Answering Epistemic Questions

In this case study, we use GLIF to answer yes/no question that require handling the knowledge of different people. For example, given the input

```
John knows that Mary or Eve knows that Ping hs a do
g.
Mary doesn't know if Ping has a dog.
Does Eve know if Ping has a dog?
```

the system should reply *"Yes"*.

In [1]:
```
archive tmpGLIF/examples epistemic
```

Successfully changed archive

Grammar

This case study was created for an older version of GLIF that didn't support multi-sentence input. Therefore, a sequence of statements followed by a question is parsed into a single AST (cat `QSeq`).

In [2]:
```
abstract Epistemic = {
    cat
        S;
        SSeq;
        QSeq;
        Cl;
        Agent;
        NP;
        VP;
        Polarity;
    fun
        positive, negative : Polarity;

        makeS : Polarity -> Cl -> S;
        simpleCl : NP -> VP -> Cl;
        knowCl : NP -> S -> Cl;
        knowWhetherCl : NP -> Cl -> Cl;

        makeQSeq : SSeq -> Cl -> QSeq;
        makeSSeq : SSeq;
        append : SSeq -> S -> SSeq;

        npify : Agent -> NP;
        and : NP -> NP -> NP;
        or : NP -> NP -> NP;
```

```
        john, mary, eve, ping : Agent;
        have_cat, have_dog : VP;
        love, hate : NP -> VP;
}
```

Successfully imported Epistemic.gf

In [3]:
```
concrete EpistemicEng of Epistemic = open SyntaxEng, Paradig
  lincat
    S = S;
    SSeq = Str;
    QSeq = Str;
    Cl = Cl;
    Agent = PN;
    NP = NP;
    VP = VP;
    Polarity = Pol;

  lin
    positive = positivePol;
    negative = negativePol;

    makeS pol cl = mkS pol cl;
    simpleCl p v = mkCl p v;
    knowCl p s = mkCl p (mkVS (mkV "know")) s;
    knowWhetherCl np cl = mkCl np (mkVP (mkVQ (mkV "know"))
    makeQSeq sseq cl = sseq ++ (mkQS cl).s ! QDir ++ "?";
    makeSSeq = "";
    append sseq s = sseq ++ s.s ++ ".";
    npify a = mkNP a;
    and a b = mkNP and_Conj a b;
    or a b = mkNP or_Conj a b;
    john = mkPN "John";
    mary = mkPN "Mary";
    eve = mkPN "Eve";
    ping = mkPN "Ping";
    have_cat = mkVP have_V2 (mkNP aSg_Det (mkN "cat"));
    have_dog = mkVP have_V2 (mkNP aSg_Det (mkN "dog"));
    love p = mkVP (mkV2 (mkV "love")) p;
    hate p = mkVP (mkV2 (mkV "hate")) p;
}
```

Successfully imported EpistemicEng.gf

In [4]:
```
-- some random sentences
gr -number=5 -cat=S | l
```

Mary or Ping doesn't hate John
Ping and Ping know if Eve has a dog
John knows if Eve has a dog
John or John has a dog
Eve doesn't know if Ping has a cat

In [5]:
```
gr -number=5 -cat=QSeq -depth=5 | l
```

do Eve and Eve or Mary or Ping have a cat ?
John doesn't have a cat . do John and Eve or Mary and Mary know that John has a cat ?
Ping has a cat . does Eve know if Mary or John loves Mary ?
do Eve and John or John know if John and Mary know if John has a dog ?
do Ping and Ping and Mary know if Eve knows if Ping has a dog ?

Logic

We need a multi-modal logic (S5n).For example, *John knows that Mary loves Eve* would be represented as

```
⟦john⟧(love mary eve)
```

In [6]:
```
theory proplog : ur:?LF =
    proposition : type | # o ❙
    true : o ❙
    neg : o → o | # ¬ 1 prec 80 ❙
    and : o → o → o | # 1 ∧ 2 prec 60 ❙
    or : o → o → o | # 1 ∨ 2 prec 50 ❙
    imp : o → o → o | # 1 ⇒ 2 prec 40 ❙
❚
```

Successfully imported proplog.mmt

In [7]:
```
theory epistemic : ur:?LF =
    include ?proplog ❙
    agent : type | # ι ❙
    box : ι → o → o | # ⟦ 1 ⟧ 2 prec 10 ❙
    dia : ι → o → o | # ⟪ 1 ⟫ 2 prec 10 ❙
❚
```

Successfully imported epistemic.mmt

In [8]:
```
theory questions : ur:?LF =
    include ?proplog ❙
    question : type ❙
    ask : o → o → question | # 1 ?⊢? 2 ❙     // ask premise
❚
```

Successfully imported questions.mmt

In [9]:
```
theory epistemicDDT : ?epistemic =
    include ?questions ❙
    john : ι ❙
    mary : ι ❙
```

```
    eve : ι ❙
    ping : ι ❙
    havedog : ι → o ❙
    havecat : ι → o ❙
    love : ι → ι → o ❙
    hate : ι → ι → o ❙
❚
```

Successfully imported epistemicDDT.mmt

In [10]:
```
view EpistemicSemantics : http://mathhub.info/tmpGLIF/exampl
    S = o ❙
    SSeq = o ❙
    QSeq = question ❙
    Cl = o ❙
    Agent = ι ❙
    NP = (ι → o) → o ❙
    VP = ι → o ❙
    Polarity = o → o ❙

    positive = [a] a ❙
    negative = [a] ¬ a ❙
    makeS = [pol,cl] pol cl ❙
    simpleCl = [np,vp] np vp ❙
    knowCl = [np,s] np ([x] ⟦ x ⟧ s) ❙
    knowWhetherCl = [np,cl] (np ([x] ⟦ x ⟧ cl)) ∨ (np ([x] ⟦
    makeQSeq = [sseq,cl] ask sseq cl ❙
    makeSSeq = true ❙
    append = [sseq,s] sseq ∧ s ❙
    npify = [x] [p] p x ❙
    and = [a,b] [p] (a p) ∧ (b p) ❙    // and : NP → NP → NF
    or = [a,b] [p] (a p) ∨ (b p) ❙
    john = john ❙
    mary = mary ❙
    eve = eve ❙
    ping = ping ❙
    have_cat = havecat ❙
    have_dog = havedog ❙
    love = [np] [x] np (love x) ❙
    hate = [np] [x] np (hate x) ❙
❚
```

Successfully imported EpistemicSemantics.mmt

In [11]:
```
-- Let's try it out
p "John knows that Mary has a dog" | construct
```

⟦john⟧(havedog mary)

In [12]:
```
p "John knows that Mary doesn't know that Eve has a dog" | c
```

⟦john⟧¬⟦mary⟧(havedog eve)

In [13]:
```
p "John knows if Mary has a dog" | construct
```

(⟦john⟧(havedog mary)) ∨ ⟦john⟧¬(havedog mary)

In [14]:
```
-- Let's also try out a sequence of statements followed by a
p -cat=QSeq "John has a dog . Mary has a dog . does Eve have
```

(true ∧ (havedog john)) ∧ (havedog mary)? ⊢ ?(havedog eve)

Inference

For the inference, we need an S5n prover, which was adapted from: *de Boer, M. S. (2006). Praktische Bewijzen in Public Announcement Logica*.

If we have premises (statements) P1, ..., Pn and a question Q, then we want to reply

- **yes** if P1 ∧ ... ∧ Pn ⊢ Q (and not P1 ∧ ... ∧ Pn ⊢ ⊥)
- **no** if P1 ∧ ... ∧ Pn ⊢ ¬Q
- **That doesn't make sense** if P1 ∧ ... ∧ Pn ⊢ ⊥ (i.e. the premises are inconsistent)
- **Maybe** otherwise

In [15]:
```
elpi-notc: util

% the original prover was implemented in normal Prolog, so w

select A [A] [] :- !.
select A [A|L] L.
select A [X|T] [X|R] :- select A T R.

freeze X f :- var X, !, declare_constraint (f X) [X].
freeze X f :- f X.

member X [X|_] :- !.
member X [_|L] :- member X L.
```

Successfully imported util.elpi
util.elpi is the new default file for ELPI commands

In [16]:
```
elpi-notc: s5npal

accumulate util.

% Adapted from: de Boer, M. S. (2006). Praktische Bewijzen ir

p F :- prove [l [] (neg F)] [], !.
```

```
prove [LFml|Fs] Branch :-
    (ruleOne LFml _tpe NLF, !, prove [NLF|Fs] Branch);
    (ruleTwo LFml conj A B, !, prove [A,B|Fs] Branch).
prove [l L Fml|Fs] Branch :-
    (Fml = neg Neg; Neg = neg Fml),
    (memberunify (l L Neg) Branch; prove Fs [l L Fml | Branc
prove [] Branch :-
    select (l Label Fml) Branch RestB,
    ruleTwo (l _ Fml) disj (l _ A) (l _ B),
    expand Label Inst M _,
    (Label = l _ L; Label = L), !,
    prove [l M A] [l (l Inst L) Fml|RestB],
    prove [l M B] [l (l Inst L) Fml|RestB].

expand (l [[X|Xs]|Xss] [star I|Ls]) [[X|Xs]|Yss] [l X I|Ms]
    var X, !,
    expand (l Xss Ls) Yss Ms G.
expand (l [Xs|Xss] [star I|Ls]) [[X|Xs]|Yss] [l X I|Ms] _ :-
    freeze X (x \ not (member X Xs)),
    expand (l Xss Ls) Yss Ms true.
expand (l Xss [C|Ls]) Yss [C|Ms] G :- !,
    expand (l Xss Ls) Yss Ms G.
expand (l [] []) [] [] G :-
    not (var G).
expand [star I|Ls] [[X]|Xss] [l X I|Ms] _ :- !,
    expand Ls Xss Ms _.
expand [L|Ls] Xss [L|Ms] _ :- !,
    expand Ls Xss Ms _.
expand [] [] [] _.

memberunify (l L A) Ls :-
    member (l K A) Ls,
    red 0 L [],
    red 0 K [], !.

red O [l _ I|L] [l X I|R] :- !,
    red O L [l X I|R].
red O L [l O _|R] :-
    red O L R.
red O L [star I|R] :- !,
    red O L [l _ I|R].
red O [X|L] R :- !,
    red O L [X|R].
red O [] O.

ruleTwo (l L (and A B)) conj (l L A) (l L B).
ruleTwo (l L (neg (imp A B))) conj (l L A) (l L (neg B)).
ruleTwo (l L (neg (or A B))) conj (l L (neg A)) (l L (neg B)
ruleTwo (l L (or A B)) disj (l L A) (l L B).
ruleTwo (l L (imp A B)) disj (l L (neg A)) (l L B).
ruleTwo (l L (neg (and A B))) disj (l L (neg A)) (l L (neg E
ruleTwo (l L (eq A B)) disj (l L (and A B)) (l L (and (neg A
ruleTwo (l L (neg (eq A B))) disj (l L (and A (neg B))) (l L
```

```
ruleOne (l L (neg (bra F))) doub (l L (neg F)).
ruleOne (l L (bra F)) doub (l L F).
ruleOne (l L (neg (neg F))) doub (l L F).

ruleOne (l [l _ I|L] (box I F)) know (l [star I|L] F).
ruleOne (l [l _ I|L] (neg (dia I F))) know (l [star I|L] (ne
ruleOne (l [star I|L] (box I F)) know (l [star I|L] F).
ruleOne (l [star I|L] (neg (dia I F))) know (l [star I|L] (r
ruleOne (l L (box I F)) know (l [star I|L] F).
ruleOne (l L (neg (dia I F))) know (l [star I|L] (neg F)).

ruleOne (l [l _ I|L] (dia I F)) poss (l [l F I|L] F).
ruleOne (l [l _ I|L] (neg (bos I F))) poss (l [l F I|L] (neg
ruleOne (l [star I|L] (dia I F)) poss (l [l F I|L] F).
ruleOne (l [star I|L] (neg (bos I F))) poss (l [l (neg F) I|
ruleOne (l L (dia I F)) poss (l [l F I|L] F).
ruleOne (l L (neg (box I F))) poss (l [l (neg F) I|L] (neg F
```

Successfully imported s5npal.elpi

s5npal.elpi is the new default file for ELPI commands

In [17]:

```
-- generate ELPI signatures for discourse domain theory and
elpigen -mode=types epistemicDDT
```

Successfully created epistemicDDT.elpi

In [18]:

```
elpi-notc: answerer

accumulate s5npal.
accumulate epistemicDDT.

answer (ask Premises _) :-
    p (imp Premises falsum), !, print "That doesn't make ser
answer (ask Premises Conclusion) :-
    p (imp Premises Conclusion), !, print "Yes!".
answer (ask Premises Conclusion) :-
    p (imp Premises (neg Conclusion)), !, print "No!".
answer (ask _ _) :- !, print "Maybe...".
answer X :- print "Bad term:" X.

% we will use the apply command
apply Item :-
    glif.getStr Item S, print S,
    glif.getLog Item Content,    % extract the logical repre
    answer Content.
```

Successfully imported answerer.elpi

answerer.elpi is the new default file for ELPI commands

Examples

In [19]:
```
p -cat=QSeq "John knows that Ping has a dog . does Ping have
```

John knows that Ping has a dog . does Ping have a dog ?
Yes!

In [20]:
```
p -cat=QSeq "John has a dog . does John know that John has a
```

John has a dog . does John know that John has a dog ?
Maybe...

In [21]:
```
-- The example from the introduction
p -cat=QSeq "John knows that Mary or Eve knows that Ping has
p -cat=QSeq "John knows that Mary or Eve knows that Ping has
```

(true ∧ ⟦john⟧(⟦mary⟧(havedog ping)) ∨ ⟦eve⟧(havedog ping)) ∧ ¬((⟦mary⟧(havedog ping)) ∨ ⟦mary⟧¬(havedog ping))? ⊢ ?(⟦eve⟧(havedog ping)) ∨ ⟦eve⟧¬(havedog ping)

John knows that Mary or Eve knows that Ping has a dog . Mary doesn't know if Ping has a dog . does Eve know if Ping has a dog ?
Yes!

In [22]:
```
p -cat=QSeq "John doesn't love Mary . does John love Mary ?'
p -cat=QSeq "John loves Mary . does John love Mary ?" | cons
p -cat=QSeq "John loves Mary . John doesn't love Mary . does
p -cat=QSeq "John loves Eve . does John love Mary ?" | const
```

John doesn't love Mary . does John love Mary ?
No!

John loves Mary . does John love Mary ?
Yes!

John loves Mary . John doesn't love Mary . does John love Mary ?
That doesn't make sense!

John loves Eve . does John love Mary ?
Maybe...

And many more examples...

In [23]:
```
p -cat=QSeq "John doesn't love Mary . does John love Mary ?'
```

John doesn't love Mary . does John love Mary ?
No!

In [24]:
```
p -cat=QSeq "John knows that Mary has a dog . does Mary have
```

John knows that Mary has a dog . does Mary have a dog ?
Yes!

In [25]:
```
p -cat=QSeq "Mary has a dog . does John know that Mary has a
```

Mary has a dog . does John know that Mary has a dog ?
Maybe...

In [26]:
```
p -cat=QSeq "John knows that Mary doesn't have a dog . does
```

John knows that Mary doesn't have a dog . does John know if Mary has a dog ?
Yes!

In [27]:
```
p -cat=QSeq "Mary doesn't have a dog . does John know if Mar
```

Mary doesn't have a dog . does John know if Mary has a dog ?
Maybe...

In [28]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . Mary do
```

John knows that Mary or Eve has a dog . Mary doesn't have a dog . does Eve have a dog ?
Yes!

In [29]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . Mary do
```

John knows that Mary or Eve has a dog . Mary doesn't have a dog . does John know that Eve has a dog ?
Maybe...

In [30]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . John kn
```

John knows that Mary or Eve has a dog . John knows if Eve has a dog . Mary doesn't have a dog . does John know that Eve has a dog ?
Yes!

In [31]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . Mary kn
```

John knows that Mary or Eve has a dog . Mary knows that Mary doesn't have a dog . does Eve have a dog ?
Yes!

In [32]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . John kn
```

John knows that Mary or Eve has a dog . John knows that Mary knows that Mary doesn't have a dog . does John know that Eve has a dog ?

Maybe

In [33]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . John kn
```

John knows that Mary or Eve has a dog . John knows that Mary doesn't have a dog . does John know that Eve has a dog ?
Maybe...

In [34]:
```
p -cat=QSeq "John knows that Mary or Eve has a dog . John kn
```

John knows that Mary or Eve has a dog . John knows that Mary knows that Mary doesn't have a dog . does Eve have a dog ?
Yes!

In [35]:
```
p -cat=QSeq "John knows that Mary or Eve knows that Eve has
```

John knows that Mary or Eve knows that Eve has a dog . does John know that Eve has a dog ?
Yes!

In [36]:
```
p -cat=QSeq "John knows that Mary or Eve knows that Ping has
```

John knows that Mary or Eve knows that Ping has a dog . Mary doesn't know if Ping has a dog . does Eve know if Ping has a dog ?
Yes!