实验课作业

1.命题逻辑的归结推理

编写函数 `ResolutionProp` 实现命题逻辑的归结推理．该函数要点如下：

- 输入为子句集(数据类型与格式详见课件)，每个子句中的元素是原子命题或其否定．
- 输出归结推理的过程，每个归结步骤存为字符串，将所有归结步骤按序存到一个列表中并返回，即返回的数据类型为 `list[str]` ．
- 一个归结步骤的格式为 `步骤编号 R[用到的子句编号] = 子句` ．如果一个字句包含多个公式，则每个公式用编号 `a,b,c...` 区分，如果一个字句仅包含一个公式，则不用编号区分．(见课件和例题)

例子：输入子句集

```
KB = {(FirstGrade,), (~FirstGrade,Child), (~Child,)}
```

则调用 `ResolutionProp(KB)` 后返回推理过程的列表如下：

```
1 (FirstGrade,),
2 (~FirstGrade,Child)
3 (~Child,),
4 R[1,2a] = (Child,),
5 R[3,4] = ()
```

2.最一般合一算法

编写函数 `MGU` 实现最一般合一算法．该函数要点如下：

- 输入为两个原子公式，它们的谓词相同．其数据类型为 `str` ，格式详见课件．
- 输出最一般合一的结果，数据类型为 `dict` ，格式形如{变量：项，变量：项}，其中的变量和项均为字符串．
- 若不存在合一，则返回空字典．

例子：

调用 `MGU('P(xx,a)', 'P(b,yy)')` 后返回字典 `{'xx':'b', 'yy':'a'}` ．

调用 `MGU('P(a,xx,f(g(yy)))', 'P(zz,f(zz),f(uu))')` 后返回字典 `{'zz':'a', 'xx':'f(a)', 'uu':'g(yy)'}` ．

提示

1．只含一个元素的 `tuple` 类型要在末尾加 `,` ．例如 `('a')` 是错误的写法，而正确的写法是 `('a',)` ．

2． `{}` 会被解释成空字典．若要定义空集合请用 `set()` ．

3.一阶逻辑的归结推理

编写函数 `ResolutionFOL` 实现一阶逻辑的归结推理. 该函数要点如下:

- 输入为子句集, `KB` 子句中的每个元素是一阶逻辑公式(不含∃, ∀等量词符号)
- 输出归结推理的过程, 每个归结步骤存为字符串, 将所有归结步骤按序存到一个列表中并返回, 即返回的数据类型为 `list[str]`
- 一个归结步骤的格式为 `步骤编号 R[用到的子句编号]{最一般合一} = 子句` , 其中最一般合一输出格式为"{变量=常量, 变量=常量}".如果一个字句包含多个公式, 则每个公式用编号 `a,b,c...` 区分, 如果一个字句仅包含一个公式, 则不用编号区分。(见课件和例题)

例题: 输入

```
KB = {(GradStudent(sue),),(~GradStudent(x),Student(x)),(~Student(x),HardWorker(x)),
(~HardWorker(sue),)}
```

则调用 `ResolutionFOL(KB)` 后返回推理过程的列表如下:

```
1 (GradStudent(sue),)
2 (~GradStudent(x),Student(x))
3 (~Student(x),HardWorker(x))
4 (~HardWorker(sue),)
5 R[1,2a]{x=sue} = (Student(sue),)
6 R[3a,5]{x=sue} = (HardWorker(sue),)
7 R[4,6] = []
```

输入

```
KB = {(A(tony),),(A(mike),),(A(john),),(L(tony,rain),),(L(tony,snow),),
(~A(x),S(x),C(x)),(~C(y),~L(y,rain)),(L(z,snow),~S(z)),(~L(tony,u),~L(mike,u)),
(L(tony,v),L(mike,v)),(~A(w),~C(w),S(w))}
```

输出

```
1 (A(tony),)
2 (A(mike),)
3 (A(john),)
4 (L(tony,rain),)
5 (L(tony,snow),)
6 (~A(x),S(x),C(x))
7 (~C(y),~L(y,rain))
8 (L(z,snow),~S(z))
9 (~L(tony,u),~L(mike,u))
10 (L(tony,v),L(mike,v))
11 (~A(w),~C(w),S(w))
12 R[2,11a]{w=mike} = (S(mike),~C(mike))
13 R[5,9a]{u=snow} = (~L(mike,snow),)
14 R[6c,12b]{x=mike} = (S(mike),~A(mike),S(mike))
15 R[2,14b] = (S(mike),)
16 R[8b,15]{z=mike} = (L(mike,snow),)
17 R[13,16] = []
```

输入

```
KB = {(On(tony,mike),),(On(mike,john),),(Green(tony),),(~Green(john),),
(~On(xx,yy),~Green(xx),Green(yy))}
```

输出

```
1 (On(tony,mike),),
2 (On(mike,john),),
3 (Green(tony),),
4 (~Green(john),),
5 (~On(xx,yy),~Green(xx),Green(yy)),
6 R[4,5c]{yy=john} = (~On(xx,john),~Green(xx)),
7 R[3,5b]{xx=tony} = (~On(tony,yy),Green(yy)),
8 R[2,6a]{xx=mike} = (~Green(mike),),
9 R[1,7a]{yy=mike} = (Green(mike),),
10 R[8,9] = ()
```

注意

1. 只含一个元素的 `tuple` 类型要在末尾加 `,` . 例如 `('x')` 是错误的写法，而正确的写法是 `('x',)` .
2. `{}` 会被解释成空字典. 若要定义空集合请用 `set()` .
3. 请提交代码时只提交一个 `.py` 代码文件，请不要提交其他文件.
4. 例题和作业都会进行代码测试.
5. 上述作业的输出仅供参考。如果有不同的归结顺序，结果相同的情况，代码也算正确.