Trabalho prático individual nº 2

Inteligência Artificial
Ano Lectivo de 2022/2023

16 de Dezembro de 2022

I Observações importantes

1. This assignment should be submitted via *GitHub* within 27 hours after its publication. The assignment can be submitted after 27 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested methods in module `"tpi2.py"`, provided together with this description. Keep in mind that the language adopted in this course is Python3.
3. Include your name and number and comment or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi2.py"`.
4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part. Limit these discussions to the general understanding of the problem and avoid detailed discussions about implementation.
5. Include a comment with the names and numbers of the colleagues with whom you discussed this assignment. If you turn to other sources, identify these sources as well.
6. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved in plagiarism will have their submissions canceled.
7. The submitted programs will be evaluated taking into account: performance; style; and originality / evidence of independent work. Performance is mainly evaluated concerning correctness and completeness, although efficiency may also be taken into acount. Performance is evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teacher in order to appropriately credit the student's work.

II Exercícios

Together with this description, you can find modules semantic_network, bayes_net and constraintsearch . They are similar to the ones used in practical lectures, but small changes and additions were introduced.

Module `tpi2` contains some derived classes. In the following exercices, you are asked to complete certain methods in these classes. Any other code that you develop and integrate in other modules will be ignored.

The module `tpi2_tests` contains some test code. You can add other test code in this module. Don't change the semantic_network, bayes_net and constraintsearch modules.

You can find the intended results of `tpi2_tests` in the file `tpi2_results.txt`

The responses to the main questions asked by students during this TPI will be collected in section III below.

1. The small semantic networks module used in practical classes was designed to facilitate the initial learning of the subject by the students. For this reason, many aspects were left out. In a more professional system, they would have to be considered. The attached module `semantic_network` was designed based on the one you already know, but has some differences related to type checking. For example, in the classes module, it is not possible to know if an association is established between two objects or between two types. There is also no way to define whether an association accepts only one value or multiple values. The `Association` class constructor was therefore modified and now has the following arguments:

   - `entity1` - First argument of the association;
   - `name` - Association name;
   - `entity2` - Second argument (or value) of the association;
   - `card` - Cardinality of the association, which can be:
     - `None` - To be used between objects, i.e. `entity1` e `entity2` are necessarily names of objects.
     - `"one"` - To be used between types. Specifies that the association, when used for objects, will accept a single value. Example: a person has a single father.
     - `"many"` - To be used between types. Specifies that the association, when used for objects, will accept multiple values (zero or more). Example: a person can have multiple children.
   - `default` - To be used for associations between types. It specifies the default value (second argument) of the association. Only associations with cardinality `"one"` can have a default value. Example: by default, the hight of a "man"is 1.75.

   Since it does not carry out any validation, the original module (from lectures) allows any string of characters `"xpto"` to be used by the same user simultaneously as the name of an object and as the name of a type. In the module `semantic_network` now provided, the class `SemanticNetwork` has three new methods to create relationships, which return `True` if it is possible to create the relationship and `False` otherwise.

   - `add_member(user,obj,type)`
   - `add_subtype(user,subtype,supertype)`
   - `add_association(user,e1,name,e2,card,default)`

   Note that the implementation of these methods is very incomplete due to the lack of auxiliary methods to use for type checking. It is on these auxiliary features that you should now focus your attention:

   a) Develop a method `is_object(user,obj)` in the `MySN` class that checks if the object `obj` exists in `user` declarations

b) Develop a method `is_type(user,type)` in the `MySN` class that checks if the type `type` exists in `user` declarations

c) In order to maintain consistency, it is necessary to check, for each new relation to be added, whether the types of entities in that relation are consistent with the types already existing in the network. Since, in this semantic network, it is possible to declare a concrete association between two objects without declaring the general characteristics of that association at the type level, it becomes necessary to develop type inference mechanisms. So, develop the following methods in the `MySN` class:

- `infer_type(user,obj)` - Infers the type of an object based on the declarations of `user`. The result must be:
  - `<type>` - the type of the object, provided that it can be inferred;
  - `"__unknown__"` - if `obj` exists, but its type cannot be inferred;
  - `None` - if `obj` doesn't exist.
- `infer_signature(user,assoc)` - Infers the signature of the association based on the declarations of `user`. The result must be:
  - `(t1,t2)` - where t1 and t2 are the types of the entities involved in the association;
  - `None` - if the association does not exist.

These two functions depend on each other, so to be completely correct they will have to be implemented as mutually recursive functions. However, you can obtain part of the score by solving the simpler cases.

2. Bayesian networks have several mathematical properties. One such property is the *Markov blanket*. It is defined as a set containing all the variables that protect a given variable from the rest of the network. All knowledge about this variable can be extracted from the knowledge of its Markov blanket. The Markov blanket of a variable A in a Bayesian network is the set of nodes consisting of the parents and children of A and the other parents of the children.

   Implement in the `MyBN` class a `markov_blanket()` method that, given a variable of the network, `var`, returns a list of the variables that make up the Markov blanket of that variable. Not that the Bayesian network implementation in `bayes_net` module is based on lists of true and false variables and not on explicit boolean values.

   For testing purposes, the network represented in Figure 1 is included in the `tpi2_tests` module.

3. The following exercices cover the topic of constraint-based search for assignment problems.

   a) Implement in class `MyCS` the method `propagate(var)` which is called in `search()` after choosing a specific value for `var`.

   b) Implement in class `MyCS` the method `higherorder2binary(ho_c_vars,unary_c)`, which will be used to transform a higher-order constraint involving the variables in list `ho_c_vars`, into a set of binary constraints involving those variables and a new auxiliary variable (see slides). This method updates the domains and constraints dictionaries and does not return a result. `unary_c` is a function that describes the higher-order constraint as unary constraint on the new auxiliary variable.

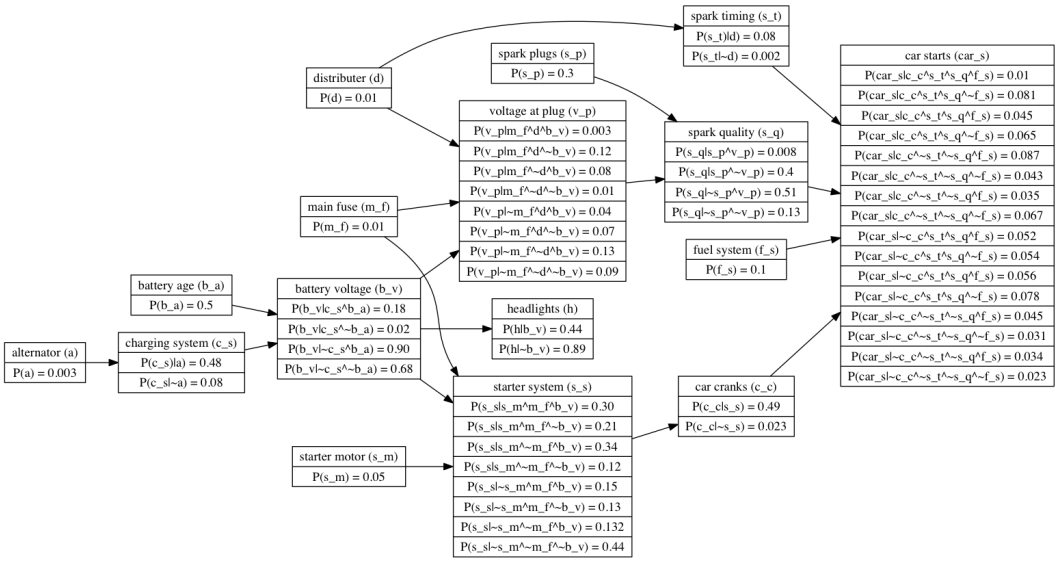

Figura 1: Bayesian network for a car that does not start

III Clarification of doubts

This work will be followed through `http://detiuaveiro.slack.com`. The clarification of the main doubts will be placed here.

1. Quanto vale cada questão?

   RespostaResposta: 1.a) = 11%; 1.b) = 11%; 1.c) = 36%; 2. = 11%; 3.a) = 11%; 3.b) = 20%.

2. No exercício 1.c), aparece o argumento `"xpto"` da função `infer_type`; temos de o considerar?

   Resposta: Esse argumento é para assinalar que, se precisarem, estão à vontade para introduzir argumentos adicionais, desde que sejam opcionais.

3. Podemos substituir `xptoval` por `None?` já que dá erro por a variável `xptoval` não existir

   Resposta: Sim.

4. Nos exs. 1.a) e 1.c) o `obj` que se refere é o `obj` do `Member` ou um objeto de uma classe?

   Resposta: É um "objecto"no sentido habitual que temos usado no contexto do módulo de redes semânticos

5. No 1.b o Descartes nao devia ser um tipo de número mas o resultado dá `True`. É correto?

   Resposta: Se `z.is_type('descartes','number')` dá `True`, é porque `'descartes'`, nas suas declarações, usa `'number'` como tipo

6. A única vez que o 'number' é usado como argumento é numa linha que tem o seguinte código "z.add_association('descartes','mamifero','altura','number','one',1.2)"; não percebo muito bem como 'number' pode ser considerado tipo?

Resposta: "number"é tipo porque a cardinalidade só pode ser usada quando a declaração da associação se refere a tipos.

7. Pode dar um exemplo de assinatura?

   Resposta: No exemplo anterior, vê-se que "altura"tem como assinatura ("mamifero","number")

8. Na função `"infer_signature()"`, se a associação apenas aparece aplicada a objectos, o que é que devolvemos?

   Resposta: Devolve-se os tipos dos objectos.

9. A entrada do `infer_type()` será sempre um objecto, ou devemos testar o caso de o parametro `obj` ser um tipo?

   Resposta: Se não for um objecto, o resultado é `None` (diz o enunciado)

10. No ex. 2, importa a ordem dos elementos na lista retornada?

    Resposta: não é relevante.

11. Poderia fornecer mais algum teste para a 1.c)?

    Resposta: Sim, acrescentei casos de teste.

12. No ex. 3.a), temos de ter exactamente 3 chamadas da função (calls==3)?

    Resposta: Pelo menos têm que ter uma solução correcta e um número de chamadas nessa ordem de grandeza. Nota: se vocês alterarem a ordem dos valores das variáveis no método de propagação, é possível que obtenham um número de chamadas diferente.

13. Pode-se redefinir o método `search()` na classe `MyCS`?

    Resposta:Não, é para seguir o esquema que está definido.

14. Podemos adicionar ao `tpi2.py` as funcoes que quisermos? (dento e fora das classes)

    Resposta: Sim, podem.

15. Porquê é que no ex. 1.c) 1.85 dá `'number'`?

    Resposta: Porque 1.85 é a `'altura'` de `'socrates'` e `'altura'` está declarada como sendo uma associação em que `e2` é `'number'`.

16. Poderia dizer porque `'filosofia'` dá `__unknown__`?

    Resposta: Porque `'filosofia'` aparece como `e2` em associações entre objectos (`card==None`), portanto é um objecto, mas não há nenhuma declaração que nos diga qual o tipo de `'filosofia'`. `card!=None` aparece na aplicação de associações entre tipos; serve para especificar a semântica da associação (valor único ou múltiplos valores, e também se tem valor por omissão).

    Resposta

17. Temos de garantir que `'sócrates'` é um `'mamífero'` para garantir que a altura dele é um `'number'`?

    Resposta: Não depende disso. Nestes exercícios da `MySN`, não se fala de herança. Assumam que uma associação, estando declarada num tipo, não poderá estar declarada em mais nenhum tipo, mesmo que seja ascendente ou descendente. Poderá aparecer, num (tipo ou) supertipo e em objectos pertencentes a tipos descendentes desse supertipo. Caso a

associação esteja declarada entre tipos, a inferência da assinatura da associação deve vir daí.

Resposta

18. Então se pedir o tipo de `'socrates'`, deve ser `'mamífero'` (a partir da assinatura de `'altura'`) ou `'homem'` ? (sendo `'socrates'` declarado membro de `'homem'`)

    Resposta: Se existe uma declaração explícita, essa prevalece, portanto dá `'homem'`. Da mesma forma, quanto à assinatura de uma associação: prevalece a declaração explícita, caso exista.