Discourse

BSc Artificial Intelligence, Spring 2011

Raquel Fernández

Institute for Logic, Language & Computation
University of Amsterdam

Plan for Today

- Discussion of HW#2 and exercise 2 from Practical Session#1
- The Curt System: putting it all together
- Next steps

HW#2: Exercise 1

Required new clauses for "Vincent offers Mia a drink":

lexical entry for noun "drink":

```
noun(lam(X,drink(X)))--> [drink].
```

lexical entry for ditransitive verbs:

```
dtv(lam(Y,lam(X,lam(Z,app(X,lam(X1,app(Y,lam(Y1,offer(Z,X1,Y1)))))))))
                                                             --> [offers].
```

compare to lexical entry for transitive verb:

```
tv(lam(X,lam(Y,app(X,lam(Z,like(Y,Z))))))--> [likes].
```

syntax-semantics rules:

```
vp(app(app(DTV,Y),X))-> dtv(DTV), np(X), np(Y).
```

another possibility using a binary tree:

```
vp(app(VB,Y))--> vbar(VB), np(Y).
vbar(app(DTV,X))--> dtv(DTV), np(X).
dtv(lam(X,lam(Y,lam(Z,app(X,lam(X1,app(Y,lam(Y1,offer(Z,X1,Y1)))))))))
                                                             --> [offers].
```

Output of semantic construction: $\exists x.(\textsc{drink}(x) \wedge \textsc{offer}(\textsc{vincent}, \textsc{mia}, x))$

```
?- s(Sem,[vincent,offers,mia,a,drink],[]),betaConvert(Sem,Reduced).
Reduced = some(X, and(drink(X), offer(vincent, mia, X))).
```

HW#2: Exercise 2

Required new clauses for “Somebody snores” and “Everyone dances”:

lexical entry for intransitive verb “dance”:

```
iv(lam(Y,dance(Y)))-> [dances].
```

lexical entries for pronouns:

```
pr(lam(Q,all(X,imp(person(X),app(Q,X)))))-> [everyone].
pr(lam(Q,some(X,and(person(X),app(Q,X)))))-> [somebody].
```

compare to the lexical entries for the determiners:

```
det(lam(P,lam(Q,all(X,imp(app(P,X),app(Q,X))))))-> [every].
det(lam(P,lam(Q,some(X,and(app(P,X),app(Q,X))))))-> [a].
```

syntax-semantics rules:

```
np(PR)-> pr(PR).
```

Output of semantic construction:

```
?- s(Sem,[somebody,snorts],[]),betaConvert(Sem,Reduced).
Reduced = some(X, and(person(X), snort(X))) .

?- s(Sem,[everyone,dances],[]),betaConvert(Sem,Reduced).
Reduced = all(X, imp(person(X), dance(X)))
```

HW#2: Exercise 4

All boxers are slow. Butch is a boxer. Butch is not slow.

$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \wedge \neg\textsc{slow}(\textsc{butch})$$

The discourse above is inconsistent if its negation is valid:

$$\neg[\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \wedge \neg\textsc{slow}(\textsc{butch})]$$

or equivalently

$$\neg[\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \rightarrow \neg\textsc{slow}(\textsc{butch})]$$
$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \rightarrow \neg\neg\textsc{slow}(\textsc{butch})$$

To prove validity by *refutation*, we need to show that the negation of a supposedly valid formula leads to a contradiction:

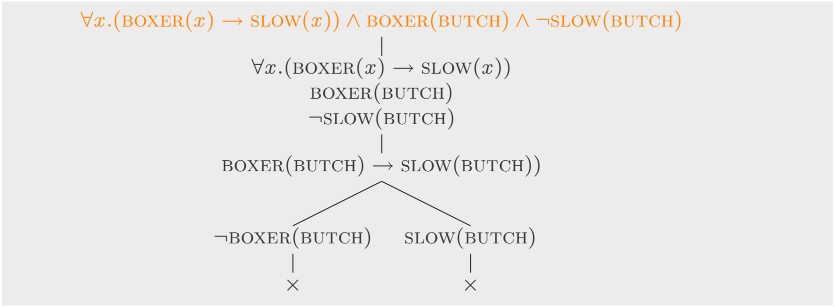
$$\neg[\neg[\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \wedge \neg\textsc{slow}(\textsc{butch})]]$$
$$\approx$$
$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \wedge \neg\textsc{slow}(\textsc{butch})$$

or equivalently

$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{slow}(x)) \wedge \textsc{boxer}(\textsc{butch}) \rightarrow \neg\textsc{slow}(\textsc{butch})$$

Any of the above formulas can be used at the root of a tableau tree.

HW#2: Exercise 4


∀x.(BOXER(x) → SLOW(x)) ∧ BOXER(BUTCH) ∧ ¬SLOW(BUTCH)
∀x.(BOXER(x) → SLOW(x))
BOXER(BUTCH)
¬SLOW(BUTCH)
BOXER(BUTCH) → SLOW(BUTCH))
¬BOXER(BUTCH)
SLOW(BUTCH)
×
×

- we apply the first alpha rule twice to deconstruct the two conjunctions;
- we then apply the first gamma rule to the universally quantified formula, using BUTCH as constant;
- finally, we apply the second beta rule to the implication;
- we end up with non-expandable formulas and contradictory information in all branches.

Exercise 2 from Practicum#1

Implementation of `tpmbTestSuite/0` in `callInference.pl`:

```
tpmbTestSuite:-
   format('~n~n>>>>> INFERENCE TEST SUITE <<<<<',[]),
   formula(Formula,Status),
   format('~nInput formula: ~p~nStatus: ~p',[Formula,Status]),
   callTPandMB(Formula,Formula,30,Proof,Model,Engine),
   (  Proof=proof,
      Result=theorem  ;
      Proof=unknown,
      Model=model(_,_),
      Result=Model    ;
      Proof=unknown,
      Model=unknown,
      Result=unknown   ),
   format('~nInference engine ~p says: ~p~n',[Engine,Result]),
   fail.
```

Note that TP and MB are given the same formula:

- TP tries to prove φ (by falsifying $\neg\varphi$); MB tries to build a model for φ

This settings is not useful for all purposes.

Exercise 2 from Practicum#1

- Current setting in `tpmbTestSuite/0` for each formula φ in `folTestSuite.pl`:
  - ∗ TP tries to prove φ
  - ∗ MB tries to build a model for φ
- Optimal setting to check for satisfiability of φ:
  - ∗ positive: MB tries to find a model for φ
  - ∗ negative: TP tries to prove validity of $\neg\varphi$
    (see the implementation of Clever Curt)
- Optimal setting to check for validity of φ:
  - ∗ positive: TP tries to prove φ
  - ∗ negative: MB tries to find a model for $\neg\varphi$
    (see the implementation of Sensitive Curt)
- With the current setting, neither TP nor MB can deal with the unsatisfiable formula. If φ is unsatisfiable, $\neg\varphi$ is valid.
  - ∗ MB can't find a model for φ
  - ∗ TP can't falsify $\neg\varphi$

The Curt System: Putting It All Together

The following slides assume you have read section 6.1 to 6.4 of chapter 6 from Blackburn & Bos (2005).

The Curt System

Curt: Clever Use of Reasoning Tools

A system that can handle some simple but interesting interactions with a user by making use of all the elements we have seen so far:

- semantic construction (grammar with lambda calculus)
- consistency checking,
- informativity checking,
- model checking (querying task).

Semantic Construction in Curt (1)

Curt builds semantic representations for natural language input using the extended grammar architecture by B&B (see the slides on semantic construction).

- In particular, it uses the code in `kellerStorage.pl`, which incorporates the capability to handle *quantifier scope ambiguity* into the semantic component of the grammar.
  - ∗ we have not treated this – you may have covered it in other courses.
- Curt can be used with `lambda.pl` instead of `kellerStorage.pl`

⇒ **Comment out all clauses involving** `kellerStorage` **and include the corresponding** `lambda` **clauses in all files of the Curt family. E.g.:**

```
% :- use_module(kellerStorage,[kellerStorage/2]).
:- use_module(lambda,[lambda/2]).
```

Semantic Construction in Curt (2)

Curt is able to combine the semantic representations of the input sentences into a discourse representation.

```
combine(New,New):-
   readings([]).

combine(Readings,Updated):-
   readings([Old|_]),
   findall(and(Old,New),memberList(New,Readings),Updated).
```

- Semantic representations are combined into a discourse representation using conjunction: `and(Old,New)`
- Note that if we use `lambda.pl` instead of `kellerStorage.pl` we deal with only 1 reading (1 semantic representation), so the predicate `combine/2` could be simpler. . .

Sample Interaction

We can examine the discourse history and the semantic representation of the discourse:

```
> Vincent likes Mia.
Curt: OK.

> readings
1 like(vincent, mia)

> Vincent is not a boxer.
Curt: OK.

> history
1 [vincent, likes, mia]
2 [vincent, is, not, a, boxer]

> readings
1 and(like(vincent, mia), not(some(A, and(boxer(A), eq(vincent, A)))))
```

Dialogue Control in Curt

The dialogue control structure of Curt integrates the user input, decides how the system should reply, and sets the program's executing state.

```
curtTalk(quit).

curtTalk(run):-
   readLine(Input),
   curtUpdate(Input,CurtsMoves,State),
   curtOutput(CurtsMoves),
   curtTalk(State).
```

The key predicates are:

- `curtUpdate(Input,ReplyMoves,State)`
- `curtOutput(ReplyMoves)`

Consistency Checking in Curt

curtUpdate/3 filters out inconsistent interpretations with consistentReadings/3, which uses consistent/3 to call a theorem prover and a model builder with callTPandMB/6:

```
consistent([Old|_],New,Model):-
  DomainSize=15,
  callTPandMB(not(and(Old,New)),and(Old,New),DomainSize,Proof,Model,Engine),
  format('~nMessage (consistency checking): ~p found a result.',[Engine]),
  \+ Proof=proof, Model=model([_|_],_).
```

If an incoming sentence is consistent with the preceding discourse, MB can find a model for and(Old,New); if it is inconsistent, TP can prove that not(and(Old,New)) is valid.

Curt keeps track of the model that is being built by the discourse and allows us to inspect it.

Sample Interaction

```
> Every boxer likes Mia.
Message (consistency checking): mace4 found a result.
Curt: OK.

> Butch is a boxer.
Message (consistency checking): mace4 found a result.
Curt: OK.

>readings
1 and(all(A,imp(boxer(A), like(A,mia))), some(B,and(boxer(B), eq(butch,B))))

> models
1 D=[d1, d2]
  f(0, butch, d1)
  f(0, mia, d1)
  f(0, c1, d1)
  f(1, boxer, [d1])
  f(2, like, [ (d1, d1)])

> Butch does not like Mia.
Message (consistency checking): prover9 found a result.
Curt: No! I do not believe that!
```

Informativity Checking in Curt

curtUpdate/3 filters out uninformative interpretations with informativeReadings/2, which uses informative/2 to again call callTPandMB/6:

```
informative([Old|_],New):-
 DSize=15,
 callTPandMB(not(and(Old,not(New))),and(Old,not(New)),DSize,Proof,Model,Engine)
 format('~nMessage (informativity checking): ~p found a result.',[Engine]),
 \+ Proof=proof, Model=model([_|_],_).
```

If an incoming sentence is informative with respect to the preceding discourse, MB can find a model for and(Old,not(New)); if it is inconsistent, TP can prove that not(and(Old,not(New))) is valid.

Sample Interaction

```
> Every customer hates Vincent.
Message (consistency checking): mace4 found a result.
Message (informativity checking): mace4 found a result.
Curt: OK.

> Vincent is not a customer.
Message (consistency checking): mace4 found a result.
Message (informativity checking): mace4 found a result.
Curt: OK.

> Jimmy is a customer.
Message (consistency checking): mace4 found a result.
Message (informativity checking): mace4 found a result.
Curt: OK.

> readings
1 and(and(all(A,imp(customer(A), hate(A,vincent))),
        not(some(B,and(customer(B), eq(vincent,B))))),
          some(C,and(customer(C), eq(jimmy,C))))

> models
1 D=[d1, d2]
  f(0, jimmy, d1)
  f(0, vincent, d2)
  f(0, c1, d1)
  f(1, customer, [d1])
  f(2, hate, [ (d1, d2)])

>Jimmy hates Vincent.
Message (consistency checking): mace4 found a result.
Message (informativity checking): prover9 found a result.
Curt: Well, that is obvious!
```

Knowledgeable & Helpful Curt

You are encouraged to read what is left of Chapter 6 and find out how Curt can be complemented with lexical and background knowledge, and how it can be expanded to handle questions from the user.

We may come back to this later on, but for now we will not cover it explicitly in class. Instead, we'll go deeper into discourse-related issues.

What's Next?

So far, we have used simple conjunction to combine the semantic representations of the sentences in a discourse:

Every customer hates Vincent. Vincent is not a customer.
$\forall x.(\textsc{customer}(x) \rightarrow \textsc{hate}(x, \textsc{vincent})) \wedge \neg\textsc{customer}(\textsc{vincent})$

The discourse above is a bit awkward, it would be more natural to use a pronoun. But if we represent pronouns as variables, a simple conjunctive strategy does not give the right result:

Every customer hates Vincent. He is not a customer.
$\forall x.(\textsc{customer}(x) \rightarrow \textsc{hate}(x, \textsc{vincent})) \wedge \neg\textsc{customer}(y)$

In order to deal with pronouns, we'll introduced a different formalism: Discourse Representation Theory

Next Steps

- Introduction to DRT
- Pronoun resolution (logic and non-logic based approaches)
- Presuppositions

To start with, read chapter 1 from BB2 (see HW#3)

- recall this is a draft book: read it intelligently and be tolerant!

Period 5: change of day, time, and room

Next lecture on Tuesday 29 March, 15–17:00h, room G0.05

HW#3 due on 25 March