Symbolic Natural Language Processing in GLIF

Prof. Dr. Michael Kohlhase

Knowledge Representation and -Processing
Computer Science, FAU Erlangen-Nürnberg
Michael.Kohlhase@FAU.de

EuroProofNet Summer School on AI for Reasoning and Processing of Mathematics – 25/27. June 2024

Contents

Chapter 1

An Introduction to Natural Language Semantics

In this chapter we will introduce the topic of this course and situate it in the larger field of natural language understanding. But before we do that, let us briefly step back and marvel at the wonders of natural language, perhaps one of the most human of abilities.

Fascination of (Natural) Language

- ▷ **Definition 1.0.1.** A natural language is any form of spoken or signed means of communication that has evolved naturally in humans through use and repetition without conscious planning or premeditation.
- ▷ **In other words:** the language you use all day long, e.g. English, German, ...
- ▷ **Why Should we care about natural language?:**
  - ▷ Even more so than thinking, language is a skill that only humans have.
  - ▷ It is a miracle that we can express complex thoughts in a sentence in a matter of seconds.
  - ▷ It is no less miraculous that a child can learn tens of thousands of words and a complex grammar in a matter of a few years.

With this in mind, we will embark on the intellectual journey of building artificial systems that can process (and possibly understand) natural language as well.

1.1 Natural Language and its Meaning

Before we embark on the journey into understanding the meaning of natural language, let us get an overview over what the concept of "semantics" or "meaning" means in various disciplines. A good probe into the issues involved in natural language understanding is to look at translations between natural language utterances – a task that arguably involves understanding the utterances first.

Meaning of Natural Language; e.g. Machine Translation

- **Idea:** Machine Translation is very simple! (we have good lexica)
- **Example 1.1.1.** *Peter liebt Maria.* $\rightsquigarrow$ *Peter loves Mary.*
- ⚠ this only works for simple examples!
- **Example 1.1.2.** *Wirf der Kuh das Heu über den Zaun.* $\not\rightsquigarrow$ *Throw the cow the hay over the fence.* (differing grammar; Google Translate)
- **Example 1.1.3.** ⚠ Grammar is not the only problem
  - *Der Geist ist willig, aber das Fleisch ist schwach!*
  - *Der Schnaps ist gut, aber der Braten ist verkocht!*
- **Observation 1.1.4.** *We have to understand the meaning for high-quality translation!*

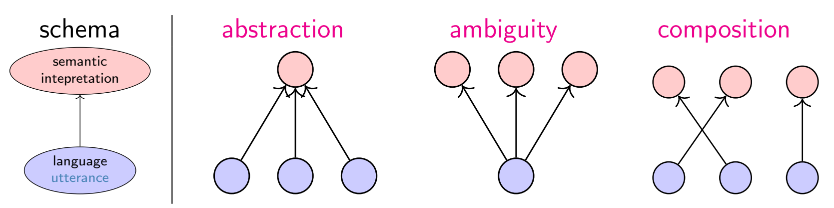
If it is indeed the meaning of natural language, we should look further into how the form of the utterances and their meaning interact.

Language and Information

- **Observation:** Humans use words (sentences, texts) in natural languages to represent and communicate information.
- **But:** What really counts is not the words themselves, but the meaning information they carry.
- **Example 1.1.5 (Word Meaning).**

Newspaper $\rightsquigarrow$ 

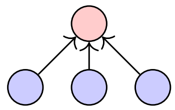
- For questions/answers, it would be very useful to find out what words (sentences/-texts) mean.
- **Definition 1.1.6.** Interpretation of natural language utterances: three problems



Let us support the last claim a couple of initial examples. We will come back to these phenomena again and again over the course of the course and study them in detail.

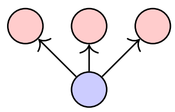
Language and Information (Examples)

▷ **Example 1.1.7 (Abstraction).**



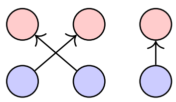
Car and *automobile* have the same meaning

▷ **Example 1.1.8 (Ambiguity).**



A *bank* can be a financial institution or a geographical feature

▷ **Example 1.1.9 (Composition).**



Every student sleeps $\leadsto \forall x.student(x) \Rightarrow sleep(x)$

FAU Michael Kohlhase: Symbolic NLP in GLIF 6 2024-06-27

But there are other phenomena that we need to take into account when compute the meaning of NL utterances.

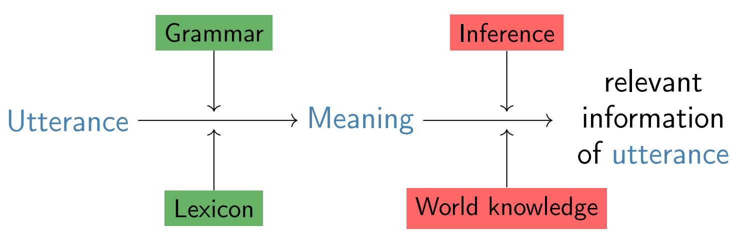
Context Contributes to the Meaning of NL Utterances

▷ **Observation:** Not all information conveyed is linguistically realized in an utterance.

▷ **Example 1.1.10.** *The lecture begins at 11:00 am.* What lecture? Today?

▷ **Definition 1.1.11.** We call a piece i of information linguistically realized in an utterance U, iff, we can trace i to a fragment of U.

▷ **Definition 1.1.12 (Possible Mechanism).** Inferring the missing pieces from the context and world knowledge:



We call this process pragmatic analysis.

FAU Michael Kohlhase: Symbolic NLP in GLIF 7 2024-06-27

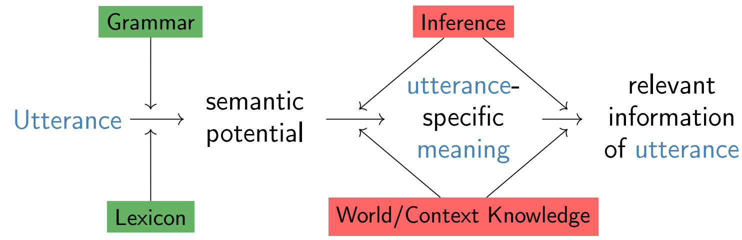
We will look at another example, that shows that the situation with pragmatic analysis is even more complex than we thought. Understanding this is one of the prime objectives of the LBS lecture.

Context Contributes to the Meaning of NL Utterances

▷ **Example 1.1.13.** *It starts at eleven.* What starts?

▷ Before we can resolve the time, we need to resolve the anaphor *it*.

▷ **Possible Mechanism:** More Inference!



⇝ Pragmatic analysis is quite complex! (prime topic of LBS)

Example 1.1.13 is also a very good example for the claim Observation 1.1.4 that even for high-quality (machine) translation we need semantics. We end this very high-level introduction with a caveat.

Semantics is not a Cure-It-All!

How many animals of each species did Moses take onto the ark?

▷ Actually, it was Noah (But you understood the question anyways)

But Semantics works in some cases

▷ The only thing that currently really helps is a restricted domain:

▹ I. e. a restricted vocabulary and world model.

▷ **Demo:**

DBPedia `http://dbpedia.org/snorql/`
Query: Soccer players, who are born in a country with more than 10 million inhabitants, who played as goalkeeper for a club that has a stadium with more than 30.000 seats and the club country is different from the birth country

But Semantics works in some cases

▷ **Answer:**

(is computed by DBPedia from a SPARQL query)

```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
?soccerplayer a dbo:SoccerPlayer ;
   dbo:position|dbp:position <http://dbpedia.org/resource/Goalkeeper_(association_football)> ;
   dbo:birthPlace/dbo:country* ?countryOfBirth ;
   #dbo:number 13 ;
   dbo:team ?team .
   ?team dbo:capacity ?stadiumcapacity ; dbo:ground ?countryOfTeam .
   ?countryOfBirth a dbo:Country ; dbo:populationTotal ?population .
   ?countryOfTeam a dbo:Country .
FILTER (?countryOfTeam != ?countryOfBirth)
FILTER (?stadiumcapacity > 30000)
FILTER (?population > 10000000)
} order by ?soccerplayer
```

Results: Browse Go! Reset

SPARQL results:

soccerplayer	countryOfBirth	team	countryOfTeam	stadiumcapacity
:Abdesslam_Benabdellah	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Airton_Moraes_Michellon	:Brazil	:FC_Red_Bull_Salzburg	:Austria	31000
:Alain_Gouaméné	:Ivory_Coast	:Raja_Casablanca	:Morocco	67000
:Allan_McGregor	:United_Kingdom	:Beşiktaş_J.K.	:Turkey	41903
:Anthony_Scribe	:France	:FC_Dinamo_Tbilisi	:Georgia_(country)	54549
:Brahim_Zaari	:Netherlands	:Raja_Casablanca	:Morocco	67000
:Bréiner_Castillo	:Colombia	:Deportivo_Táchira	:Venezuela	38755
:Carlos_Luis_Morales	:Ecuador	:Club_Atlético_Independiente	:Argentina	48069
:Carlos_Navarro_Montoya	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Cristián_Muñoz	:Argentina	:Colo-Colo	:Chile	47000
:Daniel_Ferreyra	:Argentina	:FBC_Melgar	:Peru	60000
:David_Bičík	:Czech_Republic	:Karşıyaka_S.K.	:Turkey	51295
:David_Loria	:Kazakhstan	:Karşıyaka_S.K.	:Turkey	51295
:Denys_Boyko	:Ukraine	:Beşiktaş_J.K.	:Turkey	41903
:Eddie_Gustafsson	:United_States	:FC_Red_Bull_Salzburg	:Austria	31000
:Emilian_Dolha	:Romania	:Lech_Poznań	:Poland	43269
:Eusebio_Acasuzo	:Peru	:Club_Bolívar	:Bolivia	42000
:Faryd_Mondragón	:Colombia	:Real_Zaragoza	:Spain	34596
:Faryd_Mondragón	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Federico_Vilar	:Argentina	:Club_Atlas	:Mexico	54500
:Fernando_Martinuzzi	:Argentina	:Real_Garcilaso	:Peru	45000
:Fábio_André_da_Silva	:Portugal	:Servette_FC	:Switzerland	30084
:Gerhard_Tremmel	:Germany	:FC_Red_Bull_Salzburg	:Austria	31000
:Gift_Muzadzi	:United_Kingdom	:Lech_Poznań	:Poland	43269
:Günay_Güvenç	:Germany	:Beşiktaş_J.K.	:Turkey	41903
:Hugo_Marques	:Portugal	:C.D._Primeiro_de_Agosto	:Angola	48500
:Héctor_Landazuri	:Colombia	:La_Paz_F.C.	:Bolivia	42000

Even if we can get a perfect grasp of the semanticss (aka. meanings) of NL utterances, their structure and context dependency – we will try this in this lecture, but of course fail, since the issues are much too involved and complex for just one lecture – then we still cannot account for all the human mind does with language. But there is hope, for limited and well-understood domains, we can to amazing things. This is what this course tries to show, both in theory as well as in practice.

1.2 Natural Language Understanding as Engineering

Even though this course concentrates on computational aspects of natural language semantics, it is useful to see it in the context of the field of natural language processing.

Language Technology

- Language Assistance:
  - written language: Spell/grammar/style-checking,
  - spoken language: dictation systems and screen readers,
  - multilingual text: machine-supported text and dialog translation, eLearning.
- Information management:
  - search and classification of documents, (e.g. Google/Bing)
  - information extraction, question answering. (e.g. `http://ask.com`)
- Dialog Systems/Interfaces:
  - information systems: at airport, tele-banking, e-commerce, call centers,
  - dialog interfaces for computers, robots, cars. (e.g. Siri/Alexa)
- **Observation:** The earlier technologies largely rely on pattern matching, the latter ones need to compute the meaning of the input utterances, e.g. for database lookups in information systems.

The general context of LBS is natural language processing (NLP), and in particular natural language understanding (NLU). The dual side of NLU: natural language generation (NLG) requires similar foundations, but different techniques is less relevant for the purposes of this course.

What is Natural Language Processing?

- **Generally:** Studying of natural languages and development of systems that can use/generate these.
- **Definition 1.2.1.** Natural language processing (NLP) is an engineering field at the intersection of computer science, artificial intelligence, and linguistics which is concerned with the interactions between computers and human (natural) languages. Most challenges in NLP involve:
  - Natural language understanding (NLU) that is, enabling computers to derive meaning (representations) from human or natural language input.
  - Natural language generation (NLG) which aims at generating natural language or speech from meaning representation.
- For communication with/among humans we need both NLU and NLG.

What is the State of the Art In NLU?

- Two avenues of attack for the problem: knowledge-based and statistical techniques (they are complementary)

Deep	Knowledge-based We are here	Not there yet cooperation?
Shallow	no-one wants this	Statistical Methods applications
Analysis ↑ vs. Coverage →	narrow	wide

▷ We will cover foundational methods of deep processing in the course and a mixture of deep and shallow ones in the lab.

On the last slide we have classified the two main approaches to NLU. In the last 10 years the community has almost entirely concentrated on statistical- and machine-learning based methods, because that has led to applications like google translate, Siri, and the likes. We will now borrow an argument by Aarne Ranta to show that there are (still) interesting applications for knowledge-based methods in NLP, even if they are less visible.

Environmental Niches for both Approaches to NLU

▷ **Definition 1.2.2.** There are two kinds of applications/tasks in NLU:

- ▷ Consumer tasks: consumer grade applications have tasks that must be fully generic and wide coverage. (e.g. machine translation like Google Translate)
- ▷ Producer tasks: producer grade applications must be high-precision, but can be domain-specific (e.g. multilingual documentation, machinery-control, program verification, medical technology)

Precision			
100%	Producer Tasks		
50%		Consumer Tasks	
	$10^{3\pm1}$ Concepts	$10^{6\pm1}$ Concepts	**Coverage**

▷ **Example 1.2.3.** Producing/managing machine manuals in multiple languages across machine variants is a critical producer task for machine tool company.

▷ A producer domain I am interested in: mathematical/technical documents.

An example of a producer task – indeed this is where the name comes from – is the case of a machine tool manufacturer T, which produces digitally programmed machine tools worth multiple million Euro and sells them into dozens of countries. Thus T must also comprehensive machine operation manuals, a non-trivial undertaking, since no two machines are identical and they must be translated into many languages, leading to hundreds of documents. As those manual share a lot of semantic content, their management should be supported by NLP techniques. It is critical that these NLP maintain a high precision, operation errors can easily lead to very costly machine damage and loss of production. On the other hand, the domain of these manuals is quite restricted. A machine tool has a couple of hundred components only that can be described by a comple of thousand attribute only.

Indeed companies like T employ high-precision NLP techniques like the ones we will cover in this course successfully; they are just not so much in the public eye as the consumer tasks.

NLP for NLU: The Waterfall Model

▷ **Definition 1.2.4 (The NLU Waterfall).** NL understanding is often modeled as a simple linear process: the NLU waterfall consists of five consecutive steps:

0) speech processing: acoustic signal $\rightsquigarrow$ word hypothesis graph
1) syntactic processing: word sequence $\rightsquigarrow$ phrase structure
2) semantics construction: phrase structure $\rightsquigarrow$ (quasi-)logical form
3) semantic/pragmatic analysis:
   (quasi-)logical form $\rightsquigarrow$ knowledge representation
4) problem solving: using the generated knowledge (application-specific)

▷ **Definition 1.2.5.** We call any formalization of an utterance as a logical formula a logical form. A quasi-logical form (QLF) is a representation which can be turned into a logical form by further computation.2

▷ **In this course:** steps 1), 2) and 3).

The waterfall model shown above is of course only an engineering-centric model of natural language understanding and not to be confused with a cognitive model; i.e. an account of what happens in human cognition. Indeed, there is a lot of evidence that this simple sequential processing model is not adequate, but it is the simplest one to implement and can therefore serve as a background reference to situating the processes we are interested in.

1.3 Looking at Natural Language

The next step will be to make some observations about natural language and its meaning, so that we get an intuition of what problems we will have to overcome on the way to modeling natural language.

Fun with Diamonds (are they real?) [Dav67]

▷ **Example 1.3.1.** We study the truth conditions of adjectival complexes:

- ▷ *This is a diamond.* ($\models$ *diamond*)
- ▷ *This is a blue diamond.* ($\models$ *diamond*, $\models$ *blue*)
- ▷ *This is a big diamond.* ($\models$ *diamond*, $\not\models$ *big*)
- ▷ *This is a fake diamond.* ($\models \neg$*diamond*)
- ▷ *This is a fake blue diamond.* ($\models$ *blue*?, $\models$ *diamond*?)
- ▷ *Mary knows that this is a diamond.* ($\models$ *diamond*)

▷ *Mary believes that this is a diamond.* (⊭ *diamond*)

Logical analysis vs. conceptual analysis: These examples — mostly borrowed from Davidson:tam67 — help us to see the difference between "logical-analysis" and "conceptual-analysis".

We observed that from *This is a big diamond.* we cannot conclude *This is big*. Now consider the sentence *Jane is a beautiful dancer*. Similarly, it does not follow from this that Jane is beautiful, but only that she dances beautifully. Now, what it is to be beautiful or to be a beautiful dancer is a complicated matter. To say what these things are is a problem of conceptual analysis. The job of semantics is to uncover the logical form of these sentences. Semantics should tell us that the two sentences have the same logical forms; and ensure that these logical forms make the right predictions about the entailments and truth conditions of the sentences, specifically, that they don't entail that the object is big or that Jane is beautiful. But our semantics should provide a distinct logical form for sentences of the type: *This is a fake diamond.* From which it follows that the thing is fake, but not that it is a diamond.

Ambiguity: The dark side of Meaning

▷ **Definition 1.3.2.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.

▷ **Example 1.3.3.** All of the following sentences are ambiguous:

- ▷ *John went to the bank.* (river or financial?)
- ▷ *You should have seen the bull we got from the pope.* (three readings!)
- ▷ *I saw her duck.* (animal or action?)
- ▷ *John chased the gangster in the red sports car.* (three-way too!)

One way to think about the examples of ambiguity on the previous slide is that they illustrate a certain kind of indeterminacy in sentence meaning. But really what is indeterminate here is what sentence is represented by the physical realization (the written sentence or the phonetic string). The symbol *duck* just happens to be associated with two different things, the noun and the verb. Figuring out how to interpret the sentence is a matter of deciding which item to select. Similarly for the syntactic ambiguity represented by PP attachment. Once you, as interpreter, have selected one of the options, the interpretation is actually fixed. (This doesn't mean, by the way, that as an interpreter you necessarily do select a particular one of the options, just that you can.) **A brief digression:** Notice that this discussion is in part a discussion about compositionality, and gives us an idea of what a non-compositional account of meaning could look like. The Radical Pragmatic View is a non-compositional view: it allows the information content of a sentence to be fixed by something that has no linguistic reflex.

To help clarify what is meant by compositionality, let me just mention a couple of other ways in which a semantic account could fail to be compositional.

- Suppose your syntactic theory tells you that S has the structure $[a[bc]]$ but your semantics computes the meaning of S by first combining the meanings of a and b and then combining the result with the meaning of c. This is non-compositional.

- Recall the difference between:

1. Jane knows that George was late.

2. Jane believes that George was late.

Sentence 1. entails that George was late; sentence 2. doesn't. We might try to account for this by saying that in the environment of the verb *believe*, a clause doesn't mean what it usually means, but something else instead. Then the clause *that George was late* is assumed to contribute different things to the informational content of different sentences. This is a non-compositional account.

Quantifiers, Scope and Context

- ▷ **Example 1.3.4.** *Every man loves a woman.* (Keira Knightley or his mother!)
- ▷ **Example 1.3.5.** *Every car has a radio.* (only one reading!)
- ▷ **Example 1.3.6.** *Some student in every course sleeps in every class at least some of the time.* (how many readings?)
- ▷ **Example 1.3.7.** *The president of the US is having an affair with an intern.* (2002 or 2000?)
- ▷ **Example 1.3.8.** *Everyone is here.* (who is everyone?)

FAU Michael Kohlhase: Symbolic NLP in GLIF 19 2024-06-27

Observation: If we look at the first sentence, then we see that it has two readings:

1. there is one woman who is loved by every man.
2. for each man there is one woman whom that man loves.

These correspond to distinct situations (or possible worlds) that make the sentence true.
Observation: For the second example we only get one reading: the analogue of 2. The reason for this lies not in the logical structure of the sentence, but in concepts involved. We interpret the meaning of the word *has* as the relation "has as physical part", which in our world carries a certain uniqueness condition: If a is a physical part of b, then it cannot be a physical part of c, unless b is a physical part of c or vice versa. This makes the structurally possible analogue to 1. impossible in our world and we discard it.
Observation: In the examples above, we have seen that (in the worst case), we can have one reading for every ordering of the quantificational phrases in the sentence. So, in the third example, we have four of them, we would get $4! = 24$ readings. It should be clear from introspection that we (humans) do not entertain 12 readings when we understand and process this sentence. Our models should account for such effects as well.
Context and Interpretation: It appears that the last two sentences have different informational content on different occasions of use. Suppose I say *Everyone is here.* at the beginning of class. Then I mean that everyone who is meant to be in the class is here. Suppose I say it later in the day at a meeting; then I mean that everyone who is meant to be at the meeting is here. What shall we say about this? Here are three different kinds of solution:

Radical Semantic View On every occasion of use, the sentence literally means that everyone in the world is here, and so is strictly speaking false. An interpreter recognizes that the speaker has said something false, and uses general principles to figure out what the speaker actually meant.

Radical Pragmatic View What the semantics provides is in some sense incomplete. What the sentence means is determined in part by the context of utterance and the speaker's intentions. The differences in meaning are entirely due to extra-linguistic facts which have no linguistic reflex.

The Intermediate View The logical form of sentences with the quantifier *every* contains a slot for information which is contributed by the context. So extra-linguistic information is required to fix the meaning; but the contribution of this information is mediated by linguistic form.

More Context: Anaphora

▷ **Example 1.3.9 (Anaphoric References).**

- ▷ *John is a bachelor. His wife is very nice.* (Uh, what?, who?)
- ▷ *John likes his dog Spiff even though he bites him sometimes.* (who bites?)
- ▷ *John likes Spiff. Peter does too.* (what to does Peter do?)
- ▷ *John loves his wife. Peter does too.* (whom does Peter love?)
- ▷ n*John loves golf, and Mary too.* (who does what?)

▷ **Definition 1.3.10.** A word or phrase is called anaphoric (or an anaphor), if its interpretation depends upon another phrase in context. In a narrower sense, an anaphor refers to an earlier phrase (its antecedent), while a cataphor to a later one (its postcedent).

The process of determining the antecedent or postcedent of an anaphoric phrase is called anaphor resolution.

Definition 1.3.11. An anaphoric connection between anaphor and its antecedent or postcedent is called direct, iff it can be understood purely syntactically. An anaphoric connection is called indirect or a bridging reference if additional knowledge is needed.

Context is Personal and keeps changing

- ▷ *The king of America is rich.* (true or false?)
- ▷ *The king of America isn't rich.* (false or true?)
- ▷ *If America had a king, the king of America would be rich.* (true or false!)
- ▷ *The king of Buganda is rich.* (Where is Buganda?)
- ▷ *. . . Joe Smith. . . The CEO of Westinghouse announced budget cuts.* (CEO=J.S.!)

Chapter 2

Logic as a Tool for Modeling NL Semantics

In this chapter we will briefly introduce formal logic and motivate how we will use it as a tool for developing precise theories about natural language semantics.

We want to build a compositional, semantic meaning theory based on truth conditions, so that we can directly model the truth conditional synonymy test. We will see how this works in detail in section 2.3 after we have recapped the necessary concepts about logic.

2.1 The Method of Fragments

We will proceed by the "method of fragments", introduced by Richard Montague in [Mon70], where he insists on specifying a complete syntax and semantics for a specified subset ("fragment") of a natural language, rather than writing rules for the a single construction while making implicit assumptions about the rest of the grammar. [Mon70]

> In the present paper I shall accordingly present a precise treatment, culminating in a theory of truth, of a formal language that I believe may be reasonably regarded as a fragment of ordinary English. R. Montague 1970 [Mon70, p.188]

The first step in defining a fragment of natural language is to define which sentences we want to consider. We will do this by means of a context-free grammar. This will do two things: act as an oracle deciding which sentences (of natural language) are OK, and secondly to build up syntax trees, which we will later use for semantics construction.

Natural Language Fragments

- ▷ **Methodological Problem:** How to organize the scientific method for natural language?
- ▷ **Delineation Problem:** What is natural language, e.g. English?
  Which Aspects do we want to study?
- ▷ **Idea:** Formalize a set (NL) sentences we want to study by a grammar
  $\leadsto$ Richard Montague's method of fragments (1972).
- ▷ **Definition 2.1.1.** The language L of a context-free grammar is called a fragment of a natural language N, iff $L \subseteq N$.

▷ **Scientific Fiction:** We can exhaust English with ever-increasing fragments, develop a semantic meaning theory for each.

▷ **Idea:** Use nonterminals to classify NL phrases.

▷ **Definition 2.1.2.** We call a nonterminal symbol of a context-free grammar a phrasal category. We distinguish two kinds of rules:

structural rules: $\mathcal{L}\colon H \rightarrow c_1, \ldots, c_n$ with head H, label $\mathcal{L}$, and a sequence of phrasal categories c_i.

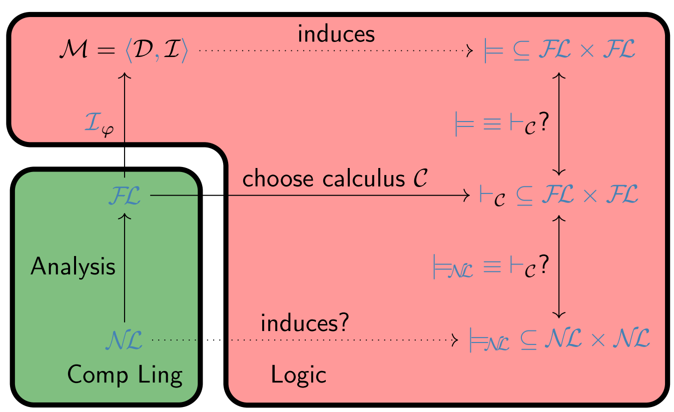
lexical rules: $\mathcal{L}\colon H \rightarrow t_1 \mid \ldots \mid t_n$, where the t_i are terminals (i.e. NL phrases)

▷ **Definition 2.1.3.** In the method of fragments we use a CFG to parse sentences from the fragment into an abstract syntax tree (AST) for further processing.

We generically distinguish two parts of a grammar: the structuralrules and the lexical rules, because they are guided by differing intuitions. The former set of rules govern how NL phrases can be composed to sentences (and later even to discourses). The latter rules are a simple representation of a lexicon, i.e. a structure which tells us about words (the terminal objects of language): their phrasal categories, their meaning, etc.

Formal Natural Language Semantics with Fragments

▷ **Idea:** We will follow the picture we have discussed before



Choose a target logic $\mathcal{FL}$ and specify a translation from syntax trees to formulae!

Semantics by Translation

▷ **Idea:** We translate sentences by translating their syntax trees via tree node translation rules.

▷ **Note:** This makes the induced meaning theory compositional.

▷ **Definition 2.1.4.** We represent a node α in a syntax tree with children $\beta_1, \ldots, \beta_n$

by $[X_{1\beta_1}, \ldots, X_{n\beta_n}]_\alpha$ and write a translation rule as

$$\mathcal{L}\colon [X_{1\beta_1}, \ldots, X_{n\beta_n}]_\alpha \rightsquigarrow \Phi(X_1{}', \ldots, X_n{}')$$

if the translation of the node α can be computed from those of the β_i via a semantical function Φ.

▷ **Definition 2.1.5.** For a natural language utterance A, we will use $\langle \mathrm{A} \rangle$ for the result of translating A.

▷ **Definition 2.1.6 (Default Rule).** For every word w in the fragment we assume a constant w' in the logic $\mathcal{L}$ and the "pseudo-rule" $t1\colon w \rightsquigarrow w'$. (if no other translation rule applies)

2.2 What is Logic?

What is Logic?

▷ **Definition 2.2.1.** Logic $\widehat{=}$ formal languages, inference and their relation with the world

- ▷ Formal language $\mathcal{FL}$: set of formulae ($2+3/7$, $\forall x.x+y=y+x$)
- ▷ Formula: sequence/tree of symbols ($x, y, f, g, p, 1, \pi, \in, \neg, \forall, \exists$)
- ▷ Model: things we understand (e.g. number theory)
- ▷ Interpretation: maps formulae into models ($[\![\text{three plus five}]\!]^{\mathcal{I}} = 8$)
- ▷ Validity: $\mathcal{M}\models\mathbf{A}$, iff $[\![\mathbf{A}]\!]^{\mathcal{I}} = \mathsf{T}$ (five greater three is valid)
- ▷ Entailment: $\mathbf{A}\models\mathbf{B}$, iff $\mathcal{M}\models\mathbf{B}$ for all $\mathcal{M}\models\mathbf{A}$. (generalize to $\mathcal{H}\models\mathbf{A}$)
- ▷ Inference: rules to transform (sets of) formulae ($\mathbf{A}, \mathbf{A}\Rightarrow\mathbf{B}\vdash\mathbf{B}$)
- ▷ Syntax: formulae, inference (just a bunch of symbols)
- ▷ Semantics: models, interpr., validity, entailment (math. structures)

▷ **Important Question:** relation between syntax and semantics?

So logic is the study of formal representations of objects in the real world, and the formal statements that are true about them. The insistence on a *formal language* for representation is actually something that simplifies life for us. Formal languages are something that is actually easier to understand than e.g. natural languages. For instance it is usually decidable, whether a string is a member of a formal language. For natural language this is much more difficult: there is still no program that can reliably say whether a sentence is a grammatical sentence of the English language.

We have already discussed the meaning mappings (under the monicker "semantics"). Meaning mappings can be used in two ways, they can be used to understand a formal language, when we use a mapping into "something we already understand", or they are the mapping that legitimize a representation in a formal language. We understand a formula (a member of a formal language) $\mathbf{A}$ to be a representation of an object $\mathcal{O}$, iff $[\![\mathbf{A}]\!] = \mathcal{O}$.

However, the game of representation only becomes really interesting, if we can do something with the representations. For this, we give ourselves a set of syntactic rules of how to manipulate the formulae to reach new representations or facts about the world.

Consider, for instance, the case of calculating with numbers, a task that has changed from a difficult job for highly paid specialists in Roman times to a task that is now feasible for young children. What is the cause of this dramatic change? Of course the formalized reasoning procedures for arithmetic that we use nowadays. These *calculi* consist of a set of rules that can be followed purely syntactically, but nevertheless manipulate arithmetic expressions in a correct and fruitful way. An essential prerequisite for syntactic manipulation is that the objects are given in a formal language suitable for the problem. For example, the introduction of the decimal system has been instrumental to the simplification of arithmetic mentioned above. When the arithmetical calculi were sufficiently well-understood and in principle a mechanical procedure, and when the art of clock-making was mature enough to design and build mechanical devices of an appropriate kind, the invention of calculating machines for arithmetic by (1623), (1642), and (1671) was only a natural consequence.

We will see that it is not only possible to calculate with numbers, but also with representations of statements about the world (propositions). For this, we will use an extremely simple example; a fragment of propositional logic (we restrict ourselves to only one connective) and a small calculus that gives us a set of rules how to manipulate formulae.

In computational semantics, the picture is slightly more complicated than in Physics. Where Physics considers mathematical models, we build logical models, which in turn employ the term "model". To sort this out, let us briefly recap the components of logics, we have seen so far.

Logics make good (scientific[1]) models for natural language, since they are mathematically precise and relatively simple.

Formal languages simplify natural languages, in that problems of grammaticality no longer arise. Well-formedness can in general be decided by a simple recursive procedure.

Semantic models simplify the real world by concentrating on (but not restricting itself to) mathematically well-understood structures like sets or numbers. The induced semantic notions of validity and logical consequence are precisely defined in terms of semantic models and allow us to make predictions about truth conditions of natural language.

The only missing part is that we can conveniently compute the predictions made by the model. The underlying problem is that the semantic notions like validity and semantic consequence are defined with respect to *all* models, which are difficult to handle.

Therefore, logics typically have a third part, an inference system, or a calculus, which is a syntactic counterpart to the semantic notions. Formally, a calculus is just a set of rules (called inference rules) that transform (sets of) formulae (the assumptions) into other (sets of) formulae (the conclusions). A sequence of rule applications that transform the empty set of assumptions into a formula **T**, is called a proof of **A**. To make these assumptions clear, let us look at a very simple example.

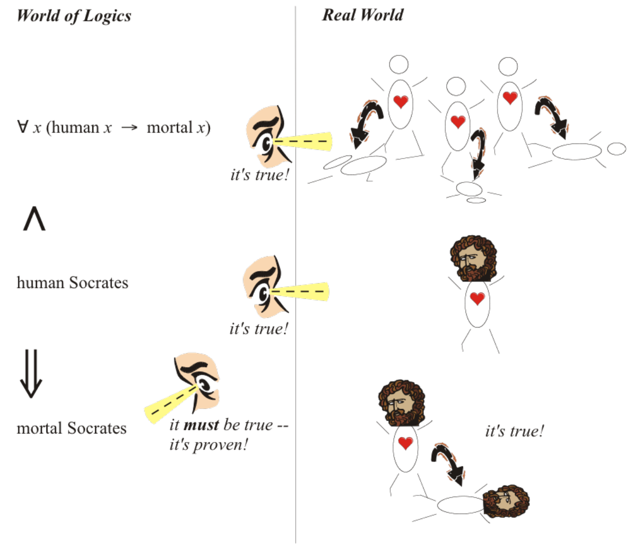
Within the world of logics, one can derive new propositions (the *conclusions*, here: *Socrates is mortal*) from given ones (the *premises*, here: *Every human is mortal* and *Sokrates is human*). Such derivations are *proofs*.

In particular, logics can describe the internal structure of real-life facts; e.g. individual things, actions, properties. A famous example, which is in fact as old as it appears, is illustrated in the slide below.

The Miracle of Logic

▷ Purely formal derivations are true in the real world!

[1]As we use the word "model" in two ways, we will sometimes explicitly label it by the attribute "scientific" to signify that a whole logic is used to model a natural language phenomenon and with the attribute "semantic" for the mathematical structures that are used to give meaning to formal languages



Michael Kohlhase: Symbolic NLP in GLIF 26 2024-06-27

If a formal system is correct, the conclusions one can prove are true (= hold in the real world) whenever the premises are true. This is a miraculous fact (think about it!)

Consequences of the "Miracle of Logics"

- ▷ Inference can be used to draw conclusions and make predictions
- ▷ **Idea:** Write down only the basics and get all consequences for free.
- ▷ **Example 2.2.2 (Mathematics uses this excessively).** For all of number theory we only need five simple assumptions. (e.g. Peano Axioms)
- ▷ We can compute with meanings! ($\leadsto$ build services that exploit meaning)
- ▷ **Slogan:** Get out more than you put in! (using semantics-aware services)

Michael Kohlhase: Symbolic NLP in GLIF 27 2024-06-27

2.3 Using Logic to Model Meaning of Natural Language

Modeling Natural Language Semantics

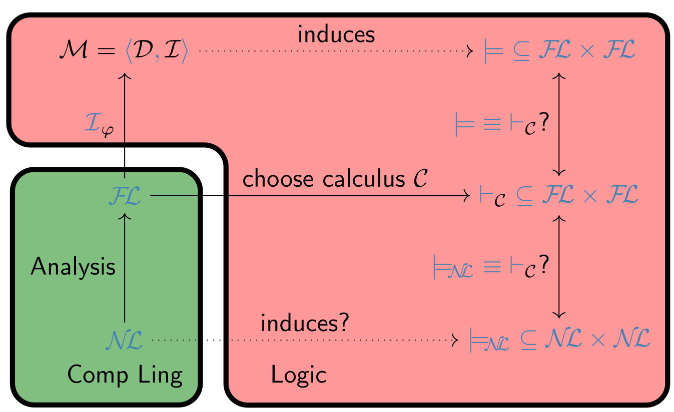
- ▷ **Problem:** Find formal (logic) system for the meaning of natural language.
- ▷ History of ideas
  - ▷ Propositional logic [ancient Greeks like Aristotle]
    * *Every human is mortal*
  - ▷ First-Order Predicate logic [Frege ≤ 1900]
    * *I believe, that my audience already knows this.*

- Modal logic [Lewis18, Kripke65]
  - *A man sleeps. He snores.* $((\exists X.\mathrm{man}(X) \wedge \mathrm{sleeps}(X))) \wedge \mathrm{snores}(X)$
- Various dynamic approaches (e.g. DRT, DPL)
  - *Most men wear black*
- Higher-order Logic, e.g. generalized quantifiers
- ...

FAU Michael Kohlhase: Symbolic NLP in GLIF 28 2024-06-27

Let us now reconcider the role of all of this for natural language semantics. We have claimed that the goal of the course is to provide you with a set of methods to determine the meaning of natural language. If we look back, all we did was to establish translations from natural languages into formal languages like first-order or higher-order logic (and that is all you will find ituisn most semantics papers and textbooks). Now, we have just tried to convince you that these are actually syntactic entities. So, *where is the semantics?*.

Natural Language Semantics?



FAU Michael Kohlhase: Symbolic NLP in GLIF 29 2024-06-27

As we mentioned, the green area is the one generally covered by natural language semantics. In the analysis process, the natural language utterance (viewed here as formulae of a language $\mathcal{NL}$) are translated to a formal language $\mathcal{FL}$ (a set $wff(,)$ of well-formed formulae). We claim that this is all that is needed to recapture the semantics even if this is not immediately obvious at first: Theoretical Logic gives us the missing pieces.

Since $\mathcal{FL}$ is a formal language of a logical system, it comes with a notion of model and an value function $\mathcal{I}_\varphi$ that translates $\mathcal{FL}$ formulae into objects of that model. This induces a notion of logical consequence[2] as explained in **??**. It also comes with a calculus $\mathcal{C}$ acting on $\mathcal{FL}$ formulae, which (if we are lucky) is sound and complete (then the mappings in the upper rectangle commute).

What we are really interested in natural language semantics is the truth conditions and natural consequence relations on natural language utterances, which we have denoted by $\models_{\mathcal{NL}}$. If the calculus $\mathcal{C}$ of the logical system $\langle \mathcal{FL}, \mathcal{K}, \models \rangle$ is adequate (it might be a bit presumptious to say sound and complete), then it is a model of the linguistic entailment relation $\models_{\mathcal{NL}}$. Given that both rectangles in the diagram commute, then we really have a model for truth conditions and logical consequence for text/speech fragments, if we only specify the analysis mapping (the green part) and the calculus.

[2] Relations on a set S are subsets of the Cartesian product of S, so we use $R \subseteq S^n \times S$ to signify that R is a (n-ary) relation on X.

Logic-Based Knowledge Representation for NLP

- Logic (and related formalisms) allow to integrate world knowledge
  - explicitly (gives more understanding than statistical methods)
  - transparently (symbolic methods are monotonic)
  - systematically (we can prove theorems about our systems)
- Signal + World knowledge makes more powerful model
  - Does not preclude the use of statistical methods to guide inference
- Problems with logic-based approaches
  - Where does the world knowledge come from? (Ontology problem)
  - How to guide search induced by log. calculi (combinatorial explosion)

Chapter 3

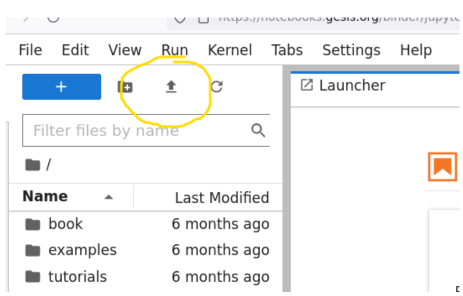
Symbolic Systems for Semantics

In this chapter, we introduce four symbolic systems for dealing with the semantics of languages (both natural and formal); they form the basis of the GLIF system we will be using for modeling natural language semantics in the LBS course. They will be combined to the GLIF (Grammatical Logical, and Inferential Framework) later, when we actually use them on a first natural language fragment.

3.1 Computational & Other Resources for Experimentation

Working with Symbolic Systems in Jupyter

- ▷ We use the GLIF system as an experimentation ground for this course. It comprises
  - ▷ The Grammatical Framework (GF) for syntactic processing
  - ▷ The MMT (Meta-Meta Toolkit) for logic representation and semantics construction
  - ▷ the ELPI system for automation of inference for pragmatics/semantic analysis
- ▷ The easiest way to use GLIF is via jupyter notebooks in the myBinder service.
  - ▷ Go to `https://mybinder.org/v2/gh/jfschaefer/GlifBinder/version2`
  - ▷ wait a little, ⇝ live jupyterLab IDE with the GLIF kernel preloaded
  - ▷ Upload your notebook there (⇝ yellow circle)



- ▷ You can also host GLIF & jupyterLab yourself (we have docker images, ask us)
- ▷ Try this out with `https://github.com/jfschaefer/GLIFkernel/blob/main/notebooks/examples/Epistemic-Question-Answering.ipynb` for sentences like

- *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?*

3.2 The Grammatical Framework (GF)

In this section we give a hands-on introduction to the GF system, a comprehensive framework for engineering natural language grammars and using them for symbolic machine translation. But before we do that, let us recap the basics of context-free grammars. GF grammars are slightly stronger, but most of intuitions still apply.

3.2.1 Recap: (Context-Free) Grammars

Phrase Structure Grammars (Motivation)

- **Problem Recap:** We do not have enough text data to build word sequence language models ⇜ data sparsity.
- **Idea:** Categorize words into classes and then generalize "acceptable word sequences" into "acceptable word class sequences" ⇝ phrase structure grammars.
- **Advantage:** We can get by with much less information.
- **Example 3.2.1 (Generative Capacity).** 10^3 structural rules over a lexicon of 10^5 words generate most German sentences.
- Vervet monkeys, antelopes etc. use isolated symbols for sentences.
  ⇝ restricted set of communicable propositions, no generative capacity.
- **Disadvantage:** Grammars may over generalize or under generalize.
- The formal study of grammars was introduced by Noam Chomsky in 1957 [Cho65].

We fortify our intuition about these – admittedly very abstract – constructions by an example and introduce some more vocabulary.

Phrase Structure Grammars (cont.)

- **Example 3.2.2.** A simple phrase structure grammar G:

$$\begin{array}{rcl} S & \to & NP\ Vi \\ NP & \to & Article\ N \\ Article & \to & \textbf{the} \mid \textbf{a} \mid \textbf{an} \\ N & \to & \textbf{dog} \mid \textbf{teacher} \mid \ldots \\ Vi & \to & \textbf{sleeps} \mid \textbf{smells} \mid \ldots \end{array}$$

  Here S, is the start symbol, NP, VP, $Article$, N, and Vi are nonterminals.

▷ **Definition 3.2.3.** The subset of lexical rules, i.e. those whose body consists of a single terminal is called its lexicon and the set of body symbols the vocabulary (or alphabet). The nonterminals in their heads are called lexical categories.

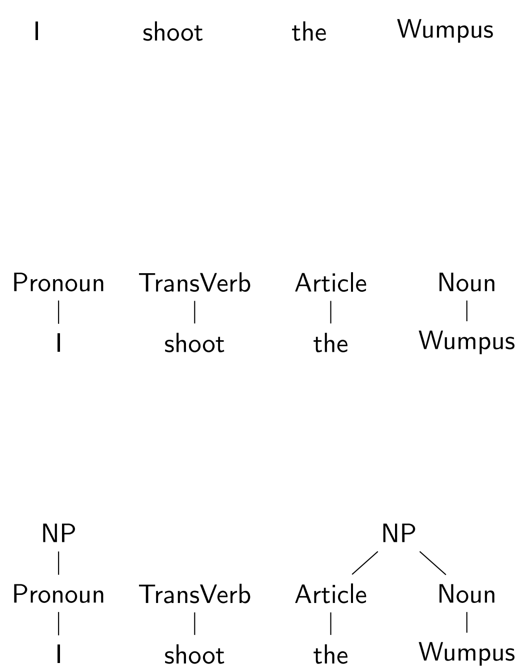
▷ **Definition 3.2.4.** The non-lexicon production rules are called structural, and the nonterminals in the heads are called phrasal categories.

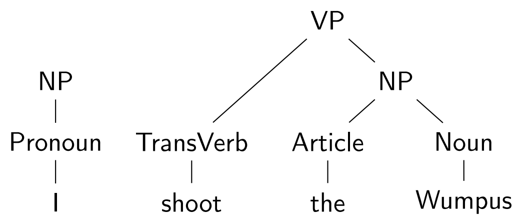
Context-Free Parsing

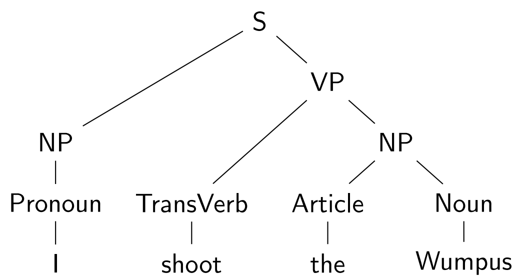
▷ **Recall:** The sentences accepted by a grammar are defined "top-down" as those the start symbol can be rewritten into.

▷ **Definition 3.2.5.** Bottom up parsing works by replacing any substring that matches the body of a production rule with its head.

▷ **Example 3.2.6.** Using the Wumpus grammar (below), we get the following parse trees in bottom up parsing:







Traditional linear notation: Also write this as:

$[S[NP[Pronoun\ \mathbf{I}]][VP[TransVerb\ \mathbf{shoot}][NP[Article\ \mathbf{the}][Noun\ \mathbf{Wumpus}]]]]$

▷ Bottom up parsing algorithms tend to be more efficient than top-down ones.

Context-Free Parsing

▷ Bottom up parsing algorithms tend to be more efficient than top-down ones.

▷ Efficient context-free parsing algorithms run in $\mathcal{O}(n^3)$, run at several thousand words/second for real grammars.

▷ **Theorem 3.2.7.** *Context-free parsing $\widehat{=}$ Boolean matrix multiplication!*

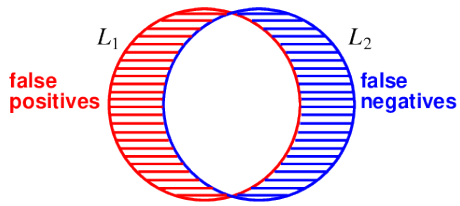
▷ $\rightsquigarrow$ unlikely to find faster practical algorithms. (details in [Lee02])

We now come to a problem that is common to all natural languages: grammaticality is not easily formalized by grammars – even though we know a lot about their syntactic structure, the set of sentences perceived as grammatical by native speakers is not sufficiently regular to be described by a small set of rules.

Grammaticality Judgments

▷ **Problem:** The formal language $\mathbf{L}(G)$ accepted by a grammar G may differ from the natural language L_n it supposedly models.

▷ **Definition 3.2.8.** We say that a grammar G over-generates, iff it accepts strings outside of L_n (false positives) and under-generates, iff there are L_n strings (false negatives) that $\mathbf{L}(G)$ does not accept.



▷ Adjusting $\mathbf{L}(G)$ to agree with L_n is an inductive learning problem!

- ▷ * *the gold grab the wumpus*
- ▷ * *I smell the wumpus the gold*
- ▷ *I give the wumpus the gold*
- ▷ * *I donate the wumpus the gold*

▷ Intersubjective agreement somewhat reliable, independent of semantics!

▷ Real grammars (100–5000 rules) are insufficient even for "proper" English.

3.2.2 A first GF Grammar

We now introduce the general setup of GF grammars by a very simple toy example and characterize two types of grammars by their intent.

The Grammatical Framework (GF)

▷ **Definition 3.2.9.** Grammatical Framework (GF [Ran04; Ran11]) is a modular formal framework and functional programming language for writing multilingual grammars of natural languages.

▷ **Definition 3.2.10.** GF comes with the GF Resource Grammar Library, a reusable library for dealing with the morphology and syntax of a growing number of natural languages. (currently > 30)

▷ **Definition 3.2.11.** A GF grammar consists of

- ▷ an abstract grammar that specifies well-formed abstract syntax trees (AST),
- ▷ a collection of concrete grammars for natural languages that specify how ASTs can be linearized into (natural language) strings.

▷ **Definition 3.2.12.** Parsing is the dual to linearization, it transforms NL utterances into abstract syntax trees.

▷ **Definition 3.2.13.** The Grammatical Framwork comes with an implementation; the GF system that implements parsing, linearization, and by combination machine translation. (download/install from [GF])

To introduce the syntax and operations of the GF system, and the underlying concepts, we will look at a very simple example.

Hello World Example for GF (Syntactic)

- **Example 3.2.14 (A Hello World Grammar).**

abstract	concrete

<pre>**abstract** zero = {
  flags startcat=O;
  cat
    S ; NP ; V2 ;
  fun
    spo : V2 −> NP −> NP −> S ;
    John, Mary : NP ;
    Love : V2 ;
}</pre>
	<pre>**concrete** zeroEng **of** zero = {
	  lincat
	    S, NP, V2 = Str ;
	  lin
	    spo vp s o = s ++ vp ++ o;
	    John = *"John"* ;
	    Mary = *"Mary"* ;
	    Love = *"loves"* ;
	}</pre>

- Make a French grammar with John="Jean"; Mary="Marie"; Love="aime";
- Parse a sentence in GF: parse "John loves Mary" ⇝ Love John Mary
- Linearize in GF: linearize Love John Mary ⇝ John loves Mary
- translate in GF: parse −lang=Eng "John Loves Mary" | linearize −lang=Fre
- generate random sentences to test:
  generate_random −number=10 | linearize −lang=Fre ⇝ Jean aime Marie

The GF system can be downloaded from [GF] and can be started from the command line or as an inferior process of a text editor. Grammars are loaded via import or short i. Then the GF commands above can be issued to the shell.
Command sequences can also be combined into an GF script, a text file with one command per line that can be loaded into GF at startup to initialize the interpreter by running it as gf −−run script.gfo.
In standard accounts of the NLU waterfall or the method of fragments, parsing of natural language utterances into syntax trees is followed by a translation into a logical representation. One way of implementing this is to linearize the syntax tree into the input language of an implementation of a logic and read them into the system for further processing. We will now explore this using a Prolog interpreter, in which it is easy to program inference procedures.

Translation to Logic

- **Idea:** Use logic as a "natural language" (to translate into)
- **Example 3.2.15 (Hello Prolog).** Linearize to Prolog terms:

```
concrete zeroPro of zero = {
  lincat
    S , NP , V2 = Str;
  lin
    spo = \vt,subj,obj -> vt ++ "(" ++ subj ++ "," ++ obj ++ ").";
    John = "john";
    Mary = "mary";
    Love = "loves";
  }
```

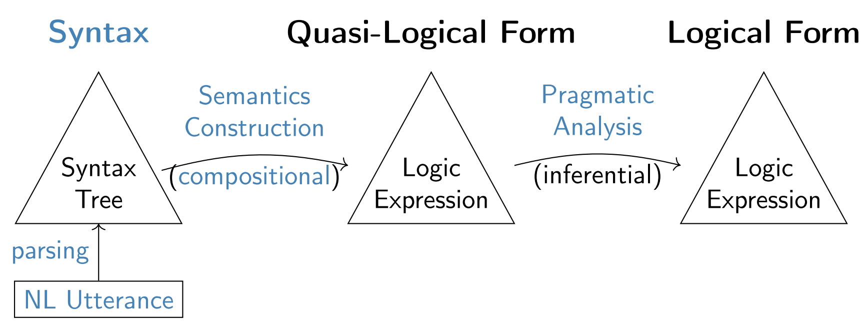
▷ **Linearization in GF:** linearize Love John Mary ⇝ loves (john , mary)

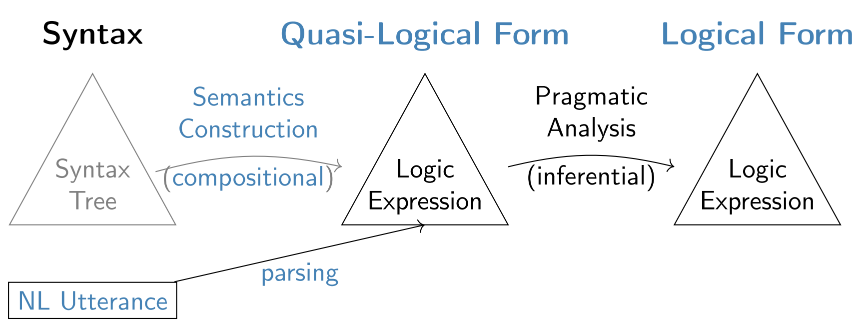
▷ **Note:** loves (john , mary) is *not* a quasi-logical forms, but a Prolog term that can be read into an Prolog interpreter for pragmatic analysis.

We will now introduce an important conceptual distinction on the intent of grammars.

Syntactic and Semantic Grammars

▷ Recall our interpretation pipeline





▷ **Definition 3.2.16.** We call a grammar syntactic, iff the categories and constructors are motivated by the syntactic structure of the utterance, and semantic, iff they are motivated by the structure of the domain to be modeled.

▷ Grammar zero from Example 3.1.14 is syntactic.

▷ We will look at semantic versions next.

Hello World Example for GF (semantic)

▷ A semantic Hello World Grammar

abstract one = {
  flags startcat = O;
  cat
    I; –– Individuals
    O; –– Statements
  fun
    John, Mary : I;
    Love : I –> I –> O;
}
	concrete oneEng **of** one = {
	  lincat
	    I = Str ;
	    O = Str ;
	  lin
	    John = *"John"*;
	    Mary = *"Mary"*;
	    Love s o = s ++ *"loves"* ++ o;
	  }

▷ Instead of the "syntactic categories" S (sentence), NP (noun phrase), and V2 (transitive verb), we now have the semantic categories I (individual) and O (proposition).

3.2.3 Inflection and Case in GF

We now extend the toy grammars from the last subsection with facilities for inflection and case. Here we start to see the strenghts of a framework like GF: it provides representational primitves that allow to do so with minimal pain. We use German – which has more inflection and cases than English – as an example.
We first set up the example and test it for English

Towards Complex Linearizations (Setup/English)

▷ Extending our hello world grammar (the trivial bit) We add the determiner *the* as an operator that turns a noun (N) into a noun phrase (NP)

abstract two = {
  flags startcat=O;
  cat
    S ; NP ; V2 ; N;
  fun
    spo : V2 –> NP –> NP –> S ;
    John, Mary : NP ;
    Love : V2 ;
    dog, mouse : N;
    the : N –> NP ;
}
	concrete twoEN **of** two = {
	  lincat
	    S, NP, V2, N = Str ;
	  lin
	    spo vp s o = s ++ vp ++ o;
	    John = *"John"* ;
	    Mary = *"Mary"* ;
	    Love = *"loves"* ;
	    dog = *"dog"* ;
	    mouse = *"mouse"* ;
	    the x = *"the"* ++ x;
	}

▷ **Idea:** A noun phrase is a phrase that can be used wherever a proper name can be used.

Now we test it with a German concrete grammar:

Towards Complex Linearizations (German)

- We try the same for German

```
abstract two = {
  flags startcat=O;
  cat
    S ; NP ; V2 ; N;
  fun
    spo : V2 -> NP -> NP -> S ;
    John, Mary : NP ;
    Love : V2 ;
    dog, mouse : N;
    the : N -> NP ;
}
```

```
concrete twoDE0 of two = {
  lincat S, NP, V2, N = Str ;
  lin
    spo vp s o = s ++ vp ++ o;
    John = "Johann" ;
    Mary = "Maria" ;
    Love = "liebt" ;
    dog = "Hund" ;
    mouse = "Maus" ;
    the x = "der" ++ x;
}
```

- Let us test-drive this; as expected we obtain

```
two> l -lang=DE0 spo Love John (the dog)
Johann liebt der Hund
```

- **Problem:** *Johann liebt der Hund* is not grammatical in German
  ⇝ We need to take (grammatical) gender into account to obtain the correct form *den* of the determiner.

Adding Gender

- To add gender, we add a parameter and extend the type N to a record

```
concrete twoDE1 of two = {
  param
    Gender = masc | fem | neut;
  lincat
    S, V2, NP = Str ;
    N = {s : Str; gender : Gender};
  lin
    spo vp s o = s ++ vp ++ o;
    John = "Johann" ;
    Mary = "Maria" ;
    Love = "liebt" ;
    dog = {s = "Hund"; gender = masc} ;
    mouse = {s = "Maus" ; gender = fem} ;
    the x = case x.gender of {masc => "der" ++ x.s;
                              fem => "die" ++ x.s;
                              neut => "das" ++ x.s} ;
}
```

Adding Gender

▷ Let us test-drive this; as expected we obtain

```
two> l —lang=DE1 spo Love (the mouse) Mary
Die Maus liebt Maria.
two> l —lang=DE1 spo Love Mary (the dog)
Maria liebt der Hund.
```

▷ We need to take into account case in German too.

Adding Case

▷ To add case, we add a parameter, reinterpret type NP as a case-dependent table of forms.

```
concrete twoDE2 of two = {
  param
    Gender = masc | fem | neut;
    Case = nom | acc;
  lincat
    S, V2 = {s: Str} ;
    N = {s : Str; gender : Gender};
    NP = {s : Case => Str};
```


Adding Case

▷

```
lin
  spo vp subj obj = {s = subj.s!nom ++ vp.s ++ obj.s!acc};
  John = {s = table {nom => "Johann"; acc => "Johann"}};
  Mary = {s = table {nom => "Maria"; acc => "Maria"}};
  Love = {s = "liebt"} ;
  dog = {s = "Hund"; gender = masc} ;
  mouse = {s = "Maus" ; gender = fem} ;
  the x = {s = table
             { nom => case x.gender of {masc => "der" ++ x.s;
                                        fem => "die" ++ x.s;
                                        neut => "das" ++ x.s};
               acc => case x.gender of {masc => "den" ++ x.s;
                                        fem => "die" ++ x.s;
                                        neut => "das" ++ x.s}}};}
```

▷ Let us test-drive this; as expected we obtain

```
two> l —lang=DE2 spo Love Mary (the dog)
Maria liebt den Hund.
```

Adding Operations (reusable components)

▷ We add operations (functions with $\lambda \widehat{=}$) to get the final form.

```
concrete twoDE of two = {
  param
    Gender = masc | fem | neut;
    Case = nom | acc;
  oper
    Noun : Type = {s : Str; gender : Gender};

    mkPN : Str −> NP = \x −> lin NP {s = table {nom => x; acc => x}};
    mkV2 : Str −> V2 = \x −> lin V2 {s = x};
    mkN : Str −> Gender −> Noun = \x,g −> {s = x; gender = g};
    mkXXX : Str −> Str −> Str −> Noun −> Str =
        \ma,fe,ne,noun −> case noun.gender of {masc => ma ++ noun.s;
                                               fem => fe ++ noun.s;
                                               neut => ne ++ noun.s};
```


Adding Operations (reusable components)

▷
```
  lincat
    S, V2 = {s : Str};
    N = Noun;
    NP = {s: Case => Str};
  lin
    spo vp subj obj = {s = subj.s!nom ++ vp.s ++ obj.s!acc};
    John = mkPN "Johannes";
    Mary = mkPN "Maria";
    Love = mkV2 "liebt";
    dog = mkN "Hund" masc;
    mouse = mkN "Maus" fem;
    the n = {s = table { nom => mkXXX "der" "die" "das" n;
                         acc => mkXXX "den" "die" "das" n}
    };
}
```


3.3 MMT: A Modular Framework for Representing Logics and Domains

In **??** we have identified truth conditions as the main tool for establishing semantic meaning theories for natural language.

In the LBS course, we want to make the establishment of meaning theories machine-supported. To do this we need to have

1. A formal language that allows us to to describe situations/worlds,
2. an formal system that allows us to compute predictions, and
3. a software system that mechanizes it.

For the first two we will use the MMT language, and for the third the MMT system that implements it.

3.3.1 Propositional Logic in MMT: A first Example

We will now introduce the MMT representation format and the MMT system by going over a simple example very carefully: the syntax and a proof theory for propositional logic. Even though the formal system itself is quite simple, it already teaches us many of the basic ideas and tricks of meta-logical representation of formal systems in LF.

Implementing minimal PL^0 in MMT

- **Recall:** The language $\mathit{wff}_0(\Sigma_0)$ of propositional logic (PL^0) consists of propositions built from propositional variables from $\mathcal{V}_0$ and connectives from Σ_0.
- We model $\mathit{wff}_0(\Sigma_0)$ in a MMT theory ($\Sigma_0 := \{\neg, \wedge\}$ for the moment)

```
theory proplogMinimal : ur:?LF =
▮
```

  - theory is the MMT keyword for modules, the module delimiter ▮ delimits them.
  - A theory has a local name and a meta-theory (after the :)
    Here it is LF (provides the logical constants $\to$, *type*, λ, Π)
- MMT theories contain declarations of the form ⟪name⟫ : ⟪type⟫ | # ⟪notation⟫
  - declarations are delimited by the declaration delimiter ❙,
  - declaration components by the object delimiter | .
- **Example 3.3.1.** A declaration for the type of propositions

```
prop : type | # o ❙
```

  - the local name prop is the system identifier
  - the type type declares prop to be a type (optional part)
  - the notation definition o declares the notation for prop (can be used instead) (optional part)

Implementing minimal PL^0 in MMT (continued)

- **Example 3.3.2.** Declarations for the connectives $\neg$ and $\wedge$

```
not : o → o | # ¬ 1 prec 100 ❙
```

  - the type o → o declares the constant not to be a unary function
  - the notation definition ¬ 1 prec 100 establishes
    - the function symbol ¬ for not followed by argument 1.
    - brackets are governed by the precedence 100 (binding strength)

```
and : o → o → o | # 1 ∧ 2 prec 90❙
```

- ▷ The type o → o → o declares the constant and to be a binary function (note currying)
- ▷ the notation definition # 1 ∧ 2 prec 90 establishes
  - ▷ the infix function symbol ∧ for and preceded by argument 1 and followed by 2,
  - ▷ brackets are governed by the precedence 90 (weaker than for not)
- ▷ **Testing precedences:** the MMT system accepts A : o | test : ¬ A ∧ A | And ¬ A ∧ A is parsed as (¬ A) ∧ A instead of ¬ (A ∧ A)

▷ **All together now! PL^0 Syntax as a Mmt theory:**

```
theory proplogMinimal : ur:?LF =
    prop : type | # o |
    not : o → o | # ¬ 1 prec 100 |
    and : o → o → o | # 1 ∧ 2 prec 90|
|
```


Completing PL^0 by Definitions

▷ Building on this, we can define additional connectives: ∨, ⇒, ⇔

```
theory proplog : ur:?LF =
    include ?proplogMinimal |
    or : o → o → o | # 1 ∨ 2 prec 80 | = [a:o,b:o] ¬ (¬ a ∧ ¬ b) |
    implies : o → o → o | # 1 ⇒ 2 prec 70 | = [a:o,b:o] ¬ a ∨ b |
|
```

- ▷ include is the keyword for an inclusion declaration
  here we include the theory proplogMinimal (notation: theory refs prefixed by ?)
  this makes all of its declarations available locally in theory proplog.
- ▷ new declaration components: definientia give a constant meaning by replacement.
- ▷ [a:o,b:o] ¬ a ∨ b is the MMT notation for $\lambda a_o b_o.\neg a \vee b$, i.e. the function that given two propositions a and b returns the proposition $\neg a \vee b$.
- ▷ **Note**: types optional in lambdas (MMT system infers them from context)

▷ This completes the syntax (language of formulae) of PL^0.

▷ **Observation:** The declarations in proplog amount to a context-free grammar of PL^0.

Describing Situations for Truth Conditions

▷ We want to derive the truth conditions e.g. for *Peter loves Mary*.

▷ **Definition 3.3.3.** A situation theory is an MMT theory that formalizes a situation.

▷ **First Attempt:** We provide declarations for the individuals and their relations.

```
theory world1 : ur:?LF =
    include ?proplog ❚

    individual : type ❘ # ι ❚
    peter : ι ❚
    mary : ι ❚
    loves : ι → ι → o ❚

    plm = loves peter mary ❚ // just an abbreviation ❚
❚
```

▷ **Problem:** We have not asserted that plm is true in world1, . . .
. . . only that the proposition plm exists.

▷ **Idea:** Let's assert that plm is "provable" in theory world1.

Asserting Truth by Declaring Provability in MMT Theories

▷ **Observation:** We can only assert existance in a theory by declarations.

▷ **Idea 1:** Use declarations to declare certain types to be inhabited $\widehat{=}$ non-empty.

▷ **Idea 2:** A proposition A is "provable", iff the "type of all proofs of A" is inhabited.

▷ **Idea 3:** We can express "the type of all proofs of A" as $\vdash A$
if we declare a suitable type constructor in MMT:

```
ded : prop → type ❘ # ⊢1 ❚
```

▷ **All Together Now:** We can assert that *Peter loves Mary* in theory world1

```
plm_axiom : ⊢plm ❚ // the type of proofs of plm is inhabited❚
```

Note that in this interpretation the constant plm_axiom is a "proof of plm"

▷ **Definition 3.3.4.** This way of representing axioms (and eventually theorems) is called the propositions as types paradigm.

Asserting Truth in MMT theories (continued)

▷ We can make world1 happier by asserting *Mary loves Peter*.

```
    mlp = loves mary peter ❚
    mlp_axiom : ⊢mlp ❚
```

▷ Do *Peter and Mary love each other* in world1?

▷ We would have to have a proof of plm ∧ mlp, which we don't.

▷ **Observation:** There should be one, given that we have proofs for plm and mlp!

▷ **Observation:** We need a proof constructor – a function constant that constructs a proof of plm ∧ mlp from those.

▷ **Idea:** Let's just declare one: pc : ⊢plm →⊢mlp →⊢plm ∧ mlp

▷ We can generalize this to the inference rule of conjunction introduction

```
conjI : {A:o,B:o} ⊢A →⊢B →⊢A∧ B
```

{A:o,B:o} is the MMT notation for Π from LF. (dependent type constructor)
Read as "for arbitrary but fixed propositions A and B..." ...

$$\frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \mathcal{ND}_0 \wedge I$$

▷ **Idea:** This leads to a MMT formalization of the propositional natural deduction calculus $\mathcal{ND}_0$. (up next)

Propositional Natural Deduction

▷ **Observation:** With the ideas discussed above we can do almost all of the inference rules of $\mathcal{ND}_0$.

▷ **Let's start small** with $\Sigma_0 = \{\neg, \wedge\}$: here are the rules again.

Introduction

$$\frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \mathcal{ND}_0 \wedge I$$

$$\frac{\begin{array}{cc} [\mathbf{A}]^1 & [\mathbf{A}]^1 \\ \vdots & \vdots \\ \mathbf{C} & \neg\mathbf{C} \end{array}}{\neg\mathbf{A}} \mathcal{ND}_0 \neg I^1$$

Elimination

$$\frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{A}} \mathcal{ND}_0 \wedge E_l \qquad \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{B}} \mathcal{ND}_0 \wedge E_r$$

$$\frac{\neg\neg\mathbf{A}}{\mathbf{A}} \mathcal{ND}_0 \neg E$$

▷ The start of an MMT theory:

```
theory proplog−ND : ur:?LF =
    include ?proplogMinimal
    ded : prop → type | # ⊢1
    conjI : {A:o,B:o} ⊢A →⊢B →⊢A∧ B
    conjEl : {A:o,B:o} ⊢A∧ B →⊢A
    conjEr : {A:o,B:o} ⊢A∧ B →⊢B
    negE : {A:o} ⊢¬ ¬ A →⊢A
```


Local Hypotheses in Natural Deduction

▷ For $\mathcal{ND}_0\neg I$ we need a new idea for the representation of the local hypothesis $\mathbf{A}$.
A subproof P with a local hypothesis $[\mathbf{A}]$ allows to plug in a proof of $\mathbf{A}$ and complete it P to a full proof for $\mathbf{C}$.
Idea: Represent this as a function from $\vdash \mathbf{A}$ to $\vdash \mathbf{C}$.

$$\dfrac{\begin{array}{c}[\mathbf{A}]^1\\ \vdots\\ \mathbf{C}\end{array} \qquad \begin{array}{c}[\mathbf{A}]^1\\ \vdots\\ \neg\mathbf{C}\end{array}}{\neg\mathbf{A}}$$

▷ In MMT we have:

```
negI : {A:o,C:o} (⊢A →⊢C) →(⊢A →⊢¬ C) →⊢¬ A
```

$\mathcal{ND}_0\neg I^1$ takes proof transformers as arguments and returns a proof of $\neg\mathbf{A}$.

▷ With this idea, we can do the rest of the inference rules of $\mathcal{ND}_0$, e.g.

```
implI: {a,b} (⊢a →⊢b) →⊢(a⇒ b)
```


Writing Proofs in MMT

▷ **Recap:** In MMT, we can write axioms as declarations c : ⊢a using the propositions as types paradigm: the proof type ⊢a must be inhabited, since it has the proof c of a as an inhabitant.

▷ **Observation:** This can be extended to theorems, by giving denfinientia:
A declaration c : ⊢a | = Φ also ensures that ⊢a is inhabited, but using already existing material Φ.

▷ **Example 3.3.5.** Let's try this on the well-known $\mathcal{ND}_0$ proof

$$\dfrac{\dfrac{\dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\mathcal{ND}_0\wedge E_r \quad \dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\mathcal{ND}_0\wedge E_l}{\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\wedge I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\Rightarrow I^1$$

```
ac : {a,b} ⊢((a∧ b)⇒ (b∧ a))
   = [a, b] ([p:⊢(a∧ b)] (p andEr) (p andEl) andI) implI
```


Writing Proofs in MMT (step by step)

▷ **Example 3.3.6 (Continued).**

$$\dfrac{\dfrac{\dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\mathcal{ND}_0\wedge E_r \quad \dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\mathcal{ND}_0\wedge E_l}{\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\wedge I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\Rightarrow I^1$$

```
ac : {a,b} ⊢((a∧ b)⇒ (b∧ a))   1
  = [a, b] ([p:⊢(a∧ b)]        2
               (p andEr)       3
               (p andEl)       4
               andI)           5
             implI             6
```

- ▷ Line 1: name and type (optional)
- ▷ Line 2: λ-abstraction [a,b] corresponding to Π-abstraction {a,b}
- ▷ Line 6: the proof is constructed by impl with one argument (a subproof Ψ)
  - ▷ **But remember**: implI: {a,b} (⊢a →⊢b) →⊢(a⇒ b) takes three!
  - ▷ **Idea**: add special postfix notation definition | # 3 impl ($3 \mapsto \Psi$)
  - ▷ **Justification**: The MMT system can reconstruct implicit arguments
- ▷ Lines 2-5: Subproof Ψ with local hyp. $[a \wedge b]^1$, represented as λp-term in Line 4
  Idea: the (informal) function of the co-indexing is formalized by λ-abstraction
- ▷ Line 5: result of Ψ constructed by andI – notation definition | # 3 4 andI
- ▷ Line 3/4: two subproofs constructed from p by andEl/andEr.

▷ **Observation 1:** The postfix notations make the MMT proof term similar!

▷ **Observation 2:** But writing them is very tedious and complex still.

Modular Representation in MMT

▷ **Recall:** We said that for PL^0, it does not matter if $\Sigma_0 = \{\neg, \wedge\}$ or $\Sigma_0 = \{\neg, \vee\}$.

▷ **In particular** we can always inter-define $\wedge$ and $\vee$ via de-Morgan.

▷ Let's make this formal using views.

▷ **Example 3.3.7.** A modular development of the two variants of PL^0

```
theory dednot : ur:?LF =
    prop : type | # o
    ded : o → type | # ⊢1
    not : o → o | # ¬ 1
```

```
theory notand : ur:?LF =
    include ?dednot
    and : o → o → o | # 1 ∧ 2
    andI : {a,b} ⊢a →⊢b →⊢(a∧ b)

view and2or : ?notand −> ?notor =
    and = [a,b] ¬ ((¬ a) ∨ (¬ b))
    andI = Φ
```

```
theory notor : ur:?LF =
    include ?dednot
    or : o → o → o | # 1 ∨ 2
    orIl : {a,b} ⊢a →⊢(a∨ b)
    orIr : {a,b} ⊢b →⊢(a∨ b)

view or2and : ?notor −> ?notand =
    or = [a,b] ¬ ((¬ a) ∧ (¬ b))
    andI = Ψ
```

For some suitable proof expressions Φ and Ψ.

3.4 Fragment 1: The Grammatical Logical Framework

Now that we have introduced the "Method of Fragments" in theory, let see how we can implement it in a contemporary grammatical and logical framework. For the implementation of the semantics construction, we use GF, the "grammatical framework". For the implementation of the logic we will use the MMT system.

In this section we develop and implement a toy/tutorial language fragment chosen mostly for didactical reasons to introduce the two systems. The code for all the examples can be found at `https://gl.mathhub.info/Teaching/LBS/tree/master/source/tutorial`.

3.4.1 Implementing Fragment 1 in GF

Implementing Fragment 1 in GF

▷ The grammar of Fragment 1 only differs trivially from Hello World grammar two.gf from slide ??.

- ▷ Verbs: $V^t \mathrel{\hat{=}}$ V2, $V^i \mathrel{\hat{=}}$ **cat** V; **fun** sp : NP −> V −> S;
- ▷ Negation: **fun** not : S −> S; **lin** not a = mkS (*"it is not the case that"*++ a.s);
- ▷ the: **fun** the : N −> NP; **lin** the n = mkNP (*"the"*++ n.s);
- ▷ conjunction: **fun** and : S −> S −> S; **lin** and a b = mkS (a.s ++ *"and"*++ b.s);

3.4.2 Implementing Fragment1 in GF and MMT

Discourse Domain Theories for $\mathcal{F}_1$ (Lexicon)

▷ A "lexicon theory" (only selected constants here)

```
theory plnqFrag1 : ?plnq =
    ethel : ι | # ethel'
    prudence : ι | # prudence'
    dog : ι | # dog'
    poison : ι → ι → o | # poison' 1 2
    laugh : ι → o | # laugh' 1
```

declares one logical constant for each from abstract GF grammar.

▷ Enough to interpret *Prudence poisoned the dog and Ethel laughed* from above.

```
    ex : | o = poison' prudence' dog' ∧ laugh' ethel'
```


Representing Multiple Readings

▷ We can even represent the three readings of *John chased the gangster in the red sports car* from ??.

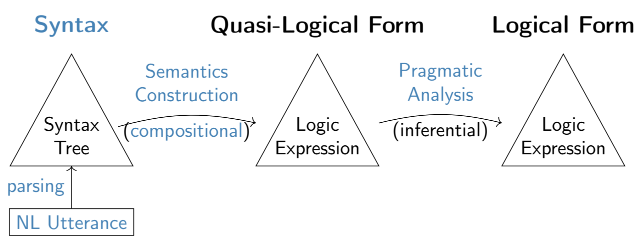
```
theory sportscar : ?plnq =
    john : ι | gangster : ι | sportscar : ι | red : ι → o |
    chased : ι → ι → o | in : ι → ι → o |
    jcgirs1 : o | = chased john gangster ∧ in sportscar gangster ∧ red sportscar
    jcgirs2 : o | = chased john gangster ∧ in sportscar john ∧ red sportscar
    jcgirs3 : o | = chased john gangster ∧ in sportscar john ∧
                        in sportscar gangster ∧ red sportscar
```

- **Problem:** Can we systematically generate terms like jcgirs1, jcgirs2, and jcgirs3?
- **Idea:** Use the ASTs from GF in MMT.

Michael Kohlhase: Symbolic NLP in GLIF 63 2024-06-27

Embedding GF into MMT

- **Observation:** The GF system provides `Java` bindings and MMT is programed in `Scala`, which compiles into the `Java` virtual machine.
- **Idea:** Use GF as a sophisticated NL-parser/generator for MMT
  - ⇝ MMT with a natural language front-end.
  - ⇝ GF with a multi-logic back-end
- **Definition 3.4.1.** The MMT integration mapping interprets GF abstract syntax trees as MMT terms.
- **Observation:** This fits very well with our interpretation process in LBS



- **Implementation:** transform GF system (`Java`) data structures to MMT (`Scala`) ones in MMT.

Michael Kohlhase: Symbolic NLP in GLIF 64 2024-06-27

GF Abstract syntax trees as MMT Terms

- **Idea:** Make the MMT integration mapping (essentially) the identity.
- **Prerequisite:** MMT theory isomorphic to GF grammar (declarations aligned)
- **Recall:** ASTs in GF are essentially terms.
- **Indeed:** GF abstract grammars are essentially MMT theories.
- **Example 3.4.2.** Syntactic categories of $\mathcal{F}_1$ (Syntactic categories $\widehat{=}$ types)

```
theory Frag1CatMMT : ur:?LF =
    S : type ❘
    Conj : type ❘
    NP : type ❘
    Npr : type ❘
```

```
    N : type ❘
    Vi : type ❘
    Vt : type ❘
❙
```

The $\mathcal{F}_1$ lexicon (words $\widehat{=}$ constants)

```
theory Frag1LexMMT : ur:?LF =
    include ? Frag1CatMMT
    ethel : Npr ❘
    prudence : Npr ❘
    dog : N ❘
    poison : Vt ❘
    laugh : Vi ❘
    and : Conj ❘
❙
```

The structural rules of $\mathcal{F}_1$ (functions $\widehat{=}$ functions)

```
theory Frag1RulesMMT : ur:?LF =
    include ? Frag1CatMMT
    s1 : NP → Vi → S ❘
    s2 : NP → Vt → NP → S ❘
    n1 : Npr → NP ❘
    n2 : N → NP ❘
    s3 : S → S ❘
    s4 : S → Conj → S → S ❘
    s5 : NP → NP → S ❘
    s6 : NP → Adj → S ❘
❙
```

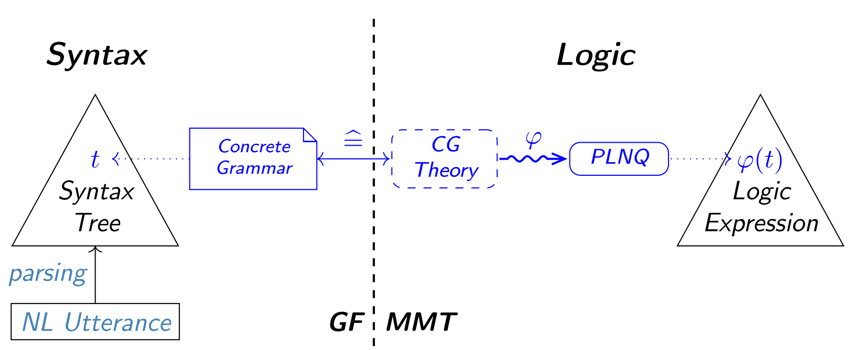
putting it all together

```
theory Frag1LexMMT : ur:?LF =
    include ? Frag1LexMMT
    include ? Frag1RulesMMT
❙
```

▷ **Observation:** GF grammars and MMT theories best when organized modularly.

Semantics Construction as an MMT View

▷ **Observation 3.4.3.** *We can express semantics construction as an* MMT *view*



▷ **Example 3.4.4.** Syntactic categories ⇝ $\mathrm{PL}^{\mathrm{nq}}$ types

```
view Frag1CatSem : ?Frag1CatMMT —> ?plnqFrag1 =
    S = o ❙
    NP = ι ❙
    Vi = ι → o ❙
    Vt = ι → ι → o ❙
    Npr = ι ❙
    N = ι ❙
    Conj = o → o → o ❙
❚
```

Lexicon ⇝ mapping into $\mathrm{PL}^{\mathrm{nq}}$ terms

```
view Frag1LexSem : ?Frag1CatMMT —> ?plnqFrag1 =
    include ?Frag1CatSem
    ethel = ethel' ❙
    prudence = prudence' ❙
    dog = dog' ❙
    poison = poison ❙
    laugh = laugh ❙
    and = and ❙
❚
```

Structural rules ⇝ defining functions via λ-terms

```
view Frag1RulesSem : ?Frag1CatMMT —> ?plnqFrag1 =
    include ?Frag1CatSem
    s1 = [n, v] v n ❙
    s2 = [n1,v,n2] v n1 n2 ❙
    n1 = [n] n ❙
    n2 = [n] n ❙
    s3 = [s] ¬ s ❙
    s4 = [a,c,b] c a b ❙
    s5 = [n1,n2] n1 ≐ n2 ❙
    s6 = [n,a] a s ❙
❚
```

putting it all together

```
view Frag1Sem : ?Frag1CatMMT —> ?plnqFrag1 =
    include ?Frag1LexSem
    include ?Frag1RulesSem
❚
```


Montague-Style Processing of $\mathcal{F}_1$ in GLF

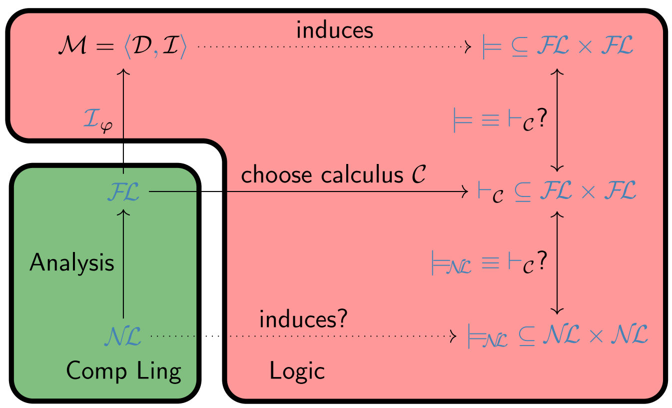
▷ **Example 3.4.5.** *Prudence poisoned the dog and Ethel laughed*

- ▷ Parsing with GF
  - ▷ parse —lang=Eng "Prudence poisons the dog and Ethel laughs"
  - ▷ s4 (s2 (n1 prudence) poison (n2 dog)) and (s1 (n1 ethel) laugh)
- ▷ Semantics construction via GLF: GF parsing + MMT view
  - ▷ parse —lang=Eng "Ethel poisons the dog and Prudence laughs" construct|

▷ poison′ prudence′ ∧ dog′ laugh′ ethel′

Montague-Style Analysis of $\mathcal{F}_1$ in GF and MMT

▷ **Recap:** We have realized the green part of



- ▷ The GF grammar for $\mathcal{F}_1$ defines the fragment $\mathcal{NL}$.
- ▷ The MMT implementation of $\mathrm{PL}^{\mathrm{nq}}$ is $\mathcal{FL}$.
- ▷ The MMT view implements the compositional translation function for $\mathcal{F}_1$

3.4.3 Implementing Natural Deduction in MMT

Implementing Calculi in MMT (Judgments as Types)

▷ **Idea:** Represent proofs and derivations as expressions in theory of "proofs" .

▷ **Concretely:** For any proposition $\mathbf{A}$, introduce $\vdash \mathbf{A}$ for *the type of proofs of* $\mathbf{A}$.

- ▷ Any term of type $\vdash \mathbf{A}$ $\widehat{=}$ a *proof of* $\mathbf{A}$
- ▷ $\mathbf{A}$ *is provable* $\widehat{=}$ $\vdash \mathbf{A}$ is nonempty
- ▷ inference rules are proof constructors (functions)
- ▷ a declaration c : ⊢A makes ¬ A non-empty ⇝ c : ⊢A $\widehat{=}$ an axiom
- ▷ a definition c : ⊢A | = P does as well but also exhibits a "proof" P ⇝ c : ⊢A | = P $\widehat{=}$ a theorem

▷ **in MMT:** we introduce a (proof) type constructor ded a type $\vdash \mathbf{A}$.

```
theory pl0NDminimal : ur:?LF =
    include ?proplogMinimal ❙
    ded : o → type | # ⊢1 prec 10 | role Judgment ❙
```

the role Judgment specifies ?????

Implementing Calculi in MMT ($\mathcal{ND}_0$ Rules)

- **Recap:** We only need the $\mathcal{ND}_0$ rules for negation and conjunction:

$$\frac{\mathbf{A}\quad\mathbf{B}}{\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}_0{\wedge}I \qquad \frac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{A}}\,\mathcal{ND}_0{\wedge}E_l \qquad \frac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{B}}\,\mathcal{ND}_0{\wedge}E_r \qquad \frac{\begin{array}{cc}[\mathbf{A}]^1 & [\mathbf{A}]^1\\ \vdots & \vdots\\ \mathbf{C} & \neg\mathbf{C}\end{array}}{\neg\mathbf{A}}\,\mathcal{ND}_0\neg I^1 \qquad \frac{\neg\neg\mathbf{A}}{\mathbf{A}}\,\mathcal{ND}_0\neg E$$

- **The ND Rules:**

```
notE : {A} ⊢¬ ¬ A →⊢A | # ¬ E 2 ❙
notI : {A,Q} (⊢A →⊢Q) →(⊢A →⊢¬ Q) →⊢¬ A | # ¬ I 3 4 ❙
andI : {A,B} ⊢A →⊢B →⊢A∧ B | # ∧ I 3 4 ❙
andEl : {A,B} ⊢A∧ B →⊢A | # ∧ El 3 ❙
andEr : {A,B} ⊢A∧ B →⊢B | # ∧ Er 3 ❙
```

Inference rules as and hypothetical derivations as proof-to-proof functions.

- **Derived ND Rules:** All other inference rules of $\mathcal{ND}_0$ can be written down similarly. What is more, as they are derivable from those above, they can become MMT definitions.

Implementing Calculi in MMT (a proof)

- **Example 3.4.6.** We can now write down the proof for the commutativity of V!

$$\frac{\dfrac{\dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\,\mathcal{ND}_0{\wedge}E_r \quad \dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\,\mathcal{ND}_0{\wedge}E_l}{\mathbf{B})\wedge\mathbf{A}}\,\mathcal{ND}_0{\wedge}I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B})\wedge\mathbf{A}}\,\mathcal{ND}_0\Rightarrow I^1$$

from ?? as the MMT declaration

```
andcomm {A,B} ⊢A∧ B ⇒ B∧ A | = ⇒ I([x] ∧ I (∧ Er x) (∧ El x)) ❙
```