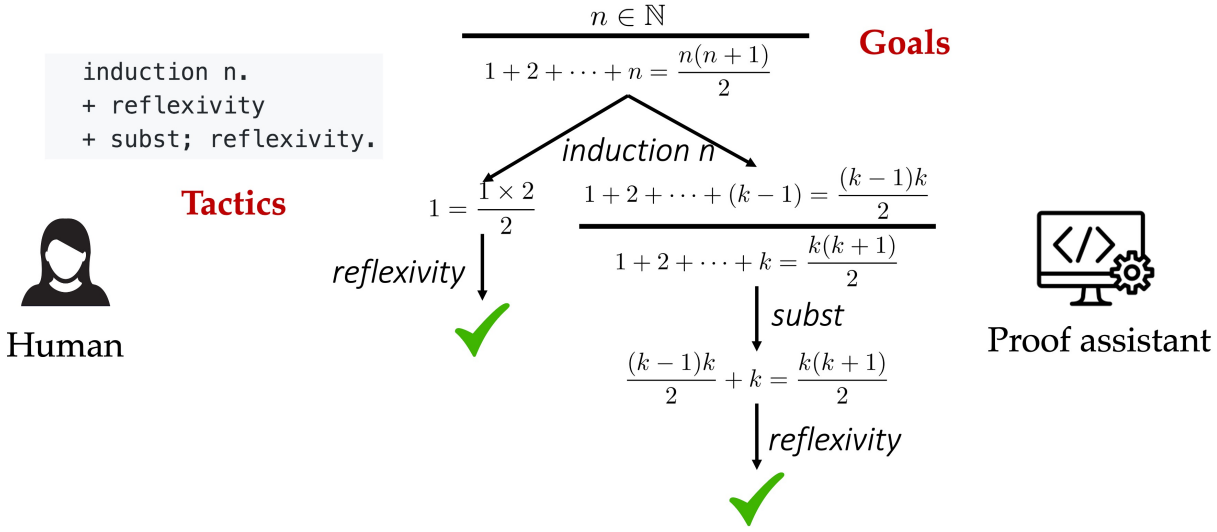
For each `*.lean` file, the compilation process checks all proofs in it and produces a `*.olean` file (analogous to compiling `*.cpp` to `*.o`). You may find more information about the toolchain of Lean 3 here and related LeanDojo implementation in *lean_dojo.data_extraction.lean*.

Lean 4 uses a different build system ([lake](https://github.com/leanprover/lake)) with different configuration files and directory structures. However, the high-level idea is the same.

[1] ... unless the repo is Lean itself, which LeanDojo handles for you but we'll not cover here.

3.2.3 Theorems and Proofs

Theorems can be proved in Lean in multiple styles (term-style, tactic-style, mixed-style, etc.). We focus on tactic-style proofs. A proof is a sequence of tactics. Think of a tactic as a proof step. Each tactic transforms current goals into (hopefully simpler) sub-goals. And the proof is complete when all remaining goals become trivial. Below is an example of interacting with a proof assistant to prove the theorem about summing n positive integers.



Initially, we know $n \in \mathbb{N}$ and want to prove $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. We enter the tactic `induction n` to reason by induction on the integer n, which leads to two sub-goals, one corresponding to $n = 0$ and the other corresponding to $n0$. Then we enter the tactic `reflexivity` to prove the trivial case ($n = 0$). Finally, we enter the tactic `subst; reflexivity` to finish the proof. This process generates a proof tree whose nodes are tactic states (states for short) and whose edges are tactics. Each state has a goal (e.g., $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$) and a local context consisting of hypotheses local to the goal (e.g., $n \in \mathbb{N}$).

Theorems/proofs do not exist in isolation. When proving a theorem, we almost inevitably rely on existing theorems and definitions. Consider a simple example we've seen in *Getting Started*:

Listing 5: example.lean

```
open nat (add_assoc add_comm)

theorem hello_world (a b c : ) : a + b + c = a + c + b :=
begin
  rw [add_assoc, add_comm b, ←add_assoc]
end
```

The tactic `rw [add_assoc, add_comm b, ←add_assoc]` uses the premise `add_assoc`, which is an existing theorem stating that the addition between natural numbers is associative (Similarly, `add_comm` is about commutativity). Premise selection, i.e. generating the premises `add_assoc` and `add_comm`, is a major challenge in theorem proving (for both humans and machines). This is because the space of potentially useful premises is huge. In principle, it includes all existing math when we attempt to prove a new math theorem. In practice, there can be hundreds of thousands of potential premises, which cannot fit into the input window of any Transformer-based large language model. Therefore, it is difficult for these models to perform premise selection effectively when generating tactics.

We have presented a vastly simplified view of theorems and proofs in Lean. For more detail, Theorem Proving in Lean (3, 4) is the definitive source. You may also refer to Sec. 3 of the LeanDojo paper. In LeanDojo, theorems are implemented by the *lean_dojo.data_extraction.lean.Theorem* class.

3.2.4 Traced Repos

Conceptually, a traced repo is simply a collection of traced files. These files form a directed acyclic graph (DAG) by importing each other. They include not only files from the target repo but also dependencies that are actually used by the target repo. Taking lean-liquid in Lean 3 as an example, it depends on mathlib and Lean's standard library. The corresponding traced repo include all traced files from lean-liquid, as well as some files from mathlib and Lean's standard library that are actually used by lean-liquid. After tracing finishes, these files are stored on the disk in the following directories:

```
lean-liquid
├─src
└─_target
  └─deps
    ├─lean
    │ └─library
    └─ mathlib
```

We call `lean-liquid` the **root directory** of the traced repo. Lean's standard library is in `lean-liquid/_target/deps/lean/library`. Source files of lean-liquid itself are in `lean-liquid/src`. Mathlib and other dependencies (if any) are in `lean-liquid/_target/deps`.

In Lean 4, dependencies are storied in `lake-packages` instead of `_target/deps`.

In LeanDojo, traced repos are implemented by the *lean_dojo.data_extraction.traced_data.TracedRepo* class.

3.2.5 Traced Files

Tracing a `*.lean` file generates the following files:

- `*.olean`: Lean's compiled object file. We're not concerned with them.
- `*.dep_paths`: Paths of dependencies imported by the `*.lean` file. We generate them using `lean --deps` (Lean 3) or ExtractData.lean (Lean 4).
- `*.ast.json`: AST of the `*.lean` file annotated with semantic information such as tactic states and name resolutions. We generate them using `lean --ast --tsast --tspp` (Lean 3) or ExtractData.lean (Lean 4).
- `*.trace.xml`: Syntactic and semantic information extracted from Lean. They are generated by post-processing `*.dep_paths` and `*.ast.json` files to organize the information in a nice way.

In LeanDojo, tracing is done by running build_lean3_repo.py with our modified Lean 3 or build_lean4_repo.py with ExtractData.lean By default, we perform tracing in a Docker container (configurable via the `CONTAINER` environment variable). Traced files are implemented by the *lean_dojo.data_extraction.traced_data.TracedFile* class.

3.2.6 Traced Theorems and Tactics

Traced theorems and tactics provide easy access to various information, such as the human-written proof of a theorem, the number of tactics in a theorem, the premises used in a theorem/tactic, whether the theorem has a tactic-style proof, etc. Please refer to *lean_dojo.data_extraction.traced_data.TracedTheorem* and *lean_dojo.data_extraction.traced_data.TracedTactic* for details.

3.2.7 Constructing Benchmark Datasets Using LeanDojo

Traced repos/files/theorems provide all information we need, and we can construct concrete machine learning datasets by some additional post-processing. See our examples of constructing datasets from Lean 3's mathlib and Lean 4's mathlib4.

3.2.8 Interacting with Lean

LeanDojo enables interacting with Lean programmatically using tactics. Please see the demos (Lean 3 version, Lean 4 version) and dojo.py.

3.2.9 Caching

Tracing large repos such as mathlib can take a long time (~1 hour if using 32 CPUs) and at least 32 GB memory. Therefore, we trace the repo only once and cache the traced repo for fast access in the future. The default cache directory is `~/.cache/lean_dojo`, which can be configured through the `CACHE_DIR` environment variable. Traced repos in the cache are read-only, which prevents accidental modifications. You need `chmod` if you want to, e.g., clean the cache. However, **refrain from manually modifying the cache while LeanDojo is running.** It may lead to unpredictable behaviors, e.g., LeanDojo may attempt to refill the cache.

Traced repos in the cache are portable across machines. We host a number of them on AWS S3, and LeanDojo will automatically download them if they are not in the local cache. To disable this behavior and build all repos locally, set the `DISABLE_REMOTE_CACHE` environment variable to any value.

The caching mechanism in LeanDojo is implemented in cache.py

3.2.10 Environment Variables

LeanDojo's behavior can be configured through the following environment variables:

- `CACHE_DIR`: Cache directory (see *Caching*). Default to `~/.cache/lean_dojo`.
- `DISABLE_REMOTE_CACHE`: Whether to disable remote caching and build all repos locally. Not set by default.
- `TMP_DIR`: Temporary directory used by LeanDojo for storing intermediate files. Default to the systems' global temporary directory.
- `NUM_PROCS`: Number of parallel processes for data extraction. Default to 32 or the number of CPUs (whichever is smaller).
- `TACTIC_TIMEOUT`: Maximum time (in milliseconds) before interrupting a tactic when interacting with Lean (only applicable to Lean 3). Default to 5000.
- `CONTAINER`: The container used for running LeanDojo. Default to `native`, i.e., running without any container, but also supports `docker`. See *Running within Docker (Important for Lean 3)*.
- `TACTIC_CPU_LIMIT`: Number of CPUs for executing tactics (see the *–cpus* flag of docker run) when interacting with Lean (only applicable when `CONTAINER=docker`). Default to 1.
- `TACTIC_MEMORY_LIMIT`: Maximum memory when interacting with Lean (see the *–memory* flag of docker run) (only applicable when `CONTAINER=docker`). Default to 16 GB.
- `GITHUB_ACCESS_TOKEN`: GitHub personal access token for using the GitHub API. They are optional. If provided, they can increase the API rate limit.

- `LOAD_USED_PACKAGES_ONLY`: Setting it to any value will cause LeanDojo to load only the dependency files that are actually used by the target repo. Otherwise, for Lean 4, it will load all files in the dependency repos. Not set by default.
- `VERBOSE` or `DEBUG`: Setting either of them to any value will cause LeanDojo to print debug information. Not set by default.

LeanDojo supports python-dotenv. You can use it to manage environment variables in a `.env` file.

3.2.11 Running within Docker (Important for Lean 3)

By default, LeanDojo performs data extraction and interaction by running Lean directly (`CONTAINER=native`). However, this may be problematic for Lean 3, as LeanDojo builds a modified version of Lean 3 from its C++ code, which requires certain dependencies that can be hard to get right. Therefore, if you use LeanDojo with Lean 3, we strongly recommend setting the environment variable `CONTAINER` to `docker`, which instructs LeanDojo to run relevant parts in a Docker container. You will need Docker installed on your machine, but you do not need to build or launch Docker containers manually. LeanDojo will do it for you.

That said, there are scenarios where running within Docker is not an option, e.g., when you are on a remote server that does not have Docker installed, or when your server requires wrapping the entire job as a Docker container and you want to avoid the troubles caused by running Docker within Docker.

If you want to use LeanDojo with Lean 3 but cannot use Docker, follow these steps to run it without Docker:

- See if you can follow the CMake instructions to build Lean 3 from source. Resolve any dependency issues.
- Make sure you can build the Lean 3 repo you want by `leanpkg build`.

3.2.12 Questions and Bugs

- For general questions and discussions, please use GitHub Discussions.
- To report a potential bug, please open an issue. In the issue, please include your OS information, the exact steps to reproduce the error, and complete logs in debug mode (setting the environment variable `VERBOSE` to 1). The more details you provide, the better we will be able to help you.

3.3 Troubleshooting

When troubleshooting, set the environment variable `VERBOSE=1` to get debug logs. Below are some common errors when using LeanDojo:

3.3.1 Installation

- I run into errors related to `lxml` and `grpcio` when running `pip install lean-dojo` on a Mac with Apple silicon.

This is a known issue with these two packages on Apple silicon. You should install them using whatever way that works for you. See these two posts on Stack Overflow.

3.3.2 Tracing Repos

- The process is killed when tracing a repo.

The most likely reason is your machine doesn't have enough memory. The amount of memory required depends on the repo you're tracing. For large repos, such as mathlib, you need at least 32 GB memory. If getting more memory is not an option, you can try a smaller repo. If your machine has enough memory but the process still gets killed, please check whether your Docker has access to all resources of host machine (see here).

3.3.3 Interacting with Lean

- `docker: Error response from daemon: invalid mount config for type "bind": bind source path does not exist`

Make sure Docker has access to the `/tmp` directory. If you're using Docker Desktop, go to Settings -> Resources -> File sharing.

3.4 Developer Guide

This page includes information that are **hidden from most users but useful if you want to contribute to LeanDojo**. It requires additional dependencies installed via `pip install "lean-dojo[all]"`.

3.4.1 Contribution Guideline

We welcome and appreciate contributions from the community. For bug fixes and relatively minor changes (comments, typos, etc.), feel free to submit a pull request directly. For anything more substantial, please first reach out to us before implementing. All contributions should conform to our code formatting standards (see *Code Formatting*).

3.4.2 Testing

We use pytest for testing. You can run tests by:

```
VERBOSE=1 CACHE_DIR=~/.cache/lean_dojo_testing DISABLE_REMOTE_CACHE=1 pytest -s tests
rm -rf ~/.cache/lean_dojo_testing
```

The environment variable `CACHE_DIR` makes sure the testing uses a temporary cache directory that does not interfere with the deployed code. The temporary cache directory is deleted after the testing completes. `DISABLE_REMOTE_CACHE=1` instructs repos to be built locally. Note that running all tests can take almost 1 day on 32 CPUs.

3.4.3 Building the Documentation

This documentation is generated by Sphinx. You can build it by:

```
cd docs && make clean && make html && cd ..
```

3.4.4 Building the Package and Uploading to PyPI

You probably won't need it, but this is how to build the LeanDojo package and upload it to PyPI:

```
hatch build
hatch publish
```

3.4.5 Static Type Checking

The source code of LeanDojo uses Python type annotations extensively. You can perform static type checking using mypy by

```
mypy src/lean_dojo
```

Currently there are still many type errors. Contributions to fix them are welcome.

3.4.6 Code Formatting

LeanDojo's code is formatted by Black. We use Github Actions to ensure all modifications are formatted.

3.5 Limitations and Caveats

3.5.1 Limitations

LeanDojo has the following limitations. Addressing them won't be our priority in the near future, but we welcome contributions:

- LeanDojo cannot extract data from the lean4 repo itself nor interact with theorems in it.
- Currently, LeanDojo cannot process Lean repos that use FFI, e.g., [LeanCopilot](https://github.com/lean-dojo/LeanCopilot).
- LeanDojo does not support term-based proofs or proofs that mixes tactics and terms.
- Entering all tactic-style proofs in mathlib 3 to LeanDojo, we found ~1.4% of them are misjudged as incorrect. The errors fall into a few categories documented in test_unexpected_errors.py. We didn't perform this analysis on Lean 4.
- Theorems extracted by LeanDojo are "syntactic theorems", i.e., they are Lean constants defined using keywords `theorem` or `lemma`. First, they are not guaranteed to be real theorems (Lean constants of type `Prop`). Second, theorems defined in other ways (e.g., using `def`) are not extracted.
- Tracing mathlib 3 produces buggy `IdentNode`. Their names start with `user__`. So you can search for `name="user__` in the generated `*.trace.xml` files. We haven't figured out the exact reason underlying this bug, but it's likely to be related to Lean's parser and mathlib's alias command.
- We require Lean 3 >= v3.42.1. Otherwise, the modification patch cannot be applied.
- We have problems tracing a few premises related to `quot` in Lean 3. See the warnings in this Jupyter notebook.
- Lean 3 ASTs generated by `lean --ast --tsast --tspp` have a small number of errors.

3.5.2 Caveats

- When LeanDojo is killed, it may leave temporary files in the system's temporary directory (or the directory specified by the `TMP_DIR` environment variable). It may be necessary to manually clean up these files occasionally.

3.6 Credits and Acknowledgements

3.6.1 Contributors

If your code has been merged into the LeanDojo repo and you would like your name to appear on this page, please feel free to edit this page and open a PR.

Developers

- Kaiyu Yang: Postdoctoral Scholar at Caltech
- Peiyang Song: Undergrad at UC Santa Barbara
- Rahul Chalamala: Undergrad at Caltech

Advisors

- Anima Anandkumar: Bren Professor at Caltech, Senior Director of AI Research at NVIDIA

3.6.2 Related Tools

LeanDojo draws on numerous existing tools for interacting with and extracting data from proof assistants. The author benefited a lot from the lessons he learned when designing and implementing CoqGym. LeanDojo's interaction implementation partially incorporates lean-gym's code but fixes some critical issues and expanded its functionality. When prototyping the data extraction part of LeanDojo, the author found Jason Rute's lean_proof_recording helpful. Its design was a hacky but extremely clever piece of art. Fortunately, we didn't have to follow that design thanks to Lean's AST exporting mechanism (`lean --ast --tsast --tspp`) implemented by Mario Carneiro, Gabriel Ebner, and Daniel Selsam. LeanDojo's mechanism for exporting Lean 4 ASTs is inspired by the conversations with Mario Carneiro during Kaiyu Yang's visit to Hoskinson Center for Formal Mathematics (directed by Jeremy Avigad) at CMU.

Below is a list of related tools we're aware of. Please reach out if we missed your work!

Lean 3

- lean_proof_recording
- lean-gym
- lean-client-python

Lean 4

- LeanInk
- lean-gym for Lean 4
- Daniel Selsam's Lean 4 fork
- repl

Coq

- CoqGym
- GamePad

Isabelle

- PISA

HOL Light

- HOList

Others

- INT

3.7 API Reference

This API reference is generated automatically from docstrings by autodoc.

3.7.1 lean_dojo.data_extraction

lean_dojo.data_extraction.lean

This module define classes for repos, files, and theorems in Lean. Objects of these classes contain only surface information, without extracting any trace.

`lean_dojo.data_extraction.lean.`**`GITHUB_ACCESS_TOKEN = None`**

GiHub personal access token is optional. If provided, it can increase the rate limit for GitHub API calls.

`lean_dojo.data_extraction.lean.`**`LEAN4_NIGHTLY_REPO = Repository(full_name="leanprover/lean4-nightly")`**

The GitHub Repo for Lean 4 nightly releases.

`lean_dojo.data_extraction.lean.`**`LEAN4_REPO = Repository(full_name="leanprover/lean4")`**

The GitHub Repo for Lean 4 itself.

class lean_dojo.data_extraction.lean.**LeanFile**(*root_dir: Path*, *path: Path*)

Bases: object

A Lean source file (*.lean).

__getitem__(*key*) → str

Return a code segment given its start/end positions.

This enables lean_file[start:end].

Parameters
key (*slice*) – A slice of two *Pos* objects for the start/end of the code segment.

property abs_path: Path

Absolute path of a *LeanFile* object.

E.g., /home/kaiyu/traced_lean-example/lean-example/src/example.lean

code: List[str]

Raw source code as a list of lines.

convert_pos(*byte_idx: int*) → *Pos*

Convert a byte index (String.Pos in Lean 4) to a *Pos* object.

property end_pos: *Pos*

Return the end position of a source file.

Parameters
zero_indexed (*bool, optional*) – Whether to use 0-index instead of 1-index. Defaults to False.

Returns
A *Pos* object representing the end of this file.

Return type
Pos

endwith_newline: bool

Whether the last line ends with a newline.

get_line(*line_nb: int*) → str

Return a given line of the source file.

Parameters
line_nb (*int*) – Line number (1-indexed).

num_bytes: List[int]

The number of UTF-8 bytes of each line, including newlines.

num_columns(*line_nb: int*) → int

Number of columns in a source file.

property num_lines: int

Number of lines in a source file.

offset(*pos:* Pos, *delta: int*) → *Pos*

Off set a position by a given number.

path: Path

Relative path w.r.t. `root_dir`

E.g., `lean-example/src/example.lean`

root_dir: Path

Root directory of the traced repo this *LeanFile* object belongs to.

`root_dir` must be an absolute path, e.g., `/home/kaiyu/traced_lean-example/lean-example`

property start_pos: *Pos*

Return the start position of a source file.

Returns
A *Pos* object representing the start of this file.

Return type
Pos

class lean_dojo.data_extraction.lean.**LeanGitRepo**(*url: str*, *commit: str*)

Bases: `object`

Git repo of a Lean project.

clone_and_checkout() → None

Clone the repo to the current working directory and checkout a specific commit.

commit: str

The repo's commit hash.

You can also use tags such as `v3.5.0`. They will be converted to commit hashes.

property commit_url: str

exists() → bool

classmethod from_path(*path: Path*) → *LeanGitRepo*

Construct a *LeanGitRepo* object from the path to a local Git repo.

get_config(*filename: str*, *num_retries: int = 2*) → Dict[str, Any]

Return the repo's files.

get_dependencies(*path: str | Path | None = None*) → Dict[str, *LeanGitRepo*]

Return the dependencies required by the target repo.

Parameters
path (*Union[str, Path, None], optional*) – Root directory of the repo if it is on the disk.

Returns
A dictionary mapping the name of each dependency to its *LeanGitRepo* object.

Return type
Dict[str, *LeanGitRepo*]

get_license() → str | None

Return the content of the `LICENSE` file.

property is_lean4: bool

lean_version: str

Required Lean version.

property name: str

repo: Repository

A `github.Repository` object.

show() → None

Show the repo in the default browser.

url: str

The repo's Github URL.

Note that we only support Github as of now.

class lean_dojo.data_extraction.lean.**Pos**(*line_nb: int*, *column_nb: int*)

Bases: `object`

Position in source files.

We use 1-index to keep it consistent with code editors such as Visual Studio Code.

column_nb: int

Column number

classmethod from_str(*s: str*) → *Pos*

Construct a *Pos* object from its string representation, e.g., `"(323, 1109)"`.

line_nb: int

Line number

class lean_dojo.data_extraction.lean.**RepoInfoCache**(*tag2commit: ~typing.Dict[~typing.Tuple[str, str], str] = <factory>*, *lean_version: ~typing.Dict[~typing.Tuple[str, str], str] = <factory>*)

Bases: `object`

To minize the number of network requests, we cache and re-use the info of all repos, assuming it does not change during the execution of LeanDojo.

lean_version: Dict[Tuple[str, str], str]

tag2commit: Dict[Tuple[str, str], str]

class lean_dojo.data_extraction.lean.**Theorem**(*repo:* LeanGitRepo, *file_path: Path*, *full_name: str*)

Bases: `object`

Theorem in Lean.

Theorems are named constants of type `Prop`. They are typically defined using the keywords `theorem` or `lemma`, but it's possible to use other keywords such as `def` or `instance`

file_path: Path

Lean source file the theorem comes from.

full_name: str

Fully qualified name of the theorem.

repo: *LeanGitRepo*

Lean repo the theorem comes from.

property uhash: str

Unique hash of the theorem.

property uid: str

Unique identifier of the theorem.

lean_dojo.data_extraction.lean.**cleanse_string**(*s: str | Path*) → str

Replace : and / with _ in a string.

lean_dojo.data_extraction.lean.**get_latest_commit**(*url: str*) → str

Get the hash of the latest commit of the Git repo at `url`.

lean_dojo.data_extraction.lean.**get_lean4_commit_from_config**(*config_dict: Dict[str, Any]*) → str

Return the required Lean commit given a `lean-toolchain` config.

lean_dojo.data_extraction.lean.**get_lean4_version_from_config**(*toolchain: str*) → str

Return the required Lean version given a `lean-toolchain` config.

lean_dojo.data_extraction.lean.**is_supported_version**(*v*) → bool

Check if `v` is at least *v4.3.0-rc2*.

lean_dojo.data_extraction.lean.**normalize_url**(*url: str*) → str

lean_dojo.data_extraction.lean.**url_to_repo**(*url: str*, *num_retries: int = 2*) → Repository

lean_dojo.data_extraction.trace

This module provides the main interfaces for tracing Lean repos, i.e., extracting data from them. To estimate the time for tracing a repo, a good rule of thumb is 1.5x the time for compiling the repo using `leanpkg build`. A repo has to be traced only once, and the traced repo will be stored in a cache for fast access in the future.

lean_dojo.data_extraction.trace.**get_traced_repo_path**(*repo: LeanGitRepo*, *build_deps: bool = True*) → Path

Return the path of a traced repo in the cache.

The function will trace a repo if it is not available in the cache. See *Caching* for details.

Parameters

- **repo** (LeanGitRepo) – The Lean repo to trace.
- **build_deps** (*bool*) – Whether to build the dependencies of `repo`. Defaults to True.

Returns

The path of the traced repo in the cache, e.g. `/home/kaiyu/.cache/lean_dojo/leanprover-community-mathlib-2196ab363eb097c008d4497125e0dde23fb36db2`

Return type

Path

lean_dojo.data_extraction.trace.**is_available_in_cache**(*repo: LeanGitRepo*) → bool

Check if `repo` has a traced repo available in the cache (including the remote cache).

lean_dojo.data_extraction.trace.**trace**(*repo:* LeanGitRepo, *dst_dir: str | Path | None = None*, *build_deps: bool = True*) → *TracedRepo*

Trace a repo (and its dependencies), saving the results to `dst_dir`.

The function only traces the repo when it's not available in the cache. Otherwise, it directly copies the traced repo from the cache to `dst_dir`. See *Caching* for details.

Parameters

- **repo** (LeanGitRepo) – The Lean repo to trace.
- **dst_dir** (*Union[str, Path]*) – The directory for saving the traced repo. If None, the traced repo is only saved in the cahe.
- **build_deps** (*bool*) – Whether to build the dependencies of `repo`. Defaults to True.

Returns
A `TracedRepo` object corresponding to the files at `dst_dir`.

Return type
TracedRepo

lean_dojo.data_extraction.traced_data

This module defines traced repos/files/theorems.

class lean_dojo.data_extraction.traced_data.**Comment**(*start:* Pos, *end:* Pos, *text: str*)

Bases: `object`

A comment in a Lean file.

end: *Pos*

classmethod from_xml(*tree: Element*) → *Comment*

start: *Pos*

text: **str**

to_xml(*parent: Element*) → None

class lean_dojo.data_extraction.traced_data.**TracedFile**(*root_dir: Path*, *repo:* LeanGitRepo, *lean_file:* LeanFile, *ast:* FileNode, *comments: List[*Comment*]*, *traced_repo:* TracedRepo | *None = None*)

Bases: `object`

A traced file is a Lean source file annotated with syntactic/semantic information such as tactic states, Lean expressions, and abstract syntax trees (ASTs).

property abs_path: **Path**

Absolute path of the `*.lean` file.

ast: *FileNode*

Abstract syntax tree (AST) of the entire `*.lean` file.

AST nodes are defined in *lean_dojo.data_extraction.ast*.

check_sanity() → None

Perform some basic sanity checks.

The function raises exceptions in case of unsuccessful checks.

comments: List[*Comment*]

All comments in the `*.lean` file.

classmethod from_traced_file(*root_dir: str | Path*, *json_path: Path*, *repo:* LeanGitRepo) → *TracedFile*

Construct a *TracedFile* object by parsing a `*.ast.json` file produced by `lean --ast --tsast --tspp` (Lean 3) or `ExtractData.lean` (Lean 4).

Parameters

- **root_dir** (*Union[str, Path]*) – Root directory of the traced repo.
- **json_path** (*Path*) – Path of the `*.ast.json` file relative to `root_dir`.

classmethod from_xml(*root_dir: str | Path*, *path: str | Path*, *repo:* LeanGitRepo) → *TracedFile*

Load a *TracedFile* object from its `*.trace.xml` file.

Parameters

- **root_dir** (*Union[str, Path]*) – Root directory of the traced repo.
- **path** (*Union[str, Path]*) – Path of the `*.trace.xml` file relative to `root_dir`.
- **repo** (LeanGitRepo) – The repo to which the traced file belongs.

get_direct_dependencies(*repo:* LeanGitRepo) → List[Tuple[str, Path]]

Return the names and paths of all modules imported by the current `*.lean` file.

get_premise_definitions() → List[Dict[str, Any]]

Return all theorems and definitions defined in the current file that can be potentially used as premises.

Returns
description

Return type
List[Dict[str, Any]]

get_traced_theorem(*thm_or_name:* Theorem | *str*) → *TracedTheorem* | None

Return a *TracedTheorem* object given an `Theorem` object or its fully-qualified name.

get_traced_theorems() → List[*TracedTheorem*]

Return a list of traced theorem in this traced file.

property has_prelude: bool

Check whether the file starts with :code:`prelude`.

:code:`prelude` instructs Lean NOT to include its built-in library automatically.

lean_file: *LeanFile*

Lean source file of this traced file.

property path: Path

Path of the `*.lean` file relative to the root directory.

repo: *LeanGitRepo*

The Lean repo this traced file belongs to.

root_dir: Path

Root directory (in absolute path) of the corresponding traced repo.

to_xml() → str

Serialize a *TracedFile* object to XML.

traced_repo: *TracedRepo* | None = None

The traced repo this traced file belongs to.

Note that `traced_repo` will become None after the traced file is serialized/deserialized on its own.

traverse_preorder(*callback*, *node_cls: type | None = None*)

Traverse the AST in preorder.

Parameters

- **callback** (*function*) – Callback function for visiting AST nodes.
- **node_cls** (*Optional[type], optional*) – Restrict the application of `callback` to only nodes of type `node_cls`. Defaults to None, which means applying `callback` to all.

class lean_dojo.data_extraction.traced_data.**TracedRepo**(*repo: LeanGitRepo*, *dependencies: Dict[str, LeanGitRepo]*, *root_dir: Path*, *traced_files: List[TracedFile]*, *traced_files_graph: DiGraph | None*)

Bases: `object`

A traced repo is a Lean repo of traced files and additional information, such as other repos it depends on, as well as the dependency graph between files.

check_sanity() → None

Perform some basic sanity checks.

The function raises exceptions in case of unsuccessful checks.

dependencies: Dict[str, *LeanGitRepo*]

Dictionary mapping the name of each dependency to a `LeanGitRepo` object.

classmethod from_traced_files(*root_dir: str | Path*, *build_deps: bool = True*) → *TracedRepo*

Construct a *TracedRepo* object by parsing *.ast.json and *.path files
produced by `lean --ast --tsast --tspp` (Lean 3) or `ExtractData.lean` (Lean 4).

Parameters

- **root_dir** (*Union[str, Path]*) – Root directory of the traced repo.
- **build_deps** (*bool, optional*) – Whether to build the dependency graph between files.

get_traced_file(*path: str | Path*) → *TracedFile*

Return a traced file by its path.

get_traced_theorem(*thm:* Theorem) → *TracedTheorem* | None

Return a *TracedTheorem* object corresponding to `thm`

get_traced_theorems() → List[*TracedTheorem*]

Return all traced theorems in the repo.

classmethod load_from_disk(*root_dir: str | Path*, *build_deps: bool = True*) → *TracedRepo*

Load a traced repo from `*.trace.xml` files.

property name: str

Name of the repo.

repo: *LeanGitRepo*

The corresponding Lean repo.

root_dir: Path

Root directory of the traced repo.

save_to_disk() → None

Save all traced files in the repo to the disk as `*.trace.xml` files.

show() → None

Show the repo in the default browser.

traced_files: List[*TracedFile*]

List of traced files in the repo.

traced_files_graph: DiGraph | None

Dependency graph between files in the repo.

The graph is a DAG, and there is an edge from file `X` to file `Y` if and only if `X` imports `Y`

class lean_dojo.data_extraction.traced_data.**TracedTactic**(*ast:* Node, *traced_theorem:* TracedTheorem | *None = None*)

Bases: `object`

A traced tactic is a tactic annotated with additional information including its AST and the states before/after the tactic.

ast: *Node*

AST of the tactic.

property end: *Pos*

End position in `*.lean` file.

get_annotated_tactic() → Tuple[str, List[Dict[str, Any]]]

Return the tactic annotated with premise information.

Premises in the tactic are marked by `<a> ... </a>`. For example, `rw [add_comm b]` contains a premise `add_comm` and therefore becomes `rw [<a>add_comm</a> b]`. In addition, the function returns the provenance (full name, file path, line/column numbers) of all premises.

Returns

The first return value is the tactic string marked by `<a> ... </a>`. The second return value is a list of provenances.

Return type

Tuple[str, List[Dict[str, Any]]]

property start: *Pos*

Start position in `*.lean` file.

property state_after: str

Pretty-printed state after applying the tactic.

property state_before: str

Pretty-printed state before applying the tactic.

property tactic: str

The raw tactic string.

to_string() → str

traced_theorem: *TracedTheorem* | None = None

The traced theorem this tactic belongs to.

class lean_dojo.data_extraction.traced_data.**TracedTheorem**(*root_dir: Path*, *theorem:* Theorem, *ast:* CommandTheoremNode | LemmaNode | MathlibTacticLemmaNode, *comments: List[*Comment*]*, *traced_file:* TracedFile | *None = None*)

Bases: object

A traced theorem is a theorem with additional information such as the AST.

ast: *CommandTheoremNode* | *LemmaNode* | *MathlibTacticLemmaNode*

AST of the theorem.

comments: List[*Comment*]

All comments in the theorem/proof.

property end: *Pos*

End position in *.lean file.

property file_path: Path

The theorem's file path (relative to the root directory).

get_namespaces() → Tuple[List[str], List[str]]

Return the namespaces that the theorem is located in, as well as the namespaces that are merely open.

get_num_tactics() → int

Return the number of tactics in the proof.

get_premise_full_names() → List[str]

Return the fully qualified names of all premises used in the proof.

get_proof_node() → *Node*

Return the AST of the theorem's proof.

get_single_tactic_proof() → str | None

Wrap the proof into a single (potentially very long) tactic.

get_tactic_proof() → str | None

Return the tactic-style proof (if any).

get_theorem_statement() → str

Return the theorem statement.

get_traced_tactics(*atomic_only: bool = False*) → List[*TracedTactic*]

Return a list of traced tactics in the proof.

has_tactic_proof() → bool

Check if the theorem has a tactic-style proof.

property is_private: bool

Check if the theorem is private.

locate_proof() → Tuple[*Pos*, *Pos*]

Return the start/end positions of the proof.

property repo: *LeanGitRepo*

The Lean repo this theorem belongs to.

root_dir: Path

Root directory of the corresponding traced repo.

show() → None

Show the theorem in the default browser.

property start: *Pos*

Start position in `*.lean` file.

theorem: *Theorem*

The corresponding `Theorem` object.

traced_file: *TracedFile* | None = None

The traced file this theorem belongs to.

property traced_repo: *TracedRepo*

The traced repo this theorem belongs to.

lean_dojo.data_extraction.traced_data.**get_code_without_comments**(*lean_file:* LeanFile, *start:* Pos, *end:* Pos, *comments:* *List[*Comment*]*) → str

Return the code in `lean_file` from `start` to `end` with comments removed.

Parameters

- **lean_file** (`LeanFile`) – The lean source file.
- **start** (`Pos`) – The start position.
- **end** (`Pos`) – The end position.
- **comments** (*`List[`Comment`]`*) – A list of *`Comment`* objects.

Returns

Human-written code with comments removed.

Return type

str

lean_dojo.data_extraction.ast

lean_dojo.data_extraction.ast

class lean_dojo.data_extraction.ast.**AtomNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*, *leading: str*, *trailing: str*, *val:* *str*)

Bases: *Node*

classmethod from_data(*atom_data: Dict[str, Any]*, *lean_file:* LeanFile) → *AtomNode* | None

leading: str

trailing: str

val: str

class lean_dojo.data_extraction.ast.**CommandAbbrevNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandAbbrevNode*

name: str

class lean_dojo.data_extraction.ast.**CommandAxiomNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandAxiomNode*

name: str

class lean_dojo.data_extraction.ast.**CommandClassinductiveNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandClassinductiveNode*

name: str

class lean_dojo.data_extraction.ast.**CommandClasstkNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandClasstkNode*

class lean_dojo.data_extraction.ast.**CommandDeclarationNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*, *name: str*, *full_name: str | None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclarationNode*

full_name: str | None = None

get_theorem_node() → *CommandTheoremNode*

property is_example: bool

property is_theorem: bool

name: str

class lean_dojo.data_extraction.ast.**CommandDeclidAntiquotNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclidAntiquotNode*

class lean_dojo.data_extraction.ast.**CommandDeclidNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclidNode*

class lean_dojo.data_extraction.ast.**CommandDeclmodifiersAntiquotNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclmodifiersAntiquotNode*

class lean_dojo.data_extraction.ast.**CommandDeclmodifiersNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclmodifiersNode*

is_private() → bool

class lean_dojo.data_extraction.ast.**CommandDeclsigNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclsigNode*

class lean_dojo.data_extraction.ast.**CommandDeclvaleqnsNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclvaleqnsNode*

class lean_dojo.data_extraction.ast.**CommandDeclvalsimpleNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDeclvalsimpleNode*

class lean_dojo.data_extraction.ast.**CommandDefNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDefNode*

name: str

class lean_dojo.data_extraction.ast.**CommandDoccommentNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*, *comment:* *str*)

Bases: *Node*

comment: str

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandDoccommentNode*

class lean_dojo.data_extraction.ast.**CommandEndNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str* | *None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandEndNode*

name: str | None

class lean_dojo.data_extraction.ast.**CommandExampleNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name:* *str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandExampleNode*

name: str

class lean_dojo.data_extraction.ast.**CommandInductiveNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str* | *None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandInductiveNode*

name: str | None

class lean_dojo.data_extraction.ast.**CommandInstanceNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandInstanceNode*

name: str

class lean_dojo.data_extraction.ast.**CommandModuledocNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *comment: str*)

Bases: *Node*

comment: str

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandModuledocNode*

class lean_dojo.data_extraction.ast.**CommandNamespaceNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandNamespaceNode*

name: str

class lean_dojo.data_extraction.ast.**CommandNoncomputablesectionNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str | None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandNoncomputablesectionNode*

name: str | None

class lean_dojo.data_extraction.ast.**CommandOpaqueNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandOpaqueNode*

name: str

class lean_dojo.data_extraction.ast.**CommandOpenNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandOpenNode*

class lean_dojo.data_extraction.ast.**CommandOpenonlyNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandOpenonlyNode*

class lean_dojo.data_extraction.ast.**CommandPrivateNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandPrivateNode*

class lean_dojo.data_extraction.ast.**CommandSectionNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str | None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandNamespaceNode*

name: str | None

class lean_dojo.data_extraction.ast.**CommandStructureNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandStructureNode*

name: str

class lean_dojo.data_extraction.ast.**CommandStructuretkNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandStructuretkNode*

class lean_dojo.data_extraction.ast.**CommandTheoremNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*, *full_name: str | None = None*, *_is_private_decl: bool | None = False*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandTheoremNode*

full_name: str | None = None

get_proof_node() → *Node*

has_tactic_proof() → bool

property is_mutual: bool

is_private() → bool

name: str

class lean_dojo.data_extraction.ast.**CommandWherestructinstNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *CommandWherestructinstNode*

class lean_dojo.data_extraction.ast.**FileNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*data: Dict[str, Any]*, *lean_file:* LeanFile) → *FileNode*

class lean_dojo.data_extraction.ast.**GroupNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *GroupNode*

class lean_dojo.data_extraction.ast.**IdentAntiquotNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *IdentAntiquotNode*

get_ident() → str

class lean_dojo.data_extraction.ast.**IdentNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *leading: str*, *trailing: str*, *raw_val: str*, *val: str*, *full_name: str* | *None* = *None*, *mod_name: str* | *None* = *None*, *def_path: str* | *None* = *None*, *def_start:* lean_dojo.data_extraction.lean.Pos | *None* = *None*, *def_end:* lean_dojo.data_extraction.lean.Pos | *None* = *None*)

Bases: *Node*

def_end: *Pos* | None = None

def_path: str | None = None

def_start: *Pos* | None = None

classmethod from_data(*ident_data: Dict[str, Any]*, *lean_file:* LeanFile) → *IdentNode* | None

full_name: str | None = None

property is_mutual: bool

leading: str

mod_name: *str | None = None*

raw_val: *str*

trailing: *str*

val: *str*

class lean_dojo.data_extraction.ast.**LeanBinderidentAntiquotNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *LeanBinderidentAntiquotNode*

get_ident() → str | None

class lean_dojo.data_extraction.ast.**LeanBinderidentNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start:* lean_dojo.data_extraction.lean.Pos | *None, end: lean_dojo.data_extraction.lean.Pos* | *None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *LeanBinderidentNode*

get_ident() → str | None

class lean_dojo.data_extraction.ast.**LeanElabCommandCommandIrreducibleDefNode**(*lean_file: lean_dojo.data_extraction.lean.Lea start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')], name: str | None, full_name: str | None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *LeanElabCommandCommandIrreducibleDefNode*

full_name: *str | None = None*

name: *str | None*

class lean_dojo.data_extraction.ast.**LemmaNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*, *full_name: str* | *None* = *None*, *_is_private_decl: bool* | *None* = *False*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *LemmaNode*

full_name: str | None = None

get_proof_node() → *Node*

has_tactic_proof() → bool

property is_mutual: bool

is_private() → bool

name: str

class lean_dojo.data_extraction.ast.**MathlibTacticLemmaNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*, *full_name: str* | *None* = *None*, *_is_private_decl: bool* | *None* = *False*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *MathlibTacticLemmaNode*

full_name: str | None = None

get_proof_node() → *Node*

has_tactic_proof() → bool

property is_mutual: bool

is_private() → bool

name: str

class lean_dojo.data_extraction.ast.**ModuleHeaderNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *ModuleHeaderNode*

class lean_dojo.data_extraction.ast.**ModuleImportNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *module:* *str* | *None*, *path: pathlib.Path* | *None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *ModuleImportNode*

module: str | None

path: Path | None = None

class lean_dojo.data_extraction.ast.**ModulePreludeNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *ModulePreludeNode*

class lean_dojo.data_extraction.ast.**Node**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: `object`

children: List[*Node*]

end: *Pos* | None

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *Node*

classmethod from_xml(*tree: Element*, *lean_file:* LeanFile) → *Node*

get_closure() → Tuple[*Pos*, *Pos*]

classmethod kind() → str

lean_file: *LeanFile*

start: *Pos* | None

to_xml(*parent: Element*) → None

traverse_postorder(*callback: Callable[[*Node*, List[Any]], Any]*) → Any

traverse_preorder(*callback: Callable[[*Node*, List[*Node*]], Any]*, *node_cls: type* | *None*, *parents: List[*Node*] = []*) → None

class lean_dojo.data_extraction.ast.**NullNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *NullNode*

class lean_dojo.data_extraction.ast.**OtherNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: Optional[lean_dojo.data_extraction.lean.Pos], end: Optional[lean_dojo.data_extraction.lean.Pos], children: List[ForwardRef('Node')], kind: str = <bound method Node.kind of <class 'lean_dojo.data_extraction.ast.OtherNode'>>, state_before: Optional[str] = None, state_after: Optional[str] = None, tactic: Optional[str] = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *OtherNode*

state_after: str | None = None

state_before: str | None = None

tactic: str | None = None

class lean_dojo.data_extraction.ast.**StdTacticAliasAliasNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: str*, *full_name: str | None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *StdTacticAliasAliasNode*

full_name: str | None = None

name: str

class lean_dojo.data_extraction.ast.**StdTacticAliasAliaslrNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*, *name: List[str]*, *full_name: List[str] | None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *StdTacticAliasAliaslrNode*

full_name: List[str] | None = None

property is_mutual: bool

name: List[str]

class lean_dojo.data_extraction.ast.**TacticTacticseq1IndentedAntiquotNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TacticTacticseq1IndentedAntiquotNode*

get_tactic_nodes(*atomic_only: bool = False*) → Generator[*Node*, None, None]

class lean_dojo.data_extraction.ast.**TacticTacticseq1IndentedNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TacticTacticseq1IndentedNode*

get_tactic_nodes(*atomic_only: bool = False*) → Generator[*Node*, None, None]

class lean_dojo.data_extraction.ast.**TacticTacticseqNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TacticTacticseqNode*

get_tactic_nodes(*atomic_only: bool = False*) → Generator[*Node*, None, None]

class lean_dojo.data_extraction.ast.**TacticTacticseqbracketedNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children:* *List[ForwardRef('Node')]*, *state_before: str | None = None*, *state_after: str | None = None*, *tactic: str | None = None*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TacticTacticseqbracketedNode*

get_tactic_nodes(*atomic_only: bool = False*) → Generator[*Node*, None, None]

state_after: **str | None = None**

state_before: **str | None = None**

tactic: **str | None = None**

property tactic_nodes: **List[*Node*]**

class lean_dojo.data_extraction.ast.**TermAttrkindAntiquotNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermAttrkindAntiquotNode*

class lean_dojo.data_extraction.ast.**TermAttrkindNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermAttrkindNode*

class lean_dojo.data_extraction.ast.**TermBytacticNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermBytacticNode*

class lean_dojo.data_extraction.ast.**TermExplicitbinderNode**(*lean_file: lean_dojo.data_extraction.lean.LeanFile, start: lean_dojo.data_extraction.lean.Pos | None, end: lean_dojo.data_extraction.lean.Pos | None, children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermExplicitbinderNode*

class lean_dojo.data_extraction.ast.**TermHoleNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermHoleNode*

class lean_dojo.data_extraction.ast.**TermTypespecNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TermTypespecNode*

class lean_dojo.data_extraction.ast.**TokenAntiquotNode**(*lean_file:* lean_dojo.data_extraction.lean.LeanFile, *start:* lean_dojo.data_extraction.lean.Pos | *None*, *end:* lean_dojo.data_extraction.lean.Pos | *None*, *children: List[ForwardRef('Node')]*)

Bases: *Node*

classmethod from_data(*node_data: Dict[str, Any]*, *lean_file:* LeanFile) → *TokenAntiquotNode*

lean_dojo.data_extraction.ast.**contains_tactic**(*node:* Node) → bool

lean_dojo.data_extraction.ast.**is_leaf**(*node:* Node) → bool

lean_dojo.data_extraction.ast.**is_mutual_lean4**(*node:* Node) → bool

lean_dojo.data_extraction.ast.**is_potential_premise_lean4**(*node:* Node) → bool

Check if node is a theorem/definition that can be used as a premise.

3.7.2 lean_dojo.interaction

lean_dojo.interaction.dojo

class lean_dojo.interaction.dojo.**CommandState**(*id: int*, *message: str | None = None*)

Bases: object

id: int

message: str | None = None

class lean_dojo.interaction.dojo.**Dojo**(*entry:* Theorem | *Tuple[*LeanGitRepo*, Path, int]*, *hard_timeout: float | None = None*, *additional_imports: List[str] = []*)

Bases: object

Gym-like environment for programmatic interaction with Lean through tactics or commands.

additional_imports: List[str]

entry: *Theorem* | Tuple[*LeanGitRepo*, Path, int]

file_path: Path

hard_timeout: float | None

has_timedout: bool = False

is_crashed: bool = False

is_successful: bool | None = None

repo: *LeanGitRepo*

run_cmd(*state:* CommandState, *command: str*) → *CommandState* | *LeanError* | *TimeoutError*

run_tac(*state:* TacticState, *tactic: str*) → *TacticState* | *ProofFinished* | *LeanError* | *TimeoutError* | *ProofGivenUp*

property uses_commands: bool

property uses_tactics: bool

exception lean_dojo.interaction.dojo.**DojoCrashError**

Bases: Exception

property is_out_of_memory: bool

exception lean_dojo.interaction.dojo.**DojoHardTimeoutError**

Bases: Exception

exception lean_dojo.interaction.dojo.**DojoInitError**

Bases: Exception

class lean_dojo.interaction.dojo.**LeanError**(*error: str*)

Bases: object

error: str

class lean_dojo.interaction.dojo.**ProofFinished**(*tactic_state_id: int*, *message: str | None = None*)

Bases: object

message: str | None = None

tactic_state_id: int

class lean_dojo.interaction.dojo.**ProofGivenUp**

Bases: object

class lean_dojo.interaction.dojo.**TacticState**(*pp: str*, *id: int*, *message: str | None = None*)

Bases: object

goals: List[*Goal*]

id: int

message: str | None = None

property num_goals: int

pp: str

class lean_dojo.interaction.dojo.**TimeoutError**(*error: str*)

Bases: object

error: str

lean_dojo.interaction.parse_goals

Utilities for parsing Lean's pretty-printed proof goals.

class lean_dojo.interaction.parse_goals.**Declaration**(*ident: str*, *lean_type: str*)

Bases: object

A declaration in the local context.

ident: str

lean_type: str

class lean_dojo.interaction.parse_goals.**Goal**(*assumptions: List[*Declaration*]*, *conclusion: str*)

Bases: object

A goal in Lean.

assumptions: List[*Declaration*]

conclusion: str

classmethod from_pp(*pp: str*) → *Goal*

Parse a pretty-printed goal.

lean_dojo.interaction.parse_goals.**parse_goals**(*pp: str*) → List[*Goal*]

Parse a list of pretty-printed goals.

3.7.3 lean_dojo.constants

Constants controlling LeanDojo's behaviors. Many of them are configurable via *Environment Variables*.

lean_dojo.constants.**CACHE_DIR = PosixPath('/home/docs/.cache/lean_dojo')**

Cache directory for storing traced repos (see *Caching*).

lean_dojo.constants.**CONTAINER = 'native'**

Container to use for running LeanDojo. Default to `native` but also support `docker`. Using `docker` is recommended for Lean 3.

lean_dojo.constants.**DISABLE_REMOTE_CACHE = False**

Whether to disable remote caching (see *Caching*) and build all repos locally.

lean_dojo.constants.**LEAN4_PACKAGES_DIR = PosixPath('.lake/packages')**

The directory where Lean 4 dependencies are stored (since v4.3.0-rc2).

lean_dojo.constants.**LEAN4_URL = 'https://github.com/leanprover/lean4'**

The URL of the Lean 4 repo.

lean_dojo.constants.**LOAD_USED_PACKAGES_ONLY = False**

Only load depdendency files that are actually used by the target repo.

lean_dojo.constants.**NUM_PROCS = 2**

Number of threads to use

lean_dojo.constants.**REMOTE_CACHE_URL = 'https://lean-dojo.s3.amazonaws.com'**

URL of the remote cache (see *Caching*).

lean_dojo.constants.**TACTIC_CPU_LIMIT = 1**

Number of CPUs for executing tactics when interacting with Lean (only useful when running within Docker).

lean_dojo.constants.**TACTIC_MEMORY_LIMIT = '32g'**

Maximum memory when interacting with Lean (only useful when running within Docker).

lean_dojo.constants.**TMP_DIR = None**

Temporary directory used by LeanDojo for storing intermediate files

lean_dojo.constants.**check_git_version**(*min_version: Tuple[int, int, int]*) → Tuple[int, int, int]

Check the version of Git installed on the system.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

INDEX

Symbols

A

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V