实验课程名称：\_\_人工智能\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **一阶谓词归结算法** | | | **实验成绩** |  |
| **实 验 者** | **林文敏** | **专业班级** | **16级计科5班** | **学 号** | **3216004669** |
| **实验日期** | **2018.10.28** | | |  |  |
| 一、实验内容利用一阶谓词归结法证明如下储蓄问题：前提：每个储蓄钱的人都获得利息。结论：如果没有利息，那么就没有人去储蓄钱 **(注：**前提可用谓词公式表达为：  **(**∀x) {(∃y)[S(x,y)∧M(y)]→ (∃z)[I(z)∧E(x,z)]}  结论可表达为：  [～(∃z)I(z)]→ **{**(∀x)(∀y)[M(y) → ～S(x,y) ] } 二、实验设计（原理分析及流程） 方便起见，本实验只实现了用子句集做归结的算法，而子句集则是由人工从上面的谓词公式求得的，易得子句集为：  {  ~S(x, y) V ~M(y) V I(f(x)),  ~S(x, y) V ~M(y) V E(x, f(x)),  ~I(z),  M(b),  S(a, b)  }  **原理分析：**  归结法是将结论取反，并将它与前提做合取，得到的谓词公式通过一系列步骤，取得子句集。  从子句集中任取两个子句C1、C2，如果一个子句里存在一个谓词L1，在另外一个子句中存在L2，并且L1和~L2可以通过合一变成同一个谓词公式，则可以将这两个子句做归结，得出的新字句为R = (C1o – {L1o}) U (C2o – {L2o})，其中o为L1和~L2的mgu。  如果重复以上步骤得出了空子句，则归结法成功，否则为失败。  **流程：**   1. 算法的主体流程是对子句集做迭代归结，要求任意两个子句不能重复做归结（没有意义），并且该迭代要在理论上可以使任意两个字句做归结，不能遗漏，实际上只要得出了空子句，就可以直接结束。 迭代的实现方法为：设置一个新字句列表和旧字句列表，旧字句列表里的子句是已经做过迭代归结的，新字句列表里的没有，每次从新字句列表里取出一个新字句作为当前字句，将其与每一个旧字句尝试做归结，得出的归结字句放入新字句列表，迭代完成后将当前字句放入旧字句列表，做下一轮迭代，直至归结成功或新字句列表为空而失败。将两个列表都初始化为初始集合，不需要担心字句会对自己尝试做归结，因为这永远不会发生。 2. 两个子句做归结的算法是核心，主要思路是寻找一个在两个子句都出现的谓词P，并且一个为P一个为~P，找到后尝试对P与~P求mgu，如果求得mgu，则将两个子句都应用mgu置换，再去掉P和~P，得出归结的新字句。 3. 求mgu的算法重点在获取v和t，需要先将两个个体词（Object）相同的外壳剥去，然后再寻找变量。 4. 对字句做置换的算法比较简单，就不再赘述了。 | | | | | |
| **三、实验代码及数据记录**  **1.代码**  # Python代码  class Sentence:  def \_\_init\_\_(self, preds):  self.preds = preds  def \_\_repr\_\_(self):  return ' V '.join((repr(p) for p in self.preds)  ) if self.preds else '""'  class Predicate:  def \_\_init\_\_(self, name, objs, is\_not=False):  self.name = name  self.objs = objs  self.is\_not = is\_not  def \_\_repr\_\_(self):  return ('~' if self.is\_not else '') + \  '{p}({objs})'.format(  p=self.name, objs=', '.join(  (repr(o) for o in self.objs)))  class Object:  CONSTANT = 0  VARIANT = 1  FUNC = 2  def \_\_init\_\_(self, name, t):  self.name = name  self.t = t  def \_\_eq\_\_(self, value):  return self.name == value.name and self.t == value.t and (  self.t != Object.FUNC or self.argv == value.argv)  def \_\_repr\_\_(self):  return self.name if self.t != Object.FUNC else '{f}({argv})'.format(  f=self.name, argv=self.argv)  @staticmethod  def create\_func(name, argv):  o = Object(name, Object.FUNC)  o.argv = argv  return o  @staticmethod  def create\_var(name):  return Object(name, Object.VARIANT)  @staticmethod  def create\_const(name):  return Object(name, Object.CONSTANT)  def deep\_copy(self):  if self.t == Object.FUNC:  return Object.create\_func(self.name, self.argv.deep\_copy())  return Object(self.name, self.t)  def get\_var(self, var=None):  if self.t == Object.VARIANT:  return self if var is None or self.name == var else None  if self.t == Object.FUNC:  return self.argv.get\_var(var)  return None  def unify(sentences):  new\_sens = sentences.copy()  old\_sens = sentences.copy()  while new\_sens:  sen1 = new\_sens.pop()  for sen2 in old\_sens:  sen = resolve(sen1, sen2)  if sen is not None:  if not sen.preds:  return True  new\_sens.append(sen)  old\_sens.append(sen1)  return False  def resolve(sen1, sen2):  for p1 in sen1.preds:  for p2 in sen2.preds:  if p1.name == p2.name and p1.is\_not != p2.is\_not:  print('try resolve [{} , {}].'.format(sen1, sen2))  mgu = get\_mgu(p1, p2)  if mgu is None:  continue  new\_sen = merge(sen1, p1, sen2, p2, mgu)  print('resolve [{} , {}] -> {}.'.format(sen1, sen2, new\_sen))  print('\tmgu is [{}].'.format(  ', '.join(('({}/{})'.format(u[1], u[0]) for u in mgu))))  return new\_sen  return None  def get\_mgu(p1, p2):  objs1 = list(p1.objs)  objs2 = list(p2.objs)  if len(objs1) != len(objs2):  return None  mgu = []  for i in range(len(objs1)):  if objs1[i] == objs2[i]:  continue  o1, o2 = diff(objs1[i], objs2[i])  v, t = get\_v\_t(o1, o2)  if v is None:  return None  mgu.append((v, t))  substitute(objs1, v, t)  substitute(objs2, v, t)  return mgu  def merge(sen1, p1, sen2, p2, mgu):  new\_preds = list(sen1.preds)  new\_preds.remove(p1)  new\_preds.extend(sen2.preds)  new\_preds.remove(p2)  for i, pred in enumerate(new\_preds[:]):  l = list(pred.objs)  for v, t in mgu:  substitute(l, v, t)  new\_preds[i] = Predicate(pred.name, tuple(l), pred.is\_not)  return Sentence(tuple(new\_preds))  def substitute(objs, v, t):  for i in range(len(objs)):  objs[i] = subs\_obj(objs[i], v, t)  def subs\_obj(o, v, t):  if o == v:  return t  if o.t == Object.FUNC:  return Object.create\_func(o.name, subs\_obj(o.argv, v, t))  return o  def get\_v\_t(o1, o2, first=True):  if o1.t == Object.VARIANT:  assert o2.get\_var(o1.name) is None  return o1, o2  else:  if not first:  return None, None  return get\_v\_t(o2, o1, False)  def diff(o1, o2):  if o1.t == o2.t and o1.name == o2.name:  assert o1.t == Object.FUNC  return diff(o1.argv, o2.argv)  return o1, o2  if \_\_name\_\_ == '\_\_main\_\_':  x = Object.create\_var('x')  y = Object.create\_var('y')  z = Object.create\_var('z')  a = Object.create\_const('a')  b = Object.create\_const('b')  fx = Object.create\_func('f', x)  sp = Predicate('S', (x, y), is\_not=True)  mp = Predicate('M', (y, ), is\_not=True)  ip = Predicate('I', (fx, ))  ep = Predicate('E', (x, fx))  rip = Predicate('I', (z, ), is\_not=True)  rmp = Predicate('M', (b, ))  rsp = Predicate('S', (a, b))  sen1 = Sentence((sp, mp, ip))  sen2 = Sentence((sp, mp, ep))  r1 = Sentence((rip, ))  r2 = Sentence((rmp, ))  r3 = Sentence((rsp, ))  sents = [sen1, sen2, r1, r2, r3]  for sen in sents:  print(sen)  print('-' \* 80)  assert unify(sents)  print('-' \* 80)  print('success')  **2. 结果截图** | | | | | |