

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, suffix='.py'):
    for name in os.listdir(rootdir):
        path = os.path.join(rootdir, name)
        if path.endswith(suffix):
            name = name[:-len(suffix)]
            if name == '__init__':
                continue
            path = path[:-len(suffix)]
            paths = re.compile(r'[\\/]').split(path)
            # print(paths)
            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, suffix)

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)
```

```

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.algrebre.unequal.equal.implly.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\\*{\\Eg(?:\\[(\\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('\n', '\\n'))
    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]

```



```

        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}} \sqrt{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!= %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, suffix='.py'):
    for name in os.listdir(rootdir):
        path = os.path.join(rootdir, name)

        if path.endswith(suffix):
            yield path
        elif os.path.isdir(path):
            yield from read_all_files(path, suffix)

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 occurs, 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 occurs, 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 = " . json_encode($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('/^(?:\s)+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('/\s.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.implly.equal.exp.apply(Eq[-3]),
algebre.equal.implly.equal.exp.apply(Eq[-2]), algebre.equal.implly.equal.exp.apply(Eq[-1])

Eq << Eq[-1] * OneMatrix(dz)

Eq.lower_part, Eq.upper_part, Eq.diagonal_part =
algebre.equal.equal.implly.equal.transit.apply(Eq[-4], Eq[7][i]), \
    algebre.equal.equal.implly.equal.transit.apply(Eq[-3], Eq[11][i - h]), \
    algebre.equal.equal.implly.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.implly.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(algrebre.imply.equal.pieewise.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(Pieewise)

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 << algrebre.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(Pieewise)

Eq << Eq[-1].this.rhs.args[0].astype(Pieewise)

Eq << Eq[-1].this.rhs.astype(Pieewise)

Eq << algrebre.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(1, i)

Eq << Eq[-1].reversed.subs(Eq[9].reversed)
Eq <=< Eq[11].this.rhs.subs(Eq[-1]), Eq[12].this.rhs.subs(Eq[-1])

 $\beta$  = Eq[9].lhs.base
 $\zeta$  = Eq[10].lhs.base

Eq <=< Eq[2].subs(j, j +  $\beta[i]$ ), Eq[7].subs(j, j +  $\beta[i]$ )

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 = 1 + 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( $\zeta[i]$ ,  $\beta[i]$  + Min(n, 1 + 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, 1 + u + 1, n)

Eq.less_than = Eq.less_than -  $\beta[i]$ 

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)

 $\Xi$  = Symbol. $\Xi$ (definition=band_part)

Eq. $\Xi$ _definition =  $\Xi$ .this.definition

Eq << Eq[-1].subs(Eq. $\Xi$ _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

```