半机械化数学定理推导系统主程序代码

**sympy/axiom/prove.py**

# coding=utf-8

import os

import re

import axiom # @UnusedImport

from sympy.utilities.miscellany import Text

import time

from os.path import getsize

from multiprocessing import cpu\_count

from queue import PriorityQueue

from functools import singledispatch

import random

def **axiom\_directory**():

return os.path.dirname(\_\_file\_\_)

# initialization

count = 0

unproved = []

failures = []

websites = []

insurmountable = {\*Text(axiom\_directory() + *'/insurmountable.txt'*)}

unprovable = {\*Text(axiom\_directory() + *'/unprovable.txt'*)}

insurmountable |= unprovable

def **readFolder**(rootdir, sufix=*'.py'*):

for name in os.listdir(rootdir):

path = os.path.join(rootdir, name)

if path.endswith(sufix):

name = name[:-len(sufix)]

if name == *'\_\_init\_\_'*:

continue

path = path[:-len(sufix)]

paths = re.compile(*r'[\\/]+'*).split(path)

# print(path)

index = paths.index(*'axiom'*)

package = *'.'*.join(paths[index:])

global count

count += 1

path += *'.php'*

if os.path.isfile(path):

with open(path, *"r"*, encoding=*'utf8'*) as file:

line = file.readline()

m = re.compile(*r"<p style='display:none'>timing = ([\d.]+)</p>"*).match(line)

if m:

timing = float(m.group(1))

else:

timing = getsize(path) / 500

else:

timing = random.random()

yield package, timing

elif os.path.isdir(path):

yield from readFolder(path, sufix)

def **project\_directory**():

return os.path.dirname(axiom\_directory())

def **working\_directory**():

return os.path.dirname(project\_directory())

def **create\_module**(package, module):

sep = os.path.sep

dirname = project\_directory()

\_\_init\_\_ = dirname + sep + package.replace(*'.'*, sep) + sep + *'\_\_init\_\_.py'*

print(*'editing'*, \_\_init\_\_)

hit = False

file = Text(\_\_init\_\_)

for line in file:

m = re.compile(*'from \. import (\w+)'*).match(line)

if m and m.group(1) == module:

hit = True

break

if not hit:

addition = *'from . import %s'* % module

last\_char = file.get\_last\_char()

if last\_char and last\_char != *'\n'*:

addition = *'\n'* + addition

file.append(addition)

def **run**(package):

command = *'python %s %s debug=True'* % (project\_directory() + os.path.sep + *'run.py'*, package)

return os.system(command)

# for line in os.popen(cmd).readlines():

# print(line)

def **import\_module**(package):

try:

return eval(package)

except AttributeError as e:

print(e)

s = str(e)

m = re.compile(*"module '([\w\.]+)' has no attribute '(\w+)'"*).fullmatch(s)

assert m

create\_module(\*m.groups())

print(package, *'is created newly'*)

return run(package)

*@singledispatch*

def **process**(package, debug=False):

if debug:

print(package)

module = import\_module(package)

if isinstance(module, int):

sep = os.path.sep

ret = None if module < 0 else bool(module)

file = project\_directory() + sep + package.replace(*'.'*, sep) + *'.py'*

else:

file = module.\_\_file\_\_

ret = module.prove(file, debug=debug)

if package in insurmountable:

ret = True

return file, ret

*@process.register*(list)

def **\_**(packages, debug=False):

return [process(package, debug=debug) for package in packages]

start = time.time()

def **prove**(debug=False, parallel=True):

rootdir = axiom\_directory()

def **generator**():

for name in os.listdir(rootdir):

path = os.path.join(rootdir, name)

if os.path.isdir(path):

yield from readFolder(path)

tasks = {task : timing for task, timing in generator()}

packages = tuple([] for \_ in range(cpu\_count() \* 2))

timings = [0 for \_ in range(cpu\_count() \* 2)]

total\_timing = sum(timing for task, timing in tasks.items())

average\_timing = total\_timing / len(packages)

print(*'total\_timing ='*, total\_timing)

print(*'average\_timing ='*, average\_timing)

# tasks = {'axiom.algebre.unequal.equal.imply.equal': 0}

tasks = [\*tasks.items()]

tasks.sort(key=lambda pair : pair[1], reverse=True)

pq = PriorityQueue()

for i, t in enumerate(timings):

pq.put((t, i))

for task, timing in tasks:

t, i = pq.get()

packages[i].append(task)

timings[i] += timing

pq.put((timings[i], i))

for proc, timing in zip(packages, timings):

print(*'timing ='*, timing)

print(*'python run.py '* + *' '*.join(proc))

print(*'total timing ='*, sum(timings))

for array in process(packages, debug=debug, parallel=parallel):

post\_process(array)

print(*'in all %d axioms'* % count)

print\_summary()

def **print\_summary**():

if unproved:

print(*'unproved:'*)

for p in unproved:

print(p)

if failures:

print(*'failures:'*)

for p in failures:

print(p)

if websites:

print(*'websites:'*)

for p in websites:

print(p)

timing = time.time() - start

print(*'seconds costed ='*, timing)

print(*'minutes costed ='*, timing / 60)

print(*'total unproved ='*, len(unproved))

print(*'total failures ='*, len(failures))

def **post\_process**(result):

for package, ret in result:

if ret is False:

unproved.append(package)

elif ret is None:

failures.append(package)

else:

continue

# print('\_\_file\_\_ =', \_\_file\_\_)

# print('working\_directory() =', working\_directory())

# print('package =', package)

websites.append(*"http://localhost"* + package[len(working\_directory()):-3] + *".php"*)

return count

def **process\_debug**(packages):

return process(packages, debug=True)

*@process.register*(tuple)

def **\_**(items, debug=False, parallel=True): # @DuplicatedSignature

proc = process\_debug if debug else process

if parallel:

from multiprocessing import Pool

with Pool(processes=cpu\_count()) as pool:

# with Pool(processes=cpu\_count() \* 2) as pool:

return pool.map(proc, items)

else:

return map(proc, items)

# Reverse[Reverse[Minors[mat], 1], 2] == Map[Reverse, Minors[mat], {0, 1}]

# adj[m\_] := Map[Reverse, Minors[Transpose[m], Length[m] - 1], {0, 1}] Table[(-1)^(i + j), {i, Length[m]}, {j, Length[m]}]

# to create a matrix symbol

# $Assumptions = M \[Element] Matrices[{n, n}, Reals, Symmetric[{1, 2}]]

# Normal[SparseArray[{{i\_, i\_} -> i^2}, {10, 10}]] // MatrixForm

if \_\_name\_\_ == *'\_\_main\_\_'*:

# prove(debug=True, parallel=False)

# prove(debug=True)

prove()

**sympy/axiom/utility.py**

import sympy

import os

from sympy.logic.boolalg import equivalent\_ancestor, Boolean

import traceback

from sympy.logic import boolalg

from sympy.utilities.iterables import topological\_sort\_depth\_first

import time

def **init**(func):

def **\_func**(\*args, \*\*kwrags):

Eq.clear()

func(\*args, \*\*kwrags)

return \_func

sympy.init\_printing()

# https://www.programiz.com/python-programming/operator-overloading

class **Eq**:

slots = {*'list'*, *'file'*, *'timing'*, *'debug'*}

def **\_\_init\_\_**(*self*, php\_file, debug=True):

from sympy.utilities.miscellany import Text

*self*.\_\_dict\_\_[*'list'*] = []

*self*.\_\_dict\_\_[*'file'*] = Text(php\_file)

*self*.\_\_dict\_\_[*'timing'*] = time.time()

*self*.\_\_dict\_\_[*'debug'*] = debug

*self*.file.clear()

php = *self*.file.file.name

# sep = os.sep

php = php.replace(*'\\'*, *'/'*)

render\_php = re.compile(*r'/\w+'*).sub(*'/render'*, re.compile(*r'\w+/'*).sub(*'../'*, php[len(os.path.dirname(\_\_file\_\_)) + 1:]))

php\_code = *"""\*

*<?php*

*require\_once '%s';*

*render(\_\_FILE\_\_);*

*"""* % render\_php

*self*.file.write(php\_code)

def **\_\_del\_\_**(*self*):

# print('calling destructor')

*self*.file.home()

# sep = os.sep

lines = []

lines.append(*"<p style='display:none'>timing = %s</p>"* % (time.time() - *self*.timing))

for line in *self*.file:

if not line.startswith(*'//'*):

lines.append(line)

continue

i = 0

res = []

for m in re.finditer(*r"\\tag\\*{(Eq(?:\[(\d+)\]|\.(\w+)))}"*, line):

expr, index, attr = m.group(1), m.group(2), m.group(3)

if i < m.start():

res.append(line[i:m.start()])

assert line[m.start():m.end()] == m.group(0)

assert line[m.start(1):m.end(1)] == m.group(1)

if index:

assert line[m.start(2):m.end(2)] == m.group(2)

if attr:

assert line[m.start(3):m.end(3)] == m.group(3)

if index:

index = int(index)

eq = *self*[index]

else:

index = attr

eq = getattr(*self*, attr)

res.append(line[m.start():m.start(1)])

if eq.plausible:

\_expr = Eq.reference(*self*.get\_index(Eq.get\_equivalent(eq)))

if *self*.debug:

print(*"%s=>%s : %s"* % (\_expr, expr, eq))

res.append(\_expr)

res.append(*'=>'*)

elif eq.plausible == False:

res.append(*'~'*)

res.append(expr)

res.append(line[m.end(1):m.end()])

i = m.end()

res.append(line[i:])

# lines.append('$text[] = "%s";' % ''.join(res).replace('\\', '\\\\'))

lines.append(*''*.join(res))

*self*.file.write(lines)

*self*.file.append(*"?>"*)

*@staticmethod*

def **reference**(index):

if isinstance(index, list):

return *', '*.join(Eq.reference(d) for d in index)

elif isinstance(index, int):

if index < 0:

return *"?"*

else:

return *"Eq[%d]"* % index

else:

return *"Eq.%s"* % index

*@staticmethod*

def **get\_equivalent**(eq):

if eq.equivalent is not None:

return eq.equivalent

elif eq.given is not None:

return eq.given

elif eq.imply is not None:

return eq.imply

def **get\_index**(*self*, equivalent):

if equivalent is None:

return -1

if isinstance(equivalent, (list, tuple, set)):

\_index = []

for eq in equivalent:

if eq.plausible:

\_index.append(*self*.get\_index(eq))

if len(\_index) == 1:

\_index = \_index[0]

if not \_index:

return -1

else:

\_index = *self*.index(equivalent, False)

if \_index == -1:

equivalent = Eq.get\_equivalent(equivalent)

return *self*.get\_index(equivalent)

return \_index

*@property*

def **plausibles\_dict**(*self*):

plausibles = {i: eq for i, eq in enumerate(*self*) if eq.plausible}

for k in *self*.\_\_dict\_\_.keys() - *self*.slots:

v = *self*.\_\_dict\_\_[k]

if v.plausible:

plausibles[k] = v

return plausibles

def **index**(*self*, eq, dummy\_eq=True):

for i, \_eq in enumerate(*self*.list):

if \_eq == eq or (dummy\_eq and eq.dummy\_eq(\_eq)):

return i

for k in *self*.\_\_dict\_\_.keys() - *self*.slots:

v = *self*.\_\_dict\_\_[k]

if eq == v or (dummy\_eq and eq.dummy\_eq(v)):

return k

return -1

def **append**(*self*, eq):

*self*.list.append(eq)

return len(*self*.list) - 1

def **\_\_getitem\_\_**(*self*, index):

if isinstance(index, int):

return *self*.list[index]

return *self*.\_\_dict\_\_[index]

def **process**(*self*, rhs, index=None, flush=True):

latex = rhs.latex

infix = str(rhs)

if isinstance(rhs, Boolean):

index = *self*.add\_to\_list(rhs, index)

if index != -1:

if isinstance(index, int):

index = *'Eq[%d]'* % index

else:

index = *'Eq.%s'* % index

tag = *r'\tag\*{%s}'* % index

latex += tag

infix = *'%s : %s'* % (index, infix)

if *self*.debug:

print(infix)

latex = *r'\[%s\]'* % latex

# latex = r'\(%s\)' % latex

# http://www.public.asu.edu/~rjansen/latexdoc/ltx-421.html

if flush:

*self*.file.append(*'//'* + latex)

else:

return latex

def **\_\_setattr\_\_**(*self*, index, rhs):

if index in *self*.\_\_dict\_\_:

eq = *self*.\_\_dict\_\_[index]

if eq.plausible:

assert rhs.is\_equivalent\_of(eq) or rhs.is\_given\_by(eq)

*self*.process(rhs, index)

def **add\_to\_list**(*self*, rhs, index=None):

old\_index = *self*.index(rhs)

if old\_index == -1:

if rhs.is\_BooleanAtom:

boolalg.process\_options(value=bool(rhs), \*\*rhs.\_assumptions)

return -1

if index is not None:

*self*.\_\_dict\_\_[index] = rhs

return index

return *self*.append(rhs)

else:

lhs = *self*[old\_index]

plausible = rhs.plausible

if plausible is False:

lhs.plausible = False

elif plausible is None:

if lhs.plausible:

lhs.plausible = True

else:

if lhs.plausible is None:

given = rhs.given

equivalent = rhs.equivalent

rhs.plausible = True

if given is None:

if equivalent is not None:

if not isinstance(equivalent, (list, tuple)):

equivalent.equivalent = lhs

elif not isinstance(given, (list, tuple)):

derivative = given.derivative

if isinstance(derivative, (list, tuple)):

if all(eq.plausible is None for eq in derivative):

given.plausible = True

elif lhs.plausible is False:

rhs.plausible = False

else:

if isinstance(rhs.equivalent, (list, tuple)):

if any(lhs is \_eq for \_eq in rhs.equivalent):

return old\_index

if rhs.given is not None:

if isinstance(rhs.given, (list, tuple)):

if any(lhs is \_eq for \_eq in rhs.given):

return old\_index

else:

if rhs.given.plausible is False:

eqs = [eq for eq in rhs.given.derivative if lhs.plausible is not None]

if len(eqs) == 1:

eqs[0].plausible = False

if rhs.equivalent is not lhs and rhs is not lhs:

lhs\_is\_plausible = *'plausible'* in lhs.\_assumptions

rhs\_equivalent = equivalent\_ancestor(rhs)

if len(rhs\_equivalent) == 1:

rhs\_equivalent, \*\_ = rhs\_equivalent

if lhs != rhs\_equivalent or rhs.given is not None:

rhs\_plausibles, rhs\_is\_equivalent = rhs\_equivalent.plausibles\_set()

if len(rhs\_plausibles) == 1:

rhs\_plausible, \*\_ = rhs\_plausibles

if rhs\_plausible is not lhs:

if rhs\_is\_equivalent:

lhs\_plausibles, lhs\_is\_equivalent = lhs.plausibles\_set()

if len(lhs\_plausibles) == 1:

lhs\_plausible, \*\_ = lhs\_plausibles

if lhs\_is\_equivalent:

lhs\_plausible.equivalent = rhs\_plausible

else:

rhs\_plausible.given = lhs\_plausible

else:

rhs\_plausible.equivalent = lhs

else:

lhs\_plausibles, lhs\_is\_equivalent = lhs.plausibles\_set()

if lhs\_is\_equivalent:

assert rhs\_plausible not in lhs\_plausibles, *'cyclic proof detected'*

lhs\_plausibles = [\*lhs\_plausibles]

if len(lhs\_plausibles) == 1:

lhs\_plausible, \*\_ = lhs\_plausibles

lhs\_plausible.given = rhs\_plausible

else:

rhs\_plausible.imply = lhs\_plausibles

else:

plausibles\_set, is\_equivalent = lhs.plausibles\_set()

if len(plausibles\_set) == 1:

lhs\_plausible, \*\_ = plausibles\_set

if is\_equivalent:

if rhs\_is\_equivalent:

rhs\_plausibles.discard(lhs\_plausible)

lhs\_plausible.equivalent = [\*rhs\_plausibles]

else:

assert lhs\_plausible not in rhs\_plausibles, *'cyclic proof detected'*

lhs\_plausible.given = [\*rhs\_plausibles]

else:

lhs\_plausible.imply = rhs\_equivalent

else:

rhs\_plausibles, rhs\_is\_equivalent = rhs.plausibles\_set()

if len(rhs\_plausibles) == 1:

rhs\_plausible, \*\_ = rhs\_plausibles

else:

lhs\_plausibles, lhs\_is\_equivalent = lhs.plausibles\_set()

if len(lhs\_plausibles) == 1:

lhs\_plausible, \*\_ = lhs\_plausibles

if rhs\_is\_equivalent and lhs\_is\_equivalent:

...

else:

if lhs\_plausible not in rhs\_plausibles:

lhs\_plausible.given = [\*rhs\_plausibles]

if lhs\_is\_plausible:

if *'imply'* not in rhs.\_assumptions:

rhs = lhs

if isinstance(old\_index, int):

*self*.list[old\_index] = rhs

else:

*self*.\_\_dict\_\_[old\_index] = rhs

return old\_index

def **return\_index**(*self*, index, rhs):

if isinstance(index, int):

*self*.list[index] = rhs

else:

*self*.\_\_dict\_\_[index] = rhs

return index

def **\_\_lshift\_\_**(*self*, rhs):

if isinstance(rhs, (list, tuple)):

def **yield\_from**(container):

for e in container:

if isinstance(e, (list, tuple)):

yield from yield\_from(e)

else:

yield *self*.process(e, flush=False)

*self*.file.append(*'//'* + *''*.join(yield\_from(rhs)))

else:

*self*.process(rhs)

return *self*

def **\_\_ilshift\_\_**(*self*, rhs):

return *self* << rhs

def **show\_latex**():

import matplotlib.pyplot as plt

ax = plt.subplot(111)

# defaultFamily

ax.text(0.1, 0.8, *r"$\int\_a^b f(x)\mathrm{d}x$"*, fontsize=30, color=*"red"*)

ax.text(0.1, 0.3, *r"$\sum\_{n=1}^\infty\frac{-e^{i\pi}}{2^n}!$"*, fontsize=30)

plt.show()

# https://www.cnblogs.com/chaosimple/p/4031421.html

def **test\_latex\_parser**():

from sympy.parsing.latex import parse\_latex

expr = parse\_latex(*r"\frac {1 + \sqrt {\a}} {\b}"*) # doctest: +SKIP

print(expr)

def **topological\_sort**(graph):

in\_degrees = {u: 0 for u in graph}

vertex\_num = len(in\_degrees)

for u in graph:

for v in graph[u]:

in\_degrees[v] += 1

Q = [u for u in in\_degrees if in\_degrees[u] == 0]

Seq = []

while Q:

u = Q.pop()

Seq.append(u)

for v in graph[u]:

in\_degrees[v] -= 1

if in\_degrees[v] == 0:

Q.append(v)

if len(Seq) == vertex\_num:

return Seq

# print("there's a circle.")

return None

def **wolfram\_decorator**(py, func, debug=True, \*\*kwargs):

eqs = Eq(py.replace(*'.py'*, *'.php'*), debug=debug)

website = *"http://localhost"* + func.\_\_code\_\_.co\_filename[len(os.path.dirname(os.path.dirname(os.path.dirname(\_\_file\_\_)))):-3] + *".php"*

try:

wolfram = kwargs[*'wolfram'*]

with wolfram:

func(eqs, wolfram)

except Exception as e:

print(e)

traceback.print\_exc()

print(website)

return

if debug:

print(website)

plausibles = eqs.plausibles\_dict

if plausibles:

return False

return True

def **prove**(func):

def **prove**(py, func, debug=True):

eqs = Eq(py.replace(*'.py'*, *'.php'*), debug=debug)

website = *"http://localhost"* + func.\_\_code\_\_.co\_filename[len(os.path.dirname(os.path.dirname(os.path.dirname(\_\_file\_\_)))):-3] + *".php"*

try:

func(eqs)

except Exception as e:

print(e)

traceback.print\_exc()

print(website)

return

if debug:

print(website)

plausibles = eqs.plausibles\_dict

if plausibles:

return False

return True

return lambda py, \*\*kwargs: prove(py, func, \*\*kwargs)

def **wolfram**(func):

def **decorator**(func):

from wolframclient.evaluation.cloud import cloudsession

session = cloudsession.session

# from wolframclient.evaluation.kernel.localsession import WolframLanguageSession

# session = WolframLanguageSession()

return lambda py, \*\*kwargs: wolfram\_decorator(py, func, wolfram=session, \*\*kwargs)

return decorator

def **apply**(\*args, \*\*kwargs):

if args:

assert len(args) == 1

axiom = args[0]

if axiom.\_\_module\_\_ == *'\_\_main\_\_'*:

paths = axiom.\_\_code\_\_.co\_filename[len(os.path.dirname(\_\_file\_\_)):].split(os.sep)

else:

paths = axiom.\_\_module\_\_.split(*'.'*)

if *'given'* in paths:

return given(axiom, \*\*kwargs)

else:

return imply(axiom, \*\*kwargs)

else:

return lambda axiom: apply(axiom, \*\*kwargs)

def **imply**(apply, \*\*kwargs):

is\_given = kwargs[*'given'*] if *'given'* in kwargs else True

simplify = kwargs[*'simplify'*] if *'simplify'* in kwargs else True

def **add**(given, statement):

if isinstance(statement, tuple):

if given is None:

return statement

if isinstance(given, list):

return tuple(given) + statement

return (given,) + statement

if given is None:

return statement

if isinstance(given, list):

return tuple(given) + (statement,)

return (given, statement)

def **process**(s, dependency):

s.definition\_set(dependency)

if *'plausible'* not in s.\_assumptions:

s.\_assumptions[*'plausible'*] = True

def **imply**(\*args, \*\*kwargs):

nonlocal simplify

simplify = kwargs.pop(*'simplify'*, True) and simplify

statement = apply(\*args, \*\*kwargs)

if isinstance(statement, tuple):

for s in statement:

if s.equivalent is not None:

s.equivalent = None

elif statement.equivalent is not None:

statement.equivalent = None

if is\_given:

given = [eq for eq in args if isinstance(eq, Boolean)]

if len(given) == 1:

given = given[0]

elif not given:

given = None

else:

given = None

s = traceback.extract\_stack()

if apply.\_\_code\_\_.co\_filename != s[-2][0]:

if given is not None:

if isinstance(statement, tuple):

statement = [s.copy(given=given) for s in statement]

else:

statement = statement.copy(given=given)

if not simplify:

return statement

if isinstance(statement, list):

return [\*(s.simplify() for s in statement)]

return statement.simplify()

dependency = {}

if isinstance(statement, tuple):

for s in statement:

process(s, dependency)

else:

process(statement, dependency)

if given is not None:

if isinstance(given, (tuple, list)):

for g in given:

g.definition\_set(dependency)

else:

given.definition\_set(dependency)

G = topological\_sort\_depth\_first(dependency)

if G:

definition = [s.equality\_defined() for s in G]

statement = add(given, statement)

if isinstance(statement, tuple):

return definition + [\*statement]

return definition + [statement]

else:

return add(given, statement)

return imply

def **given**(apply):

def **add**(given, statement):

if isinstance(statement, tuple):

if given is None:

return statement

if isinstance(given, tuple):

return tuple(given) + statement

return (given,) + statement

if given is None:

return statement

if isinstance(given, tuple):

return tuple(given) + (statement,)

return (given, statement)

def **process**(s, dependency):

s.definition\_set(dependency)

if *'plausible'* in s.\_assumptions:

del s.\_assumptions[*'plausible'*]

def **given**(\*args, \*\*kwargs):

simplify = kwargs.pop(*'simplify'*, True)

statement = apply(\*args, \*\*kwargs)

assert not isinstance(statement, tuple)

if statement.equivalent is not None:

statement.equivalent = None

imply, \*args = args

given = tuple(eq for eq in args if isinstance(eq, Boolean))

assert all(g.plausible is None for g in given)

assert imply.is\_Boolean

s = traceback.extract\_stack()

if apply.\_\_code\_\_.co\_filename != s[-2][0]:

statement = statement.copy(imply=imply)

if not simplify:

return statement

if isinstance(statement, list):

return [\*(s.simplify() for s in statement)]

return statement.simplify()

dependency = {}

process(statement, dependency)

for g in given:

g.definition\_set(dependency)

imply.definition\_set(dependency)

imply.\_assumptions[*'plausible'*] = True

G = topological\_sort\_depth\_first(dependency)

if G:

definition = [s.equality\_defined() for s in G]

statement = add((imply,) + given, statement)

if isinstance(statement, tuple):

return definition + [\*statement]

return definition + [statement]

else:

return add((imply,) + given, statement)

return given

import inspect

import re

from itertools import dropwhile

# https://cloud.tencent.com/developer/ask/222013

def **get\_function\_body**(func):

print()

print(*"{func.\_\_name\_\_}'s body:"*.format(func=func))

source\_lines = inspect.getsourcelines(func)[0]

source\_lines = dropwhile(lambda x: x.startswith(*'@'*), source\_lines)

source = *''*.join(source\_lines)

pattern = re.compile(*r'(async\s+)?def\s+\w+\s\*\(.\*?\)\s\*:\s\*(.\*)'*, flags=re.S)

lines = pattern.search(source).group(2).splitlines()

if len(lines) == 1:

return lines[0]

else:

indentation = len(lines[1]) - len(lines[1].lstrip())

return *'\n'*.join([lines[0]] + [line[indentation:] for line in lines[1:]])

def **assert\_hashly\_equal**(lhs, rhs):

assert lhs.\_hashable\_content() == rhs.\_hashable\_content(), *"hash(%s) != hash(%s), \nsince %s \n!= \n%s"* % (lhs, rhs, lhs.\_hashable\_content(), rhs.\_hashable\_content())

if \_\_name\_\_ == *'\_\_main\_\_'*:

...

**sympy/axiom/plausibles.py**

# coding=utf-8

import os

import re

from sympy.utilities.miscellany import Text

from \_collections import defaultdict

def **axiom\_directory**():

return os.path.dirname(\_\_file\_\_)

def **read\_directory**(dir):

for name in os.listdir(dir):

path = os.path.join(dir, name)

if os.path.isdir(path):

yield path

def **read\_all\_php**(dir):

for directory in read\_directory(dir):

for php in read\_all\_files(directory, *'.php'*):

yield php

def **read\_all\_files**(rootdir, sufix=*'.py'*):

for name in os.listdir(rootdir):

path = os.path.join(rootdir, name)

if path.endswith(sufix):

yield path

elif os.path.isdir(path):

yield from read\_all\_files(path, sufix)

def **read\_all\_plausibles**(plausible):

count = 0

for php in read\_all\_php(os.path.dirname(\_\_file\_\_)):

py = php[:-3] + *'py'*

if not os.path.exists(py):

print(php + *" is an obsolete file since its py file is deleted!"*)

os.unlink(php)

continue

count += 1

if is\_axiom\_plausible(php):

axiom = to\_python\_module(php)

sec = section(axiom)

if sec in insurmountable:

if axiom in insurmountable[sec]:

continue

if sec in unprovable:

if axiom in unprovable[sec]:

continue

plausible[sec].append(axiom)

def **section**(axiom):

\_, section, \*\_ = axiom.split(*'.'*, 3)

return section

def **is\_axiom\_plausible**(php):

for statement in yield\_from\_php(php):

matches = is\_latex(statement)

for match in matches:

if re.compile(*".+tag\\*\{(.+=>.+)\}.+"*).search(match.group()):

return True

return False

def **is\_latex**(latex):

return re.compile(*'\\\\\[.+?\\\\\]'*).finditer(latex)

sagemath = os.path.basename(os.path.dirname(os.path.dirname(\_\_file\_\_)))

insurmountable = defaultdict(list)

for axiom in Text(axiom\_directory() + *'/insurmountable.txt'*):

insurmountable[section(axiom)].append(axiom)

unprovable = defaultdict(list)

for axiom in Text(axiom\_directory() + *'/unprovable.txt'*):

unprovable[section(axiom)].append(axiom)

def **get\_extension**(file):

return os.path.splitext(file)[-1]

def **to\_python\_module**(py):

global sagemath

module = []

pythonFile = py

while True:

dirname = os.path.dirname(pythonFile)

basename = os.path.basename(pythonFile)

if basename == sagemath:

break

module.append(basename)

pythonFile = dirname

module[0] = module[0][:-len(get\_extension(module[0]))]

module.reverse()

module = *'.'*.join(module)

return module

def **yield\_from\_php**(php):

for statement in Text(php):

if not statement.startswith(*r"//"*):

continue

statement = statement[2:]

yield statement

if \_\_name\_\_ == *'\_\_main\_\_'*:

plausible = defaultdict(list)

read\_all\_plausibles(plausible)

prefix = os.path.dirname(axiom\_directory())

print(*'axioms plausible:'*)

for section in plausible:

for axiom in plausible[section]:

print(prefix + *'/'* + axiom.replace(*'.'*, *'/'*) + *'.py'*)

**sympy/axiom/index.php**

<?php

**include\_once** 'index.html';

**require\_once** 'utility.php';

?>

the whole math theory is composed of the following sections:

<form style="float: *right*" name=*search* enctype=*"multipart/form-data"*

method=*"post"* action=*"search.php"*>

<input type=*text* spellcheck=*false* name=*keyword* size=*"48"* value=*""*

placeholder=*'input a hint for search of a theorem/axiom'*><br> <input

type=*checkbox* name=*CaseSensitive*><u>C</u>ase sensitive <input

type=*checkbox* name=*WholeWord*><u>W</u>hole word <input type=*checkbox*

name=*RegularExpression*>Regular e<u>x</u>pression

</form>

<br>

<?php

**function** *yield\_empty\_directory*($dir)

{

$empty = **true**;

**foreach** (*read\_files*($dir, 'py') **as** $py) {

**if** (! *strcmp*(*basename*($py), '\_\_init\_\_.py')) {

**continue**;

}

$empty = **false**;

}

**foreach** (*read\_directory*($dir) **as** $directory) {

**if** (! *strcmp*(*basename*($directory), '\_\_pycache\_\_')) {

**continue**;

}

$array = *iterator\_to\_array*(*yield\_empty\_directory*($directory));

**if** (**empty**($array)) {

$empty = **false**;

} **else** {

**if** (*strcmp*(*end*($array), $directory)) {

$empty = **false**;

}

// is not empty;

**foreach** ($array **as** $directory) {

**yield** $directory;

}

}

}

**if** ($empty)

**yield** $dir;

}

**function** *read\_all\_php*($dir)

{

**foreach** (*read\_directory*($dir) **as** $directory) {

**foreach** (*read\_all\_files*($directory, 'php') **as** $php) {

**yield** $php;

}

}

}

**function** *read\_from*($file, $trim = **true**)

{

**if** (*file\_exists*($file)) {

$handle = *fopen*($file, "r");

**while** (($buffer = *fgets*($handle, 4096)) !== **false**) {

**if** ($trim) {

$buffer = *trim*($buffer);

**if** (**empty**($buffer)) {

**continue**;

}

}

**yield** $buffer;

}

**if** (! *feof*($handle)) {

**echo** "Error: unexpected fgets() fail\n";

}

*fclose*($handle);

}

}

**function** *is\_axiom\_plausible*($php)

{

**foreach** (*yield\_from\_php*($php) **as** &$statement) {

**if** (*is\_latex*($statement, $matches)) {

**foreach** ($matches **as list** ($match)) {

**if** (*preg\_match*("/.+tag\\*\{(.+=>.+)\}.+/", $match, $result)) {

**return true**;

}

}

}

}

**return false**;

}

**function** *accumulate*($dict)

{

$sum = 0;

**foreach** ($dict **as** $key => $value) {

$sum += *count*($value);

}

**return** $sum;

}

**global** $sagemath;

$unprovable = [];

$insurmountable = [];

$plausible = [];

**function** *section*($axiom)

{

**list** (, $section,) = *explode*('.', $axiom, 3);

**return** $section;

}

**foreach** (*read\_from*(*dirname*(*\_\_file\_\_*) . '/insurmountable.txt') **as** $axiom) {

$insurmountable[*section*($axiom)][] = $axiom;

}

**foreach** (*read\_from*(*dirname*(*\_\_file\_\_*) . '/unprovable.txt') **as** $axiom) {

$unprovable[*section*($axiom)][] = $axiom;

}

$count = 0;

**foreach** (*read\_all\_php*(*dirname*(*\_\_file\_\_*)) **as** $php) {

// https://www.php.net/manual/en/function.substr.php

$py = *substr*($php, 0, - 3) . 'py';

**if** (! *file\_exists*($py)) {

**echo** "$php is an obsolete file since its py file is deleted!<br>";

// if error of Permission denied ocurrs, run the following command:

// chmod -R 777 axiom

*unlink*($php);

**continue**;

}

++ $count;

**if** (*is\_axiom\_plausible*($php)) {

$axiom = *to\_python\_module*($php);

$section = *section*($axiom);

**if** (*array\_key\_exists*($section, $insurmountable)) {

**if** (*in\_array*($axiom, $insurmountable[$section]))

**continue**;

}

**if** (*array\_key\_exists*($section, $unprovable)) {

**if** (*in\_array*($axiom, $unprovable[$section]))

**continue**;

}

// echo $axiom . " is plausible<br>";

$plausible[$section][] = $axiom;

}

}

$tab = *str\_repeat*("&nbsp;", 8);

**foreach** (*read\_directory*(*dirname*(*\_\_file\_\_*)) **as** $directory) {

$section = *basename*($directory);

**if** (! *strcmp*($section, '\_\_pycache\_\_')) {

**continue**;

}

**echo** "$tab<a href=$section>$section</a><br>";

**if** (*array\_key\_exists*($section, $insurmountable)) {

**echo** "$tab${tab}<font color=blue>axioms insurmountable:</font><br>";

**foreach** ($insurmountable[$section] **as** $axiom) {

**echo** "$tab$tab$tab" . *to\_a\_tag*($axiom) . "<br>";

}

}

**if** (*array\_key\_exists*($section, $unprovable)) {

**echo** "$tab${tab}<font color=green>axioms unprovable:</font><br>";

**foreach** ($unprovable[$section] **as** $axiom) {

**echo** "$tab$tab$tab" . *to\_a\_tag*($axiom) . "<br>";

}

}

**if** (*array\_key\_exists*($section, $plausible)) {

**echo** "$tab${tab}<font color=red>axioms plausible:</font><br>";

**foreach** ($plausible[$section] **as** $axiom) {

**echo** "$tab$tab$tab" . *to\_a\_tag*($axiom) . "<br>";

}

}

}

**foreach** (*yield\_empty\_directory*(*dirname*(*\_\_file\_\_*)) **as** $directory) {

**echo** "$directory is an obsolete folder since there is no py file in it!<br>";

// if error of Permission denied ocurrs, run the following command:

// chmod -R 777 axiom

*removedir*($directory);

}

**echo** "in summary:<br>";

**echo** "there are $count axioms in all, wherein:<br>";

**echo** "${tab}there are " . *accumulate*($plausible) . " axioms plausible;<br>";

**echo** "${tab}there are " . *accumulate*($unprovable) . " axioms unprovable;<br>";

**echo** "${tab}there are " . *accumulate*($insurmountable) . " axioms insurmountable.<br>";

?>

<script>

$("input[type=text]")[0].focus();

</script>

**sympy/axiom/render.php**

<?php

// if (! defined('SAGEMATH\_ROOT')) {

// define('SAGEMATH\_ROOT', dirname(dirname(\_\_FILE\_\_)) . '/');

// }

**include\_once** 'index.html';

**require\_once** 'utility.php';

// ^ +error\_log

// include\_once dirname(dirname(\_\_FILE\_\_)) . '/index.html';

**function** *str\_html*($param)

{

// preg\_replace()

**return** *preg\_replace*("/<(?=[a-zA-Z!\/])/", "&lt;", *str\_replace*("&", "&amp;", $param));

}

**function** *replace\_white\_spaces*($param)

{

**return** *str\_replace*(" ", "&nbsp;", $param);

}

// use the following regex to remove error\_log prints:^ +error\_log

// to speed up the .php page rendering, disable error\_log!!

**global** $sagemath;

**function** *create\_text\_tag*(&$statement)

{

$length = *strlen*($statement) + 2;

$statement\_quote = *quote*($statement);

**return** "<input spellcheck=false name=python[] size=$length value='$statement\_quote'>";

}

**function** *create\_a\_tag\_with\_this\_module*(&$statement, $module)

{

$length = *strlen*($statement);

$statement\_quote = *quote*($statement);

**global** $sagemath;

$request\_url = "/$sagemath/axiom/request.php?callee=$module";

**return** "<a href='$request\_url'>$statement\_quote</a>";

}

**function** *create\_a\_tag*($theorem, &$statement, &$axiom\_prefix)

{

// error\_log("theorem = $theorem");

// error\_log("statement = $statement");

// error\_log("axiom\_prefix = " . jsonify($axiom\_prefix));

// error\_log("\_\_file\_\_ = " . \_\_file\_\_);

// error\_log("dirname(\_\_file\_\_) = " . dirname(\_\_file\_\_));

$dot\_index = *strpos*($theorem, '.');

**if** ($dot\_index === **false**) {

$head = $theorem;

} **else** {

$head = *substr*($theorem, 0, $dot\_index);

}

$theorem = *str\_replace*(".", "/", $theorem);

**global** $sagemath;

**if** (*strlen*($head)) {

$prefix = $axiom\_prefix[$head];

$full\_theorem\_path = "/$sagemath/$prefix";

} **else**

$full\_theorem\_path = "/$sagemath";

$full\_theorem\_path .= "/$theorem.php";

$statement\_quote = *str\_html*($statement);

$statement\_quote = *replace\_white\_spaces*($statement\_quote);

**return** "<a href='$full\_theorem\_path'>$statement\_quote</a>";

}

// input is a php file

**function** *render*($php)

{

// error\_log("php file = $php");

$py = *str\_replace*('.php', '.py', $php);

// $py = str\_replace('latex', 'sympy', $py);

// error\_log("python file = $py");

*assert*(*file\_exists*($py), "file\_exists($py)");

$lengths = [];

$indexOfYield = - 1;

$counterOfLengths = 0;

$inputs = [];

$input = [];

**foreach** (*yield\_from\_py*($py) **as** $dict) {

// error\_log(jsonify($dict));

**if** (*array\_key\_exists*('axiom\_prefix', $dict)) {

$axiom\_prefix = $dict['axiom\_prefix'];

**continue**;

}

**if** (*array\_key\_exists*('numOfYields', $dict)) {

$numOfYields = $dict['numOfYields'];

**continue**;

}

$statement = $dict['statement'];

**if** (*array\_key\_exists*('pivot', $dict)) {

// error\_log("dict: " . jsonify($dict));

$pivot = $dict['pivot'];

$a = $dict['a'];

$first\_statement = *substr*($statement, 0, $pivot);

$second\_statement = *substr*($statement, $pivot);

$html = *create\_a\_tag*($a[0], $first\_statement, $axiom\_prefix);

**if** ($a[1] == **null**) {

$html .= *create\_text\_tag*($second\_statement);

} **else** {

$html .= *create\_a\_tag*($a[1], $second\_statement, $axiom\_prefix);

}

$input[] = $html;

} **else if** (*array\_key\_exists*('module', $dict)) {

$module = $dict['module'];

$indexOfYield = $counterOfLengths;

$input[] = *create\_a\_tag\_with\_this\_module*($statement, $module);

} **else if** (*array\_key\_exists*('a', $dict)) {

$a = $dict['a'][0];

$a = *create\_a\_tag*($a, $statement, $axiom\_prefix);

**if** (*startsWith*($statement, ' '))

$inputs[*count*($inputs) - 1] .= "<br>$a";

**else**

$input[] = $a;

} **else** {

$text = *create\_text\_tag*($statement);

// error\_log("create\_text\_tag: " . $statement);

**if** (*startsWith*($statement, ' ') && $input == **null**) {

// starting with more than 4 spaces indicates this line is a continuation of the previous line of code!

$inputs[*count*($inputs) - 1] .= "<br>$text";

} **else** {

$input[] = $text;

}

}

**if** (*preg\_match*('/Eq \*<< \*/', $statement, $matches)) {

$inputs[] = *join*("<br>", $input);

$input = **null**;

// unset($input);

++ $counterOfLengths;

$lengths[] = 1;

} **else if** (*preg\_match\_u*('/(Eq\.\w+ \*(?:, \*(?:Eq\.\w+|\w+|\\*\w+) \*)\*)= \*/', $statement, $matches)) {

$statement = $matches[1];

// error\_log("parameter: " . $statement);

// https://www.php.net/manual/en/function.preg-match-all.php

*preg\_match\_all*('/Eq\.\w+/u', $statement, $matches, *PREG\_SET\_ORDER*);

++ $counterOfLengths;

$lengths[] = *count*($matches);

$inputs[] = *join*("<br>", $input);

**unset**($input);

}

}

$pos = *strpos*(*to\_python\_module*($py), '.given.');

**if** ($pos >= 0) {

$given = "imply";

$imply = "given";

} **else** {

$given = "given";

$imply = "imply";

}

**echo** "<h3><font color=blue>$given:</font></h3>";

// error\_log("indexOfYield = $indexOfYield");

$numOfReturnsFromApply = $lengths[$indexOfYield];

// error\_log("numOfReturnsFromApply = " . $numOfReturnsFromApply);

// error\_log("lengths = " . jsonify($lengths));

$p = [];

$i = 0;

$statements = '';

$statements\_before\_yield = '';

**foreach** (*yield\_from\_php*($php) **as** &$statement) {

**if** ($i == $indexOfYield) {

// error\_log($statement);

-- $lengths[$i];

$statements .= $statement;

**if** ($lengths[$i] == 0) {

**if** ($numOfReturnsFromApply == 1) {

**if** (*is\_latex*($statement, $matches)) {

// error\_log("matches = ".jsonify($matches));

$numOfReturnsFromApply = *count*($matches);

// error\_log("count(matches) = ".$numOfReturnsFromApply);

$statements\_before\_yield = *array\_slice*($matches, 0, $numOfReturnsFromApply - $numOfYields);

// error\_log("statements\_before\_yield = ".jsonify($statements\_before\_yield));

$statements = *array\_slice*($matches, $numOfReturnsFromApply - $numOfYields);

// error\_log("statements\_after\_yield = ".jsonify($statements));

**foreach** ($statements **as** &$statement) {

$statement = $statement[0];

}

$statements = *join*('', $statements);

**foreach** ($statements\_before\_yield **as** &$statement) {

$statement = $statement[0];

}

$statements\_before\_yield = *join*('', $statements\_before\_yield);

}

}

$p[] = "<p>$statements\_before\_yield</p><h3><font color=blue>$imply:</font></h3><p>$statements</p><h3><font color=blue>prove:</font></h3>";

$statements = '';

$statements\_before\_yield = '';

++ $i;

} **else if** ($lengths[$i] == $numOfYields) {

$statements\_before\_yield = $statements;

// error\_log("lengths[i] = ".$lengths[$i]);

// error\_log("statements\_before\_yield = $statements\_before\_yield");

$statements = '';

}

} **else** {

$statements .= $statement;

-- $lengths[$i];

**if** ($lengths[$i] == 0) {

$p[] = "<p>$statements</p>";

$statements = '';

++ $i;

}

}

}

$size = *min*(*count*($inputs), *count*($p));

**for** ($i = 0; $i < $size; ++ $i) {

**echo** $inputs[$i];

$statement = $p[$i];

**echo** $statement;

}

}

?>

**sympy/axiom/request.php**

<?php

**include\_once** 'index.html';

**require\_once** 'utility.php';

**function** *read\_all\_axioms*($dir)

{

**foreach** (*read\_directory*($dir) **as** $directory) {

**foreach** (*read\_all\_files*($directory, 'py') **as** $py) {

**if** (*strcmp*(*basename*($py), "\_\_init\_\_.py")) {

**yield** $py;

}

}

}

}

**function** *module\_pieced\_together*($theorem, &$axiom\_prefix)

{

// error\_log("theorem = $theorem");

// error\_log("statement = $statement");

// error\_log("axiom\_prefix = " . jsonify($axiom\_prefix));

// error\_log("\_\_file\_\_ = " . \_\_file\_\_);

// error\_log("dirname(\_\_file\_\_) = " . dirname(\_\_file\_\_));

$dot\_index = *strpos*($theorem, '.');

**if** ($dot\_index === **false**) {

$head = $theorem;

} **else** {

$head = *substr*($theorem, 0, $dot\_index);

}

**if** (*strlen*($head)) {

$prefix = $axiom\_prefix[$head];

$prefix = *str\_replace*('/', '.', $prefix);

$module = "$prefix.$theorem";

} **else** {

$module = $theorem;

}

**return** $module;

}

// input is a py file

**function** *process\_py*($py)

{

$axioms = [];

**foreach** (*yield\_from\_py*($py) **as** $dict) {

// error\_log(jsonify($dict));

**if** (*array\_key\_exists*('axiom\_prefix', $dict)) {

$axiom\_prefix = $dict['axiom\_prefix'];

} **else if** (*array\_key\_exists*('a', $dict)) {

**foreach** ($dict['a'] **as** &$axiom) {

$axioms[] = *module\_pieced\_together*($axiom, $axiom\_prefix);

}

}

}

**return** $axioms;

}

**global** $sagemath;

**class** Set

{

**private** $set;

**public function** \_\_construct()

{

$this->set = [];

}

**public function** add($element)

{

$this->set[$element] = **true**;

}

**public function** remove($element)

{

**unset**($this->set[$element]);

}

**public function** enumerate()

{

**foreach** ($this->set **as** $key => &$\_) {

**yield** $key;

}

}

**public function** contains($element)

{

**return** *array\_key\_exists*($element, $this->set);

}

}

**class** Graph

{

**private** $graph;

**private** $permanent\_mark;

**private** $temporary\_mark;

**function** visit($n)

{

// error\_log("visiting key = $n");

**if** ($this->permanent\_mark->contains($n))

**return null**;

**if** ($this->temporary\_mark->contains($n))

**return** $n;

**if** (*array\_key\_exists*($n, $this->graph)) {

$this->temporary\_mark->add($n);

// error\_log("this->graph[n] = " . jsonify($this->graph[$n]));

**foreach** ($this->graph[$n] **as** $m) {

$node = $this->visit($m);

**if** ($node != **null**)

**return** $node;

}

$this->temporary\_mark->remove($n);

}

$this->permanent\_mark->add($n);

**return null**;

}

**function** initialize\_topology()

{

$this->permanent\_mark = **new** Set();

$this->temporary\_mark = **new** Set();

}

**function** &topological\_sort\_depth\_first()

{

$this->initialize\_topology();

**foreach** ($this->graph **as** $n => $\_) {

**if** ($this->visit($n))

**return null**;

}

**return** $this->L;

}

**function** detect\_cyclic\_proof($key)

{

$this->initialize\_topology();

**return** $this->visit($key);

}

**public function** \_\_construct()

{

$this->graph = [];

}

**function** convert\_set\_to\_list()

{

**foreach** ($this->graph **as** $key => &$value) {

$this->graph[$key] = *iterator\_to\_array*($value->enumerate());

}

}

**function** add\_edge($from, $to)

{

**if** (! *array\_key\_exists*($from, $this->graph)) {

$this->graph[$from] = **new** Set();

}

$this->graph[$from]->add($to);

}

**function** depict($module, $multiplier)

{

// https://www.php.net/manual/en/function.str-repeat.php

**echo** *str\_repeat*("&nbsp;", $multiplier) . *to\_a\_tag*($module);

**if** (*array\_key\_exists*($module, $this->graph)) {

**echo** "<button onmouseover=\"this.style.backgroundColor='red';\" onmouseout=\"this.style.backgroundColor='rgb(199, 237, 204)';\">>>>></button>";

**echo** "<div class=hidden>";

**foreach** ($this->graph[$module] **as** $module) {

$this->depict($module, $multiplier + 8);

}

**echo** "</div>";

}

**echo** "<br>";

}

**function** depict\_topology()

{

**foreach** ($this->permanent\_mark->enumerate() **as** $module) {

**echo** *str\_repeat*("&nbsp;", 8) . *to\_a\_tag*($module) . "<br>";

}

}

}

$mapping = **new** Graph();

$array\_keys = *array\_keys*(**$\_GET**);

**if** (*count*($array\_keys) > 1) {

// print\_r($\_GET);

$deep = *json\_decode*(**$\_GET**['deep']);

**unset**(**$\_GET**['deep']);

} **else** {

$deep = **false**;

}

$key\_input = *array\_keys*(**$\_GET**)[0];

**switch** ($key\_input) {

**case** "callee":

$key = 'caller';

$invert = **true**;

**break**;

**case** "caller":

$key = 'callee';

$invert = **false**;

**break**;

}

**foreach** (*read\_all\_axioms*(*dirname*(*\_\_file\_\_*)) **as** $py) {

$from = *to\_python\_module*($py);

$modules = *process\_py*($py);

**foreach** ($modules **as** $to) {

**if** ($invert)

$mapping->add\_edge($to, $from);

**else**

$mapping->add\_edge($from, $to);

}

}

$module = **$\_GET**[$key\_input];

$deep\_invert = *jsonify*(! $deep);

**echo** "the axiom in question is a <a href='request.php?$key\_input=$module&deep=$deep\_invert'>$key\_input</a> in the following hierarchy, would you switch to <a href='request.php?$key=$module'>$key</a> hierarchy?<br>";

$mapping->convert\_set\_to\_list();

$pinpoint = $mapping->detect\_cyclic\_proof($module);

**if** ($pinpoint) {

**echo** "<font color=red>cyclic proof detected in :</font><br>";

**echo** *to\_a\_tag*($module) . "<br>";

**if** (*strcmp*($pinpoint, $module)) {

**echo** *str\_repeat*("&nbsp;", 8) . *to\_a\_tag*($pinpoint) . "<br>";

} **else** {

// $mapping->depict\_topology();

}

} **else** {

$mapping->depict($module, 2);

}

**function** *javaScript*($js)

{

**echo** "<script>" . $js . "</script>";

}

*javaScript*("toggle\_expansion\_button();");

**if** ($deep)

*javaScript*("click\_all\_expansion\_buttons();");

**else**

*javaScript*("click\_first\_expansion\_button();");

?>

sympy/axiom/utiltiy.php

<?php

**include\_once** 'index.html';

// use the following regex to remove error\_log prints:^ +error\_log

// to speed up the .php page rendering, disable error\_log!!

**function** *get\_extension*($file)

{

**return** *pathinfo*($file, *PATHINFO\_EXTENSION*);

}

**function** *startsWith*($haystack, $needle)

{

$length = *strlen*($needle);

**return** *substr*($haystack, 0, $length) === $needle;

}

**function** *endsWith*($haystack, $needle)

{

$length = *strlen*($needle);

**if** ($length == 0) {

**return true**;

}

**return** *substr*($haystack, - $length) === $needle;

}

**function** *quote*($param)

{

**if** (*strpos*($param, "'") !== **false**) {

$param = *str\_replace*("'", "&apos;", $param);

}

**return** $param;

}

**function** *to\_python\_module*($py)

{

**global** $sagemath;

$module = [];

$pythonFile = $py;

**for** (;;) {

$dirname = *dirname*($pythonFile);

$basename = *basename*($pythonFile);

**if** (! *strcmp*($basename, $sagemath)) {

**break**;

}

$module[] = $basename;

$pythonFile = $dirname;

}

$module[0] = *substr*($module[0], 0, - *strlen*(*get\_extension*($module[0])) - 1);

$module = *array\_reverse*($module);

$module = *join*('.', $module);

**return** $module;

}

**function** &*yield\_from\_php*($php)

{

**foreach** (*file*($php) **as** &$statement) {

// error\_log($statement);

**if** (*strncmp*($statement, "//", 2) !== 0) {

**continue**;

}

$statement = *substr*($statement, 2);

**yield** $statement;

}

}

**function** *reference*(&$value)

{

**if** (*is\_array*($value)) {

**foreach** ($value **as** &$element) {

$element = *reference*($element);

}

$value = *join*(', ', $value);

**return** $value;

}

**if** (*preg\_match*('/\d+/', $value, $matches)) {

$value = (int) $value;

**if** ($value < 0)

**return** "plausible";

**return** "Eq[$value]";

} **else** {

**return** "Eq.$value";

}

}

**function** *jsonify*($param)

{

**return** *json\_encode*($param, *JSON\_UNESCAPED\_UNICODE*);

}

**function** *println*($param, $file = **null**)

{

**if** (*is\_array*($param)) {

$param = *jsonify*($param);

}

**if** ($file) {

**echo** "called in $file:<br>";

}

*print\_r*($param);

*print\_r*("<br>");

}

**function** *read\_directory*($dir)

{

**if** (*is\_dir*($dir)) {

$handle = *opendir*($dir);

**if** ($handle) {

**while** (($fl = *readdir*($handle)) !== **false**) {

$temp = $dir . *DIRECTORY\_SEPARATOR* . $fl;

**if** ($fl == '.' || $fl == '..') {

**continue**;

}

**if** (*is\_dir*($temp)) {

**yield** $temp;

}

}

}

}

}

**function** *read\_files*($dir, $ext = **null**)

{

**if** (*is\_dir*($dir)) {

$handle = *opendir*($dir);

**if** ($handle) {

**while** (($fl = *readdir*($handle)) !== **false**) {

$temp = $dir . *DIRECTORY\_SEPARATOR* . $fl;

**if** ($fl == '.' || $fl == '..') {

**continue**;

}

**if** (! *is\_dir*($temp)) {

**if** ($ext == **null** || ! *strcmp*(*get\_extension*($temp), $ext)) {

**yield** $temp;

}

}

}

}

}

}

**function** *to\_a\_tag*($module)

{

$href = *str\_replace*('.', '/', $module);

**global** $sagemath;

$href = "/$sagemath/$href.php";

**return** "<a name=python[] href='$href'>$module</a>";

}

**function** *read\_all\_files*($dir, $ext)

{

**if** (*is\_dir*($dir)) {

$handle = *opendir*($dir);

**if** ($handle) {

**while** (($fl = *readdir*($handle)) !== **false**) {

**if** ($fl == '.' || $fl == '..') {

**continue**;

}

$temp = $dir . *DIRECTORY\_SEPARATOR* . $fl;

**if** (*is\_dir*($temp)) {

// echo 'directory : ' . $temp . '<br>';

**yield from** *read\_all\_files*($temp, $ext);

} **else** {

**if** (! *strcmp*(*get\_extension*($temp), $ext)) {

**yield** $temp;

}

}

}

}

}

}

**function** *removedir*($dir)

{

**foreach** (*read\_files*($dir) **as** $file) {

*unlink*($file);

}

**foreach** (*read\_directory*($dir) **as** $subdir) {

*removedir*($subdir);

}

*rmdir*($dir);

}

**function** *is\_latex*($latex, &$matches)

{

**if** (*preg\_match\_all*('/\\\\\[.+?\\\\\]/', $latex, $matches, *PREG\_SET\_ORDER*)) {

**return true**;

}

**return false**;

}

**function** *is\_def\_start*($funcname, $statement, &$matches)

{

**return** *preg\_match*("/^def +$funcname\([^)]\*\) \*: \*/", $statement, $matches);

}

**function** *recursive\_construct*($parentheses, $depth)

{

$mid = *strlen*($parentheses) / 2;

$start = *substr*($parentheses, 0, $mid);

$end = *substr*($parentheses, $mid);

**if** (*need\_escape*($start)) {

$start = "\\" . $start;

$end = "\\" . $end;

}

**if** ($depth == 1)

**return** "${start}[^$parentheses]\*$end";

**return** "${start}[^$parentheses]\*(?:" . *recursive\_construct*($parentheses, $depth - 1) . "[^$parentheses]\*)\*$end";

}

**function** *balancedGroups*($parentheses, $depth, $multiple = **true**)

{

$regex = *recursive\_construct*($parentheses, $depth);

**if** ($multiple)

**return** "((?:$regex)\*)";

**else**

**return** "(?:$regex)";

}

**function** *balancedParentheses*($depth, $multiple = **false**)

{

**return** *balancedGroups*("()", $depth, $multiple);

}

**function** *balancedBrackets*($depth, $multiple = **false**)

{

**return** *balancedGroups*("\[\]", $depth, $multiple);

}

**function** *need\_escape*($s)

{

**switch** ($s) {

**case** "(":

**case** ")":

**case** "{":

**case** "}":

**return true**;

**default**:

**return false**;

}

}

**function** *numOfYields*($statement)

{

**global** $patternOfYield;

**if** (*preg\_match\_u*("/^$patternOfYield,?$/", $statement, $matches)) {

// error\_log("match one yield: " . $matches[1]);

**return** 1;

} **else** {

// error\_log('return ' . $statement);

**if** (*preg\_match\_u*("/^$patternOfYield,\s\*([\s\S]+)$/", $statement, $matches)) {

// error\_log("match one yield: " . $matches[1]);

// error\_log("try to match the next yield from: " . $matches[2]);

$numOfYields = *numOfYields*($matches[2]);

**if** ($numOfYields) {

**return** 1 + $numOfYields;

}

} **else if** (*preg\_match\_u*("/^${patternOfYield}[&|]\s\*([\s\S]+)$/", $statement, $matches)) {

// error\_log("match one yield: " . $matches[1]);

// error\_log("try to match the next yield from: " . $matches[2]);

$numOfYields = *numOfYields*($matches[2]);

**if** ($numOfYields) {

**return** $numOfYields;

}

}

// error\_log("match failed: " . $statement);

}

**return** 0;

}

**function** *analyze\_apply*($py, &$i)

{

// ++ $i;

$numOfYields = 0;

$count = *count*($py);

**for** (; $i < $count; ++ $i) {

$statement = $py[$i];

**if** (*is\_def\_start*('prove', $statement, $matches)) {

// error\_log('prove begins: ' . $statement);

**break**;

}

**if** (*preg\_match*('/^@prove/', $statement, $matches)) {

**continue**;

}

**if** (*preg\_match*('/^from/', $statement, $matches)) {

**continue**;

}

**if** (*preg\_match*('/^ \*$/', $statement, $matches)) {

**continue**;

}

**if** (*preg\_match*('/^(?: )+return +(.+) \*/', $statement, $matches)) {

**if** ($numOfYields)

**continue**;

// error\_log('return statement: ' . $statement);

$yield = $matches[1];

// error\_log('matches[1]=' . $yield);

**if** (! *strcmp*($yield, 'None'))

**continue**;

**do** {

$yield = *rtrim*($yield);

$yield = *rtrim*($yield, "\\");

$numOfYields = *numOfYields*($yield);

**if** ($numOfYields)

**break**;

++ $i;

**if** ($i >= $count)

**break**;

$yield .= $py[$i];

} **while** (**true**);

}

}

**return** $numOfYields;

}

**function** *detect\_axiom*(&$statement)

{

// Eq << Eq.x\_j\_subset.apply(discrete.sets.subset.nonemptyset, Eq.x\_j\_inequality, evaluate=False)

**if** (*preg\_match*('/\.apply\((.+)\)/', $statement, $matches)) {

$theorem = *preg\_split*("/\s\*,\s\*/", $matches[1], - 1, *PREG\_SPLIT\_NO\_EMPTY*)[0];

// error\_log('create\_a\_tag: ' . \_\_LINE\_\_);

**return** [

$theorem

];

} **else** {

**return** [];

}

}

**function** *detect\_axiom\_given\_theorem*(&$theorem, &$statement)

{

**if** (*startsWith*($theorem, '.')) {

// consider the case

// Eq << Eq[-1].reversed.apply(discrete.sets.unequal.notcontains, evaluate=False)

**return** *detect\_axiom*($statement);

}

**if** (*startsWith*($theorem, 'Eq')) {

// consider the case

// Eq[-2].this.args[0].apply(algebre.condition.condition.imply.et, invert=True, swap=True)

**return** *detect\_axiom*($statement);

}

**if** (*strpos*($theorem, 'Eq.') === **false**) {

**return** [

$theorem

];

}

**return** *detect\_axiom*($statement);

}

// input is a py file

**function** *yield\_from\_py*($python\_file)

{

*assert*(*file\_exists*($python\_file), "file\_exists($python\_file)");

$inputs = [];

$input = [];

$axiom\_prefix = [];

$py = *file*($python\_file);

**for** ($i = 0; $i < *count*($py); ++ $i) {

$statement = $py[$i];

// error\_log("$statement");

// from axiom.keras import bilinear # python import statement

**if** (*preg\_match*('/^from +(.+) +import +(.\*)/', $statement, $matches)) {

$prefix = $matches[1];

$namespaces = $matches[2];

$namespaces = *preg\_split*("/[\s,]+/", $namespaces, - 1, *PREG\_SPLIT\_NO\_EMPTY*);

// error\_log("end(namespaces) = " . end($namespaces));

**if** (! *strcmp*(*end*($namespaces), '\\')) {

// error\_log("strcmp = " . strcmp(end($namespaces), '\\'));

*array\_pop*($namespaces);

$statement = $py[++ $i];

// error\_log("$statement");

$namespaces\_addition = *preg\_split*("/[\s,]+/", $statement, - 1, *PREG\_SPLIT\_NO\_EMPTY*);

// error\_log("namespaces\_addition = " . jsonify($namespaces\_addition));

*array\_push*($namespaces, ...$namespaces\_addition);

// error\_log("namespaces = " . jsonify($namespaces));

}

$prefix\_path = *str\_replace*(".", "/", $prefix);

**foreach** ($namespaces **as** $namespace) {

// error\_log('prefix detected: ' . $prefix . '.' . $namespace);

$axiom\_prefix[$namespace] = $prefix\_path;

}

**continue**;

}

**if** (*preg\_match*('/^import +(.+)/', $statement, $matches)) {

// error\_log('import statement: ' . $statement);

$packages = $matches[1];

$packages = *preg\_split*("/\s\*,\s\*/", $packages, - 1, *PREG\_SPLIT\_NO\_EMPTY*);

**foreach** ($packages **as** $package) {

$package = *preg\_split*("/\s+/", $package, - 1, *PREG\_SPLIT\_NO\_EMPTY*);

// error\_log('count(package) = ' . count($package));

**switch** (*count*($package)) {

**case** 1:

$package = $package[0];

$axiom\_prefix[$package] = '';

**break**;

**case** 2:

// error\_log('count(package[0]) = ' . $package[0]);

// error\_log('count(package[1]) = ' . $package[1]);

**break**;

**case** 3:

// error\_log('count(package[0]) = ' . $package[0]);

// error\_log('count(package[1]) = ' . $package[1]);

// error\_log('count(package[2]) = ' . $package[2]);

$axiom\_prefix[*end*($package)] = '';

// error\_log('package detected: ' . $package[0]);

**break**;

**default**:

**break**;

}

}

**continue**;

}

// $yield = [

// 'line' => $i

// ];

**if** (*is\_def\_start*('apply', $statement, $matches)) {

**yield** [

'axiom\_prefix' => $axiom\_prefix,

'line' => $i

];

// error\_log('given begins: ' . $statement);

$numOfYields = *analyze\_apply*($py, $i);

// error\_log('given ended: ' . $statement);

**yield** [

'numOfYields' => $numOfYields,

'line' => $i + 1

];

**break**;

}

}

// error\_log('axiom\_prefix: ' . jsonify($axiom\_prefix));

**for** (++ $i; $i < *count*($py); ++ $i) {

$statement = $py[$i];

// error\_log("$statement");

$statement = *rtrim*($statement);

// remove comments starting with #

**if** (*preg\_match*('/^\s\*#.\*/', $statement, $matches) || ! $statement) {

**continue**;

}

// the start of the next global statement other than def prove

**if** (! *startsWith*($statement, ' ')) {

**break**;

}

$statement = *substr*($statement, 4);

$yield = [

'statement' => $statement,

'line' => $i

];

**global** $balancedParanthesis;

// Eq <<= geometry.plane.trigonometry.sine.principle.add.apply(\*Eq[-2].rhs.arg.args), geometry.plane.trigonometry.cosine.principle.add.apply(\*Eq[-1].rhs.arg.args)

**if** (*preg\_match\_u*("/((?:Eq \*<<= \*|Eq\.\w+, \*Eq\.\w+ \*= \*)([\w.]+|Eq[-\w.\[\]]\*\[-?\d+\][\w.]\*)\.apply$balancedParanthesis\s\*[,&]\s\*)(.+)/", $statement, $matches)) {

// error\_log('theorem detected: ' . $theorem);

$first\_statement = $matches[1];

$a = *detect\_axiom\_given\_theorem*($matches[2], $first\_statement);

$second\_statement = $matches[3];

**if** (*strcmp*($second\_statement, "\\")) {

*preg\_match\_u*("/([\w.]+|Eq[-\w.\[\]]\*\[-?\d+\])\.apply\(/", $second\_statement, $matches);

$second = *detect\_axiom\_given\_theorem*($matches[1], $second\_statement);

**if** (*count*($second)) {

*array\_push*($a, ...$second);

$yield['pivot'] = *strlen*($first\_statement);

}

}

$yield['a'] = $a;

} **else if** (*preg\_match\_u*("/([\w.]+)\.apply\(/", $statement, $matches)) {

// error\_log('theorem detected: ' . $theorem);

$a = *detect\_axiom\_given\_theorem*($matches[1], $statement);

**if** (*count*($a)) {

$yield['a'] = $a;

}

} **else if** (*preg\_match*('/(=|<<) \*apply\(/', $statement, $matches)) {

// error\_log('yield statement: ' . $statement);

// error\_log("php = $php");

$yield['module'] = *to\_python\_module*($python\_file);

} **else** {

// error\_log("statement = $statement");

$a = *detect\_axiom*($statement);

**if** (*count*($a)) {

$yield['a'] = $a;

}

}

**yield** $yield;

}

}

// global variables:

$sagemath = *basename*(*dirname*(*dirname*(*\_\_file\_\_*)));

$balancedParanthesis = *balancedParentheses*(7);

$balancedBrackets = *balancedBrackets*(4);

$patternOfYield = "(?:((?:\w+\.)\*\w+)\s\*(?:$balancedBrackets\s\*)?$balancedParanthesis|\w+(?:\.\w+)\*)\s\*";

**function** *preg\_match\_u*($regex, $str, &$matches)

{

**return** *preg\_match*($regex . "u", $str, $matches);

}

?>

sympy/js/utility.js

"use strict";

**function** strlen(s) {

**var** length = 0;

for (**let** i = 0; i < s.length; i++) {

**var** code = s.charCodeAt(i)

if (code < 128 || code == 0x2002)

length += 1;

else

length += 2;

}

return length;

}

**function** changeInputlength(input) {

**var** val = input.val();

console.log(val);

**var** text\_length = strlen(val);

console.log(text\_length);

// text\_length = Math.max(text\_length, input.attr('placeholder').length);

// text\_length = Math.min(text\_length, 32);

text\_length /= 2;

text\_length += 2;

input.css("width", text\_length + "em");

}

**function** toggle\_expansion\_button() {

$('button').click(**function**() {

**var** div = $(this)[0].nextElementSibling;

if ($(this).text() == '>>>>') {

div.style.display = 'block';

$(this).text('<<<<');

} else {

div.style.display = null;

$(this).text('>>>>');

}

});

}

**function** click\_first\_expansion\_button() {

**var** first\_button = document.querySelector("button");

first\_button.click();

}

**function** click\_all\_expansion\_buttons() {

**var** buttons = document.querySelectorAll("button");

for (**let** button of buttons) {

button.click();

}

}

window.onload = **function**() {

**var** currentFunctionKey = null;

// currentFunctionKey = window.currentFunctionKey;

// console.log('register: function (MainKey, value, func)');

document.onkeyup = **function**(event) {

console.log('onkeyup');

**var** key = event.key;

console.log('key = ' + key);

if (key == currentFunctionKey)

currentFunctionKey = null;

}

document.onkeydown = **function**(event) {

**var** key = event.key;

console.log('onkeydown');

console.log('key = ' + key);

console.log('currentFunctionKey = ' + currentFunctionKey);

if (currentFunctionKey == null) {

currentFunctionKey = key;

return;

}

switch (currentFunctionKey) {

case 'Alt':

switch (key) {

case 'c':

console.log("M-c");

**var** checkbox = $('input[type=checkbox][name=CaseSensitive]')[0];

checkbox.checked = !checkbox.checked;

break;

case 'd':

console.log("M-d");

break;

case 'l':

console.log("M-l");

break;

case 'r':

console.log("M-r");

break;

case 't':

console.log("M-t");

break;

case 'w':

console.log("M-w");

**var** checkbox = $('input[type=checkbox][name=WholeWord]')[0];

checkbox.checked = !checkbox.checked;

break;

case 'x':

console.log("M-x");

**var** checkbox = $('input[type=checkbox][name=RegularExpression]')[0];

checkbox.checked = !checkbox.checked;

break;

case '\r':

case '\n':

console.log("Alt + Enter");

break;

}

break;

case 'Control':

switch (key) {

case 'd':

console.log("C-d");

break;

case 'l':

console.log("C-l");

break;

case 'r':

console.log("C-r");

break;

case 't':

console.log("C-t");

break;

case 'Home':

$("input[type=text]")[0].focus();

console.log("C-Home");

break;

case 'Insert':

console.log("C-Insert");

break;

case 'Enter':

console.log("C-Enter");

//submit();

break;

}

break;

case 'Shift':

switch (key) {

case 'Enter':

console.log("Shift + Enter");

//submit();

break;

}

break;

default:

switch (key) {

case 40: // DownArrow

console.log("DownArrow");

break;

case 38: // UPArrow

console.log("UPArrow");

break;

}

break;

}

}

}

sympy\axiom\keras\layers\bert\scaled\_dot\_product\_attention.py

from axiom.utility import prove, apply

from tensorflow.nn import softmax

from sympy import \*

from axiom import keras, algebre

*@apply*

def **apply**(n, dz, h):

Q = Symbol.Q(shape=(n, dz), real=True)

K = Symbol.K(shape=(n, dz), real=True)

V = Symbol.V(shape=(n, dz), real=True)

a = Symbol.a(definition=Q @ K.T / sqrt(dz))

Ξ = Symbol.Ξ(definition=Identity(n) + BlockMatrix([[ZeroMatrix(h, h), OneMatrix(h, n - h)],

[OneMatrix(n - h, h), ZeroMatrix(n - h, n - h)]]))

a\_quote = Symbol(*"a'"*, definition=a - (1 - Ξ) \* oo)

s = Symbol.s(definition=softmax(a\_quote))

z = Symbol.z(definition=s @ V)

# diagonal part

D = Symbol.D(definition=(exp(ReducedSum(Q \* K) / sqrt(dz)) \* OneMatrix(dz, n)).T)

# upper part

Wu = Symbol(*"W^u"*, definition=exp(Q[:h] @ K[h:n].T / sqrt(dz)))

Vu = Symbol(*"V^u"*, definition=V[:h])

Du = Symbol(*"D^u"*, definition=D[:h])

# lower part

Wl = Symbol(*"W^l"*, definition=exp(Q[h:n] @ K[:h].T / sqrt(dz)))

Vl = Symbol(*"V^l"*, definition=V[h:n])

Dl = Symbol(*"D^l"*, definition=D[h:n])

return Equality(z, BlockMatrix((Wu @ Vl + Du \* Vu) / (ReducedSum(Wu) + Du), (Wl @ Vu + Dl \* Vl) / (ReducedSum(Wl) + Dl)))

*@prove*

def **prove**(Eq):

n = Symbol.n(integer=True, positive=True)

h = Symbol.h(domain=Interval(1, n - 1, integer=True))

dz = Symbol.d\_z(integer=True, positive=True)

Eq << apply(n, dz, h)

i = Symbol.i(domain=Interval(0, n - 1, integer=True))

j = Symbol.j(integer=True)

a = Eq[0].lhs

Eq << keras.layers.bert.mask.cross\_attention.apply(a, h)

Eq.ai\_definition = Eq[-1][i]

Eq << Eq[4][i]

Eq.z\_definition = Eq[-1].this.rhs.args[0].definition

Eq.z\_definition = Eq.z\_definition.this.rhs.subs(Eq[-1])

Eq.z\_definition = Eq.z\_definition.this.rhs.subs(Eq.ai\_definition)

Eq << Eq.z\_definition.rhs.args[-1].args[0].this.astype(MatMul)

Eq << Eq[-1].this.rhs.expand()

Eq << Eq[-1].this.rhs.subs(Eq[1][i, j])

Eq << Eq[-1].this.rhs.args[0]().expr.args[0].simplify()

Eq << Eq[-1].this.rhs.args[-1].expr.astype(Piecewise)

Eq << Eq[-1].this.rhs.args[0]().expr.args[1]().function.simplify()

Eq << Eq[-1].this.rhs.args[1]().expr.args[1]().function.simplify()

Eq << Eq[-1].this.rhs.simplify(wrt=True)

Eq.divisor\_definition = Eq[-1].this.rhs.astype(Plus)

Eq << Eq.divisor\_definition.rhs.args[0].args[-1].expr.this.astype(ReducedSum)

Eq << Eq.divisor\_definition.rhs.args[0].args[0].expr.this.astype(ReducedSum)

Eq.divisor\_definition = Eq.divisor\_definition.this.rhs.subs(Eq[-2], Eq[-1], simplify=False)

Eq << Eq[5][i]

Eq << Eq[-1].this.rhs.args[1].arg.args[1].astype(MatMul)

Eq.M\_definition = Eq[-1].this.rhs.args[1].arg.args[1].T

Eq << Eq[0][i]

Eq <<= Eq[-1][h:n], Eq[-1][:h], Eq[-1][i]

Eq <<= algebre.equal.imply.equal.exp.apply(Eq[-3]), algebre.equal.imply.equal.exp.apply(Eq[-2]), algebre.equal.imply.equal.exp.apply(Eq[-1])

Eq << Eq[-1] \* OneMatrix(dz)

Eq.lower\_part, Eq.upper\_part, Eq.diagonal\_part = algebre.equal.equal.imply.equal.transit.apply(Eq[-4], Eq[7][i]), \

algebre.equal.equal.imply.equal.transit.apply(Eq[-3], Eq[11][i - h]), \

algebre.equal.equal.imply.equal.transit.apply(Eq[-1], Eq.M\_definition)

Eq << Eq.divisor\_definition \* OneMatrix(dz)

Eq << Eq[-1].this.rhs.astype(Plus)

Eq << Eq[-1].this.rhs.subs(Eq.lower\_part, Eq.upper\_part, Eq.diagonal\_part)

Eq.z\_definition = algebre.equal.condition.imply.condition.subs\_with\_expand\_dims.apply(Eq[-1], Eq.z\_definition)

Eq << Eq.z\_definition.rhs.args[0].this.expand()

Eq << Eq[-1].this.rhs.function.function.args[1].definition

Eq << Eq[-1].this(i).rhs.function.args[0]().expr.simplify()

Eq << Eq[-1].this.rhs.function.args[-1].expr.astype(Piecewise)

Eq << Eq[-1].this.rhs.function.apply(algebre.imply.equal.piecewise.swap.back)

Eq << Eq[-1].this.rhs.function.simplify(wrt=i)

Eq << Eq[-1].this.rhs.function.astype(Plus)

Eq << Eq[-1].this.rhs.astype(Plus)

Eq << Eq[-1].this.rhs.args[1].function.args[0].expr.astype(Plus)

Eq << Eq[-1].this.rhs.args[1].function.args[0]().expr.args[1].simplify()

Eq << Eq[-1].this.rhs.args[1].function.args[1].expr.astype(Plus)

Eq << Eq[-1].this.rhs.args[1].function.args[1]().expr.args[1].simplify()

Eq << Eq[-1].this.rhs.args[1].function.args[0].expr.astype(MatMul)

Eq << Eq[-1].this.rhs.args[1].function.args[1].expr.astype(MatMul)

Eq << Eq[-1].this.rhs.args[1].astype(Piecewise)

Eq << Eq[-1].this.rhs.args[1].args[0].expr.astype(MatMul)

Eq << Eq[-1].this.rhs.args[1].args[0].expr.T

Eq << Eq[-1].this.rhs.args[1].args[1].expr.astype(MatMul)

Eq << Eq[-1].this.rhs.args[1].args[1].expr.T

Eq << Eq[-1].this.rhs.subs(Eq.lower\_part, Eq.upper\_part)

Eq << algebre.equal.condition.imply.condition.subs\_with\_expand\_dims.apply(Eq.diagonal\_part, Eq[-1])

Eq << Eq.z\_definition.this.rhs.subs(Eq[-1])

Eq << Eq[-1].this.rhs.args[0].args[0].astype(Piecewise)

Eq << Eq[-1].this.rhs.args[0].astype(Piecewise)

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << algebre.equal.imply.equal.lamda.apply(Eq[-1], (i,))

Eq << Eq[-1].this.rhs.astype(BlockMatrix)

Eq << Eq[-1].this.rhs.args[0].astype(Times)

Eq << Eq[-1].this.rhs.args[0].args[0].astype(Power)

Eq << Eq[-1].this.rhs.args[1].astype(Times)

Eq << Eq[-1].this.rhs.args[1].args[0].astype(Power)

Eq << Eq[-1].this.rhs.args[0].args[1].astype(Plus)

Eq << Eq[-1].this.rhs.args[0].args[1].base.astype(Plus)

Eq << Eq[-1].this.rhs.args[0].args[1].base.args[1].astype(ReducedSum)

Eq << Eq[-1].this.rhs.args[1].args[1].astype(Plus)

Eq << Eq[-1].this.rhs.args[1].args[1].base.astype(Plus)

Eq << Eq[-1].this.rhs.args[1].args[1].base.args[1].astype(ReducedSum)

Eq << Eq[-1].this.rhs.subs(Eq[6].reversed, Eq[8].reversed, Eq[9].reversed, Eq[10].reversed)

if \_\_name\_\_ == *'\_\_main\_\_'*:

prove(\_\_file\_\_)

# reference:

# Self-Attention with Relative Position Representations.pdf

# <https://arxiv.org/abs/1803.02155>

sympy\axiom\keras\layers\bert\position\_representation\sinusoidal\linearity.py

from sympy import \*

from axiom.utility import prove, apply

from axiom import algebre, geometry

from axiom.keras.layers.bert.position\_representation.sinusoidal.definition import sinusoid\_position\_encoding

*@apply*

def **apply**(n, d):

PE = sinusoid\_position\_encoding(n, d)

j, i = PE.definition.variables

k = Symbol.k(integer=True)

PE\_quote = sinusoid\_position\_encoding(n, d, inverse=True)

(e0, c0), (e1, \_) = PE[k, j].definition.args

F = Symbol.F(definition=LAMBDA[j:d, k:n](Piecewise((cos(e0.arg), c0), (e1, True))))

F\_quote = Symbol(*"F'"*, definition=LAMBDA[j:d, k:n](Piecewise((e0, c0), (sin(e1.arg), True))))

I = S.ImaginaryUnit

z = Symbol.z(definition=F - I \* F\_quote)

Z = Symbol.Z(definition=PE \* I - PE\_quote)

return Equality(Z[i + k], Z[i] \* z[1] \*\* k)

*@prove*

def **prove**(Eq):

n = Symbol.n(positive=True, integer=True)

d = Symbol(*"d\_model"*, integer=True, positive=True, even=True)

Eq << apply(n, d)

PE\_quote = Eq[0].lhs.base

PE = Eq[1].lhs.base

i, j = Eq[0].lhs.indices

k = Eq[3].lhs.indices[0]

Eq.PE\_definition = PE[i + k, j].this.definition

Eq.PE\_quote\_definition = PE\_quote[i + k, j].this.definition

Eq << Eq.PE\_definition.rhs.args[0].expr.this.arg.astype(Plus)

Eq << Eq.PE\_definition.rhs.args[1].expr.this.arg.astype(Plus)

Eq <<= geometry.plane.trigonometry.sine.principle.add.apply(\*Eq[-2].rhs.arg.args), geometry.plane.trigonometry.cosine.principle.add.apply(\*Eq[-1].rhs.arg.args)

Eq <<= algebre.equal.equal.imply.equal.transit.apply(Eq[-4], Eq[-2]), algebre.equal.equal.imply.equal.transit.apply(Eq[-3], Eq[-1])

Eq << Eq.PE\_definition.this.rhs.args[0].expr.subs(Eq[-2])

Eq.cossin = Eq[-1].this.rhs.args[1].expr.subs(Eq[-2])

Eq << Eq[1] \* Eq[3]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << Eq[0] \* Eq[4]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << Eq[-1] + Eq[-3]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << algebre.equal.equal.imply.equal.transit.apply(Eq.cossin, Eq[-1])

Eq << algebre.equal.imply.equal.lamda.apply(Eq[-1], (j, 0, d))

Eq.PE\_equality = Eq[-1].this.rhs.astype(Plus)

Eq << Eq.PE\_quote\_definition.rhs.args[0].expr.this.arg.astype(Plus)

Eq << Eq.PE\_quote\_definition.rhs.args[1].expr.args[1].this.arg.astype(Plus)

Eq <<= geometry.plane.trigonometry.cosine.principle.add.apply(\*Eq[-2].rhs.arg.args), geometry.plane.trigonometry.sine.principle.add.apply(\*Eq[-1].rhs.arg.args)

Eq <<= algebre.equal.equal.imply.equal.transit.apply(Eq[-4], Eq[-2]), algebre.equal.equal.imply.equal.transit.apply(Eq[-3], Eq[-1])

Eq << Eq.PE\_quote\_definition.this.rhs.args[0].expr.subs(Eq[-2])

Eq.coscos = Eq[-1].this.rhs.args[1].expr.subs(Eq[-2])

Eq << Eq[1] \* Eq[4]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << Eq[0] \* Eq[3]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << Eq[-1] - Eq[-3]

Eq << Eq[-1].this.rhs.astype(Piecewise)

Eq << algebre.equal.equal.imply.equal.transit.apply(Eq.coscos, Eq[-1])

Eq << algebre.equal.imply.equal.lamda.apply(Eq[-1], (j, 0, d))

Eq << Eq[-1].this.rhs.astype(Plus)

I = S.ImaginaryUnit

Eq << I \* Eq.PE\_equality - Eq[-1]

Eq << Eq[-1].this.rhs.expand()

Eq << Eq[-1].this.rhs.collect(PE[i])

Eq.collect = Eq[-1].this.rhs.collect(PE\_quote[i])

F\_quote = Eq[1].lhs.base

F = Eq[3].lhs.base

z = Eq[5].lhs

Eq << z[k].this.definition

Eq << Eq[-1] \* I

Eq << Eq[-1].this.rhs.expand()

Eq << Eq.collect.subs(Eq[-1].reversed, Eq[-3].reversed)

Eq << Eq[-1].this.rhs.collect(z[k])

Z = Eq[2].lhs

Eq << Z[i].this.definition

Eq << Eq[-2].subs(Eq[-1].reversed)

Eq << Z[k + i].this.definition

Eq << algebre.equal.equal.imply.equal.transit.apply(Eq[-1], Eq[-2])

Eq << Eq[-1].subs(k, 1)

Eq << algebre.equal.imply.equal.geometric\_progression.apply(Eq[-1], n=i)

Eq << Eq[-1].subs(i, i + k)

Eq << Eq[-2] \* z[1] \*\* k

Eq << Eq[-1].this.rhs.powsimp()

Eq << algebre.equal.equal.imply.equal.transit.apply(Eq[-3], Eq[-1])

if \_\_name\_\_ == *'\_\_main\_\_'*:

prove(\_\_file\_\_)

# reference:

# Self-Attention with Relative Position Representations.pdf

# https://arxiv.org/abs/1803.02155

sympy\axiom\keras\layers\bert\position\_representation\relative\band\_part\_mask.py

from axiom.utility import prove, apply

from tensorflow.nn import softmax

from sympy import \*

import tensorflow as tf

from axiom import keras, algebre, sets

*@apply*

def **apply**(seq\_length, dx, dz, k, num\_lower, num\_upper):

x = Symbol.x(shape=(seq\_length, dx), real=True)

W\_Q = Symbol(*"W^Q"*, shape=(dx, dz), real=True)

W\_K = Symbol(*"W^K"*, shape=(dx, dz), real=True)

W\_V = Symbol(*"W^V"*, shape=(dx, dz), real=True)

Q = Symbol.Q(definition=x @ W\_Q)

K = Symbol.K(definition=x @ W\_K)

V = Symbol.V(definition=x @ W\_V)

i = Symbol.i(integer=True)

j = Symbol.j(integer=True)

w\_K = Symbol(*"w^K"*, shape=(2 \* k + 1, dz), real=True)

w\_V = Symbol(*"w^V"*, shape=(2 \* k + 1, dz), real=True)

a\_K = Symbol(*"a^K"*, definition=LAMBDA[j:seq\_length, i:seq\_length](w\_K[k + tf.clip(j - i, -k, k)]))

a\_V = Symbol(*"a^V"*, definition=LAMBDA[j:seq\_length, i:seq\_length](w\_V[k + tf.clip(j - i, -k, k)]))

a = Symbol.a(definition=Q @ (K + a\_K).T / sqrt(dz))

a\_quote = Symbol(*"a'"*, definition=a - (1 - tf.linalg.band\_part[num\_lower, num\_upper](OneMatrix(seq\_length, seq\_length))) \* oo)

s = Symbol.s(definition=softmax(a\_quote))

z = Symbol.z(definition=s @ (V + a\_V))

gram\_width = num\_lower + num\_upper + 1

start = i - num\_lower

stop = start + gram\_width # i + k\_max + 1

a\_K\_quote = Symbol(*"a^K'"*, definition=LAMBDA[j:Min(seq\_length, gram\_width), i:seq\_length](w\_K[k + tf.clip(j - Min(i, num\_lower), -k, k)]))

a\_V\_quote = Symbol(*"a^V'"*, definition=LAMBDA[j:Min(seq\_length, gram\_width), i:seq\_length](w\_V[k + tf.clip(j - Min(i, num\_lower), -k, k)]))

β = Symbol.beta(definition=LAMBDA[i:seq\_length](tf.nn.relu(start)))

ζ = Symbol.zeta(definition=LAMBDA[i:seq\_length](Min(stop, seq\_length)))

indices = slice(β[i], ζ[i])

indices0 = slice(0, ζ[i] - β[i])

return Equality(z[i], softmax(Q[i] @ (K[indices] + a\_K\_quote[i][indices0]).T / sqrt(dz)) @ (V[indices] + a\_V\_quote[i][indices0]))

*@prove*

def **prove**(Eq):

n = Symbol.n(integer=True, positive=True)

k = Symbol.k(integer=True, positive=True)

l = Symbol.l(integer=True, positive=True)

u = Symbol.u(integer=True, positive=True)

dx = Symbol.d\_x(integer=True, positive=True)

dz = Symbol.d\_z(integer=True, positive=True)

Eq << apply(n, dx, dz, k, l, u)

i, j = Eq[2].lhs.indices

Eq << keras.nn.relu.min.astype.apply(l, i)

Eq << Eq[-1].reversed.subs(Eq[9].reversed)

Eq <<= Eq[11].this.rhs.subs(Eq[-1]), Eq[12].this.rhs.subs(Eq[-1])

β = Eq[9].lhs.base

ζ = Eq[10].lhs.base

Eq <<= Eq[2].subs(j, j + β[i]), Eq[7].subs(j, j + β[i])

Eq <<= algebre.equal.equal.imply.equal.transit.apply(Eq[-4], Eq[-2]), algebre.equal.equal.imply.equal.transit.apply(Eq[-3], Eq[-1])

gram\_width = l + u + 1

Eq.K\_equality = algebre.equal.imply.equal.lamda.apply(Eq[-2], (j, 0, Min(n, gram\_width)))

Eq.V\_equality = algebre.equal.imply.equal.lamda.apply(Eq[-1], (j, 0, Min(n, gram\_width)))

Eq.less\_than = LessThan(ζ[i], β[i] + Min(n, l + u + 1), plausible=True)

Eq << Eq.less\_than.this.lhs.definition

Eq << Eq[-1].this.rhs.args[0].definition.reversed

Eq << keras.nn.relu.min.greater\_than.apply(i + u + 1, l + u + 1, n)

Eq.less\_than = Eq.less\_than - β[i]

Eq << algebre.less\_than.equal.imply.equal.slice.apply(Eq.less\_than, Eq.K\_equality)

Eq << algebre.less\_than.equal.imply.equal.slice.apply(Eq.less\_than, Eq.V\_equality)

Eq.objective = Eq[13].subs(Eq[-1], Eq[-2])

a = Eq[3].lhs

band\_part = Eq[4].rhs.args[1].args[1].args[1].args[1]

Eq << keras.layers.bert.mask.theorem.apply(a, band\_part)

Eq << Eq[-1].subs(Eq[4].reversed)

Ξ = Symbol.Ξ(definition=band\_part)

Eq.Ξ\_definition = Ξ.this.definition

Eq << Eq[-1].subs(Eq.Ξ\_definition.reversed)

Eq << Eq[-1][i]

Eq << Eq[8][i]

Eq << Eq[-1].this.rhs.args[0].definition

Eq.z\_definition = Eq[-1].this.rhs.subs(Eq[-3])

Eq << Eq.Ξ\_definition.this.rhs.definition

Eq << Eq[-1][i]

Eq.Ξ\_definition = Eq[-1].this.rhs.function.astype(Piecewise)

Eq << Eq.z\_definition.rhs.args[-1].args[0].this.arg.args[0].subs(Eq.Ξ\_definition)

Eq << Eq[-1].this.rhs.astype(Sum)

Eq << Eq[-1].this.rhs.function.args[0].cond.apply(sets.imply.equivalent.contains.astype.contains)

Eq.start\_definition = Eq[9].this.rhs.definition

Eq.stop\_definition = (Eq[10] - 1).this.rhs.astype(Min)

Eq << Eq[-1].subs(Eq.start\_definition.reversed, Eq.stop\_definition.reversed)

Eq << Eq[-1].this.rhs.astype(ReducedSum)

Eq.z\_definition = Eq.z\_definition.subs(Eq[-1])

Eq << Eq[3][i]

Eq << Eq[-1][β[i]:ζ[i]]

Eq << Eq.objective.this.rhs.subs(Eq[-1].reversed)

Eq << Eq[-1].this.rhs.args[0].definition

Eq << Eq.z\_definition.rhs.args[0].this.expand()

k = Eq[-1].rhs.function.variable

Eq << Eq.Ξ\_definition[k]

Eq << Eq[-2].this.rhs.function.function.subs(Eq[-1])

Eq << Eq[-1].subs(Eq.start\_definition.reversed, Eq.stop\_definition.reversed)

Eq << Eq[-1].this.rhs.function.astype(MatMul)

Eq << Eq[-1].this.rhs.function.T

Eq << Eq[-1].this.rhs.function.args[1].astype(Plus)

Eq << Eq[-1].this.rhs.astype(MatMul)

Eq << Eq.z\_definition.this.rhs.subs(Eq[-1])

if \_\_name\_\_ == *'\_\_main\_\_'*:

prove(\_\_file\_\_)

# for detailed reference, please check this thesis

# Self-Attention with Relative Position Representations.pdf

# <https://arxiv.org/abs/1803.02155>