Machine Arguing Assessed Coursework 2

Due date: 22 November 2018

Electronic submission via LabTS + CATE

Groups of two students allowed/welcome

(but note that only one student per group will have access rights to the LabTS repository)

Throughout: **Prolog = Sicstus Prolog**, to run under **Linux**

Part 1 (25% marks) - Abstract Argumentation (AA)

Consider the following Prolog representation of arguments and attacks in AA:

`argument(a).` for `a` is an argument (in an AA framework).
`attacks(a,b).` for argument `a` attacks argument `b` (in an AA framework).

Give a Sicstus Prolog *meta-interpreter* (program) defining the predicate `grounded/1` such that if

```
grounded(a).
```

succeeds for some abstract argument `a` and AA framework represented as a Prolog program as above, then `a` belongs to the grounded set of arguments.

Your implementation does not need to (but may) generate the grounded set of arguments.

You should test your program with several AA frameworks, including (but not limited to) the ones below. It may be useful to use ASPARTIX to check whether you get the right answers.

- For the AA framework $(Args, attacks)$ with

  $Args = \{a, b, c\}$

  $attacks = \{(a, b), (b, c)\}$

  the grounded extension is $\{a, c\}$, so, for example, `grounded(a).` should succeed, but `grounded(b).` should (finitely) fail.

- For the AA framework $(Args, attacks)$ with

  $Args = \{a, b, c, d\}$

  $attacks = \{(a, a), (a, b), (b, a), (c, d), (d, c)\}$

  `grounded(X).` should not succeed for any argument `X`, since the grounded extension here is the empty set.

Marks will be awarded for correctness rather than efficiency or coding style, but try and avoid obvious inefficiencies and use good Prolog coding standards.
Don't worry too much if your program loops for complicated AA frameworks. However, it should not loop for the frameworks given above and for other simple frameworks. *Hint: Think carefully about arguments attacking one another as well as about self-attacks.*

Part 2 (25% marks) - Relation-based Argument Mining

Construct a Bipolar Abstract Argumentation (BAA) framework from the following dialogue, taken from the film *Twelve Angry Men*, and where 'NO. X' stands for 'juror X'. For simplicity, assume that each paragraph contains exactly *one* argument.

> NO. 3: I never saw a guiltier man in my life. You sat right in court and heard the same thing I did. The man's a dangerous killer. You could see it.
>
> NO. 8: He's nineteen years old [he is too young to be guilty].
>
> NO. 3: That's old enough. He knifed his own father, four inches into the chest. An innocent little nineteen-year old kid. They proved it a dozen different ways. Do you want me to list them?
>
> NO. 8: It's not so easy for me to raise my hand and send a boy off to die without talking about it first.
>
> NO. 7: I think the guy's guilty. You couldn't change my mind if you talked for a hundred years.
>
> NO. 8: Look, this boy's been kicked around all his life. You know, living in a slum, his mother dead since he was nine. That's not a very good head start. He's a tough, angry kid. I think maybe we owe him a few words [before considering him guilty].
>
> NO. 10: We don't owe him a thing. He got a fair trial, didn't he? You're not going to tell us that we're supposed to believe him, knowing what he is. I've lived among 'em all my life. You can't believe a word they say.
>
> NO.2: I just think he's guilty. I thought it was obvious. I mean nobody proved otherwise.
>
> NO. 8: Nobody has to prove otherwise. The burden of proof is on the prosecution. The defendant doesn't have to open his mouth. That's in the Constitution. The Fifth Amendment.
>
> NO.3: Okay, let's get to the facts. Number one, let's take the old man who lived on the second floor right underneath the room where the murder took place. At ten minutes after twelve on the night of the killing he heard loud noises in the upstairs apartment. He said it sounded like a fight. Then he heard the kid say to his father, "I'm gonna kill you.!" A second later he heard a body falling, and he ran to the door of his apartment, looked out, and saw the kid running down the stairs and out of the house. Then he called the police. They found the father with a knife in his chest.
>
> FOREMAN: And the coroner fixed the time of death at around midnight.
>
> NO. 4: The boy's entire story is flimsy. He claimed he was at the movies. That's a little ridiculous, isn't it? He couldn't even remember what pictures he saw.

NO. 10: Look, what about the woman across the street? If her testimony [to have seen the killing] don't prove [that the boy is guilty], then nothing does.

NO. 8: [She saw the killing] through the windows of a passing elevated train [so her testimony is not reliable].

NO. 10: Okay. And they proved in court that you can look through the windows of a passing train at night and see what's happening on the other side. They proved it.

NO.6: I started to be convinced, you know, with the testimony from those people across the hall. Didn't they say something about an argument between the father and the boy around seven o'clock that night?

NO. 8: That's right. Eight o'clock. They heard the father hit the boy twice and then saw the boy walk angrily out of the house. [It does not prove that the boy killed him].

Represent the BAA framework in Prolog, using `argument/1` and `attacks/2` as in Part 1, as well as a predicate `supports/2` with

`supports(a,b).` for argument `a` supports argument `b` (in a BAA framework).

Identify each argument as `an`, where `n` indicates the position of the argument in the dialogue. Thus, the argument taken from the first utterance is `a1` and the argument taken from the last utterance is `a17`. Moreover, include an argument `a0` standing for "not guilty" (implicit in the dialogue).

Part 3 (25% marks) - QuAD frameworks and DF-QuAD

Consider the Prolog representation of a QuAD framework with `argument/1`, `attacks/2` and `supports/2` as in Part 2, as well as `base/2` with

`base(a,0.2).` for argument `a` has a base score of `0.2` (in a QuAD framework).

Give a Sicstus Prolog meta-interpreter (program) defining the predicate `strength/2` such that if

```
strength(a,X).
```

succeeds e.g. with `X=0.7` for some abstract argument `a` and QuAD framework represented as a Prolog program as above, then the DF-QuAD algorithm computes a strength of `0.7` for `a`.

You should test your program with several QuAD frameworks, including (but not limited to) QuAD frameworks (for different choices of base scores) corresponding to the BAA framework of Part 2. It may be useful to use Arg&Dec to check whether you get the right answers.

Part 4 (25% marks) - Assumption-based Argumentation (ABA)

Consider the following Prolog representation of the main concepts in a flat Assumption-Based Argumentation (ABA) framework:

`myAsm(a).`	for a is an assumption
`contrary(a,b).`	for the contrary relation $\overline{a} = b$
`myRule(p,[b,c]).`	for inference rule $p \leftarrow b, c$
`myRule(b,[]).`	for inference rule $b \leftarrow$

i) Give a Sicstus Prolog *meta-interpreter* (program) defining the predicate `argument/1` such that if `argument((C,X)).` succeeds then the (set corresponding to the) list of assumptions `X` supports an argument for the claim `C`, that is `X` $\vdash$ `C` is an ABA argument, represented as `(C,X)`.

Hint: The assumptions in the supporting set of an ABA argument can have any order, so the order of assumptions in X does not matter.

ii) Give a Sicstus Prolog *meta-interpreter* (program) defining the predicate `attacks/2` such that if `attacks((C1,X1),(C2,X2)).` succeeds then the argument `(C1,X1)` attacks the argument `(C2,X2)`.

Your implementations of `argument/1` and `attacks/2` should be able to generate arguments and attacks, respectively, in addition to testing whether they are so. For example, given

```
myAsm(a).
myAsm(b).
contrary(a,p).
myRule(p,[b]).
myRule(p,[]).
```

your implementation should behave as follows:

`argument((p,X)).`	should give `X=[b]` or `X=[]`
`argument((C,[b])).`	should give `C=b` or `C=p`
`argument(a,[a]).`	should succeed
`attacks((p,X),(C,Y)).`	should give `X=[], C=a, Y=[a]` etc

Marks will be awarded for correctness rather than efficiency or coding style, but try and avoid obvious inefficiencies and use good Prolog coding standards.

iii) Consider the following setting:
You are working in a hospital, allocating appointments to patients. Today, there are two patients, Anne (a) and Boris (b). They both tell you that they could probably make the 8am or the 6pm appointment. While talking to Anne, she tells you that she has a child, so you take a note that the 8am appointment would not be optimal for her because she has to bring her child to the nursery at this time. During your conversation with Boris, he mentions that he is taking sports classes and would consequently prefer not to have the 6pm appointment. However, you are not completely sure whether he is telling the truth since he looks very overweight and does not seem to be the kind of person who does sports regularly. On the basis of all this information you now have to decide who gets which appointment.

Your strategy for making a decision will be ABA, where the ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^- \rangle$ has

- $\mathcal{A} = \{$*free6pm(a)*, *free8am(a)*, *free6pm(b)*, *free8am(b)*,
  child(a), *sports(b)*, *overweight(b)*,
  not_get6pm(a), *not_get8am(a)*, *not_get6pm(b)*, *not_get8am(b)*$\}$
- $\overline{\textit{free6pm(a)}} = \textit{not\_free6pm(a)}$, $\overline{\textit{free8am(a)}} = \textit{not\_free8am(a)}, \ldots$
  $\overline{\textit{child(a)}} = \textit{not\_child(a)}$, $\overline{\textit{sports(b)}} = \textit{not\_sports(b)}$, $\overline{\textit{overweight(b)}} = \textit{not\_overweight(b)}$,
  $\overline{\textit{not\_get6pm(a)}} = \textit{get6pm(a)}$, $\overline{\textit{not\_get8am(a)}} = \textit{get8am(a)}, \ldots$

- $\mathcal{R} = \{$*get6pm(a)* $\leftarrow$ *free6pm(a)*, *not_get8am(a)*, *not_get6pm(b)*;
  get8am(a) $\leftarrow$ *free8am(a)*, *not_get6pm(a)*, *not_get8am(b)*;
  get6pm(b) $\leftarrow$ *free6pm(b)*, *not_get8am(b)*, *not_get6pm(a)*;
  get8am(b) $\leftarrow$ *free8am(b)*, *not_get6pm(b)*, *not_get8am(a)*;
  not_free8am(a) $\leftarrow$ *child(a)*;
  not_free6pm(b) $\leftarrow$ *sports(b)*;
  not_sports(b) $\leftarrow$ *overweight(b)*$\}$

a) Download the file `appointmentExample.pl` from CATE/LabTS and run your implementations in Parts 1 and 4.i)-ii) to determine whether *free6pm(b)* belongs to the grounded extension. Write down the query(-ies) you used and the results.

b) Who gets which appointment according to the grounded extension? Who would get which appointment if you had used the stable extensions semantics?

SUBMISSION –

Modify the Prolog files on LabTS[1] as follows:

(i) file `grounded.pl` should amount to your answer to Part 1 (don't include any examples in your code!), implemented in Sicstus Prolog and running on the Linux lab machines

(ii) file `angrymen.pl` should amount to your answer to Part 2, implemented in Sicstus Prolog and running on the Linux lab machines

(iii) file `df_quad.pl` should amount to your answer to Part 3 (don't include any examples in your code!), implemented in Sicstus Prolog and running on the Linux lab machines

(iv) file `aba.pl` should include your answers to questions 4i) and 4ii) (don't include any examples in your code!), implemented in Sicstus Prolog and running on the Linux lab machines

Marks will be deduced if your code does not compile under Linux/Sicstus and/or generates run-time errors, so make sure you include the necessary libraries if you are using predicates which are not built-in.

Finally, upload on CATE:

(v) a file `appointment.pdf` with your answers to questions a) and b) in Part 4iii).

[1]The url you need to clone to get your repo is:
`https://gitlab.doc.ic.ac.uk/lab1819_autumn/474_cw2_<login>.git`
where `<login>` needs to be replaced with your college user name.