Discourse

BSc Artificial Intelligence, Spring 2011

Raquel Fernández

Institute for Logic, Language & Computation
University of Amsterdam

Summary from Last Week

Semantic construction: how to systematically associate logical semantic representations with natural language sentences.

We introduced the lambda calculus, a notational extension of FOL very well suited for semantic construction.

- specify a reasonable syntax for the fragment of interest;
- associate semantic representation with each lexical item using lambda abstractions;
- use functional application and β-conversion to determine the meaning of a complex constituent from that of its subparts.

Extended grammar from Blackburn & Bos: syntax-semantics architecture covering an extended fragment of English.

$\Rightarrow$ Exercises 1 and 2 of HW #2 deal with semantic construction.

Plan for Today

We now have a basic architecture for translating natural language sentences into a formal meaning representation (FOL) and for checking whether they are valid in a given situation.

We are ready to move on to discourse – to dealing with more than single sentences. We'll start by addressing these two tasks:

- Consistency Checking Task: given the logical representation of a discourse, is it consistent or inconsistent?
- Informativity Checking Task: given the logical representation of a discourse, is it informative or uninformative?

These tasks are much more difficult than the querying task: they are *undecidable* for FOL as we shall see. To deal with them computationally, we'll use automated reasoning tools for theorem proving and model building.

Consistent Discourse

A consistent discourse is one that describes a possible state of affairs:

> Everybody likes Mia. She's smart and clever, and she dances gorgeously.
> Mia likes Butch, who is a boxer. All boxers are poor dancers.

Inconsistent discourses describe situations that are impossible:

> Everybody likes Mia. She's smart and clever, and she dances gorgeously.
> Mia likes Butch, who is a boxer. Butch doesn't like Mia.

Humans have a strong tendency to search for consistency. . .
("perhaps *'everybody'* should have been interpreted loosely?")

We'd like to be able to detect inconsistency when it arises because it typically signals trouble.

Consistency & Satisfiability

When is a discourse (in)consistent? The informal concept of consistency corresponds to the model-theoretic concept of **satisfiability**:

- A first-order formula is *satisfiable* if it is satisfied at least in one model: if it describes a conceivable or possible situation, e.g.: $\textsc{robber}(\textsc{mia})$
- A formula that is not satisfiable in any model is called *unsatisfiable*: it describes an impossible or unrealisable situation, e.g.: $\textsc{robber}(\textsc{mia}) \wedge \neg\textsc{robber}(\textsc{mia})$

We can extend this concept to finite sets of formulas by using conjunction:

- A finite set of formulas $\{\varphi_1, \ldots, \varphi_n\}$ is (un)satisfiable if $\varphi_1 \wedge \ldots \wedge \varphi_n$ is (un)satisfiable.

Accordingly, we define the consistency checking task as follows:

$\Rightarrow$ **Consistency Checking Task:** Given a first-order formula φ, is φ consistent (i.e. satisfiable) or inconsistent (i.e. unsatisfiable)?

Informative Discourse

Information is a very broad concept. But using a narrow sense of informativity, we can say that an informative discourse is one that provides genuinely new information (i.e. that it is not redundant):

A: Everybody likes Mia. Either she's very smart or she dances gorgeously.
B: Yeah. (Either she's very smart or she dances gorgeously.)

C: She's a poor dancer.

We'd like to be able to detect informativity and uninformativity, because only the former allows us to rule out possibilities.

Validity

How can we formalise (some aspects of) informativity? We can think of informativity in terms of the model-theoretic concept of validity:

- A formula φ is *valid* if it is satisfied in all models under any variable assignment: if it is impossible to find a situation and a context where it is not satisfied, e.g.: $\textsc{robber}(x) \vee \neg\textsc{robber}(x)$.
- A formula φ that fails to be satisfied in at least one model is *invalid*: e.g., $\textsc{robber}(x)$ is invalid as it is not possible to satisfy it in models where there are no robbers.

We use $\models \varphi$ to indicate that φ is valid, and $\not\models \varphi$ to indicate it is invalid.

Valid Arguments

Validity can be extended to the notion of valid arguments. Given a finite set of first-order formulas $\varphi_1, \ldots, \varphi_n$ and ψ:

- the argument with *premises* $\varphi_1, \ldots, \varphi_n$ and *conclusion* ψ is a *valid argument* if and only if whenever all premises are satisfied in some model using some variable assignment, then the conclusion is also satisfied in the same model and using the same variable assignment.
  - we can also say that ψ is a *logical consequence* of $\varphi_1, \ldots, \varphi_n$ or a *valid inference* from $\varphi_1, \ldots, \varphi_n$.
- an *invalid argument* is one for which it is possible to find a model and an assignment that satisfies all the premises but not the conclusion.

We indicate that an argument is valid by writing $\varphi_1, \ldots, \varphi_n \models \psi$, and that it is invalid by writing $\varphi_1, \ldots, \varphi_n \not\models \psi$.

Semantic Deduction Theorem

Validity and valid arguments are closely related: with the help of the boolean connectives $\rightarrow$ and $\wedge$ we can convert any valid argument into a valid formula.

For instance, the following argument is valid:

$$\forall x(\text{ROBBER}(x) \rightarrow \text{CUSTOMER}(x)), \text{ROBBER}(\text{MIA}) \models \text{CUSTOMER}(\text{MIA})$$

and so is the following formula:

$$\models \forall x(\text{ROBBER}(x) \rightarrow \text{CUSTOMER}(x)) \wedge \text{ROBBER}(\text{MIA}) \rightarrow \text{CUSTOMER}(\text{MIA})$$

This is an example of the Semantic Deduction Theorem:

$$\varphi_1, \ldots, \varphi_n \models \psi \text{ if and only if } \models (\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \psi$$

Validity & Informativity (1)

What does validity tell us about informativity?

There is a clear sense in which valid formulas are *uninformative*: because they are satisfied in all models, they don't tell us anything about any particular model - they don't rule out possibilities. This idea can be extended to arguments too:

- if $\varphi_1, \ldots, \varphi_n \models \psi$, and we know that $\varphi_1, \ldots, \varphi_n$, then ψ does not tell us anything new.

All customers in this diner are robbers. One of the customers is Mia. **She's a robber.**

$$\forall x(\textsc{customer}(x) \rightarrow \textsc{robber}(x)), \textsc{customer}(\textsc{mia}) \models \textsc{robber}(\textsc{mia})$$

- however, if $\varphi_1, \ldots, \varphi_n \not\models \psi$ and we already know $\varphi_1, \ldots, \varphi_n$, then when we are told ψ we have learned something new.

All customers in this diner are robbers. One of the customers is Mia. **She has a gun.**

$$\forall x(\textsc{customer}(x) \rightarrow \textsc{robber}(x)), \textsc{customer}(\textsc{mia}) \not\models \textsc{have}(\textsc{mia}, \textsc{gun})$$

Validity & Informativity (2)

Hence we can introduce the following terminology:

- if $\varphi_1, \ldots, \varphi_n \models \psi$ we say that ψ is *uninformative with respect to* $\varphi_1, \ldots, \varphi_n$

All customers in this diner are robbers. One of the customers is Mia. **She's a robber.**
$\forall x(\text{CUSTOMER}(x) \rightarrow \text{ROBBER}(x)), \text{CUSTOMER}(\text{MIA}) \models \text{ROBBER}(\text{MIA})$

- if $\varphi_1, \ldots, \varphi_n \not\models \psi$ we say that ψ is *informative with respect to* $\varphi_1, \ldots, \varphi_n$

All customers in this diner are robbers. One of the customers is Mia. **She has a gun.**
$\forall x(\text{CUSTOMER}(x) \rightarrow \text{ROBBER}(x)), \text{CUSTOMER}(\text{MIA}) \not\models \text{HAVE}(\text{MIA}, \text{GUN})$

By appealing to the Semantic Deduction Theorem, we can reduce testing for informativity to a task involving a single formula:

$\varphi_1, \ldots, \varphi_n \not\models \psi$ if and only if $\not\models (\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \psi$

$\Rightarrow$ **Informativity Checking Task:** Given a first-order formula φ, is φ informative (i.e. invalid) or uninformative (i.e. valid)?

How Hard are these Tasks?

- Consistency Checking Task: given a first-order formula φ, is φ consistent (i.e. satisfiable) or inconsistent (i.e. unsatisfiable)?
  - φ is satisfiable if it is satisfied at least in one model
- Informativity Checking Task: given a first-order formula φ, is φ informative (i.e. invalid) or uninformative (i.e. valid)?
  - φ is valid if it is satisfied in all models under any variable assignment

These tasks require us to check all possible models, but there is an infinite number of possible models!

Therefore there is no algorithm capable of solving these tasks in finite time for all possible input formulas...

$\Rightarrow$ Unlike the querying task (model checking task) where the model and assignment used for evaluation are fixed, the consistency and informativity checking tasks for FOL are **undecidable**.

Relating Consistency & Informativity (1)

There is one piece of good news however – both tasks boil down to one single problem:

1. φ is consistent (satisfiable) iff $\neg\varphi$ is informative (invalid)
2. φ is inconsistent (unsatisfiable) iff $\neg\varphi$ is uninformative (valid)
3. φ is informative (invalid) iff $\neg\varphi$ is consistent (satisfiable)
4. φ is uninformative (valid) iff $\neg\varphi$ is inconsistent (unsatisfiable)

Consider 1 above:

> Suppose φ is consistent. This means it is satisfiable in at least one model. But this is the same as saying that there is at least one model where $\neg\varphi$ is not satisfiable. Which is precisely to say that φ is informative.

Exercise 3 in Homework #2 asks you to give similar arguments for establishing the truth of the relationships in 2, 3, and 4 above.

Relating Consistency & Informativity (2)

Both tasks can thus be reformulated in terms of the validity of single formulas:

- saying that ψ is uninformative with respect to $\varphi_1, \ldots, \varphi_n$ means that $\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \psi$ is valid
- saying that ψ is inconsistent with respect to $\varphi_1, \ldots, \varphi_n$ means that $\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \neg\psi$ is valid

Therefore, what we need is a computational way to deal with validity. However, since validity checking for FOL is undecidable, there are no computational tools that give us a full algorithmic solution ...

Fortunately, there are some *partial* solutions we can exploit: we'll use theorem provers and model builders in combination to tackle the problem.

What is a Theorem Prover?

A theorem prover is a system that provides us with a systematic method for checking whether or not it is possible to build a model in which some given formula is true or false.

Methods for theorem proving are investigated within a branch of computer science called *automated reasoning*. The crucial feature of such proof methods is that they concentrate on the syntactic structure of formulas.

Such a systematic method gives us a way to test a formula for validity:

- a formula φ would be valid if and only if the method told us that there is no way to build a model that falsifies φ (that is, a model for $\neg\varphi$).

Typically, theorem provers use *refutation proof methods*: to show that φ is valid, we show that $\neg\varphi$ leads to a contradiction.

Theorem Provers and Undecidability

Recall that FOL is undecidable: it is impossible to build a computer program that can correctly decide *in finite time* whether an *arbitrary first-order formula* is valid or not.

If a theorem prover fails to prove a formula this can be due to two rather different reasons:

- the formula may be not valid, hence no proof exists, and so the prover cannot find a proof;
- the formula may be valid but very difficult to prove, so the prover may use up all its resources without finding a proof (even if it exists).

We can never be sure! A theorem prover thus cannot prove non-validity. It can only prove validity (of formulas that are not extremely complex nor tricky to prove).

Proof Methods

Many proofs methods for theorem proving have been studied: *tableau* and *resolution* methods are two of the most common ones.

We shall briefly review the tableau method:

- it is conceptually simple and easy to understand;
- you should have covered its rudiments in previous courses, at least for propositional logic;
- it is used by many "real" theorem provers

Blackburn and Bos include two simple theorem provers implemented in Prolog. These are the relevant programs:

Propositional Logic	**First-Order Logic**
`propTableau.pl`	`freeVarTabl.pl`
`propResolution.pl`	`foResolution.pl`
`propTestSuite.pl`	`folTestSuite.pl`

We will not discuss this code. We shall instead concentrate on the overall general idea behind theorem proving and on how it can be used for issues in natural language discourse.

Tableau Method

These are the main ingredients of the tableau method for proving validity by refutation:

- A proof is constructed as a *tableau* – a binary tree, the nodes of which are labelled with formulas.
- Start: we start by putting in the root of the tree we want to falsify (the negation of the original formula)
- Expansion: we apply *expansion rules* to the formulas on the tree, thereby adding new formulas and splitting branches.
- Closure: we close branches that are obviously contradictory.
- Success: a proof is successful iff we can close all branches.

[N.B.: There are of course other variants, where e.g. tableaus are not trees but different data structures. The main idea is what counts.]

Expansion Rules

Propositional rules:

Alpha Rules: $\dfrac{A \wedge B}{\begin{array}{c} A \\ B \end{array}}$ $\dfrac{\neg(A \vee B)}{\begin{array}{c} \neg A \\ \neg B \end{array}}$ $\dfrac{\neg(A \rightarrow B)}{\begin{array}{c} A \\ \neg B \end{array}}$ $\neg\neg$-Elimination: $\dfrac{\neg\neg A}{A}$

Beta Rules: $\dfrac{A \vee B}{A \mid B}$ $\dfrac{A \rightarrow B}{\neg A \mid B}$ $\dfrac{\neg(A \wedge B)}{\neg A \mid \neg B}$ Closure: $\dfrac{\begin{array}{c} A \\ \neg A \end{array}}{\times}$

Quantifier rules for first-order logic:

Gamma Rules: $\dfrac{\forall x.A}{A[t/x]}$ $\dfrac{\neg\exists x.A}{\neg A[t/x]}$ Delta Rules: $\dfrac{\exists x.A}{A[c/x]}$ $\dfrac{\neg\forall x.A}{\neg A[c/x]}$

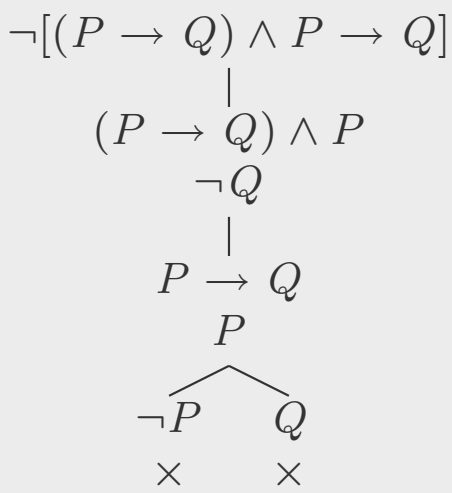
- Beta Rules are branching rules that split the tree into branches.
- t is an arbitrary term and c is a constant symbol new to the branch.
- Unlike all other rules, the gamma rules may have to be applied more than once to the same formula on the same branch.
- Recall that $\varphi[t/x]$ denotes the formula we get by replacing each free occurrence of the variable x in φ by the term t.

A Propositional Example

Let us show that the following formula is valid:

$$(P \rightarrow Q) \wedge P \rightarrow Q$$

We need to construct a proof that shows that its negation leads to a contradiction. Here is the tableau:

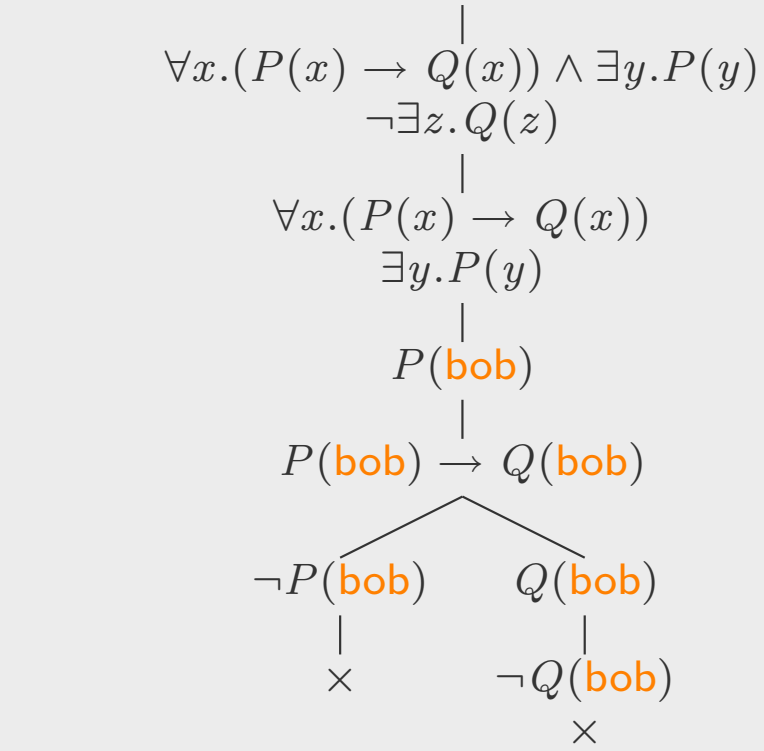


- First, we apply the third alpha rule to the root (because it is a negated implication)
- Then, the first alpha rule to $(P \rightarrow Q) \wedge P$ (because it is a conjunction)
- Then, the second beta rule to $P \rightarrow Q$, which splits the tree in two branches
- Each branch can then be closed because they are contradictory: the left branch contains $\neg P$ and P and the right branch contains Q and $\neg Q$.

A First-Order Example

Let us show that the following formula is valid:

$$\forall x.(P(x) \rightarrow Q(x)) \wedge \exists y.P(y) \;\rightarrow\; \exists z.Q(z)$$



- We apply the third alpha rule to the root.
- Then, the first alpha rule to the conjunction $\forall x.(P(x) \rightarrow Q(x)) \wedge \exists y.P(y)$.
- We are left with only gamma and delta rules to be applied: we first apply delta to $\exists y.P(y)$ using bob as constant.
- We then apply the first gamma rule to $\forall x.(P(x) \rightarrow Q(x))$ with term bob.
- We can close the left branch.
- We use the second gamma rule with $\neg\exists z.Q(z)$ and bob, and then we can also close the right branch.

Back to Discourse...

We have seen that the consistency and informativity checking tasks can be formulated in terms of validity. And that theorem provers are automated reasoning tools designed to prove validity.

$\Rightarrow$ Are we able to fully address computationally the consistency and informativity checking tasks by using a theorem prover?

Let us look at each of these tasks in turn...

Theorem Proving & Consistency (1)

How can we automatically show that this discourse is *inconsistent*?

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

- use semantic construction to build first-order representations for the sentences in the discourse:

$$\forall x.(\text{BOXER}(x) \rightarrow \text{SLOW}(x))$$
$$\text{BOXER}(\text{BUTCH})$$
$$\neg\text{SLOW}(\text{BUTCH})$$

- check whether the conjunction of the first two sentences implies the negation of the last one - the final sentence is inconsistent with the preceding discourse if and only if this formula is valid:

$$\forall x.(\text{BOXER}(x) \rightarrow \text{SLOW}(x)) \wedge \text{BOXER}(\text{BUTCH}) \rightarrow \neg\neg\text{SLOW}(\text{BUTCH})$$

- give the formula to a theorem prover to automatically prove its validity.

Theorem Proving & Consistency (2)

If a discourse is inconsistent, then (assuming the problem is not too hard for the prover) a decent first-order theorem prover can demonstrate its inconsistency.

But theorem provers can't prove non-validity, so we don't have a way to automatically prove that a discourse, such as the following, is *consistent*:

All boxers are slow. Butch is a boxer. Mia likes Butch.

⇒ *Theorem provers provide us with a negative check for consistency.*

Theorem Proving & Informativity (1)

Let's look into the consequences of undecidability for informativity.

How can we automatically show that this discourse is *uninformative*?

All boxers are crazy. Butch is a boxer. Butch is crazy.

- use semantic construction to build first-order representations for the sentences in the discourse:

$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{crazy}(x))$$
$$\textsc{boxer}(\textsc{butch})$$
$$\textsc{crazy}(\textsc{butch})$$

- check whether the conjunction of the first two sentences implies the the last one - the final sentence is uninformative with respect to the preceding discourse if and only if this formula is valid:

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

- give the formula to a theorem prover to automatically prove its validity.

Theorem Proving & Informativity (2)

As before, problems arise when the discourse is *informative*.

All boxers are crazy. Butch is a boxer. Butch loves Fabian.

Here the inference tasks we face is to determine whether the following formula is valid:

$$\forall x.(\textsc{boxer}(x) \rightarrow \textsc{crazy}(x)) \wedge \textsc{boxer}(\textsc{butch}) \rightarrow \textsc{love}(\textsc{butch}, \textsc{fabian})$$

This formula is clearly not valid (there are models where it could be false). But again: no general computational tools for determining non-validity exist!

$\Rightarrow$ *Theorem provers provide us with a negative check for informativity.*

Summarizing. . .

Theorem provers provide us with a negative check for consistency and informativity:

- they can demonstrate the inconsistency or uninformativity of a discourse simply by proving a theorem (provided the theorem is not too hard).

However theorem proving does not provide us with a positive test:

- due to the undecidability of FOL, full positive checks for consistency and informativity do not exists.

Are there any tools that provide us with *partial* positive checks?
Yes! they are called model builders.

Model Builders

Recall what a model checker does: it takes a formula and a model and checks whether the formula is satisfied in that model or not.

A **model builder** does something more difficult: it takes a formula, and tries to build a model that satisfies it.

- Model builders are relatively new automated reasoning tools;
- They do not build arbitrary infinite models, so even when some possible models exist they may not be able to always build them.
- Model builders are only capable to build relatively small finite models.
- Often the domain size can be specified (e.g. up to 20 elements).

Although all this may sound simple, model builders can help us to get partial positives checks for consistency and informativity.

Model Building & Consistency

The logical representation of this discourse is consistent:

All boxers are slow. Butch is a boxer. Mia likes Butch.

$$\forall x.(\text{BOXER}(x) \rightarrow \text{SLOW}(x)) \wedge \text{BOXER}(\text{BUTCH}) \wedge \text{LIKE}(\text{MIA}, \text{BUTCH})$$

Suppose we ask a model builder to build a 2-element model:

Some possible models:

$D = \{d_1, d_2\}$
$F(\text{BUTCH}) = d_1$
$F(\text{MIA}) = d_2$
$F(\text{BOXER}) = \{d_1\}$
$F(\text{SLOW}) = \{d_1\}$
$F(\text{LIKE}) = \{(d_2, d_1)\}$

$D = \{d_1, d_2\}$
$F(\text{BUTCH}) = d_1$
$F(\text{MIA}) = d_2$
$F(\text{BOXER}) = \{d_1\}$
$F(\text{SLOW}) = \{d_1, d_2\}$
$F(\text{LIKE}) = \{(d_2, d_1)\}$

The automatically built models may not always be intuitive, but what matters to establish consistency is that there is *at least one model* where the formula is satisfied.

⇒ *Model builders provide us with a (partial) positive check for consistency.*

Model Building & Informativity

The last sentence in this discourse is clearly informative:

All boxers are crazy. Butch is a boxer. Butch loves Fabian.

That means the following formula is *not* valid, which in turn means that its negation has at least one model:

$$\forall x.(\text{BOXER}(x) \rightarrow \text{CRAZY}(x)) \wedge \text{BOXER}(\text{BUTCH}) \rightarrow \text{LOVE}(\text{BUTCH}, \text{FABIAN})$$

So we should ask a model builder to build a model for one of these two equivalent formulas:

$$\neg(\forall x.(\text{BOXER}(x) \rightarrow \text{CRAZY}(x)) \wedge \text{BOXER}(\text{BUTCH}) \rightarrow \text{LOVE}(\text{BUTCH}, \text{FABIAN}))$$

$$\forall x.(\text{BOXER}(x) \rightarrow \text{CRAZY}(x)) \wedge \text{BOXER}(\text{BUTCH}) \wedge \neg\text{LOVE}(\text{BUTCH}, \text{FABIAN})$$

This is easy! As long as there is a model for the negated formula, the original formula is not valid and hence the discourse is informative.

⇒ *Model builders provide us with a (partial) positive check for informativity.*

Combining both Tools

We have seen that:

- Theorem provers are capable of carrying out negative checks for consistency and informativity.
- Model builders are capable of carrying out partial positive checks for these tasks.

The optimal computational strategy is thus to use simultaneously theorem proving and model building on the input formula – that is, to do both negative and (partial) positive checks in parallel.

Next week we'll explore an architecture that deals with the consistency and informativity tasks by carrying out these processes in parallel.

Summary

We have seen that the linguistic notions of consistency and informativity of a discourse can be analysed with the logical concepts of satisfiability and validity.

- Consistency Checking Task: given a first-order formula φ, is φ consistent (i.e. satisfiable) or inconsistent (i.e. unsatisfiable)?
  - φ is satisfiable if it is satisfied at least in one model
- Informativity Checking Task: given a first-order formula φ, is φ informative (i.e. invalid) or uninformative (i.e. valid)?
  - φ is valid if it is satisfied in all models under any variable assignment

Both tasks boil down to checking validity. In First-Order Logic, this task is undecidable. However:

$\Rightarrow$ we can use theorem provers to obtain negative checks, and model builders to obtain partial positive checks.

Next Week

We'll have a practical session where we'll put into practice today's ideas and go over what we have learned in the course so far.

Room details will be announced as soon as possible.