Discourse

BSc Artificial Intelligence, Spring 2011

Raquel Fernández

Institute for Logic, Language & Computation
University of Amsterdam

Plan for Today

- Introduction to Discourse Representation Theory (DRT)
  - Syntax and semantics of DRT and its connection to First-Order Logic (B&B2, ch. 1)
  - Semantic construction from natural language input in DRT (B&B2, ch. 2)

Why DRT?

The system we have been working with (B&B's Curt system) is able to process and interpret sequences of sentences:

- it can assess whether a sentence is consistent or informative with respect to the previous discourse (which is great!),
- but it uses a simple conjunctive strategy to build up the semantic representation of the ongoing discourse

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 need a different strategy...

Context Change Potential

First-order logic representations focus on the truth-conditional dimension of meaning, but do not handle well the context change potential of natural language expressions:

- sentences change the context in which subsequent sentences will be interpreted

Every customer hates Vincent. He is not a customer.

$\forall x.(\textsc{customer}(x) \rightarrow \textsc{hate}(x, \textsc{vincent})) \wedge \neg\textsc{customer}(y) \wedge y = \textsc{vincent}$

To capture truth conditions *plus* the context change potential of meaning, we need representations that are more sophisticated than first-order logic formulas:

- Discourse Representation Theory (DRT) offers the right semantic architecture.

Discourse Representation Theory

- DRT was developed by Hans Kamp in the 1980s, and it has since then become a very influential semantic theory.
- Other versions of *dynamic* semantics: Heim's File Change Semantics, and Groenendijk & Stokhof's Dynamic Predicate Logic (Amsterdam school).
- The classic reference book for DRT is:
  Kamp & Reyle (1993) *From Discourse to Logic*.

Basic idea behind DRT: a hearer (i.e. a natural language processing agent) builds up an internal representation of the discourse as it unfolds in time, with every incoming sentence prompting additions to that representation.

- common idea in psycholinguistics, but new to formal semantics

Discourse Representation Structures

According to DRT, hearers processing language build up a discourse representation structure (DRS).

DRSs are conventionally represented as *boxes*. They distinguish two types of information:

- which discourse entities we have at our disposal (top part of the DRS box, containing *discourse referents*)
- which properties these entities have and how they are interrelated (bottom part of the box - containing *conditions*)

Vincent offers Mia a drink. $\begin{array}{|l|}\hline x\ y\ z \\ \hline x=\text{VINCENT} \\ y=\text{MIA} \\ \text{DRINK}(z) \\ \text{OFFER}(x,y,z) \\ \hline \end{array}$ It is blue. $\begin{array}{|l|}\hline x\ y\ z\ v \\ \hline x=\text{VINCENT} \\ y=\text{MIA} \\ \text{DRINK}(z) \\ \text{OFFER}(x,y,z) \\ \text{BLUE}(v) \\ v=z \\ \hline \end{array}$

What we need to know about DRSs

The DRT framework will allow us to handle anaphoric pronouns (as well as presuppositions). But before we get to that, we need to to know the following:

- how are DRS languages defined formally
- how are DRS interpreted
- how are DRSs constructed from natural language input

As we shall see, DRT is closely related to first-order logic:

- DRS languages and first-order languages are constructed from the same vocabularies
- DRSs and first-order formulas are interpreted in the same models
- we can use the lambda calculus to build up DRSs from natural language input as we did with first-order representations.

DRS Languages (1)

Given a vocabulary containing relation symbols, constants, and variables (which we shall call *discourse referents*), we can build DRSs and conditions as follows:

- DRSs:
  If $\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}$ are discourse referents and $\mathrm{c}_1, \ldots, \mathrm{c}_\mathrm{m}$ are conditions, then $\begin{array}{|l|}\hline \mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n} \\ \hline \mathrm{c}_1 \\ \vdots \\ \mathrm{c}_\mathrm{m} \\ \hline \end{array}$ is a DRS.
- Terms: A term τ is either a constant or a discourse referent.
- Primitive conditions:
  - $*$ If R is a relation symbol and $\tau_1, \ldots, \tau_n$ are terms, then $R(\tau_1, \ldots, \tau_n)$ is a condition.
  - $*$ If τ_1 and τ_2 are terms, then $\tau_1 = \tau_2$ is a condition.

DRS Languages (2)

- Complex conditions:
  - If K is a DRS, then $\neg K$ is a condition.
  - If K_1 and K_2 are DRSs, then $K_1 \vee K_2$ is a condition.
  - If K_1 and K_2 are DRSs, then $K_1 \Rightarrow K_2$ is a condition.
- Nothing else is a DRS or a condition.

Note that to build up complex conditions, the only logical symbols used are $\neg$, $\vee$, and $\Rightarrow$.

- $\Rightarrow$ handles both conditionals and universal quantification.
- $\wedge$ is handled in two ways:
  - conditions within a DRS are implicitly conjoined;
  - we can conjoin two DRSs with a *merge* operation (as we will see).

Interpreting DRSs (1)

DRSs are interpreted in exactly the sames models as first-order formulas: $M = (D, F)$

B&B discuss two types of semantics for DRSs that are closely linked: *embedding semantics* and *dynamic semantics*. It suffices to understand the basic idea behind embedding semantics:

- essentially, a DRSs is satisfied in a model $M = (D, F)$ if we can find a function from discourse referents to elements in D (an embedding i) such that all conditions are satisfied.

$D = \{d_1, d_2, d_3, d_4, d_5\}$
$F(\text{MIA}) = d_2$
$F(\text{HONEY-BUNNY}) = d_1$
$F(\text{VINCENT}) = d_4$
$F(\text{YOLANDA}) = d_1$
$F(\text{CUSTOMER}) = \{d_1, d_2, d_4\}$
$F(\text{ROBBER}) = \{d_3, d_5\}$
$F(\text{LOVE}) = \{(d_3, d_4)\}$

$$\begin{array}{|l|}\hline \text{x y} \\ \hline \text{x} = \text{VINCENT} \\ \text{LOVE(y,x)} \\ \text{CUSTOMER(y)} \\ \hline \end{array}$$

When are conditions satisfied?

M is a model and i is an embedding in M.

- Basic conditions:

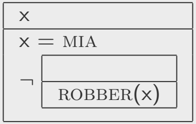
$M, i \models R(\tau_1, \ldots, \tau_n)$	if there is an i such that the elements in D corresponding to $\tau_1, \ldots, \tau_n$ are part of $F(R)$.
$M, i \models \tau_1 = \tau_2$	if there is an i such that the elements in D corresponding to τ_1, τ_2 are are the same.

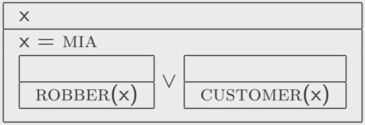
- Complex conditions:

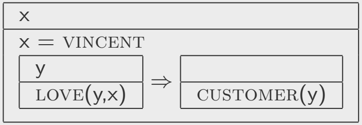
$M, i \models \neg K$	if there is no i that satisfies K.
$M, i \models K_1 \vee K_2$	if there is an i such that K_1 is satisfied or K_2 is satisfied.
$M, i \models K_1 \Rightarrow K_2$	if for any i that satisfies K_1, there is an extended function that satisfies K_2.

Examples

$D = \{d_1, d_2, d_3, d_4, d_5\}$
$F(\textsc{mia}) = d_2$
$F(\textsc{honey-bunny}) = d_1$
$F(\textsc{vincent}) = d_4$
$F(\textsc{yolanda}) = d_1$
$F(\textsc{customer}) = \{d_1, d_2, d_4\}$
$F(\textsc{robber}) = \{d_3, d_5\}$
$F(\textsc{love}) = \{(d_3, d_4)\}$







Translating DRT into FOL

DRT and FOL are closely related indeed! We can define a translation function fo that translates DRSs into first-order formulas.

Translation of DRSs: $$\left(\begin{array}{|l|}\hline \mathrm{x}_1,\dots,\mathrm{x}_n \\ \hline \mathrm{c}_1 \\ \vdots \\ \mathrm{c}_m \\ \hline \end{array}\right)^{fo} = \exists \mathrm{x}_1,\dots,\exists \mathrm{x}_n[(\mathrm{c}_1)^{fo} \land \dots \land (\mathrm{c}_m)^{fo}]$$

Basic conditions: $$(R(\mathrm{x}_1,\dots,\mathrm{x}_n))^{fo} = R(\mathrm{x}_1,\dots,\mathrm{x}_n)$$ $$(\tau_1 = \tau_n)^{fo} = \tau_1 = \tau_n$$

Complex conditions: $$(\neg K)^{fo} = \neg(K)^{fo}$$ $$(K_1 \lor K_2)^{fo} = (K_1)^{fo} \lor (K_2)^{fo}$$

$$\left(\begin{array}{|l|}\hline \mathrm{x}_1,\dots,\mathrm{x}_n \\ \hline \mathrm{c}_1 \\ \vdots \\ \mathrm{c}_m \\ \hline \end{array} \Rightarrow K\right)^{fo} = \forall \mathrm{x}_1,\dots,\forall \mathrm{x}_n[(\mathrm{c}_1)^{fo} \land \dots \land (\mathrm{c}_m)^{fo} \to (K)^{fo}]$$

Quick Exercise

- Translate the DRS examples we saw before into first-order formulas
- Draw the DRSs representing the following sentences and translate them into first-order formulas:
  - ∗ Mia dances.
  - ∗ Vincent does not have a car.
  - ∗ Every robber has a gun.
  - ∗ Yolanda hates all customers.
  - ∗ Mia has a Ferrari or a Fiat.

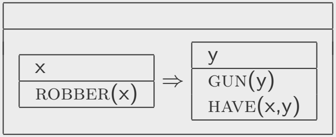
Implementing DRT in Prolog

Before moving into DRS construction, let us have a look at how we can implement the basics of DRT in Prolog.

- We shall represent DRSs as Prolog terms of the form `drs(D,C)`, where:
  - `D` is a list of terms representing the discourse referents, and
  - `C` is a list of terms representing DRS conditions.
- Discourse referents will be represented as Prolog variables.
- For complex conditions, we use the same operators as for FOL.

For instance:

Every robber has a gun.



```
drs([], imp(drs([X],[robber(X)]), drs([Y],[gun(Y),have(X,Y)])))
```

Prolog Implementation of fo

- The program `drs2fol` implements the translation function fo.
- By implementing a program that translates DRSs into first-order formulas, we are able to make use of all the tools we have available for FOL: the model checker, the theorem prover, and the model builder.
- Let's first look into how to translate DRSs.

  [Note that the material in this slides follows the latest version of the Prolog code — this may be a bit different from the draft book.]

Translating DRSs

This is the definition of *fo*:

$$\left(\begin{array}{|l|}\hline \mathrm{x}_1, \ldots, \mathrm{x}_n \\ \hline \mathrm{c}_1 \\ \vdots \\ \mathrm{c}_m \\ \hline \end{array}\right)^{fo} = \exists \mathrm{x}_1, \ldots, \exists \mathrm{x}_n[(\mathrm{c}_1)^{fo} \wedge \ldots \wedge (\mathrm{c}_m)^{fo}]$$

And here is the Prolog code that does the same thing:

```
drs2fol(drs([],[Cond]),Formula):-
   cond2fol(Cond,Formula).

drs2fol(drs([],[Cond1,Cond2|Conds]),and(Formula1,Formula2)):-
   cond2fol(Cond1,Formula1),
  drs2fol(drs([],[Cond2|Conds]),Formula2).

drs2fol(drs([X|Referents],Conds),some(X,Formula)):-
  drs2fol(drs(Referents,Conds),Formula).
```

- the third clause adds an existential quantifier for each discourse referent.
- the second clause conjoins the first-order translations of the conditions.
- the second and first clauses call the predicate `cond2fol` which specifies how to translate conditions.

Translating Conditions

The translation of basic conditions is trivial:

$(\tau_1 = \tau_n)^{fo} = \tau_1 = \tau_n$

```
cond2fol(eq(X,Y),eq(X,Y)).
```

$(R(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}))^{fo} = R(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n})$

```
cond2fol(pred(Sym,X),pred(Sym,X)).
cond2fol(rel(Sym,X,Y),rel(Sym,X,Y)).
```

As are the clauses for the disjunctive and negative complex conditions:

$(\neg K)^{fo} = \neg(K)^{fo}$

```
cond2fol(not(Drs),not(Formula)):-
   drs2fol(Drs,Formula).
```

$(K_1 \vee K_2)^{fo} = (K_1)^{fo} \vee (K_2)^{fo}$

```
cond2fol(or(Drs1,Drs2),or(Formula1,Formula2)):-
   drs2fol(Drs1,Formula1),
   drs2fol(Drs2,Formula2).
```

Translating Implicative Conditions

This is how implicative conditions are translated according to fo

$$\left(\begin{array}{|l|}\hline x_1,\ldots,x_n \\ \hline c_1 \\ \vdots \\ c_m \\ \hline\end{array} \Rightarrow K\right)^{fo} = \forall x_1,\ldots,\forall x_n[(c_1)^{fo} \wedge \ldots \wedge (c_m)^{fo} \rightarrow (K)^{fo}]$$

And here is the Prolog code that does the same thing:

```
   cond2fol(imp(drs([],Conds),Drs2),imp(Formula1,Formula2)):-
      drs2fol(drs([],Conds),Formula1),
      drs2fol(Drs2,Formula2).

   cond2fol(imp(drs([X|Referents],Conds),Drs2),all(X,Formula)):-
     cond2fol(imp(drs(Referents,Conds),Drs2),Formula).
```

- the second clause adds a universal quantifier for each discourse referent in the antecedent DRS and translates its conditions into a first-order formula.
- the first clause translates the consequent DRS into a first-order formula and places the implication

DRS Construction

DRS Construction

We have seen how DRS Languages are defined and interpreted formally. In order to build a little agent that uses DRT to process language, we also need to specify how DRSs are constructed from natural language discourses.

Informally, we have seen that:

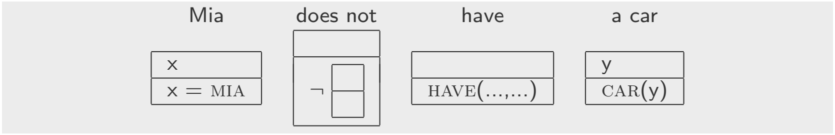
- NPs introduce discourse referents.
- Nouns and verbs introduce conditions.

(see the explanation of the construction algorithm in BB2, ch.1)

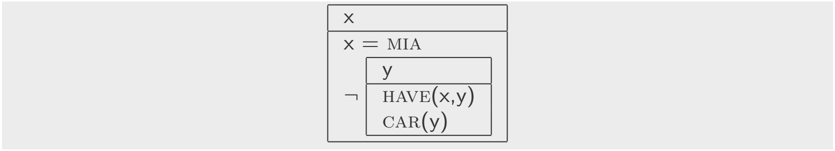
We shall see that we can formally define semantic constriction for DRT using the lambda calculus, as we did for first-order logic.

Composing DRSs (roughly)

Consider the following sentence: “Mia does not have a car”
Roughly, its parts seem to contribute the following bits:



How do we get from this to the overall representation?



- We need a mechanism for combining two DRSs into one larger DRSs – DRS merging
- We need a mechanism for marking missing information and indicating how it should be filled in – lambda calculus

Merging DRSs

We'll use the operator $\oplus$ to merge two DRSs.

- Merge combines two DRSs by taking the union of the two universes and the two lists of conditions. For example:

$$\begin{array}{|l|}\hline \mathrm{x} \\ \hline \textsc{boxer}(\mathrm{x}) \\ \textsc{lose}(\mathrm{x}) \\ \hline \end{array} \oplus \begin{array}{|l|}\hline \mathrm{y} \\ \hline \textsc{die}(\mathrm{y}) \\ \mathrm{y} = \mathrm{x} \\ \hline \end{array} = \begin{array}{|l|}\hline \mathrm{x}\ \mathrm{y} \\ \hline \textsc{boxer}(\mathrm{x}) \\ \textsc{lose}(\mathrm{x}) \\ \textsc{die}(\mathrm{y}) \\ \mathrm{y} = \mathrm{x} \\ \hline \end{array}$$

- Discourse processing: When we interpret a sentence with respect to the current discourse, we merge the new DRS with the DRS associated with the discourse so far.

λ-DRT: Lexical Items (1)

We can use the lambda calculus tools developed for first-order logic for DRT: with λ-DRT, we can use the λ operator with DRS.

The representations of the different kinds of lexical items will simply be the DRS counterparts of the first-order representations we are familiar with:

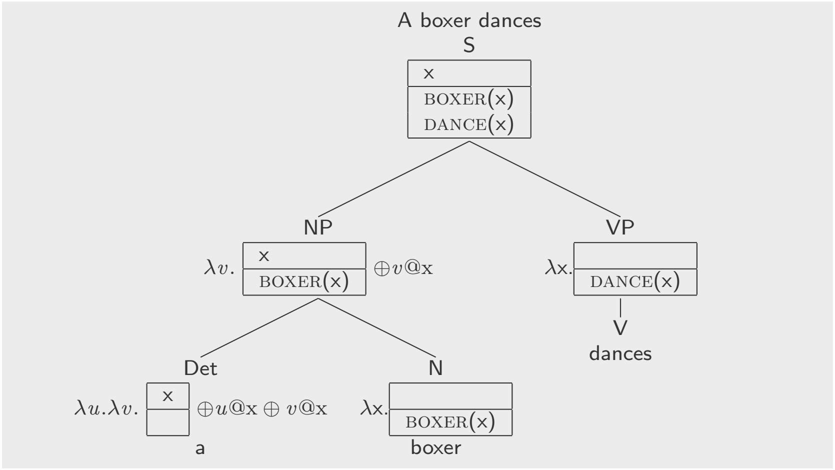
	first-order representation	**λ-DRT representation**
boxer:	$\lambda x.\textsc{boxer}(x)$	$\lambda\mathrm{x}.\begin{array}{\|l\|}\hline \\ \hline \textsc{boxer}(\mathrm{x}) \\ \hline\end{array}$
walk:	$\lambda x.\textsc{boxer}(x)$	$\lambda\mathrm{x}.\begin{array}{\|l\|}\hline \\ \hline \textsc{walk}(\mathrm{x}) \\ \hline\end{array}$
loves:	$\lambda v.\lambda x.(v@\lambda y.\textsc{love}(x,y))$	$\lambda v.\lambda\mathrm{x}.\ (v@\lambda\mathrm{y}.\begin{array}{\|l\|}\hline \\ \hline \textsc{love}(\mathrm{x},\mathrm{y}) \\ \hline\end{array})$
Mia:	$\lambda u.(u@\textsc{mia})$	$\lambda u.(\begin{array}{\|l\|}\hline \mathrm{x} \\ \hline \mathrm{x} = \textsc{mia} \\ \hline\end{array}) \oplus u@\mathrm{x}$

λ-DRT: Lexical Items (2)

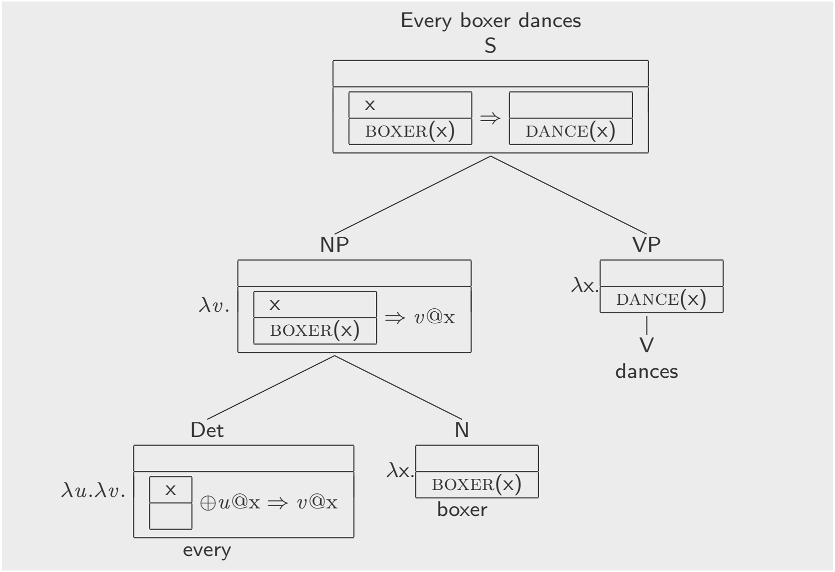
	first-order representation	**λ-DRT representation**
a:	$\lambda u.\lambda v.\exists x.(u@x \wedge v@x)$	$\lambda u.\lambda v.\boxed{\begin{array}{c} \mathrm{x} \\ \hline \; \end{array}} \oplus u@\mathrm{x} \oplus v@\mathrm{x}$
every:	$\lambda u.\lambda v.\forall x.(u@x \rightarrow v@x)$	$\lambda u.\lambda v.\boxed{\begin{array}{c} \; \\ \hline \boxed{\begin{array}{c} \mathrm{x} \\ \hline \; \end{array}} \oplus u@\mathrm{x} \Rightarrow v@\mathrm{x} \end{array}}$

Semantic Construction with λ-DRT

The main mechanisms for semantic construction are function application and β-conversion (as in FOL), plus merge.



Semantic Construction with λ-DRT



Implementing λ-DRT in Prolog

Implementing λ-DRT turns out to be very easy, because we can reuse most of the code we have for first-order logic. Recall our grammar architecture:

lambda.pl	SYNTAX	SEMANTICS
LEXICON	the Lexicon englishLexicon.pl	the Semantic Lexicon semLexLambda.pl
GRAMMAR	the Syntax Rules englishGrammar.pl	the Semantic Rules semRulesLambda.pl

- The Syntax will remain the same (with the incorporation of pronouns in the lexicon).
- The main task consists in defining the macros in the semantic lexicon with the λ-DRSs corresponding to each basic syntactic category.
- We also need to handle merge.

The Semantic Lexicon

As an example consider the semantic macro for the indefinite determiner "a" / "some":

	first-order representation	**λ-DRT representation**
a:	$\lambda u.\lambda v.\exists x.(u@x \wedge v@x)$	$\lambda u.\lambda v.\begin{array}{\|c\|}\hline \text{x} \\ \hline \\ \hline\end{array} \oplus u@\text{x} \oplus v@\text{x}$

First-order version (in semLexLambda.pl):

```
semLex(det,M):-
   M = [type:indef,
        sem:lam(P,lam(Q,some(X,and(app(P,X),app(Q,X)))))].
```

λ-DRT version (in semLexLambdaDRT.pl):

```
semLex(det,M):-
   M = [type:indef,
         num:sg,
        sem:lam(U,lam(V,merge(merge(drs([X],[]),app(U,X)),app(V,X))))].
```

Implementing Merge

- We can represent the merge of two DRSs K_1 and K_2 with the Prolog predicate `merge(K1,K2)` (as we've just seen).
- We still need a mechanism to actually do the merges (merge reduction). This is achieved by the program `mergeDRT.pl`:
  - ∗ It essentially appends the list of discourse referents and list conditions of the two DRSs we are merging.
  - ∗ And it does so recursively since merges can also occur in complex conditions.
  - ∗ Have a look at the code for more details.
- Note that `merge/2` and `mergeDRT.pl` work similarly to `app/2` and `betaConversion.pl` (in the sense that *merge* and β-conversion allow us to simplify the representations).

Wrapping Everything Together

The main level program is `lambdaDRT.pl` (which is the DRT version of `lambda.pl`).

- It reads in a sentence,
- computes all its semantic representations in the form of DRSs using the grammar,
- applies β-conversion and merge reduction, and
- prints the output DRS.

```
?- lambdaDRT.
> Mia walks.
1 drs([A], [mia(A), walk(A)])
```

The actual output is in fact a little bit more complicated:

```
?- lambdaDRT.
> Mia walks.
1 drs([A,B], [pred(mia,A), pred(walk,B), rel(agent,B,A), pred(event,B)])
```

This representation includes *events* as discourse referents and uses `pred/2` and `rel/3` to represent predicate symbols.

Summary of Programs

These are the main programs in BB2 used to build up DRSs for sentences:

- `lambdaDRT.pl`
- `lambdaTestSuite.pl`
- `mergeDRT.pl`
- `semLexLambdaDRT.pl`
- `semRulesDRT.pl`
- `englishLexicon.pl`
- `englishGrammar.pl`

Recall that we have also seen the program that implements fo:

- `drt2fol.pl`

Next Week

- Pronoun Resolution
  - ∗ Read chapter 3 from B&B2