Trabalho prático individual nº 1

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2017/2018

27 de Outubro de 2017

I Observações importantes

1. This assignment should be submitted via *Moodle* within 28 hours after the publication of this description. The assignment can be submitted after 28 hours, but will be penalized at 5% for each additional hour.

2. Complete the requested functions in module "tpi1.py"provided together with this description and include your name and number. Test cases, print statements, and other non-relevant code should be commented or removed.

3. You can discuss this assignment with colleagues, but you can not copy their programs neither in whole nor in part.

4. Include a comment with the names and numbers of the colleagues with whom you discussed this assigment. If you turn to other sources, identify those sources as well.

5. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved plagiarism will have their submissions canceled.

6. The the submiited programs will be evaluated taking into account: correctness and completeness; style; and originality / evidence of independent work. Correction and completeness will be usually evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teachers in order to appropriately credit their work.

II Exercícios

Together with this description, you can find module `semnet`. It is similar to the one used in practical classes, but small changes and additions were introduced. In particular, the class `iAssociation` now has the following three derived classes:

- `AssocOne` - an association that accepts a single symbolic value as second argument. Example: `hasFather` (a person has only one father)

- `AssocSome` - an association that accepts multiple symbolic values. Example: `likes` (a person may like meat, fish and vegetables)
- `AssocNum` - an association that accepts a single numeric values. Example: `height` (the height of a person)

The following methods were added to the `SemanticNetwork` class:

- `addInverse(assocname1,assocname2)` - declares that the two given associations are the inverse of each other. Example: the associations `fatherOf` and `hasFather` are inverse of each other.
- `addOpposite(assoc1,assoc1)` - declares that the two given associations are the opposite of each other. Example: the associations `likes` and `dislikes` are the opposites.

Both methods store the given information in dictionaries.

The module `tpi1_tests` contains a semantic network for tests. You can add other test code in this module. Don't change the `semnet` module. Module `tpi1` contains the class `MySemNet`. In the following exercices, you are asked to complete certain methods of this class. All code that you may need to develop should me integrated in the same module.

1. `query_local_relations(e1,relname,e2)` - Returns a list of pairs `(relname,e2)` such that a relation `relname` exists between entities `e1` and `e2`. Note that arguments `relname` and `e2` are optional. This query does not involve inheritance, i.e. it is based on local declarations only. However, it should take into account information about inverse relations.

   Exemplos:

   ```
   >>> z.query_local_relations('aristotle',e2='nicomachus')
   [ ('hasFather', 'nicomachus') ]

   >>> z.query_local_relations('aristotle')
   [ ('member', 'man'), ('hasFather', 'nicomachus'),
     ('hasFather', 'ariston') ]

   >>> z.query_local_relations('man',relname='hasWeight')
   [('hasWeight', 70), ('hasWeight', 80)]

   >>> z.query_local_relations('plato')
   [ ('member', 'man'), ('professorOf', 'philosophy'),
     ('fatherOf', 'socrates')]
   ```

2. `query_excluding_opposites(entity,assocname)` - Returns a list of values of the given association for the given entity, including inherited values, but excluding values for which there is an opposite association. For example, `man likes meat`, but `socrates dislikes meat`, therefore that property of `man` will not be inherited by `socrates`.

   Exemplo:

   ```
   >>> z.query_excluding_opposites('socrates','likes')
   [ 'philosophy', 'vegetables', 'milk', 'sophroniscus',
     'mathematics', 'phaenarete' ]
   ```

3. `query(entity,relname)` - Returns a list of values for a given relation in a given entity, including inherited values. The different types of relations are handled in different ways. In which concerns `Member` and `Subtype` relations, only the local ones are relevant. Inheritance of `AssocOne` and `AssocNum` is processed with cancelling, i.e. the existance of one of these associations in a given entity cancels the effects of a similar association with the same name in a predecessor entity. When there are several declarations of an `AssocOne` association, the most common value should be returned. When there are several declarations of an `AssocNum` association, the average of all values should be returned. In the case of `AssocSome`, inheritance is processed without cancelling. Therefore, all such associations found in predecessors are relevant. Finally, because different users may use a given association name for different types of associations, the type of association most frequently used with a given name should be considered the correct type of the association. For example, `hasHeight` was used four times as `AssocNum` and only once as `AssocOne`, therefore the type of this association is considered to be `AssocNum` and the only declaration in which it was used as `AssocOne` is ignored.

   Note: In this exercice, ignore inverse and opposite relations.

   Exemplos:

```
>>> z.query('socrates','hasHeight')
[1.775]

>>> z.query('socrates','hasWeight')
[75.0]

>>> z.query('socrates','likes')
['mathematics', 'phaenarete', 'meat', 'milk', 'philosophy', 'sophroniscus']

>>> z.query('socrates','hasFather')
['sophroniscus']
```

III Esclarecimento de dúvidas

O acompanhamento do trabalho será feito via `http://detiuaveiro.slack.com`. O esclarecimento das principais dúvidas será também colocado aqui. Bom trabalho!