1 输出结果

命题得证

```
[ 1] {Poor(z4), -Smart(z4), Happy(z4)}     A
[ 2] {-Read(z5), Smart(z5)}                A
[ 3] {Read(liming)}                        A
[ 4] {-Poor(liming)}                       A
[ 5] {-Happy(z6), Exciting(z6)}            A
[ 6] {-Exciting(liming)}                   A
[ 7] {-Read(z5), Happy(z5), Poor(z5)}      (1, 2)
[ 8] {-Smart(liming), Happy(liming)}       (1, 4)
[ 9] {Exciting(z6), Poor(z6), -Smart(z6)}  (1, 5)
[10] {Smart(liming)}                       (2, 3)
[11] {Happy(liming), Poor(liming)}         (1, 10)
[12] {Happy(liming), -Read(liming)}        (2, 8)
[13] {Exciting(z6), Poor(z6), -Read(z6)}   (2, 9)
[14] {Happy(liming), Poor(liming)}         (3, 7)
[15] {Happy(liming)}                       (3, 12)
[16] {Exciting(liming), Poor(liming)}      (3, 13)
[17] {-Read(liming), Happy(liming)}        (4, 7)
[18] {Happy(liming)}                       (3, 17)
[19] {-Smart(liming), Exciting(liming)}    (4, 9)
[20] {Exciting(liming), -Read(liming)}     (2, 19)
[21] {Exciting(liming)}                    (3, 20)
[22] {Happy(liming)}                       (4, 11)
[23] {-Read(liming), Exciting(liming)}     (4, 13)
[24] {Exciting(liming)}                    (3, 23)
[25] {Happy(liming)}                       (4, 14)
[26] {Exciting(liming)}                    (4, 16)
[27] {-Happy(liming)}                      (5, 6)
[28] {Poor(liming), -Smart(liming)}        (1, 27)
[29] {Poor(liming), -Read(liming)}         (2, 28)
[30] {Poor(liming)}                        (3, 29)
[31] {-Smart(liming)}                      (4, 28)
[32] {-Read(liming)}                       (2, 31)
[33] {}                                    (3, 32)
```

2 程序代码

```python
# coding: utf-8

from nltk.sem import logic, skolemize
from nltk.inference.resolution import ResolutionProverCommand


expr = logic.Expression.fromstring


class SimpleSolver(object):

    def __init__(self):
        self._proof = None

    def resolution(self, predicates, target):
        ''' 从已知谓词关系公式证明给定谓词公式

        如果能够证明给定谓词公式，返回真，否则返回假

        :param predicates: 已知谓词公式
        :type predicates: list of string
        :param target: 待证明谓词公式
        :type target: string
        '''
        # 将所有谓词关系公式转换成内部表现形式
        conditions = [expr(i) for i in predicates]
        target = expr(target)
        # 使用反演法，将待证明结论取反
        negated_target = target.negate()

        # 构造条件谓词集合
        predicates = conditions + [negated_target]

        # 将谓词集合进行 skolem 标准化，转换为子句集
        clauses_set = [skolemize(i) for i in predicates]

        # 进行归结
        solver = ResolutionProverCommand(None, clauses_set)

        # 检查迭代后的子句集是否为空
        result = solver.prove()

        # 生成证明过程
        self._proof = solver.proof()

        return result

    @property
    def proof(self):
        return self._proof

    def __call__(self, *args, **kwargs):
        return self.resolution(*args, **kwargs)


if __name__ == '__main__':
    # 已知条件
    conditions = [
        # 所有不贫穷且聪明的人都快乐
        'all x.((not Poor(x)) & Smart(x) -> Happy(x))',

        # 那些看书的人是聪明的
        'all y.(Read(x) -> Smart(x))',

        # 李明能看书且不贫穷
        'Read(liming) & not Poor(liming)',

        # 快乐的人过着激动人心的生活
        'all x.(Happy(x) -> Exciting(x))'
    ]

    # 需要证明的定理：李明过着激动人心的生活
    target = 'Exciting(liming)'

    solver = SimpleSolver()
    if solver(conditions, target):
        print('命题得证')
    else:
        print('命题不能被证明为真')
    print(solver.proof)
```