Logic-Based Natural Language Processing
WS 2023/24

Lecture Notes

Michael Kohlhase

Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg
`Michael.Kohlhase@FAU.de`

2024-01-20

0.1 Preface

0.1.1 This Document

This document contains the course notes for the course "Logic-Based Natural Language Processing" (Logik-Basierte Sprachverarbeitung) held at FAU Erlangen-Nürnberg in the Winter Semesters 2017/18 ff.

This course is a one-semester introductory course that provides an overview over logic-based semantics of natural language. It follows the "method of fragments" introduced by Richard Montague, and builds a sequence of fragments of English with increasing coverage and a sequence of logics that serve as target representation formats. The course can be seen as both a course on semantics and as a course on applied logics.

As this course is predominantly about modeling natural language and not about the theoretical aspects of the logics themselves, we give the discussion about these as a "suggested readings" section part in **??**. This material can safely be skipped (thus it is in the appendix), but contains the missing parts of the "bridge" from logical forms to truth conditions and textual entailment.

Contents: The document mixes the slides presented in class with comments of the instructor to give students a more complete background reference.

Caveat: This document is made available for the students of this course only. It is still an early draft, and will develop over the course of the course. It will be developed further in coming academic years.

Licensing:

This document is licensed under a Creative Commons license that requires attribution, forbids commercial use, and allows derivative works as long as these are licensed under the same license.

Knowledge Representation Experiment: This document is also an experiment in knowledge representation. Under the hood, it uses the sTeX package [Koh08; sTeX], a TeX/LaTeX extension for semantic markup, which allows to export the contents into active documents that adapt to the reader and can be instrumented with services based on the explicitly represented meaning of the documents.

Comments: Comments and extensions are always welcome, please send them to the author.

0.1.2 Acknowledgments

Materials: Some of the material in this course is based on a course "Formal Semantics of Natural Language" held by the author jointly with Prof. Mandy Simons at Carnegie Mellon University in 2001.

ComSem Students: The course is based on a series of courses "Computational Natural Language Semantics" held at Jacobs University Bremen and shares a lot of material with these. The following students have submitted corrections and suggestions to this and earlier versions of the notes: Bastian Laubner, Ceorgi Chulkov, Stefan Anca, Elena Digor, Xu He, and Frederik Schäfer.

LBS Students: The following students have submitted corrections and suggestions to this and earlier versions of the notes: Maximilian Lattka, Frederik Schaefer, Navid Roux.

0.2 Recorded Syllabus

The recorded syllabus – a record the progress of the course in the WS 2023/24 – is in the course page in the ALeA system at `https://courses.voll-ki.fau.de/course-home/ai-1`. The table of contents in the LBS notes at `https://courses.voll-ki.fau.de` indicates the material covered to date in yellow.

For the topics planned for this course, see **??**.

Contents

Elevator Pitch for LBS

▷ **Mission:** In this course we will

- ▹ explore how to model the *meaning of natural language* via transformation into *logical systems*
- ▹ use *logical inference* there to unravel the missing pieces; the information that is *not linguistically realized*, but is conveyed anyways.

▷ **Warning:** This course is only for you if you like logic, you are going to get lots of it and we are going to build our own logics, usually a new one every week or fortnight.

▷ **Approach:** We will do so in a hands-on fashion using the GLIF system, formalizing NL grammars, semantics construction, and inference systems in meta-grammatical/logical systems: GF and MMT.

▷ **Mixing Theory and Practice:** Half of the lectures will be classroom-style teaching of theory and half will be joint formalization.

Chapter 1

Administrativa

We will now go through the ground rules for the course. This is a kind of a social contract between the instructor and the students. Both have to keep their side of the deal to make the acquaintance with research in natural language semantics as efficient and painless as possible.

Prerequisites

- ▷ **I will presuppose:** the mandatory CS courses from Semester 1-4, in particular: (or equivalent)
  - ▷ Course "Grundlagen der Logik in der Informatik" (GLOIN)
  - ▷ Course "Algorithms and data structures"
- ▷ **The following will help:** (we recap if necessary)
  - ▷ AI-1 (symbolic AI)
  - ▷ Ontologies in the semantic web (INF8)
- ▷ **Key Ingredients:** Motivation, interest, curiosity, hard work (LBS is non-trivial)
  - ▷ You can do this course if you want! (and we will help you)

Michael Kohlhase: LBS 2 2024-01-20

LBS Lab (Dogfooding our own Techniques)

- ▷ **General Plan:** We use the thursday slot to get our hands dirty with actual GLIF representations.
- ▷ **Responsible:** Frederik Schaefer (`jan.frederik.schaefer@fau.de`) Room: 11.137.
- ▷ **Goal:** Reinforce what was taught on tuesdays and have some fun.
- ▷ **Homeworks** will be small individual modeling/formalization problems (but take time to solve)

  Group submission if and only if explicitly permitted.

- ▷ **Admin:** To keep things running smoothly
  - ▷ Homeworks will be posted on course forum. (discussed in the lab)
  - ▷ Submission via StudOn (details ⇝ course forum)
- ▷ **Homework Discipline:**
  - ▷ start early! (many assignments need more than one evening's work)
  - ▷ Don't start by sitting at a blank screen!
  - ▷ Humans will be trying to understand the text/code/math when grading it.

Now we come to a topic that is always interesting to the students: the grading scheme.

Grades

- ▷ **Academic Assessment:** so far: two parts (Portfolio Assessment)
  - ▷ (20-30 min oral) or 90 min written exam at the end of the semester (50%)
  - ▷ results of the LBS lab (50%)

  This might not work with 50+ students, need to see how the course develops!
- ▷ If you have a suggestions, I will probably be happy with that as well.

Actually, I do not really care what the grading scheme is, and so it is open to discussion. For all I care we would not have grades at all; but students need them to graduate. Generally, I would like to spend as little time as possible on the grades admin, to the extent that I can give grades without going to jail or blushing too much.

Textbook, Handouts and Information, Forums, Videos

- ▷ **(No) Textbook:** Course notes at `http://kwarc.info/teaching/LBS`
  - ▷ I mostly prepare them as we go along (semantically preloaded ⇝ research resource)
  - ▷ Please e-mail me any errors/shortcomings you notice. (improve for group)
- ▷ **For GLIF:** Frederik's Master's Thesis [Sch20]
- ▷ **Classical Semantics/Pragmatics:** (in the FAU Library)
  - ▷ Primary reference for LBS: [CKG09] (in the FAU Library)
  - ▷ also: [HHS07; Bir13; Rie10; ZS13; Sta14; Sae03; Por04; Kea11; Jac83; Cru11; Ari10]
- ▷ **Computational Semantics:** [BB05; EU10]
- ▷ **StudOn Forum:** `https://www.studon.fau.de/crs4625835.html` for
  - ▷ announcements, homeworks (my view on the forum)

  - questions, discussion among your fellow students (your forum too, use it!)
- **Course Videos:** at `https://fau.tv/course/3647`

Do I need to attend the lectures

- Attendance is not mandatory for the LBS lecture (official version)
- There are two ways of learning: (both are OK, your mileage may vary)
  - Approach B: Read a book/papers (here: course notes)
  - Approach I: come to the lectures, be involved, interrupt me whenever you have a question.

  The only advantage of I over B is that books/papers do not answer questions
- Approach S: come to the lectures and sleep does not work!
- The closer you get to research, the more we need to discuss!

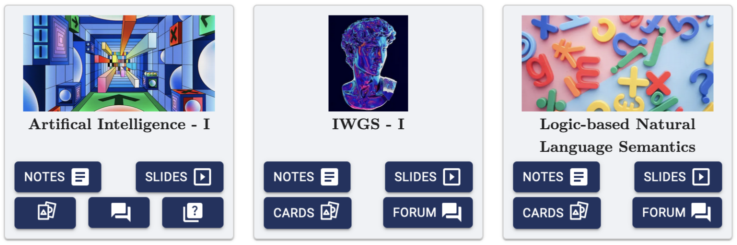
Next we come to a special project that is going on in parallel to teaching the course. I am using the course materials as a research object as well. This gives you an additional resource, but may affect the shape of the coures materials (which now serve double purpose). Of course I can use all the help on the research project I can get, so please give me feedback, report errors and shortcomings, and suggest improvements.

Experiment: Learning Support with KWARC Technologies

- **My research area:** Deep representation formats for (mathematical) knowledge
- **One Application:** Learning support systems(represent knowledge to transport it)
- **Experiment:** Start with this course (Drink my own medicine)
  1. Re-represent the slide materials in OMDoc (Open Mathematical Documents)
  2. Feed it into the ALeA system (`http://courses.voll-ki.fau.de`)
  3. Try it on you all (to get feedback from you)
- Research tasks
  - help me complete the material on the slides (what is missing/would help?)
  - I need to remember "what I say", examples on the board. (take notes)
- Benefits for you (so why should you help?)
  - you will be mentioned in the acknowledgements (for all that is worth)
  - you will help build better course materials (think of next-year's students)

VoLL-KI Portal at `https://courses.voll-ki.fau.de`

▷ **Portal for ALeA Courses:** `https://courses.voll-ki.fau.de`


Artifical Intelligence - I
NOTES
SLIDES
IWGS - I
NOTES
SLIDES
CARDS
FORUM
Logic-based Natural Language Semantics
NOTES
SLIDES
CARDS
FORUM

▷ **AI-1 in ALeA:** `https://courses.voll-ki.fau.de/course-home/ai-1`

- ▷ All details for the course.
- ▷ recorded syllabus (keep track of material covered in course)
- ▷ syllabus of the last semester (for over/preview)

▷ **ALeA Status:** The ALeA system is deployed at FAU for over 1000 students taking six courses

- ▷ (some) students use the system actively (our logs tell us)
- ▷ reviews are mostly positive/enthusiastic (error reports pour in)

The VoLL-KI course portal (and the AI-1) home page is the central entry point for working with the ALeA system. You can get to all the components of the system, including two presentations of the course contents (notes- and slides-centric ones), the flash cards, the localized forum, and the quiz dashboard.

Chapter 2

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 2.0.1.** A natural language is any form of spoken or signed means 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.

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 9 2024-01-20 SOME RIGHTS RESERVED

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.

2.1 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.

What is Natural Language Semantics? A Difficult Question!

- ▷ **Question:** What is "Natural Language Semantics"?
- ▷ **Definition 2.1.1 (Generic Answer).** Semantics is the study of reference, meaning,

or truth.

▷ **Definition 2.1.2.** A sign is anything that communicates a meaning that is not the sign itself to the interpreter of the sign. The meaning can be intentional, as when a word is uttered with a specific meaning, or unintentional, as when a symptom is taken as a sign of a particular medical condition

Meaning is a relationship between signs and the objects they intend, express, or signify.

▷ **Definition 2.1.3.** Reference is a relationship between objects in which one object (the name) designates, or acts as a means by which to refer to – i.e. to connect to or link to – another object (the referent).

▷ **Definition 2.1.4.** Truth is the property of being in accord with reality in a/the mind-independent world. An object ascribed truth is called true, iff it is, and false, if it is not.

▷ **Definition 2.1.5.** For natural language semantics, the signs are usually utterances and names are usually phrases.

▷ That is all very abstract and general, can we make this more concrete?

▷ Different (academic) disciplines find different concretizations.

What is (NL) Semantics? Answers from various Disciplines!

▷ **Observation:** Different (academic) disciplines specialize the notion of semantics (of natural language) in different ways.

▷ **Philosophy:** has a long history of trying to answer it, e.g.

- ▷ Platon $\rightsquigarrow$ cave allegory, Aristotle $\rightsquigarrow$ Syllogisms.
- ▷ Frege/Russell $\rightsquigarrow$ sense vs. referent. (*Michael Kohlhase* vs. *Odysseus*)

▷ **Linguistics/Language Philosophy:** We need semantics e.g. in translation
Der Geist ist willig aber das Fleisch ist schwach! vs.
Der Schnaps ist gut, aber der Braten ist verkocht! (meaning counts)

▷ **Psychology/Cognition:** Semantics $\widehat{=}$ "what is in our brains" ($\rightsquigarrow$ mental models)

▷ **Mathematics** has driven much of modern logic in the quest for foundations.

- ▷ Logic as "foundation of mathematics" solved as far as possible
- ▷ In daily practice syntax and semantics are not differentiated (much).

▷ **Logic@AI/CS** tries to define meaning and compute with them. (applied semantics)

- ▷ makes syntax explicit in a formal language (formulae, sentences)
- ▷ defines truth/validity by mapping sentences into "world" (interpretation)

  - ▷ gives rules of truth-preserving reasoning (inference)

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 2.1.6.** *Peter liebt Maria.* $\rightsquigarrow$ *Peter loves Mary.*
- ▷ ⚠ this only works for simple examples!
- ▷ **Example 2.1.7.** *Wirf der Kuh das Heu über den Zaun.* $\not\rightsquigarrow$ *Throw the cow the hay over the fence.* (differing grammar; Google Translate)
- ▷ **Example 2.1.8.** ⚠ 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 2.1.9.** *We have to understand the meaning for high-quality translation!*

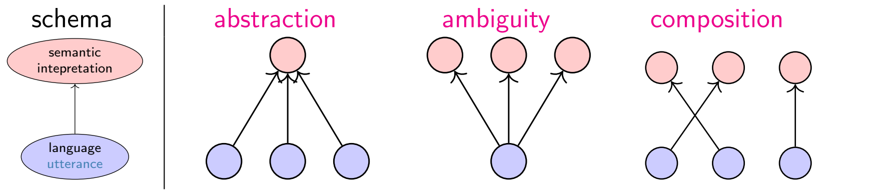
If it is indeed the meaning of natural language, we should look further into how the form of the utterances and their meaning interact.

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 2.1.10 (Word Meaning).**

  Newspaper $\rightsquigarrow$

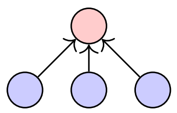
- ▷ For questions/answers, it would be very useful to find out what words (sentences/-texts) mean.
- ▷ **Definition 2.1.11.** Interpretation of natural language utterances: three problems



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.

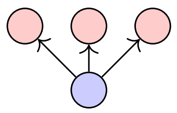
Language and Information (Examples)

▷ **Example 2.1.12 (Abstraction).**



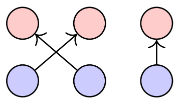
Car and *automobile* have the same meaning

▷ **Example 2.1.13 (Ambiguity).**



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

▷ **Example 2.1.14 (Composition).**



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

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

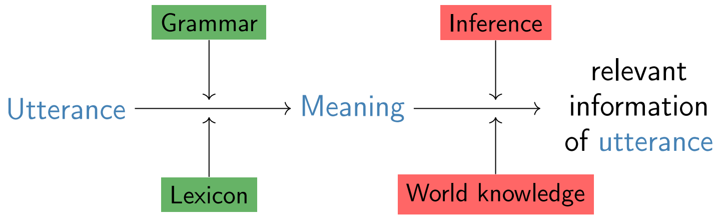
Context Contributes to the Meaning of NL Utterances

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

▷ **Example 2.1.15.** *The lecture begins at 11:00 am.* What lecture? Today?

▷ **Definition 2.1.16.** We call a piece i of information linguistically realized in an utterance U, iff, we can trace i to a fragment of U.

▷ **Definition 2.1.17 (Possible Mechanism).** Inferring the missing pieces from the context and world knowledge:



We call this process pragmatic analysis.

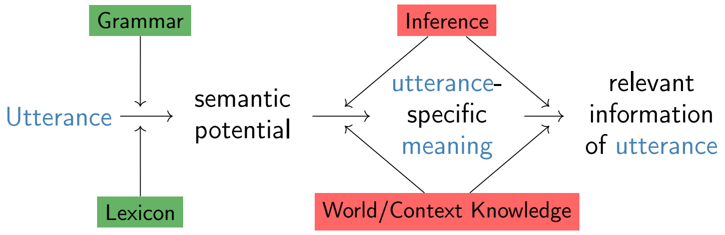
We will look at another example, that shows that the situation with pragmatic analysis is even more complex than we thought. Understanding this is one of the prime objectives of the LBS lecture.

Context Contributes to the Meaning of NL Utterances

▷ **Example 2.1.18.** *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 2.1.18 is also a very good example for the claim Observation 2.1.9 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.

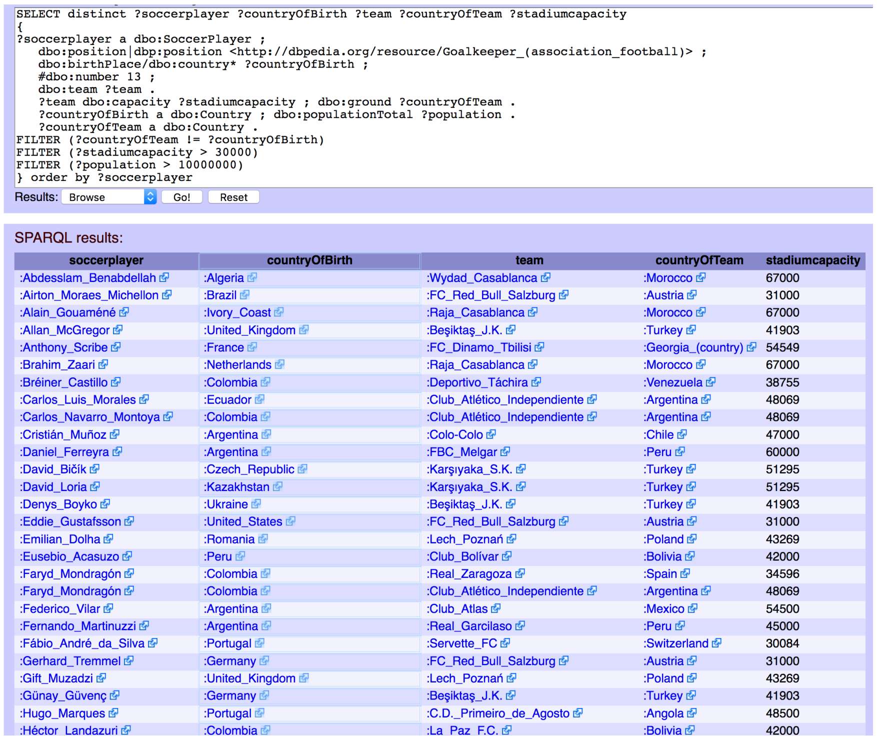
▷ **Demo:**

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
```

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.

2.2 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 2.2.1.** 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

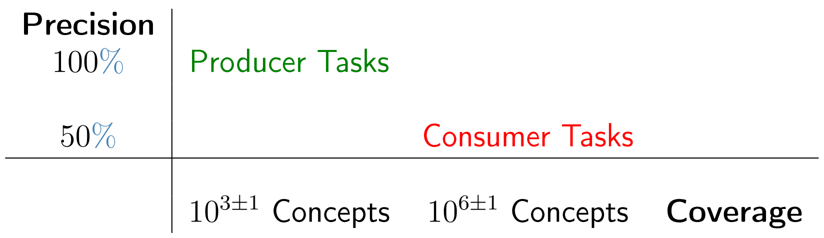
- ▷ We will cover foundational methods of deep processing in the course and a mixture of deep and shallow ones in the lab.

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 2.2.2.** 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)



▷ **Example 2.2.3.** 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 2.2.4 (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 2.2.5.** 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.

▷ **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.

2.3 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?) [Dav67b]

▷ **Example 2.3.1.** 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.*	($\not\models$ *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 2.3.2.** We call an utterance ambiguous, iff it has multiple meanings, which we call readings.

▷ **Example 2.3.3.** 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 2.3.4.** *Every man loves a woman.* (Keira Knightley or his mother!)

▷ **Example 2.3.5.** *Every car has a radio.* (only one reading!)

▷ **Example 2.3.6.** *Some student in every course sleeps in every class at least some of the time.* (how many readings?)

▷ **Example 2.3.7.** *The president of the US is having an affair with an intern.* (2002 or 2000?)

▷ **Example 2.3.8.** *Everyone is here.* (who is everyone?)

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 2.3.9 (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 2.3.10.** 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.

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.!)

2.4 A Taste of Language Philosophy

We will now discuss some concerns from language philosophy as they pertain to the LBS course. Note that this discussion is only intended to give our discussion on natural language semantics some perspective; in particular, it is in no way a complete introduction to language philosophy, or does the discussion there full justice.
We start out our tour through language philosophy with some examples – as linguists and philosophers often to – to obtain an intuition of the phenomena we want to understand.

What is the Meaning of Natural Language Utterances?

- ▷ **Question:** What is the meaning of the word *chair*?
- ▷ **Answer:** "the set of all chairs" (difficult to delineate, but more or less clear)
- ▷ **Question:** What is the meaning of the word *Michael Kohlhase*?
- ▷ **Answer:** The word refers to an object in the real world: the instructor of LBS.
- ▷ **Alternatively:** The singleton with that object (as for "set of chairs" above)
- ▷ **Question:** What about *Michael Kohlhase sits on a chair*?
- ▷ **Towards an Answer:** We have to combine the two sets, via the meaning of "sits".

▷ **Question:** What is the meaning of the word *John F. Kennedy* or *Odysseus*?

▷ **Problem:** There are no objects in the real worlds, so the meaning of both is ∅ and thus equal ☹.

The main intuition we get is that meaning is more complicated than we may have thought in the beginning.

2.4.1 Epistemology: The Philosphy of Science

We start out by looking at the foundations of epistemology, which sets the basis for modern (empirical) science. Our presentation here is modeled on Karl Popper's work on the theory of science. Naturally, our account here is simplified to fit the occasion, see [Pop34; Pop59] for the full story.

Note that like any foundational account of complex concepts like knowledge, belief, rationality, and their justification, we have to base our philosophy on some concepts we take at face value. Here these are natural and formal languages, worlds, situations, etc. which will stay very general in the current foundational setting.

We will later instantiate these by more concrete notions as we go along in the LBS course.

Epistemology – Propositions & Observations

▷ **Definition 2.4.1.** Epistemology is the branch of philosophy concerned with studying nature of knowledge, its justification, the rationality of belief, scientific theories and predictions, and various related issues.

▷ **Definition 2.4.2.** A proposition is a sentence about the actual world or a class of worlds deemed possible in a natural or formal language whose meaning can be expressed as being true or false in a specific world.

▷ **Definition 2.4.3.** A belief is a proposition φ that an agent a holds true about a class of worlds. This is a characterizing feature of the agent.

▷ **Definition 2.4.4 (Belief - The JTB Account).** Knowledge is justified, true belief.

▷ **Problem:** How can an agent justify a belief to obtain knowledge.

▷ **Definition 2.4.5.** Given a world w, the observed value (or just value, i.e. true or false) of a proposition (in w) can be determined by observations, that is an agent, the observer, either observes (experiences) that φ is true in w or conducts a deliberate, systematic experiment that determines φ to be true in w.

The crucial intuition here is that we express belief and possibly knowledge about the world using language. But we can only access truth in the world by observation, a possibly flawed operation. So we will never be able to ascertain the "true belief" part, and need to work all the harder on the "justified" part.

Epistemology – Reproducibility & Phenomena

- ▷ **Problem:** Observations are sometimes unreliable, e.g. observer o perceives φ to be true, while it is false or vice versa.
- ▷ **Idea:** Repeat the observations to raise the probability of getting them right.
- ▷ **Definition 2.4.6.** An observation φ is said to be reproducible, iff φ can observed by different observers in different situations.
- ▷ **Definition 2.4.7.** A phenomenon φ is a proposition that is reproducibly observable to be true in a class of worlds.
- ▷ **Problem:** We would like to verify a phenomenon φ, i.e. observe φ in all worlds, But relevant world classes are too large to make this practically feasible.
- ▷ **Definition 2.4.8.** A world w is a counterexample to a proposition φ, if φ is observably false in w.
- ▷ **Intuition:** The absence of counterexamples is the best we can hope for in general for accepting phenomena.
- ▷ **Intuition:** The phenomena constitute the "world model" of an agent.
- ▷ **Problem:** It is impossible/inefficient (for an agent) to know all phenomena.
- ▷ **Idea:** An agent could retain only a small subset of known propositions, from this all phenomena can be derived.

We will pursue this last idea. The (small) subset of propositions from which the phenomena that are relevant to an agent can be derived will become the beliefs of the agent. An agent will make strive to justify these beliefs to succeed in the world. This is where our notion of knowledge comes from.

Epistemology – Explanations & Hypotheses

- ▷ **Definition 2.4.9.** A proposition ψ follows from a proposition φ, iff ψ is true in any world where φ is.
- ▷ **Definition 2.4.10.** An explanation of a phenomenon φ is a set Φ of propositions, such that φ follows from Φ.
- ▷ **Example 2.4.11.** $\{\varphi\}$ is a (rather useless) explanation for φ.
- ▷ **Intuition:** We prefer explanations Φ that explain more than just φ.
- ▷ **Observation:** This often coincides with explanations that are in some sense "simpler" or "more elementary" than φ. ($\leadsto$ Occam's razor)
- ▷ **Definition 2.4.12.** A proposition is called falsifiable, iff counterexamples are theoretically possible and the observation of a reproducible series of counterexample is practically feasible.
- ▷ **Definition 2.4.13.** A hypothesis is a proposed explanation of a phenomenon that is falsifiable.

We insist that a hypothesis be falsifiable, because we cannot hope to verify it and indeed the absence of counterexamples is the best we can hope for. But if finding counterexamples is hopeless, it is not even worth bothering with a hypothesis.
This gives rise to a very natural strategy of accumulating propositions to represent (what could) knowledge about the world.

Epistemology – Scientific Theories

▷ **Knowledge Strategy:** Collect hypotheses about the world, drop those with counterexamples and those that can be explained themselves.

▷ **Definition 2.4.14.** A hypothesis φ can be tested in world/situation w by observing the value of φ in w. If the value is true, then we say that the observation o supports φ or is evidence for φ. If it is false then o falsifies φ.

▷ **Definition 2.4.15.** A (scientific) theory for a set Φ of phenomena is a set Θ of hypotheses that

- ▷ has been tested extensively and rigorously without finding counterexamples, and
- ▷ is minimal in the sense that no subset of Θ explains Φ.

▷ **Definition 2.4.16.** We call any proposition φ that follows from a theory Φ a prediction of Φ.

▷ **Note:** To falsify a theory Φ, it is sufficient to falsify any prediction. Any observation of a prediction φ of Φ supports Φ.

Indeed the epistemological approach described in this subsection has become the predominant one in modern science. We will introduce both on very simple examples next.

2.4.2 Meaning Theories

If the meaning of natural language is indeed complicated, then we should really admit to that and instead of directly answering the question, allow for multiple opinions and embark on a regime of testing them against reality. We review some concepts from language philosophy towards that end.
We now specialize the general epistemology for natural language the "world" we try to model empirically.

Theories of Meaning

▷ **The Central Question:** What is the meaning of natural language?

▷ This is difficult to answer definitely, . . .

▷ **But** we can form meaning theory that make predictions that we can test.

▷ **Definition 2.4.17.** A semantic meaning theory assigns semantic contents to expressions of a language.

▷ **Definition 2.4.18.** A foundational meaning theory tries to explain why language expressions have the meanings they have; e.g. in terms of mental states of individuals and groups.

▷ It is important to keep these two notions apart.

▷ We will concentrate on semantic meaning theories in this course.

In [Spe17], an excellent survey on meaning theories, the author likens the difference between semantic and foundational theories of meaning to the differing tasks of an anthropologist trying to fully document the table manner of a distant tribe ($\widehat{=}$ semantic meaning theory) or to explain why the table manners evolve ($\widehat{=}$ foundational meaning theory).
Let us fortify our intuition about semantic meaning theories by showing one that can deal with the meaning of names we started our subsection with.

The Meaning of Singular Terms

▷ Let's see a semantic meaning theory in action.

▷ **Definition 2.4.19.** A singular term is a phrase that purports to denote or designate a particular individual person, place, or other object.

▷ **Example 2.4.20.** *Michael Kohlhase* and *Odysseus* are singular terms.

▷ **Definition 2.4.21.** In [Fre92], Gottlob Frege distinguishes between sense (Sinn) and referent (Bedeutung) of singular terms.

▷ **Example 2.4.22.** Even though *Odysseus* does not have a referent, it has a very real sense. (but what is a sense?)

▷ **Example 2.4.23.** The ancient greeks knew the planets *Hesperos* (the evening star) and *Phosphoros* (the morning star). These words have different senses, but the – as we now know – same referent: the planet Venus.

▷ **Remark:** Bertrand Russell views singular terms as disguised definite descriptions – *Hesperos* as "the brightest heavenly body that sometimes rises in the evening". Frege's sense can often be conflated with Russell's descriptions.(there can be more than one definite description)

We think of Frege's conceptualization as a semantic meaning theory, since it assigns semantic content – the pair of sense and referent, whatever they might concretely be – to singular terms.

Cresswell's "Most Certain Principle" and Truth Conditions

▷ **Problem:** How can we test meaning theories in practice?

▷ **Definition 2.4.24.** Cresswell's (1982) most certain principle (MCP): [Cre82]

I'm going to begin by telling you what I think is the most certain thing I think about meaning. Perhaps it's the only thing. It is this. If we have two sentences A and B, and A is true and B is false, then A and B do not mean the same.

▷ **Definition 2.4.25.** The truth conditions of a sentence are the conditions of the world under which it is true. These conditions must be such that if all obtain, the sentence is true, and if one doesn't obtain, the sentence is false.

▷ **Observation:** Meaning determines truth conditions and vice versa.

▷ **In Fregean terms** The sense of a sentence (a thought) determines its referent (a truth value).

a

This principle sounds trivial – and indeed it is, if you think about it – but gives rise to the notion of truth conditions, which form the most important way of finding out about the meaning of sentences: the determinations of truth conditions.

Truth Conditions in Practice

▷ **Idea:** To test/determine the truth conditions of a sentence S in practice, we tell little stories that describe situations/worlds that embed S.

▷ **Example 2.4.26.** Consider the ambiguous sentence from Example 2.3.3
John chased the gangster in the red sports car.
For each of three readings there is story $\widehat{=}$ truth conditions

- ▷ John drives the red sports car and chases the gangster
- ▷ John chases the gangster who drives the red sports car
- ▷ John chases the gangster on the back seat of a (very very big) red sports car.

All of these stories correspond to different worlds, so by the MCP there must be at least three readings!

Compositionality

▷ **Definition 2.4.27.** A meaning theory T is compositional, iff the meaning of an expression is a function of the meanings of its parts. We say that T obeys the compositionality principle or simply compositionality if it is.

▷ To compute the meaning of an expression, look up the meanings of the basic expressions forming it and successively compute the meanings of larger parts until a meaning for the whole expression is found.

▷ **Example 2.4.28 (Compositionality at work in arithmetic).** To compute the value of $(x+y)/(z\cdot u)$, look up the values of x, y, z, and u, then compute $x+y$ and $z\cdot u$, and finally compute the value of the whole expression.

▷ Many philosophers and linguists hold that compositionality is at work in ordinary language too.

Why Compositionality is Attractive

▷ Compositionality gives a nice building block for a meaning theory:

▷ **Example 2.4.29.** *[Expressions [are [built [from [words [that [combine [into [[larger [and larger]] subexpressions]]]]]]]]*

▷ **Consequence:** To compute the meaning of an expression, look up the meanings of its words and successively compute the meanings of larger parts until a meaning for the whole expression is found.

▷ Compositionality explains how people can easily understand sentences they have never heard before, even though there are an infinite number of sentences any given person at any given time has not heard before.

Compositionality and the Congruence Principle

▷ Given reasonable assumptions compositionality entails the

▷ **Definition 2.4.30.** The congruence principle states that whenever A is part of B and A' means just the same as A, replacing A by A' in B will lead to a result that means just the same as B.

▷ **Example 2.4.31.** Consider the following (complex) sentences:

1. *blah blah blah such and such blah blah*
2. *blah blah blah so and so blah blah*

If *such and such* and *so and so* mean the same thing, then 1. and 2. mean the same too.

▷ **Conversely:** if 1. and 2. do not mean the same, then *such and such* and *so and so* do not either.

A Test for Synonymity

▷ Suppose we accept the most certain principle (difference in truth conditions implies difference in meaning) and the congruence principle (replacing words by synonyms results in a synonymous utterance). Then we have a diagnostics for synonymity: Replacing utterances by synonyms preserves truth conditions, or equivalently

▷ **Definition 2.4.32.** The following is called the truth conditional synonymy test:

> If replacing A by B in some sentence C does not preserve truth conditions, then A and B are not synonymous.

▷ We can use this as a test for the question of individuation: when are the meanings of two words the same – when are they synonymous?

▷ **Example 2.4.33 (Unsurprising Results).** The following sentences differ in truth conditions.

1. *The cat is on the mat.*
2. *The dog is on the mat.*

Hence *cat* and *dog* are not synonymous. The converse holds for

1. *John is a Greek.*
2. *John is a Hellene.*

In this case there is no difference in truth conditions.

▷ But there might be another context that does give a difference.

Contentious Cases of Synonymy Test

▷ **Example 2.4.34 (Problem).** The following sentences differ in truth values:

1. *Mary believes that John is a Greek*
2. *Mary believes that John is a Hellene*

So *Greek* is not synonymous to *Hellene*. The same holds in the classical example:

1. *The Ancients knew that Hesperus was Hesperus*
2. *The Ancients knew that Hesperus was Phosphorus*

In these cases most language users do perceive a difference in truth conditions while some philosophers vehemently deny that the sentences under 1. could be true in situations where the 2. sentences are false.

▷ It is important here of course that the context of substitution is within the scope of a verb of propositional attitude. (maybe later!)

A better Synonymy Test

▷ **Definition 2.4.35 (Synonymy).** The following is called the truth conditional synonymy test:

> If replacing A by B in some sentence C does not preserve truth conditions in a compositional part of C, then A and B are not synonymous.

Testing Truth Conditions with Logic

▷ **Definition 2.4.36.** A logical language model $\mathcal{M}$ for a natural language L consists of a logical system $\langle\mathcal{L},\mathcal{K},\models\rangle$ and a function φ from L sentences to $\mathcal{L}$-formulae.

▷ **Problem:** How do we find out whether $\mathcal{M}$ models L faithfully?

▷ **Idea:** Test truth conditions of sentences against the predictions $\mathcal{M}$ makes.

▷ **Problem:** The truth conditions for a sentence S in L can only be formulated and verified by humans that speak L.

▷ **In Practice:** Truth conditions are expressed as "stories" that specify salient situations. Native speakers of L are asked to judge whether they make S true/false.

▷ **Observation 2.4.37.** *A logical language model $\mathcal{M}{:=}\langle L,\mathcal{L},\varphi\rangle$ can be tested:*

1. *Select a sentence S and a situation W that makes S true.(according to humans)*
2. *Translate S in to an $\mathcal{L}$-formula $S'{:=}\varphi(S)$.*
3. *Express W as a set Φ of $\mathcal{L}$-formulae.* ($\Phi \mathrel{\widehat{=}}$ *truth conditions*)
4. *$\mathcal{M}$ is supported if $\Phi\models S'$, falsified if $\Phi\not\models S'$.*

▷ **Corollary 2.4.38.** *A logical language model constitutes a semantic meaning theory.*

2.5 Computational Semantics as a Natural Science

Overview: Formal natural language semantics is an approach to the study of meaning in natural language which utilizes the tools of logic and model theory. Computational semantics adds to this the task of representing the role of inference in interpretation. By combining these two different approaches to the study of linguistic interpretation, we hope to expose you (the students) to the best of both worlds.

Computational Semantics as a Natural Science

▷ **In a nutshell:** Formal logic studies formal languages, their relation with the world (in particular the truth conditions). Computational logic adds the question about the computational behavior of the relevant aspects of the formal languages.

▷ This is almost the same as the task of natural language semantics!

▷ It is one of the key ideas that logics are good scientific models for natural languages, since they simplify certain aspects so that they can be studied in isolation. In particular, we can use the general scientific method of

1. observing
2. building formal theories for an aspect of reality,
3. deriving the consequences of the hypotheses about the world in the theories
4. testing the predictions made by the theory against the real-world data. If the theory predicts the data, then this supports the theory, if not, we refine the theory, starting the process again at 2.

Excursion: In natural sciences, this is established practice; e.g. astronomers observe the planets, and try to make predictions about the locations of the planets in the future. If you graph the location over time, it appears as a complicated zig-zag line that is difficult to understand. In 1609 Johannes Kepler postulated the model that the planets revolve around the sun in ellipses, where the sun is in one of the focal points. This model made it possible to predict the future whereabouts of the planets with great accuracy by relatively simple mathematical computations. Subsequent observations have confirmed this theory, since the predictions and observations match.

Later, the model was refined by Isaac Newton, by a theory of gravitation; it replaces the Keplerian assumptions about the geometry of planetary orbits by simple assumptions about gravitational forces (gravitation decreases with the inverse square of the distance) which entail the geometry.

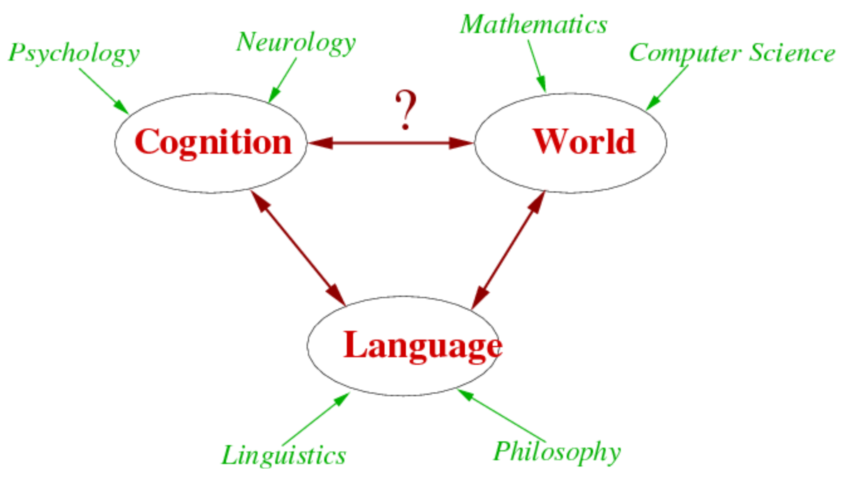
Even later, the Newtonian theory of celestial mechanics was replaced by Einstein's relativity theory, which makes better predictions for great distances and high-speed objects.

All of these theories have in common, that they build a mathematical model of the physical reality, which is simple and precise enough to compute/derive consequences of basic assumptions, that can be tested against observations to validate or falsify the model/theory.

The study of natural language (and of course its meaning) is more complex than natural sciences, where we only observe objects that exist independently of ourselves as observers. Language is an inherently human activity, and deeply interdependent with human cognition (it is arguably one of its motors and means of expression). On the other hand, language is used to communicate about phenomena in the world around us, the world in us, and about hypothetical worlds we only imagine.

Therefore, natural language semantics must necessarily be an intersective discipline and a trans-disciplinary endeavour, combining methods, results and insights from various disciplines.

NL Semantics as an Intersective Discipline



Chapter 3

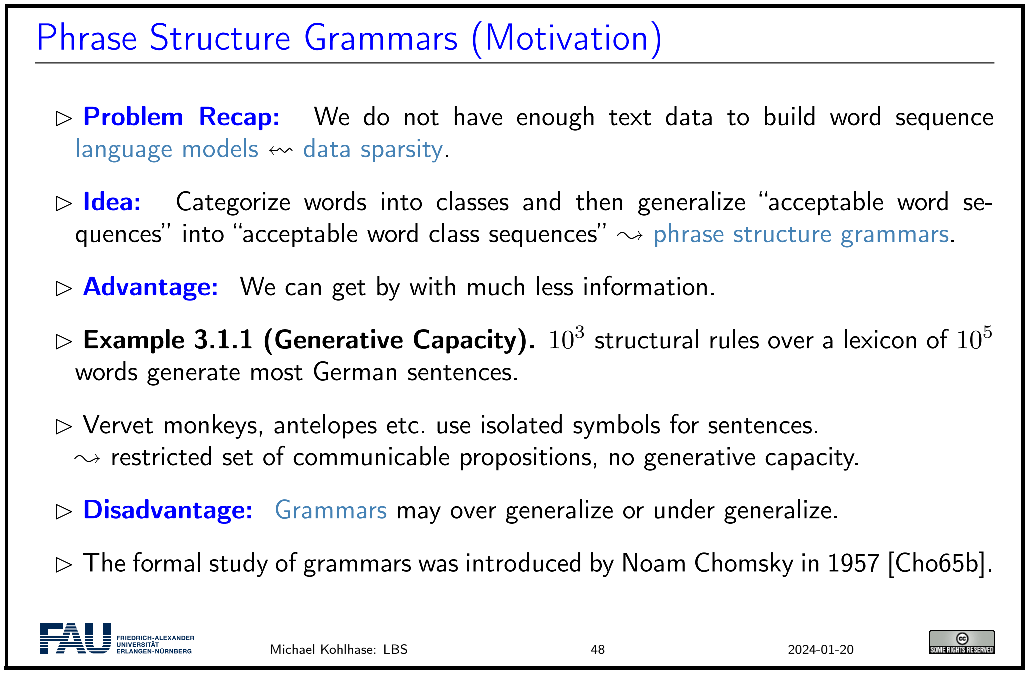
Symbolic Systems for Semantics

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 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.

3.1.1 Recap: (Context-Free) Grammars



Phrase Structure Grammars (Motivation)

- ▷ **Problem Recap:** We do not have enough text data to build word sequence language models $\leftsquigarrow$ data sparsity.
- ▷ **Idea:** Categorize words into classes and then generalize "acceptable word sequences" into "acceptable word class sequences" $\rightsquigarrow$ phrase structure grammars.
- ▷ **Advantage:** We can get by with much less information.
- ▷ **Example 3.1.1 (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.
  $\rightsquigarrow$ 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 [Cho65b].

Michael Kohlhase: LBS 48 2024-01-20

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

Phrase Structure Grammars (cont.)

▷ **Example 3.1.2.** A simple phrase structure grammar G:

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

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

▷ **Definition 3.1.3.** 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 alphabet. The nonterminals in their heads are called lexical categories.

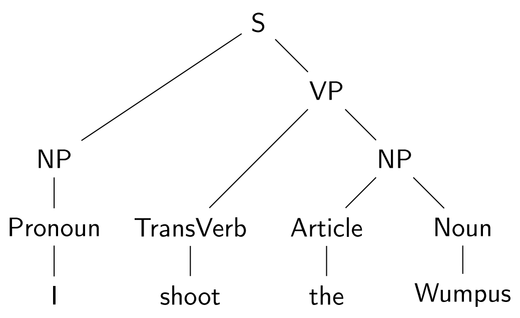
▷ **Definition 3.1.4.** The non-lexicon production rules are called structural, and the nonterminals in the heads are called phrasal categories.

Context-Free Parsing

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

▷ **Definition 3.1.5.** Bottom up parsing works by replacing any substring that matches the body of a production rule with its head.

▷ **Example 3.1.6.** 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.

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

▷ **Theorem 3.1.7.** *Context-free parsing $\hat{=}$ Boolean matrix multiplication!*

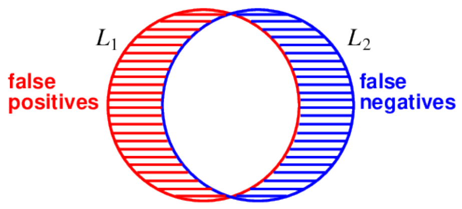
▷ $\rightsquigarrow$ unlikely to find faster practical algorithms. (details in [Lee02])

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 3.1.8.** 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.

3.1.2 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 3.1.9.** Grammatical Framework (GF [Ran04; Ran11]) is a modular formal framework and functional programming language for writing multilingual grammars of natural languages.

▷ **Definition 3.1.10.** 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 3.1.11.** 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 3.1.12.** Parsing is the dual to linearization, it transforms NL utterances into abstract syntax trees.

▷ **Definition 3.1.13.** 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 3.1.14 (A Hello World Grammar).**

```
abstract zero = {
  flags startcat=O;
  cat
    S ; NP ; V2 ;
  fun
    spo : V2 -> NP -> NP -> S ;
    John, Mary : NP ;
    Love : V2 ;
}
```

```
concrete zeroEng of zero = {
  lincat
    S, NP, V2 = Str ;
  lin
    spo vp s o
= s ++ vp ++ o;
    John = "John" ;
    Mary = "Mary" ;
    Love = "loves" ;
}
```

▷ Make a French grammar with `John="Jean"; Mary="Marie"; Love="aime";`

▷ Parse a sentence in GF: `parse "John loves Mary"` $\leadsto$ `Love John Mary`

▷ Linearize in GF: `linearize Love John Mary` $\leadsto$ `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` $\leadsto$ `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 3.1.15 (Hello Prolog).** Linearize to Prolog terms:

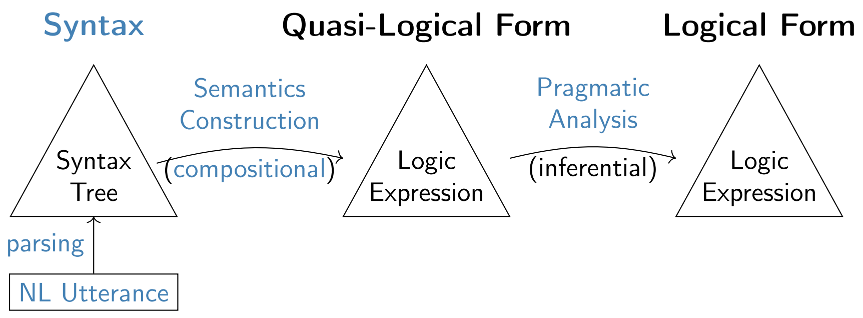
```
concrete zeroPro of zero = {
  lincat
    S , NP , V2 = Str;
  lin
    spo = \vt,subj,obj -> vt ++ "(" ++ subj ++ "," ++ obj ++ ").";
    John = "john";
    Mary = "mary";
    Love = "loves";
  }
```

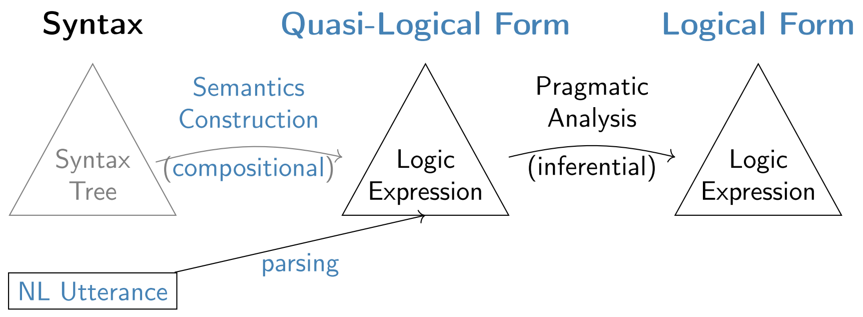
- ▷ **Linearization in GF:** `linearize Love John Mary` $\rightsquigarrow$ `loves ( john , mary )`
- ▷ **Note:** `loves ( john , mary )` is *not* a quasi-logical forms, but a Prolog term that can be read into an Prolog interpreter for pragmatic analysis.

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

Syntactic and Semantic Grammars

- ▷ Recall our interpretation pipeline





▷ **Definition 3.1.16.** 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).

3.1.3 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 $\lambda \mathrel{\widehat{=}}$) 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}
    };
}
```

3.1.4 Engineering Resource Grammars in GF

After understanding the moving parts of GF grammars, we can imagine that grammars that cover large parts of the phenomena of a natural language will become quite large – if only because for every word we need a declaration in the abstract grammar and a linearization in the concrete grammar.

Therefore we will turn to GF functionalities for engineering practical grammars now. We do what we mostly do in computer science: we modularize. The modularization functionality presented in this subsection has been developed for the GF resource grammar library (RGL), a giant shared abstract grammar together with ca. 40 concrete grammars for various natural languages. For managing such a large project, modularity becomes crucial.

Modular Grammars (Abstract)

▷ We split the grammar into modules (resource + application grammar)

Monolithic	Modular

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

▷ Functionality is the same, but we can reuse the components

Modular Grammars (Concrete English)

▷ We split the grammar into modules (resource + application grammar)

Monolithic	Modular

`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;`
`}`
 	
`resource twoParadigmsEN =`
`  twoCatEN ** {oper`
`    mkPN : Str -> StringType`
`      = \x -> {s = x};`
`    mkV2 : Str -> StringType`
`      = \x -> {s = x};`
`    mkN : Str -> StringType`
`      = \x -> {s = x};}`
	`concrete twoCatEN of twoCat = {`
	`  oper StringType : Type = {s : Str};`
	`  lincat`
	`    S, NP, N, V2 = StringType ;}`
	`concrete twoGrammarEN of twoGrammar =`
	`  twoCatEN ** {`
	`  lin`
	`    spo vp s o`
	`= {s= s.s ++ vp.s ++ o.s};`
	`    the x = {s = "the" ++ x.s};}`
	`concrete twoLexEN of twoLex =`
	`  twoCatEN ** open twoParadigmsEN in {`
	`  lin`
	`    John = mkPN "John" ;`
	`    Mary = mkPN "Mary" ;`
	`    Love = mkV2 "loves" ;`
	`    dog = mkN "dog" ;`
	`    mouse = mkN "mouse" ;}`
	`concrete twoRGEN of twoRG =`
	`  twoGrammarEN,twoLexEN;`

Modular Grammars (Concrete German)

▷ We split the grammar into modules (resource + application grammar)

```
concrete twoCatDE of twoCat = {
  param
    Gender = masc | fem | neut;
    Case = nom | acc;
  oper
    Noun : Type = {s : Str; gender : Gender};
    NounPhrase : Type = {s: Case => Str};
  lincat
    S, V2 = {s : Str};
    N = Noun;
    NP = NounPhrase;}

resource twoParadigmsDE = twoCatDE ** {
  oper
    mkPN : Str -> NounPhrase = \x -> {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};}
```

Modular Grammars (Concrete German)

▷
```
concrete twoGrammarDE of twoGrammar =
  twoCatDE ** open twoParadigmsDE in {
  lin
    spo vp subj obj = {s = subj.s!nom ++ vp.s ++ obj.s!acc};
    the n = {s = table { nom => mkXXX "der" "die" "das" n;
                         acc => mkXXX "den" "die" "das" n}};}

concrete twoLexDE of twoLex = twoCatDE ** open twoParadigmsDE in {
  lin
    John = mkPN "Johannes";
    Mary = mkPN "Maria";
    Love = mkV2 "liebt";
    dog = mkN "Hund" masc;
    mouse = mkN "Maus" fem;}

concrete twoRGDE of twoRG = twoGrammarDE,twoLexDE;
```

A Semantic Grammar

▷ We use logic-inspired categories instead of the syntactic ones

Syntactic	Semantic

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 ;
}
	abstract three = {
	  flags startcat=O;
	  cat
	    I; O; P1; P2;
	  fun
	    spo : P2 −> I −> I −> O ;
	    John, Mary : I ;
	    Love : P2 ;
	    dog, mouse : P1;
	    the : P1 −> I;
	}

A Semantic Grammar (Modular Development)

▷ We use logic-inspired categories instead of the syntactic ones

Syntactic	Semantic
concrete twoCatEN **of** twoCat = {	**concrete** threeEN **of** three =
oper StringType : Type = {s : Str};	twoLexEN,twoGrammarEN **
lincat	**open** twoParadigmsEN **in** {
S, NP, N, V2 = StringType ;}	**lincat**
concrete twoGrammarEN **of** twoGrammar =	I = NP;
twoCatEN ** {	O = S;
lin	P1 = N;
spo vp s o = {s= s.s ++ vp.s ++ o.s};	P2 = V2;
the x = {s = *"the"* ++ x.s};}	}
concrete twoLexEN **of** twoLex =	**concrete** threeDE **of** three =
twoCatEN ** **open** twoParadigmsEN **in** {	twoLexDE,twoGrammarDE **
lin	**open** twoParadigmsDE **in** {
John = mkPN *"John"* ;	**lincat**
Mary = mkPN *"Mary"* ;	I = NP;
Love = mkV2 *"loves"* ;	O = S;
dog = mkN *"dog"* ;	P1 = N;
mouse = mkN *"mouse"* ;}	P2 = V2;
concrete twoRGEN **of** twoRG =	}
twoGrammarEN,twoLexEN;	

Michael Kohlhase: LBS 70 2024-01-20

3.2 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.

3.2.1 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 $wff_0(\Sigma_0)$ of propositional logic (PL^0) consists of propositions built from propositional variables from $\mathcal{V}_0$ and connectives from Σ_0.

▷ We model $wff_0(\Sigma_0)$ 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 $\longrightarrow$, $type$, 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 3.2.1.** 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 3.2.2.** Declarations for the connectives ¬ and ∧

```
    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 3.2.3.** 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 3.2.4.** 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` $\wedge$ `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` $\wedge$ `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	Elimination
$\dfrac{\mathbf{A}\quad\mathbf{B}}{\mathbf{A}\wedge\mathbf{B}}\mathcal{ND}\_0\wedge I$	$\dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{A}}\mathcal{ND}\_0\wedge E_l \qquad \dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{B}}\mathcal{ND}\_0\wedge E_r$
$\dfrac{\begin{matrix}[\mathbf{A}]^1 & [\mathbf{A}]^1 \\ \vdots & \vdots \\ \mathbf{C} & \neg\mathbf{C}\end{matrix}}{\neg\mathbf{A}}\mathcal{ND}\_0\neg I^1$	$\dfrac{\neg\neg\mathbf{A}}{\mathbf{A}}\mathcal{ND}\_0\neg E$

▷ 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 $\vdash\mathbf{C}$.

$$\dfrac{\begin{matrix}[\mathbf{A}]^1 & [\mathbf{A}]^1 \\ \vdots & \vdots \\ \mathbf{C} & \neg\mathbf{C}\end{matrix}}{\neg\mathbf{A}}$$

▷ 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 3.2.5.** Let's try this on the well-known $\mathcal{ND}\_0$ proof

$$\cfrac{\cfrac{\cfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\,\mathcal{ND}\_0\wedge E_r \qquad \cfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\,\mathcal{ND}\_0\wedge E_l}{\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}\_0\wedge I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}\_0\Rightarrow I^1$$

```
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 3.2.6 (Continued).**

$$\cfrac{\cfrac{\cfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\,\mathcal{ND}\_0\wedge E_r \qquad \cfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\,\mathcal{ND}\_0\wedge E_l}{\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}\_0\wedge I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}\_0\Rightarrow I^1$$

```
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 `impI` with one argument (a subproof Ψ)
  - ▹ **But remember**: `implI: {a,b} (⊢a →⊢b) →⊢(a⇒ b)▌` takes three!
  - ▹ **Idea**: add special postfix notation definition `|# 3 impI` $(3 \mapsto \Psi)$
  - ▹ **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 3.2.7.** 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) ❚
```

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

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

For some suitable proof expressions Φ and Ψ.

3.2.2 General Functionality of MMT

We will use the OMDoc/Mmt to represent both logical systems and the semantic domains (universes of discourse) of the various fragments. The Mmt system implements the OMDoc/Mmt language, it can be used as

- a Java library that provides data structures and an API of logic oriented algorithms, and as
- a standalone knowledge-management service provider via web interfaces.

We will make use of both in the LBS course and give a brief overview in this subsection. For a (math-themed) tutorial that introduces format and system in more detail see [OMT].

Representation language (Mmt)

▷ **Definition 3.2.8.** Mmt $\widehat{=}$ module system for mathematical theories

▷ Formal syntax and semantics

  ▹ needed for mathematical interface language

  ▹ but how to avoid foundational commitment?

▷ Foundation-independence

  ▹ identify aspects of underlying language that are necessary for large scale processing

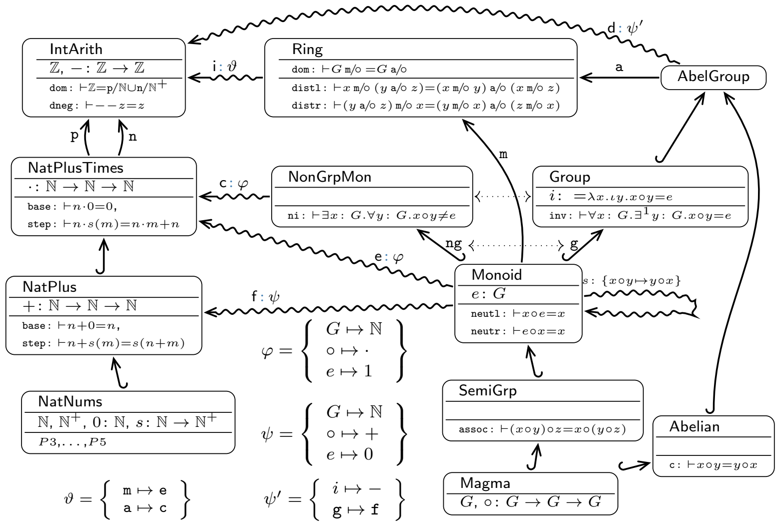
  ▹ formalize exactly those, be parametric in the rest

- ▷ observation: most large scale operations need the same aspects
- ▷ Module system
  - ▷ preserve mathematical structure wherever possible
  - ▷ formal semantics for modularity
- ▷ Web-scalable
  - ▷ build on XML, OpenMath, OMDoc
  - ▷ URI based logical identifiers for all declarations
- ▷ Implemented in the MMT system.

The basic idea of the OMDoc/MMT format is that knowledge (originally mathematical knowledge for which the format is designed, but also world knowledge of the semantic domains in the fragments) can be represented modularly, using strong forms of inheritance to avoid duplicate formalization. This leads to the notion of a theory graph, where the nodes are theories that declare language fragments and axiomatize knowledge about the objects in the domain of discourse. The following theory graph is taken from [OMT].

Modular Representation of Math (MMT Example)

▷ **Example 3.2.9 (Elementary Algebra and Arithmetics).**


IntArith
$\mathbb{Z}, -\colon \mathbb{Z} \to \mathbb{Z}$
dom: $\vdash \mathbb{Z}=$p/$\mathbb{N}\cup$n/$\mathbb{N}^+$
dneg: $\vdash --z=z$
Ring
dom: $\vdash G$ m/∘ $=G$ a/∘
distl: $\vdash x$ m/∘ $(y$ a/∘ $z)=(x$ m/∘ $y)$ a/∘ $(x$ m/∘ $z)$
distr: $\vdash (y$ a/∘ $z)$ m/∘ $x=(y$ m/∘ $x)$ a/∘ $(z$ m/∘ $x)$
AbelGroup
d: ψ'
i: ϑ
a
p
n
m
NatPlusTimes
$\cdot\colon \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
base: $\vdash n\cdot 0=0$,
step: $\vdash n\cdot s(m)=n\cdot m+n$
c: φ
NonGrpMon
ni: $\vdash \exists x\colon G.\forall y\colon G.x\circ y\neq e$
Group
$i\colon\ =\lambda x.\iota y.x\circ y=e$
inv: $\vdash\forall x\colon G.\exists^1 y\colon G.x\circ y=e$
ng
g
e: φ
Monoid
$e\colon G$
neutl: $\vdash x\circ e=x$
neutr: $\vdash e\circ x=x$
s: $\{x\circ y\mapsto y\circ x\}$
NatPlus
$+\colon \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
base: $\vdash n+0=n$,
step: $\vdash n+s(m)=s(n+m)$
f: ψ
$\varphi=\{G\mapsto\mathbb{N},\ \circ\mapsto\cdot,\ e\mapsto 1\}$
NatNums
$\mathbb{N}, \mathbb{N}^+, 0\colon\mathbb{N}, s\colon\mathbb{N}\to\mathbb{N}^+$
$P3,\ldots,P5$
$\psi=\{G\mapsto\mathbb{N},\ \circ\mapsto +,\ e\mapsto 0\}$
SemiGrp
assoc: $\vdash (x\circ y)\circ z=x\circ(y\circ z)$
Abelian
c: $\vdash x\circ y=y\circ x$
Magma
$G, \circ\colon G\to G\to G$
$\vartheta=\{$m $\mapsto$ e, a $\mapsto$ c$\}$
$\psi'=\{i\mapsto -,\ $g $\mapsto$ f$\}$

We will use the foundation-independence (bring-your-own logic) in this course, since the models for the different fragments come with differing logics and foundational theories (together referred to as "foundations"). Logics can be represented as theories in OMDoc/MMT– after all they just introduce language fragments and specify their behavior – and are subject to the same modularity and inheritance regime as domain theories. The only difference is that logics form the meta-language of the domain theories – they provide the language used to talk about the domain – and are thus connected to the domain theories by the meta relation. The next slide gives some details on the construction.

Representing Logics and Foundations as Theories

▷ **Example 3.2.10.** Logics and foundations represented as MMT theories

$$\begin{tikzcd}
 & & & \mathtt{LF} \arrow[r, hook] \arrow[dl, dotted] \arrow[dr, dotted] & \mathtt{LF+X} \\
 & & \mathtt{FOL} \arrow[rr, squiggly, "\mathtt{f2h}"] \arrow[dll, squiggly, bend right, "\mathtt{folsem}"'] \arrow[dll, dotted] \arrow[ddll, dotted] \arrow[dd, dotted] & & \mathtt{HOL} \arrow[dd, dotted] \\
\mathtt{ZFC} & & & & \\
\mathtt{Monoid} \arrow[u, squiggly, "\mathtt{mod}"] \arrow[rr, hook] \arrow[rrrr, bend left, "\mathtt{mult}"] & & \mathtt{CGroup} \arrow[rr, bend right, "\mathtt{add}"] & & \mathtt{Ring}
\end{tikzcd}$$

▷ **Definition 3.2.11.** Meta relation between theories special case of inclusion

▷ **Uniform Meaning Space:** morphisms between formalizations in different logics become possible via meta-morphisms.

▷ *Remark 3.2.12.* Semantics of logics as views into foundations, e.g., `folsem`.

▷ *Remark 3.2.13.* Models represented as views into foundations (e.g. ZFC)

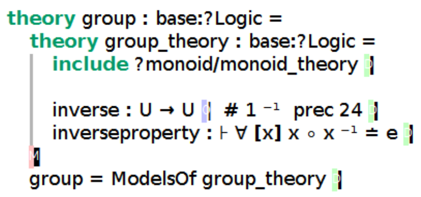
▷ **Example 3.2.14.** $\mathtt{mod} := \{G \mapsto \mathbb{Z}, \circ \mapsto +, e \mapsto 0\}$ interprets Monoid in ZFC.

In the next slide we show the MMT surface language which gives a human-oriented syntax to the OMDoc/MMT format.

A MitM Theory in MMT Surface Language

▷ **Example 3.2.15.** A theory of Groups

- ▷ Declaration $\widehat{=}$ name : type [= Def] [# notation]
- ▷ Axioms $\widehat{=}$ Declaration with type $\vdash F$
- ▷ ModelsOf makes a record type from a theory.



```
theory group : base:?Logic =
  theory group_theory : base:?Logic =
    include ?monoid/monoid_theory ❙

    inverse : U → U ❘ # 1 ⁻¹ prec 24 ❙
    inverseproperty : ⊦ ∀ [x] x ∘ x ⁻¹ ≐ e ❙
  ❚
  group = ModelsOf group_theory ❙
```

▷ **MitM Foundation:** optimized for natural math formulation

- ▷ higher-order logic based on polymorphic λ-calculus
- ▷ judgments-as-types paradigm: $\vdash F \,\widehat{=}$ type of proofs of F
- ▷ dependent types with predicate subtyping, e.g. $\{\mathsf{n}\}\{'\mathsf{a} \in \mathsf{mat}(\mathsf{n},\mathsf{n})|\mathsf{symm}(\mathsf{a})'\}$
- ▷ (dependent) record types for reflecting theories

Finally, we summarize the concepts and features of the OMDoc/MMT.

The MMT Module System

▷ **Central notion:** Theory graph with theory nodes and theory morphisms as edges.

▷ **Definition 3.2.16.** In MMT, a theory is a sequence of constant declarations optionally with type declarations and definitions.

▷ MMT employs the Curry/Howard isomorphism and treats

- ▷ axioms/conjectures as typed symbol declarations (propositions-as-types)
- ▷ inference rules as function types (proof transformers)
- ▷ theorems as definitions (proof terms for conjectures)

▷ **Definition 3.2.17.** MMT has two kinds of theory morphisms

- ▷ structures instantiate theories in a new context (also called: definitional link, import)
  they import theory S into theory T (induces theory morphism $S \to T$)
- ▷ views translate between existing theories (also called: postulated link, theorem link)
  Views transport theorem from source to target (framing).

▷ Together, structures and views allow a very high degree of re-use

▷ **Definition 3.2.18.** We call a statement t induced in a theory T, iff there is

- ▷ a path of theory morphisms from a theory S to T with (joint) assignment σ,
- ▷ such that $t = \sigma(s)$ for some statement s in S.

▷ **Definition 3.2.19.** In MMT, all induced statements have a canonical name, the MMT URI.

3.3 ELPI a Higher-Order Logic Programming Language

ELPI

▷ **Definition 3.3.1.** λProlog, also written lambda Prolog, is a logic programming language featuring polymorphic typing, modular programming, and h i g h e r - o r - d e r f u n c t i o nhigher-order programming.

▷ **Definition 3.3.2.** ELPI implements a variant of λProlog enriched with constraint handling rules.

ELPI by example

▷ **Intuition:** ELPI almost works like Prolog, if we forget the advanced features

▷ **But:** ELPI insists on types declarations for all objects it works with.

▷ **Example 3.3.3 (A Member Predicate).** Indeed in line 1 we see an ELPI type declaration for the ismember predicate. As in Prolog, we use identifiers starting with capital letters for variables. This makes ismember polymorphic in the type T.

```
type ismember T -> list T -> prop.
ismember X [X|_T].
ismember X [_H|T] :- ismember X T.
```

The recursive ismember predicate itself is just as we would write it in Prolog.

As always, we can test this with the queries

- ▷ `ismember 2 [1,2,3]` which succeeds and
- ▷ `ismember 5 [1,2,3]` which fails.

Warning: If you have a functional programming background, you might have expected something like

```
ismember X [] = false
ismember X [X|_T] = true
ismember X [_Y|T] = ismember X T
```

with an explicit failure case. But ELPI/Prolog works differently: It fails unless it finds a way to make it true. For example, in our case `ismember 1 []` fails because there is no rule that makes it work.

You may have to brush up on these lovely Prolog tutorials if this baffles you.

Propositional Logic in ELPI

▷ **Remember:** we wanted to use ELPI to automate proof construction for our target logics.

▷ **Idea:** Let's just start with PL^0 – this is really just like in MMT.

```
kind oo type. % propositions (prop and o are taken)
type neg oo -> oo.
type and oo -> oo -> oo.
type or oo -> oo -> oo.
type impl oo -> oo -> oo.
type true oo.
type false oo.
type pvar int -> oo.
```

The declarations (and their ELPI syntax) should be quite obvious
the pvar function makes a countable collection of propositional variables.

Predicates for Properties of Formulae

▷ **Problem:** We will need to know when a PL^0 formula is atomic later.

▷ **Idea:** It is easier to (first) specify whehter a formula is complex.

```
type complex oo -> prop.
```

```
complex (neg _Y).
complex (and _X _Y).
```

And then we just make atomic to be “not complex”.

▷ **Standard Method:** In ELPI, we use negation as failure: To establish that a term t is atomic we try to establish that it complex and if that succeeds, then we fail. On the other hand, if the first clause of the `atomic` predicate fails, then the second clause (automatically) succeeeds.
Together they switch orchestrate the switch of truth values needed for negation as failure

```
type atomic oo -> prop.
atomic (X) :- complex(X),!,fail.
atomic (_X).
```

The trick now is to guard the `fail` with a cut operator `!`, a literal that forbids the `atomic` predicate to backtrack after it failed. Otherwise the first clause would succeed via the second clause ruining the effect.

Part I

English as a Formal Language: The Method of Fragments

Chapter 4

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 4.3 after we have recapped the necessary concepts about logic.

4.1 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
  $\rightsquigarrow$ Richard Montague's method of fragments (1972).
- ▷ **Definition 4.1.1.** 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 4.1.2.** We call a nonterminal symbol of a context-free grammar a phrasal category. We distinguish two kinds of rules:

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

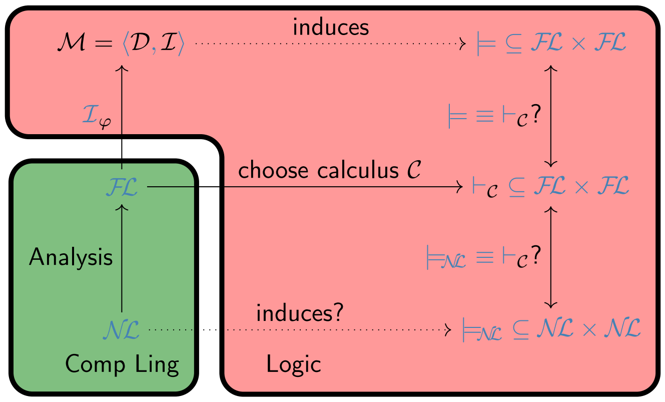
lexical rules: $\mathcal{L}\colon H{\to}t_1 \mid \ldots \mid t_n$, where the t_i are terminals (i.e. NL phrases)

▷ **Definition 4.1.3.** 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


$\mathcal{M} = \langle\mathcal{D},\mathcal{I}\rangle$
induces
$\models \subseteq \mathcal{FL}\times\mathcal{FL}$
$\mathcal{I}_\varphi$
$\models \equiv \vdash_\mathcal{C}$?
$\mathcal{FL}$
choose calculus $\mathcal{C}$
$\vdash_\mathcal{C} \subseteq \mathcal{FL}\times\mathcal{FL}$
Analysis
$\models_{\mathcal{NL}} \equiv \vdash_\mathcal{C}$?
$\mathcal{NL}$
induces?
$\models_{\mathcal{NL}} \subseteq \mathcal{NL}\times\mathcal{NL}$
Comp Ling
Logic

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 4.1.4.** 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 4.1.5.** For a natural language utterance A, we will use $\langle A\rangle$ for the result of translating A.

▷ **Definition 4.1.6 (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)

4.2 What is Logic?

What is Logic?

▷ **Definition 4.2.1.** 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.

4.3 Using Logic to Model Meaning of Natural Language

Modeling Natural Language Semantics

▷ **Problem:** Find formal (logic) system for the meaning of natural language.

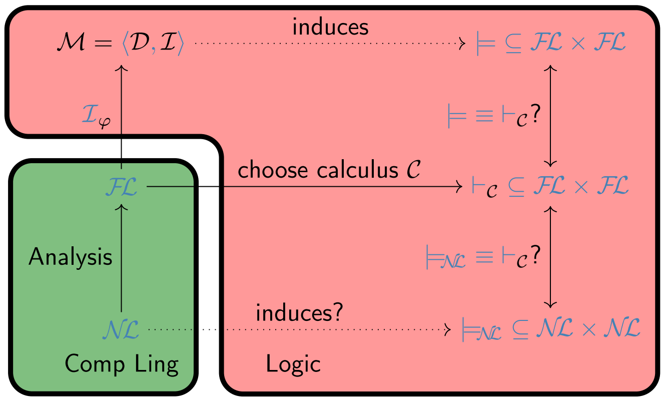
▷ History of ideas

[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

- ▷ 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.* $((\exists X.\mathsf{man}(X) \wedge \mathsf{sleeps}(X))) \wedge \mathsf{snores}(X)$
- ▷ Various dynamic approaches (e.g. DRT, DPL)
  - * *Most men wear black*
- ▷ Higher-order Logic, e.g. generalized quantifiers
- ▷ ...

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?


$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$
induces
$\models \subseteq \mathcal{FL} \times \mathcal{FL}$
$\mathcal{I}_\varphi$
$\models \equiv \vdash_\mathcal{C}$?
$\mathcal{FL}$
choose calculus $\mathcal{C}$
$\vdash_\mathcal{C} \subseteq \mathcal{FL} \times \mathcal{FL}$
Analysis
$\models_{\mathcal{NL}} \equiv \vdash_\mathcal{C}$?
$\mathcal{NL}$
induces?
$\models_{\mathcal{NL}} \subseteq \mathcal{NL} \times \mathcal{NL}$
Comp Ling
Logic

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 $\mathit{wff}(,)$ 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

[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.

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.

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 5

Fragment 1

5.1 The First Fragment: Setting up the Basics

The first fragment will primarily be used for setting the stage, and introducing the method itself. The coverage of the fragment is too small to do anything useful with it, but it will allow us to discuss the salient features of the method, the particular setup of the grammars and semantics before graduating to more useful fragments.

Fragment 1 Data (Sentences we want to cover)

▷ **Fragment 1 Data:** We delineate the intended fragment by giving examples

1. *Ethel kicked the cat and Fiona laughted*
2. *Peter is the teacher*
3. *The teacher is happy*
4. *It is not the case that Bertie ran*
5. *It is not the case that Jo is happy*

▷ We can later use these sentences as benchmark tests.

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG   Michael Kohlhase: LBS   98   2024-01-20   SOME RIGHTS RESERVED

5.1.1 Natural Language Syntax (Fragment 1)

Structural Grammar Rules

▷ **Definition 5.1.1.** $\mathcal{F}_1$ knows the following eight phrasal categories

S	sentence	NP	noun phrase
N	noun	N_{pr}	proper name
V^i	intransitive verb	V^t	transitive verb
conj	connective	Adj	adjective

▷ **Definition 5.1.2.** We have the following production rules in $\mathcal{F}_1$.
$S1$: $S \to \text{NP}, V^i$,
$S2$: $S \to \text{NP}, V^t, \text{NP}$,

$N1$: NP$\to N_{\mathsf{pr}}$,
$N2$: NP$\to$the, N,
$S3$: $S\to$It is not the case that, S,
$S4$: $S\to S$, conj, S,
$S5$: $S\to$NP, is, NP,
$S6$: $S\to$NP, is, Adj

Lexical insertion rules for Fragment 1

▷ **Definition 5.1.3.** We have the following lexical rules in Fragment 1.

$L1$: $N_{\mathsf{pr}}\to$Prudence | Ethel | Chester | Jo | Bertie | Fiona,
$L2$: $N\to$book | cake | cat | golfer | dog | lecturer | student | singer,
$L3$: $V^i\to$ran | laughed | sang | howled | screamed,
$L4$: $V^t\to$read | poisoned | ate | liked | loathed | kicked, $L5$: conj$\to$and | or,
$L6$: Adj$\to$happy | crazy | messy | disgusting | wealthy

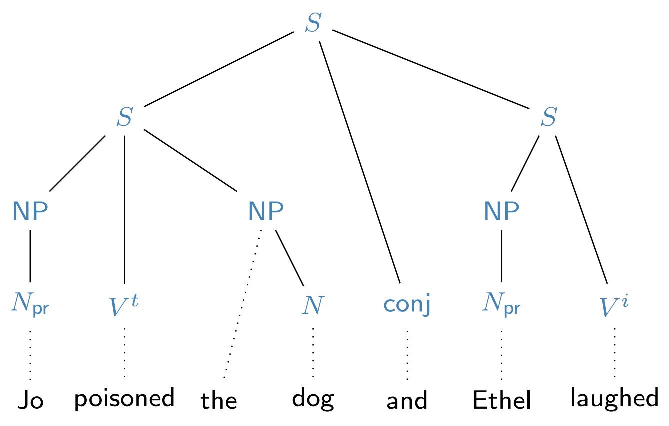
▷ Note: We will adopt the convention that new lexical rules can be generated spontaneously as needed.

These rules represent a simple lexicon, they specify which words are accepted by the grammar and what their phrasal categories are.

Syntax Example: *Jo poisoned the dog and Ethel laughed*

▷ **Observation 5.1.4.** *Jo poisoned the dog and Ethel laughed is a sentence of fragment 1*

▷ We can construct a syntax tree for it!



5.1.2 Predicate Logic without Quantifiers

The next step will be to introduce the logical model we will use for Fragment 1: Predicate Logic without Quantifiers. Syntactically, this logic is a fragment of first-order logic, but it's expressivity is equivalent to propositional logic. Therefore, we will introduce the syntax of full first-order logic (with quantifiers since we will need if for Fragment 4 later), but for the semantics stick with a setup without quantifiers. We will go into the semantic difficulties that they pose later (in section 9.1 and chapter 10).

Individuals and their Properties/Relations

▷ **Observation:** We want to talk about individuals like Stefan, Nicole, and Jochen and their properties, e.g. being blond, or studying AI and relationships, e.g. that *Stefan loves Nicole*.

▷ **Idea:** Re-use PL^0, but replace propositional variables with something more expressive! (instead of fancy variable name trick)

▷ **Definition 5.1.5.** A first-order signature $\langle\Sigma^f, \Sigma^p\rangle$ consists of

- ▷ $\Sigma^f := \bigcup_{k\in\mathbb{N}}\Sigma^f_k$ of function constants, where members of Σ^f_k denote k-ary functions on individuals,
- ▷ $\Sigma^p := \bigcup_{k\in\mathbb{N}}\Sigma^p_k$ of predicate constants, where members of Σ^p_k denote k-ary relations among individuals,

where Σ^f_k and Σ^p_k are pairwise disjoint, countable sets of symbols for each $k\in\mathbb{N}$.

▷ **Definition 5.1.6.** The formulae of PL^{nq} are given by the following grammar

function constants	f^k	$\in$	Σ^f_k	
predicate constants	p^k	$\in$	Σ^p_k	
terms	t	$::=$	f^0	constant
		$\mid$	$f^k(t_1,\ldots,t_k)$	application
formulae	$\mathbf{A}$	$::=$	$p^k(t_1,\ldots,t_k)$	atomic
		$\mid$	$\neg\mathbf{A}$	negation
		$\mid$	$\mathbf{A}_1\wedge\mathbf{A}_2$	conjunction

PL^{nq} Semantics

▷ **Definition 5.1.7.** Domains $\mathcal{D}_0 = \{\mathsf{T},\mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.

▷ **Definition 5.1.8.** Interpretation $\mathcal{I}$ assigns values to constants, e.g.

- ▷ $\mathcal{I}(\neg)\colon \mathcal{D}_0\to\mathcal{D}_0; \mathsf{T}\mapsto\mathsf{F}; \mathsf{F}\mapsto\mathsf{T}$ and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
- ▷ $\mathcal{I}\colon \Sigma^f_0\to\mathcal{D}_\iota$ (interpret individual constants as individuals)
- ▷ $\mathcal{I}\colon \Sigma^f_k\to{\mathcal{D}_\iota}^k\to\mathcal{D}_\iota$ (interpret function constants as functions)
- ▷ $\mathcal{I}\colon \Sigma^p_k\to\mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)

▷ **Definition 5.1.9.** The value function $\mathcal{I}$ assigns values to formulae: (recursively)

- ▹ $\mathcal{I}(f(\mathbf{A}^1, \ldots, \mathbf{A}^k)) := \mathcal{I}(f)(\mathcal{I}(\mathbf{A}^1), \ldots, \mathcal{I}(\mathbf{A}^k))$
- ▹ $\mathcal{I}(p(\mathbf{A}^1, \ldots, \mathbf{A}^k)) := \mathsf{T}$, iff $\langle \mathcal{I}(\mathbf{A}^1), \ldots, \mathcal{I}(\mathbf{A}^k) \rangle \in \mathcal{I}(p)$
- ▹ $\mathcal{I}(\neg \mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}(\mathbf{A}))$ and $\mathcal{I}(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}(\mathbf{A}), \mathcal{I}(\mathbf{G}))$ (just as in PL^0)

▷ **Definition 5.1.10.** Model: $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.

▷ **Theorem 5.1.11.** *PL^{nq} is isomorphic to PL^0 (interpret atoms as prop. variables)*

A Model for PL^{nq}

▷ **Example 5.1.12.** Let $L := \{a, b, c, d, e, P, Q, R, S\}$, we set the universe $\mathcal{D} := \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$, and specify the interpretation function $\mathcal{I}$ by setting

- ▹ $a \mapsto \clubsuit$, $b \mapsto \spadesuit$, $c \mapsto \heartsuit$, $d \mapsto \diamondsuit$, and $e \mapsto \diamondsuit$ for constants,
- ▹ $P \mapsto \{\clubsuit, \spadesuit\}$ and $Q \mapsto \{\spadesuit, \diamondsuit\}$, for unary predicate constants.
- ▹ $R \mapsto \{\langle \heartsuit, \diamondsuit \rangle, \langle \diamondsuit, \heartsuit \rangle\}$, and $S \mapsto \{\langle \diamondsuit, \spadesuit \rangle, \langle \spadesuit, \clubsuit \rangle\}$ for binary predicate constants.

▷ **Example 5.1.13 (Computing Meaning in this Model).**

- ▹ $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{T}$, iff
- ▹ $\mathcal{I}(R(a,b)) = \mathsf{T}$ and $\mathcal{I}(P(c)) = \mathsf{T}$, iff
- ▹ $\langle \mathcal{I}(a), \mathcal{I}(b) \rangle \in \mathcal{I}(R)$ and $\mathcal{I}(c) \in \mathcal{I}(P)$, iff
- ▹ $\langle \clubsuit, \spadesuit \rangle \in \{\langle \heartsuit, \diamondsuit \rangle, \langle \diamondsuit, \heartsuit \rangle\}$ and $\heartsuit \in \{\clubsuit, \spadesuit\}$

So, $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{F}$.

PL^{nq} and PL^0 are Isomorphic

▷ **Observation:** For every choice of Σ of signature, the set $\mathcal{A}_\Sigma$ of atomic PL^{nq} formulae is countable, so there is a $\mathcal{V}_\Sigma \subseteq \mathcal{V}_0$ and a bijection $\theta_\Sigma \colon \mathcal{A}_\Sigma \to \mathcal{V}_\Sigma$.

θ_Σ can be extended to formulae as PL^{nq} and PL^0 share connectives.

▷ **Lemma 5.1.14.** *For every model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$, there is a variable assignment $\varphi_\mathcal{M}$, such that $\mathcal{I}_{\varphi_\mathcal{M}}(\mathbf{A}) = \mathcal{I}(\mathbf{A})$.*

▷ *Proof sketch:* We just define $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$

▷ **Lemma 5.1.15.** *For every variable assignment $\psi \colon \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\}$ there is a model $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$, such that $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

▷ *Proof sketch:* see next slide

▷ **Corollary 5.1.16.** *PL^{nq} is isomorphic to PL^0, i.e. the following diagram commutes:*

$$\begin{tikzcd}
\langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle & \mathcal{V}_\Sigma \to \{\mathsf{T},\mathsf{F}\} \arrow[l, "\psi \mapsto \mathcal{M}^\psi"'] \\
PL^{nq}(\Sigma) \arrow[u, "\mathcal{I}^\psi()"] \arrow[r, "\theta_\Sigma"] & PL^0(\mathcal{A}_\Sigma) \arrow[u, "\mathcal{I}_{\varphi_\mathcal{M}}()"']
\end{tikzcd}$$

▷ **Note:** This constellation with a language isomorphism and a corresponding model isomorphism (in converse direction) is typical for a logic isomorphism.

Valuation and Satisfiability

▷ **Lemma 5.1.17.** *For every variable assignment $\psi\colon \mathcal{V}_\Sigma \to \{\mathsf{T},\mathsf{F}\}$ there is a model $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$, such that $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

▷ *Proof:* We construct $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$ and show that it works as desired.

1. Let $\mathcal{D}^\psi$ be the set of PL^{nq} terms over Σ, and
   - ▷ $\mathcal{I}^\psi(f)\colon {\mathcal{D}_\iota}^k \to {\mathcal{D}^\psi}^k; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto f(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ for $f \in \Sigma_k^f$
   - ▷ $\mathcal{I}^\psi(p) := \{\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mid \psi(\theta_\psi^{-1} p(\mathbf{A}_1, \ldots, \mathbf{A}_k)) = \mathsf{T}\}$ for $p \in \Sigma^p$.
2. We show $\mathcal{I}^\psi(\mathbf{A}) = \mathbf{A}$ for terms $\mathbf{A}$ by induction on $\mathbf{A}$
   2.1. If $\mathbf{A} = c$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(c) = c = \mathbf{A}$
   2.2. If $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ then
   $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(f)(\mathcal{I}(\mathbf{A}_1), \ldots, \mathcal{I}(\mathbf{A}_n)) = \mathcal{I}^\psi(f)(\mathbf{A}_1, \ldots, \mathbf{A}_k) = \mathbf{A}$.
3. For a PL^{nq} formula $\mathbf{A}$ we show that $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ by induction on $\mathbf{A}$.
   3.1. If $\mathbf{A} = p(\mathbf{A}_1, \ldots, \mathbf{A}_k)$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(p)(\mathcal{I}(\mathbf{A}_1), \ldots, \mathcal{I}(\mathbf{A}_n)) = \mathsf{T}$, iff $\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \in \mathcal{I}^\psi(p)$, iff $\psi(\theta_\psi^{-1} \mathbf{A}) = \mathsf{T}$, so $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ as desired.
   3.2. If $\mathbf{A} = \neg \mathbf{B}$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}^\psi(\mathbf{B}) = \mathsf{F}$, iff $\mathcal{I}^\psi(\mathbf{B}) = \mathcal{I}_\psi(\mathbf{B})$, iff $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$.
   3.3. If $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$ then we argue similarly
4. Hence $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ for all PL^{nq} formulae and we have concluded the proof.

Now that we have the target logic we can complete the analysis arrow in slide 93. We do this again, by giving transformation rules.

5.1.3 Natural Language Semantics via Translation

Translation rules for non-basic expressions (NP and S)

▷ **Definition 5.1.18.** We have the following translation rules for non-leaf node of the abstract syntax tree

$T1$: $[X_{\mathsf{NP}}, Y_{V^i}]_S \rightsquigarrow Y'(X')$
$T2$: $[X_{\mathsf{NP}}, Y_{V^t}, Z_{\mathsf{NP}}]_S \rightsquigarrow Y'(X', Z')$
$T3$: $[X_{N_{\mathsf{pr}}}]_{\mathsf{NP}} \rightsquigarrow X'$
$T4$: $[\text{the}, X_N]_{\mathsf{NP}} \rightsquigarrow theX'$
$T5$: $[\text{It is not the case that} X_S]_S \rightsquigarrow (\neg X')$

$T6$: $[X_S, Y_{\mathsf{conj}}, Z_S]_S \leadsto Y'(X', Z')$
$T7$: $[X_{\mathsf{NP}}, \mathsf{is}, Y_{\mathsf{NP}}]_S \leadsto X' = Y'$
$T8$: $[X_{\mathsf{NP}}, \mathsf{is}, Y_{\mathsf{Adj}}]_S \leadsto Y'(X')$

Read e.g. $[Y, Z]_X$ as a node with label X in the syntax tree with children X and Y. Read X' as the translation of X via these rules.

▷ Note that we have exactly one translation per syntax rule.

Translation rule for basic lexical items

▷ **Definition 5.1.19.** The target logic for $\mathcal{F}_1$ is $\mathrm{PL}^{\mathrm{nq}}$, the fragment of PL^1 without quantifiers.

▷ **Lexical Translation Rules for $\mathcal{F}_1$ Categories:**

- ▷ If w is a proper name, then $w' \in \Sigma_0^f$. (individual constant)
- ▷ If w is an intransitive verb, then $w' \in \Sigma_1^p$. (one-place predicate)
- ▷ If w is a transitive verb, $w' \in \Sigma_2^p$. (two-place predicate)
- ▷ If w is a noun phrase, then $w' \in \Sigma_0^f$. (individual constant)

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

▷ For any non-logical word w, we have the "pseudo-rule" $t1$: $w \leadsto w'$.

▷ Note: This rule does not apply to the syncategorematic items is and the.

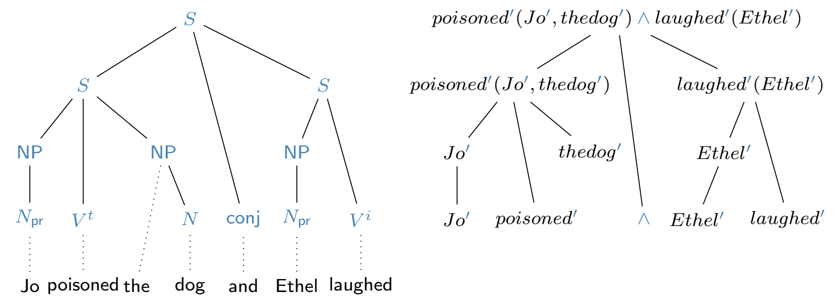
▷ Translations for logical connectives

$$t2\colon \text{and} \leadsto \wedge,\ t3\colon \text{or} \leadsto \vee,\ t4\colon \text{it is not the case that} \leadsto \neg$$

Translation Example

▷ **Observation 5.1.20.** *Jo poisoned the dog and Ethel laughed is a sentence of Fragment 1*

▷ We can construct a syntax tree for it!



5.2 Testing Truth Conditions via Inference

Testing Truth Conditions in PL^{nq}

▷ **Idea 1:** To test our language model ($\mathcal{F}_1$)

- ▷ Select a sentence S and a situation W that makes S true. (according to humans)
- ▷ Translate S in to a formula S' in PL^{nq}.
- ▷ Express W as a set Φ of formulae in PL^{nq} ($\Phi \mathrel{\hat{=}}$ truth conditions)
- ▷ Our language model is supported if $\Phi \models S'$, falsified if $\Phi \not\models S'$.

▷ **Example 5.2.1 (John chased the gangster in the red sports car).**

- ▷ We claimed that we have three readings Example 2.3.3
$R_1{:=}c(j,g) \wedge in(j,s)$, $R_2{:=}c(j,g) \wedge in(g,s)$, and $R_3{:=}c(j,g) \wedge in(j,s) \wedge in(g,s)$
- ▷ So there must be three distinct situations W that make S true

1. *John is in the red sports car, but the gangster isn't*
$W_1{:=}c(j,g) \wedge in(j,s) \wedge \neg in(g,s)$, so $W_1 \models R_1$, but $W_1 \not\models R_2$ and $W_1 \not\models R_3$
2. *The gangster is in the red sports car, but John isn't*
$W_2{:=}c(j,g) \wedge in(j,s) \wedge \neg in(g,s)$, so $W_2 \models R_2$, but $W_2 \not\models R_1$ and $W_2 \not\models R_3$
3. *Both are in the red sports car*
$\hat{=}$ they run around on the back seat of a very big sports car
$W_3{:=}c(j,g) \wedge in(j,s) \wedge in(g,s)$, so $W_3 \models R_3$, but $W_3 \not\models R_1$ and $W_3 \not\models R_1$

▷ **Idea 2:** Use a calculus to model $\models$, e.g. $\mathcal{ND}\_0$

Fragment 1

▷ Fragment $\mathcal{F}_1$ of English (defined by grammar + lexicon)

▷ Logic PL^{nq} (serves as a mathematical model for $\mathcal{F}_1$)

- ▷ Formal Language (individuals, predicates, $\neg, \wedge, \vee, \Rightarrow$)

- ▷ Semantics $\mathcal{I}_\varphi$ defined recursively on formula structure (⇝ validity, entailment)
- ▷ Tableau calculus for validity and entailment (CALCULEMUS!)

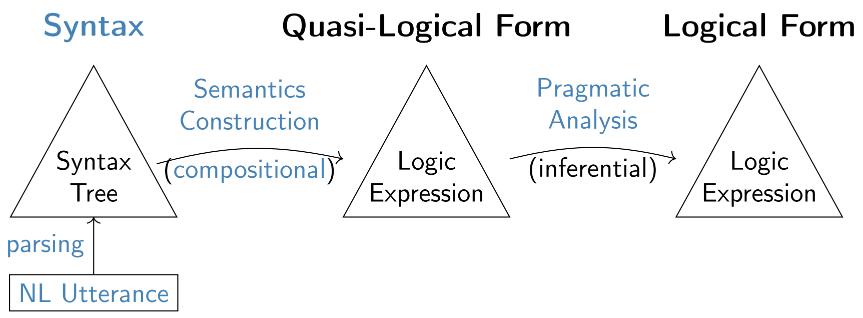
▷ Analysis function $\mathcal{F}_1$ ⇝ PL^{nq} (Translation)

▷ Test the model by checking predictions (calculate truth conditions)

▷ **Coverage:** Extremely Boring! (accounts for 0 examples from the intro) but the conceptual setup is fascinating

Summary: The Interpretation Process

▷ **Interpretation Process:**



Chapter 6

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 chapter 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`.

6.1 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 65.

- ▷ Verbs: $V^t \mathrel{\widehat{=}}$ `V2`, $V^i \mathrel{\widehat{=}}$ `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);`

6.2 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 Example 2.3.3.

```
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.

Embedding GF into MMT

▷ **Observation:** The GF system provides Java bindings and MMT is programed in Scala, which compiles into the Java virtual machine.

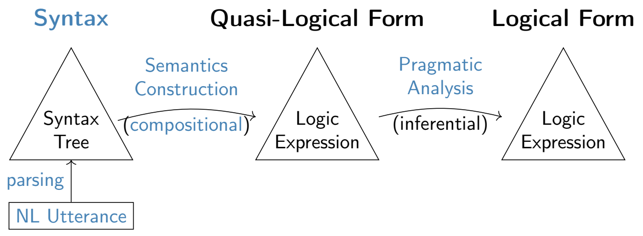
▷ **Idea:** Use GF as a sophisticated NL-parser/generator for MMT

⇝ MMT with a natural language front-end.

⇝ GF with a multi-logic back-end

▷ **Definition 6.2.1.** 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.

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 6.2.2.** 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
```

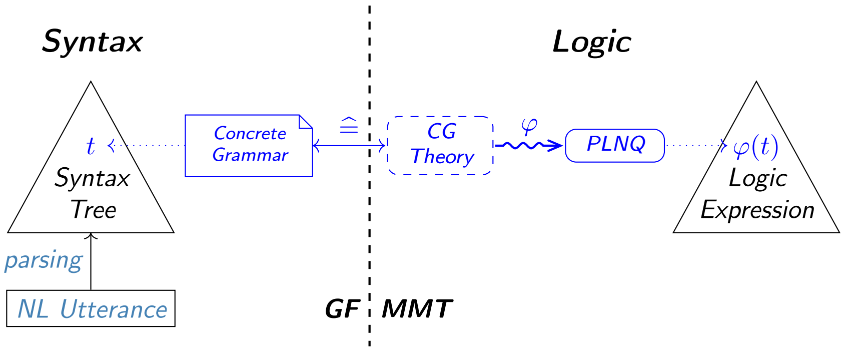
putting it all together

```
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 6.2.3.** *We can express semantics construction as an MMT view*



▷ **Example 6.2.4.** 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
```

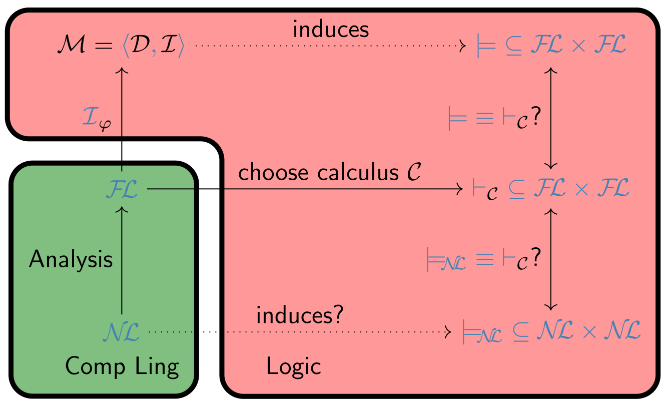
Montague-Style Processing of $\mathcal{F}_1$ in GLF

▷ **Example 6.2.5.** *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


$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$
induces
$\models \subseteq \mathcal{FL} \times \mathcal{FL}$
$\mathcal{I}_\varphi$
$\models \equiv \vdash_\mathcal{C}$?
choose calculus $\mathcal{C}$
$\mathcal{FL}$
$\vdash_\mathcal{C} \subseteq \mathcal{FL} \times \mathcal{FL}$
Analysis
$\models_{\mathcal{NL}} \equiv \vdash_\mathcal{C}$?
induces?
$\mathcal{NL}$
$\models_{\mathcal{NL}} \subseteq \mathcal{NL} \times \mathcal{NL}$
Comp Ling
Logic

- ▷ The GF grammar for $\mathcal{F}_1$ defines the fragment $\mathcal{NL}$.

- The MMT implementation of PL^{nq} is $\mathcal{FL}$.
- The MMT view implements the compositional translation function for $\mathcal{F}_1$

6.3 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 $\rightsquigarrow$ `c : ⊢A` $\widehat{=}$ an axiom
  - a definition `c : ⊢A |= P` does as well but also exhibits a "proof" `P` $\rightsquigarrow$ `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:

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

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

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

from **??** as the MMT declaration

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

Chapter 7

Adding Context: Pronouns and World Knowledge

In this chapter we will extend the model generation system by facilities for dealing with world knowledge and pronouns. We want to cover discourses like *Peter loves Fido. Even though he bites him sometimes*. As we already observed there, we crucially need a notion of context which determines the meaning of the pronoun. Furthermore, the example shows us that we will need to take into account world knowledge as A way to integrate world knowledge to filter out one interpretation, i.e. *Humans don't bite dogs.*

7.1 Fragment 2: Pronouns and Anaphora

Fragment 2 ($\mathcal{F}_2 \widehat{=} \mathcal{F}_1$ + Pronouns)

▷ **Want to cover:** *Peter loves Fido. He bites him.* (almost intro)

- ▷ **We need**: Translation and interpretation for *he*, *she*, *him*,. . . .
- ▷ **Also**: A way to integrate world knowledge to filter out one interpretation (i.e. *Humans don't bite dogs.*)

▷ **Idea:** Integrate variables into $\mathrm{PL^{nq}}$ (work backwards from that)

▷ **Logical System:** $\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}} = \mathrm{PL^{nq}} +$ variables (Translate pronouns to variables)

New Grammar in $\mathcal{F}_2$ (Pronouns)

▷ **Definition 7.1.1.** We have the following structural grammar rules in $\mathcal{F}_2$

$$S1\colon S \to \mathrm{NP}, V^i,$$
$$S2\colon S \to \mathrm{NP}, V^t, \mathrm{NP},$$
$$N1\colon \mathrm{NP} \to N_{\mathrm{pr}},$$
$$N2\colon \mathrm{NP} \to \mathrm{Pron},$$
$$N3\colon \mathrm{NP} \to \mathrm{the}, N,$$

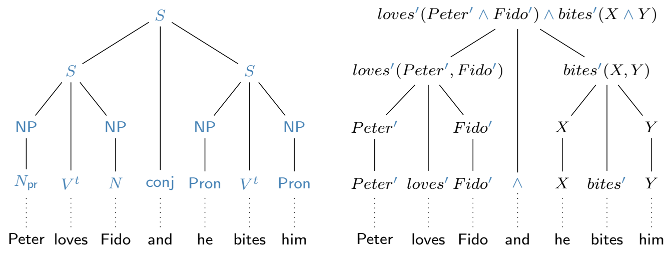
$$S3\colon S \to \text{it is not the case that}, S,$$
$$S4\colon S \to S, \text{conj}, S,$$
$$S5\colon S \to \text{NP}, \text{is}, \text{NP},$$
$$S6\colon S \to \text{NP}, \text{is}, \text{Adj}$$

and one additional lexical rule:

$$L7\colon \text{Pron} \to he \mid she \mid it \mid we \mid they$$

Translation for $\mathcal{F}_2$ (first attempt)

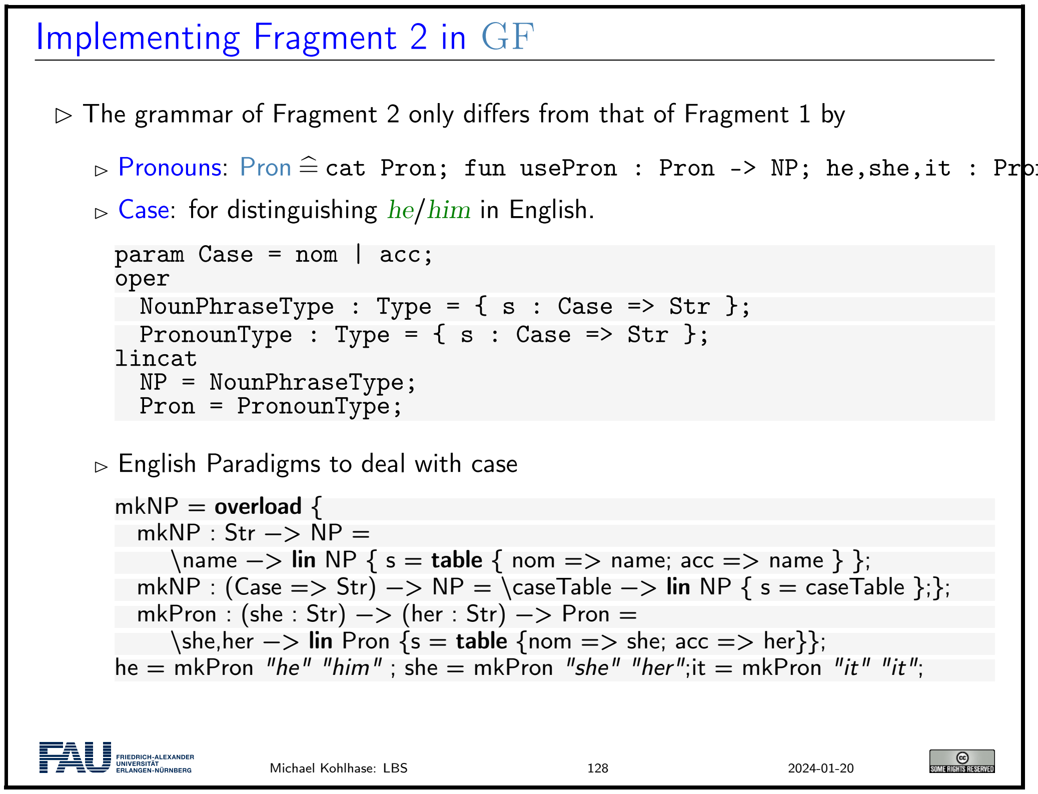
- **Idea:** Pronouns are translated into new variables (so far)
- The syntax/semantic trees for *Peter loves Fido and he bites him.* are straightforward. (almost intro)



S
S
S
NP
NP
NP
NP
N_{pr}
V^t
N
conj
Pron
V^t
Pron
Peter loves Fido and he bites him
$loves'(Peter' \wedge Fido') \wedge bites'(X \wedge Y)$
$loves'(Peter', Fido')$
$bites'(X, Y)$
$Peter'$
$Fido'$
X
Y
$Peter'$ $loves'$ $Fido'$ $\wedge$ X $bites'$ Y
Peter loves Fido and he bites him

Predicate Logic with Variables (but no Quantifiers)

- **Definition 7.1.2 (Logical System $\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}}$).** $\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}} := \mathrm{PL}^{\mathrm{nq}}$ + variables
- **Definition 7.1.3 ($\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}}$ Syntax).**
  Category $\mathcal{V} = \{X, Y, Z, X^1, X^2, \ldots\}$ of variables (allow variables wherever individual constants were allowed)
- **Definition 7.1.4 ($\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}}$ Semantics).** Model $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ (need to evaluate variables)
  - variable assignment: $\varphi\colon \mathcal{V}_\iota \to U$
  - value function: $\mathcal{I}_\varphi(X) = \varphi(X)$ (defined like $\mathcal{I}$ elsewhere)
  - call a $\mathrm{PL}^{\mathcal{V}}_{\mathrm{NQ}}$ formula $\mathbf{A}$ valid in $\mathcal{M}$ under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$,
  - call it satisfiable in $\mathcal{M}$, iff there is a variable assignment φ, such that $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$



Implementing Fragment 2 in GF

- ▷ The grammar of Fragment 2 only differs from that of Fragment 1 by
  - ▷ Pronouns: Pron $\widehat{=}$ `cat Pron; fun usePron : Pron -> NP; he,she,it : Pron;`,
  - ▷ Case: for distinguishing *he*/*him* in English.

```
param Case = nom | acc;
oper
  NounPhraseType : Type = { s : Case => Str };
  PronounType : Type = { s : Case => Str };
lincat
  NP = NounPhraseType;
  Pron = PronounType;
```

  - ▷ English Paradigms to deal with case

```
mkNP = overload {
  mkNP : Str −> NP =
     \name −> lin NP { s = table { nom => name; acc => name } };
  mkNP : (Case => Str) −> NP = \caseTable −> lin NP { s = caseTable };};
  mkPron : (she : Str) −> (her : Str) −> Pron =
     \she,her −> lin Pron {s = table {nom => she; acc => her}};
he = mkPron "he" "him" ; she = mkPron "she" "her";it = mkPron "it" "it";
```

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 128 2024-01-20 SOME RIGHTS RESERVED

7.2 A Tableau Calculus for PLNQ with Free Variables

The main idea here is to extend the fragment of first-order logic we use as a model for natural language to include free variables, and assume that pronouns like *he*, *she*, *it*, and *they* are translated to distinct free variables i.e. every occurrance of a pronoun to a new variable. Note that we do not allow quantifiers yet that will come in [1], as quantifiers will pose new problems, and we can already solve some linguistically interesting problems without them. EdN:1

To allow for world knowledge, we generalize the notion of an initial tableau [2]. Instead of allowing only the initial signed formula at the root node, we allow a linear tree whose nodes are labeled with signed formulae representing the world knowledge. As the world knowledge resides in the initial tableau (intuitively before all input), we will also speak of background knowledge. EdN:2

We will use free variables for two purposes in our new fragment. Free variables in the input will stand for pronouns, their value will be determined by random instantiation. Free variables in the world knowledge allow us to express schematic knowledge. For instance, if we want to express *Humans don't bite dogs.*, then we can do this by the formula $\mathrm{human}(X)\wedge\mathrm{dog}(Y)\Rightarrow\neg\mathrm{bites}(X,Y)$.

Of course we will have to extend our tableau calculus with new inference rules for the new language capabilities.

7.2.1 Calculi for Automated Theorem Proving: Analytical Tableaux

In this section we will introduce tableau calculi for propositional logics. To make the reasoning procedure more interesting, we will use first-order predicate logic without variables, function symbols and quantifiers as a basis. This logic (we will call it $\mathrm{PL}^{\mathrm{nq}}$) allows us express simple natural language sentences and to re-use our grammar for experimentation, without introducing the whole complications of first-order inference.

[1] EdNote: crossref
[2] EdNote: crossref

The logic $\mathrm{PL}^{\mathrm{nq}}$ is equivalent to propositional logic in expressivity: atomic formulae take the role of propositional variables.

Instead of deducing new formulae from axioms (and hypotheses) and hoping to arrive at the desired theorem, we try to deduce a contradiction from the negation of the theorem. Indeed, a formula $\mathbf{A}$ is valid, iff $\neg\mathbf{A}$ is unsatisfiable, so if we derive a contradiction from $\neg\mathbf{A}$, then we have proven $\mathbf{A}$. The advantage of such "test-calculi" (also called negative calculi) is easy to see. Instead of finding a proof that ends in $\mathbf{A}$, we have to find any of a broad class of contradictions. This makes the calculi that we will discuss now easier to control and therefore more suited for mechanization.

7.2.1.1 Analytical Tableaux

Before we can start, we will need to recap some nomenclature on formulae.

Recap: Atoms and Literals

▷ **Definition 7.2.1.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.

▷ **Definition 7.2.2.** We call a pair $\mathbf{A}^\alpha$ of a formula and a truth value $\alpha\in\{\mathsf{T},\mathsf{F}\}$ a labeled formula. For a set Φ of formulae we use $\Phi^\alpha{:=}\{\mathbf{A}^\alpha|\mathbf{A}\in\Phi\}$.

▷ **Definition 7.2.3.** A labeled atom $\mathbf{A}^\alpha$ is called a (positive if $\alpha=\mathsf{T}$, else negative) literal.

▷ **Intuition:** To satisfy a formula, we make it "true". To satisfy a labeled formula $\mathbf{A}^\alpha$, it must have the truth value α.

▷ **Definition 7.2.4.** For a literal $\mathbf{A}^\alpha$, we call the literal $\mathbf{A}^\beta$ with $\alpha\neq\beta$ the opposite literal (or partner literal).

The idea about literals is that they are atoms (the simplest formulae) that carry around their intended truth value.

Alternative Definition: Literals

▷ **Note:** Literals are often defined without recurring to labeled formulae:

▷ **Definition 7.2.5.** A literal is an atom $\mathbf{A}$ (positive literal) or negated atom $\neg\mathbf{A}$ (negative literal). $\mathbf{A}$ and $\neg\mathbf{A}$ are opposite literals.

▷ **Note:** This notion of literal is equivalent to the labeled formulae-notion of literal, but does not generalize as well to logics with more than two truth values.

Test Calculi: Tableaux and Model Generation

▷ **Idea:** A tableau calculus is a test calculus that

- ▷ analyzes a labeled formulae in a tree to determine satisfiability,
- ▷ its branches correspond to valuations ($\rightsquigarrow$ models).

▷ **Example 7.2.6.** Tableau calculi try to construct models for labeled formulae:

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P \wedge Q \Rightarrow Q \wedge P$	$\models P \wedge (Q \vee \neg R) \wedge \neg Q$
$\begin{array}{c} (P \wedge Q \Rightarrow Q \wedge P)^{\mathsf{F}} \\ (P \wedge Q)^{\mathsf{T}} \\ (Q \wedge P)^{\mathsf{F}} \\ P^{\mathsf{T}} \\ Q^{\mathsf{T}} \\ \begin{array}{c\|c} P^{\mathsf{F}} & Q^{\mathsf{F}} \\ \bot & \bot \end{array} \end{array}$	$\begin{array}{c} (P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}} \\ (P \wedge (Q \vee \neg R))^{\mathsf{T}} \\ \neg Q^{\mathsf{T}} \\ Q^{\mathsf{F}} \\ P^{\mathsf{T}} \\ (Q \vee \neg R)^{\mathsf{T}} \\ \begin{array}{c\|c} Q^{\mathsf{T}} & \neg R^{\mathsf{T}} \\ \bot & R^{\mathsf{F}} \end{array} \end{array}$
No Model	Herbrand Model $\{P^{\mathsf{T}}, Q^{\mathsf{F}}, R^{\mathsf{F}}\}$ $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}, R \mapsto \mathsf{F}\}$

▷ **Idea:** Open branches in saturated tableaux yield models.

▷ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)

▷ Satisfiable, iff there are open branches (correspond to models)

Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with exponents that hold the intended truth value.
On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction $\bot$.

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T. This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one, which corresponds a model).

Now that we have seen the examples, we can write down the tableau rules formally.

Analytical Tableaux (Formal Treatment of $\mathcal{T}_0$)

▷ **Idea:** A test calculus where

▷ A labeled formula is analyzed in a tree to determine satisfiability,

▷ branches correspond to valuations (models)

▷ **Definition 7.2.7.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(\mathbf{A} \wedge \mathbf{B})^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}} \\ \mathbf{B}^{\mathsf{T}}\end{array}}\mathcal{T}_0\wedge \qquad \frac{(\mathbf{A} \wedge \mathbf{B})^{\mathsf{F}}}{\mathbf{A}^{\mathsf{F}} \;\big|\; \mathbf{B}^{\mathsf{F}}}\mathcal{T}_0\vee \qquad \frac{\neg \mathbf{A}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}}}\mathcal{T}_0\neg^{\mathsf{T}} \qquad \frac{\neg \mathbf{A}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}}\mathcal{T}_0\neg^{\mathsf{F}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha} \\ \mathbf{A}^{\beta}\end{array} \quad \alpha \neq \beta}{\bot}\mathcal{T}_0\bot$$

Use rules exhaustively as long as they contribute new material ($\rightsquigarrow$ termination)

▷ **Definition 7.2.8.** We call any tree ($\big|$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.

▷ **Definition 7.2.9.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in $\perp$, else open. A tableau is closed, iff all of its branches are.

These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol $\perp$ (for unsatisfiability) to a branch.
We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).
Definition 7.2.10. We will call a closed tableau with the labeled formula $\mathbf{A}^\alpha$ at the root a tableau refutation for $\mathcal{A}^\alpha$.
The saturated tableau represents a full case analysis of what is necessary to give $\mathbf{A}$ the truth value α; since all branches are closed (contain contradictions) this is impossible.

Analytical Tableaux ($\mathcal{T}_0$ continued)

▷ **Definition 7.2.11 ($\mathcal{T}_0$-Theorem/Derivability).** $\mathbf{A}$ is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^\mathsf{F}$ at the root.

$\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives $\mathbf{A}$ in $\mathcal{T}_0$ ($\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau starting with $\mathbf{A}^\mathsf{F}$ and Φ^T. The tableau with only a branch of $\mathbf{A}^\mathsf{F}$ and Φ^T is called initial for $\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$.

Definition 7.2.12. We will call a tableau refutation for $\mathbf{A}^\mathsf{F}$ a tableau proof for $\mathbf{A}$, since it refutes the possibility of finding a model where $\mathbf{A}$ evaluates to F. Thus $\mathbf{A}$ must evaluate to T in all models, which is just our definition of validity.
Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the propositional Hilbert calculus it does not prove a theorem $\mathbf{A}$ by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.
We have rules for all the necessary connectives (we restrict ourselves to $\wedge$ and $\neg$, since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg\mathbf{A} \vee \mathbf{B}, \ldots$.)
We now look at a formulation of propositional logic with fancy variable names. Note that $\mathrm{loves}(\mathrm{mary}, \mathrm{bill})$ is just a variable name like P or X, which we have used earlier.

A Valid Real-World Example

▷ **Example 7.2.13.** *If Mary loves Bill and John loves Mary, then John loves Mary*

$$
\begin{array}{c}
(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}) \Rightarrow \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
\neg(\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary})) \wedge \neg\mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
(\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary})) \wedge \neg\mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\neg\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{mary},\mathsf{bill})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{F}}\\
\bot
\end{array}
$$

This is a closed tableau, so the $\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}) \Rightarrow \mathsf{loves}(\mathsf{john},\mathsf{mary})$ is a $\mathcal{T}_0$-theorem.

As we will see, $\mathcal{T}_0$ is sound and complete, so

$$\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john},\mathsf{mary}) \Rightarrow \mathsf{loves}(\mathsf{john},\mathsf{mary})$$

is valid.

We could have used the unsatisfiability theorem (??) here to show that *If Mary loves Bill and John loves Mary* entails *John loves Mary*. But there is a better way to show entailment: we directly use derivability in $\mathcal{T}_0$.

Deriving Entailment in $\mathcal{T}_0$

▷ **Example 7.2.14.** *Mary loves Bill* and *John loves Mary* together entail that *John loves Mary*

$$
\begin{array}{c}
\mathsf{loves}(\mathsf{mary},\mathsf{bill})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{F}}\\
\bot
\end{array}
$$

This is a closed tableau, so $\{\mathsf{loves}(\mathsf{mary},\mathsf{bill}), \mathsf{loves}(\mathsf{john},\mathsf{mary})\} \vdash_{\mathcal{T}_0} \mathsf{loves}(\mathsf{john},\mathsf{mary})$.

Again, as $\mathcal{T}_0$ is sound and complete we have

$$\{\mathsf{loves}(\mathsf{mary},\mathsf{bill}), \mathsf{loves}(\mathsf{john},\mathsf{mary})\} \models \mathsf{loves}(\mathsf{john},\mathsf{mary})$$

Note: We can also use the tableau calculus to try and show entailment (and fail). The nice thing is that the failed proof, we can see what went wrong.

A Falsifiable Real-World Example

▷ **Example 7.2.15.** * *If Mary loves Bill or John loves Mary, then John loves Mary*

Try proving the implication (this fails)

$$\begin{array}{c}
((\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary})) \Rightarrow \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
\neg(\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary})) \land \neg\mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
(\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary})) \land \neg\mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\neg\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{F}}\\
\neg\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\neg(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{F}}\\
(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\begin{array}{c|c}
\mathsf{loves}(\mathsf{mary},\mathsf{bill})^{\mathsf{T}} & \mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}$$

Indeed we can make $\mathcal{I}_\varphi(\mathsf{loves}(\mathsf{mary},\mathsf{bill})) = \mathsf{T}$ but $\mathcal{I}_\varphi(\mathsf{loves}(\mathsf{john},\mathsf{mary})) = \mathsf{F}$.

Obviously, the tableau above is saturated, but not closed, so it is not a tableau proof for our initial entailment conjecture. We have marked the literal on the open branch green, since they allow us to read of the conditions of the situation, in which the entailment fails to hold. As we intuitively argued above, this is the situation, where *Mary loves Bill*. In particular, the open branch gives us a variable assignment (marked in green) that satisfies the initial formula. In this case, *Mary loves Bill*, which is a situation, where the entailment fails.
Again, the derivability version is much simpler:

Testing for Entailment in $\mathcal{T}_0$

▷ **Example 7.2.16.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$\begin{array}{c}
(\mathsf{loves}(\mathsf{mary},\mathsf{bill}) \lor \mathsf{loves}(\mathsf{john},\mathsf{mary}))^{\mathsf{T}}\\
\mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{F}}\\
\begin{array}{c|c}
\mathsf{loves}(\mathsf{mary},\mathsf{bill})^{\mathsf{T}} & \mathsf{loves}(\mathsf{john},\mathsf{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\mathsf{loves}(\mathsf{mary},\mathsf{bill})) = \mathsf{T}$ but $\mathcal{I}_\varphi(\mathsf{loves}(\mathsf{john},\mathsf{mary})) = \mathsf{F}$ falsifies the derivability/entailment conjecture.

We have seen in the examples above that while it is possible to get by with only the connectives $\lor$ and $\neg$, it is a bit unnatural and tedious, since we need to eliminate the other connectives first. In this subsection, we will make the calculus less frugal by adding rules for the other connectives, without losing the advantage of dealing with a small calculus, which is good making statements about the calculus itself.

7.2.1.2 Practical Enhancements for Tableaux

The main idea here is to add the new rules as derivable inference rules, i.e. rules that only abbreviate derivations in the original calculus. Generally, adding derivable inference rules does not change the derivation relation of the calculus, and is therefore a safe thing to do. In particular, we will add the following rules to our tableau calculus.

We will convince ourselves that the first rule is derivable, and leave the other ones as an exercise.

Derived Rules of Inference

▷ **Definition 7.2.17.** An inference rule $\frac{\mathbf{A}_1 \;\; \ldots \;\; \mathbf{A}_n}{\mathbf{C}}$ is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$ derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

▷ **Definition 7.2.18.** We have the following derivable inference rules in $\mathcal{T}_0$:

$$\frac{(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}} \mid \mathbf{B}^{\mathsf{T}}} \qquad \frac{(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}} \\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\mathsf{T}} \\ (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}\end{array}}{\mathbf{B}^{\mathsf{T}}}$$

$$\frac{(\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}{\mathbf{A}^{\mathsf{T}} \mid \mathbf{B}^{\mathsf{T}}} \qquad \frac{(\mathbf{A} \vee \mathbf{B})^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{T}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{F}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{F}} & \mathbf{B}^{\mathsf{T}}\end{array}}$$

$$\begin{array}{c}
\mathbf{A}^{\mathsf{T}} \\
(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}} \\
(\neg\mathbf{A} \vee \mathbf{B})^{\mathsf{T}} \\
\neg(\neg\neg\mathbf{A} \wedge \neg\mathbf{B})^{\mathsf{T}} \\
(\neg\neg\mathbf{A} \wedge \neg\mathbf{B})^{\mathsf{F}} \\
\begin{array}{c|c}
\neg\neg\mathbf{A}^{\mathsf{F}} & \neg\mathbf{B}^{\mathsf{F}} \\
\neg\mathbf{A}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}} \\
\mathbf{A}^{\mathsf{F}} & \\
\bot &
\end{array}
\end{array}$$

With these derived rules, theorem proving becomes quite efficient. With these rules, the tableau (Example 7.2.13) would have the following simpler form:

Tableaux with derived Rules (example)

Example 7.2.19.

$$\begin{array}{c}
(\mathsf{loves}(\mathsf{mary}, \mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john}, \mathsf{mary}) \Rightarrow \mathsf{loves}(\mathsf{john}, \mathsf{mary}))^{\mathsf{F}} \\
(\mathsf{loves}(\mathsf{mary}, \mathsf{bill}) \wedge \mathsf{loves}(\mathsf{john}, \mathsf{mary}))^{\mathsf{T}} \\
\mathsf{loves}(\mathsf{john}, \mathsf{mary})^{\mathsf{F}} \\
\mathsf{loves}(\mathsf{mary}, \mathsf{bill})^{\mathsf{T}} \\
\mathsf{loves}(\mathsf{john}, \mathsf{mary})^{\mathsf{T}} \\
\bot
\end{array}$$

Another thing that was awkward in (Example 7.2.13) was that we used a proof for an implication to prove logical consequence. Such tests are necessary for instance, if we want to check consistency or informativity of new sentences[3]. Consider for instance a discourse $\Delta = \mathbf{D}^1, \ldots, \mathbf{D}^n$, where n EdN:3 is large. To test whether a hypothesis $\mathcal{H}$ is a consequence of Δ ($\Delta \models \mathbf{H}$) we need to show that $\mathbf{C} := \mathbf{D}^1 \wedge \ldots \wedge \mathbf{D}^n \Rightarrow \mathbf{H}$ is valid, which is quite tedious, since $\mathcal{C}$ is a rather large formula, e.g. if Δ is a 300 page novel. Moreover, if we want to test entailment of the form ($\Delta \models \mathbf{H}$) often, – for instance to test the informativity and consistency of every new sentence $\mathbf{H}$, then successive Δs will overlap quite significantly, and we will be doing the same inferences all over again; the entailment check is not incremental.

incremental procedure for entailment checking in the model generation based setting: To test whether $\Delta \models \mathbf{H}$, where we have interpreted Δ in a model generation tableau $\mathcal{T}$, just check whether the tableau closes, if we add $\neg\mathbf{H}$ to the open branches. Indeed, if the tableau closes, then $\Delta \wedge \neg\mathbf{H}$

[3] EdNote: add reference to presupposition stuff

is unsatisfiable, so $\neg(\Delta\wedge\neg\mathbf{H})$ is valid, but this is equivalent to $\Delta\Rightarrow\mathbf{H}$, which is what we wanted to show.

Example 7.2.20. Consider for instance the following entailment in natural language.

Mary loves Bill. John loves Mary $\models$ *John loves Mary*

EdN:4 [4] We obtain the tableau

$$\begin{array}{c}\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}}\\ \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\ \neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\ \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}\\ \bot\end{array}$$

which shows us that the conjectured entailment relation really holds.

Excursion: We will discuss the properties of propositional tableaux in**??**.

7.2.2 A Tableau Calculus for PLNQ with Free Variables

A Tableau Calculus for $\mathrm{PL}_{\mathrm{NQ}}^{\mathcal{V}}$

▷ **Definition 7.2.21 (Tableau Calculus for $\mathrm{PL}_{\mathrm{NQ}}^{\mathcal{V}}$).** $\mathcal{T}_{\mathcal{V}}^{p}=\mathcal{T}_0$ + new tableau rules for formulae with variables

$$\frac{\begin{array}{c}\vdots\\ \mathbf{A}^{\alpha}\quad c\in\mathcal{H}\\ \vdots\end{array}}{([c/X](\mathbf{A}))^{\alpha}}\mathcal{T}_{\mathcal{V}}^{p}\,WK \qquad \frac{\begin{array}{c}\vdots\\ \boxed{(\mathbf{A}^{\alpha})}\end{array}\quad\begin{array}{c}\mathcal{H}=\{a_1,\ldots,a_n\}\\ \mathrm{free}(\mathbf{A})=\{X_1,\ldots,X_m\}\end{array}}{(\sigma_1(\mathbf{A}))^{\alpha}\;\Big|\;\ldots\;\Big|\;(\sigma_{(n^m)}(\mathbf{A}))^{\alpha}}\mathcal{T}_{\mathcal{V}}^{p}\,Ana$$

$\mathcal{H}$ is the set of ind. constants in the branch above (Herbrand Base)
and the σ_i are substitutions that instantiate the X_j with any combinations of the a_k (there are n^m of them).

- ▷ the first rule is used for world knowledge (up in the branch)
- ▷ the second rule is used for input logical forms $\boxed{\cdots}$
  this rule has to be applied eagerly (while they are still at the leaf)

Michael Kohlhase: LBS 140 2024-01-20

Let us look at two examples: To understand the role of background knowledge we interpret *Peter snores* with respect to the knowledge that *Only sleeping people snore*.

Some Examples in $\mathcal{F}_2$

[4] EdNote: need to mark up the embedding of NL strings into Math

▷ **Example 7.2.22 (Peter snores).** (Only sleeping people snore)

$$\begin{array}{c}
(\mathsf{snores}(X) \Rightarrow \mathsf{sleeps}(X))^{\mathsf{T}} \\
\boxed{(\mathsf{snores}(\mathsf{peter})^{\mathsf{T}})} \\
(\mathsf{snores}(\mathsf{peter}) \Rightarrow \mathsf{sleeps}(\mathsf{peter}))^{\mathsf{T}} \\
\mathsf{sleeps}(\mathsf{peter})^{\mathsf{T}}
\end{array}$$

▷ **Example 7.2.23 (Peter sleeps. John walks. He snores).** (who snores?)

$$\begin{array}{c|c}
\multicolumn{2}{c}{\boxed{(\mathsf{sleeps}(\mathsf{peter})^{\mathsf{T}})}} \\
\multicolumn{2}{c}{\boxed{(\mathsf{walks}(\mathsf{john})^{\mathsf{T}})}} \\
\multicolumn{2}{c}{\boxed{(\mathsf{snores}(X)^{\mathsf{T}})}} \\
\mathsf{snores}(\mathsf{peter})^{\mathsf{T}} & \mathsf{snores}(\mathsf{john})^{\mathsf{T}}
\end{array}$$

The background knowledge is represented in the schematic formula in the first line of the tableau. Upon receiving the input, the tableau instantiates the schema to line three and uses the chaining rule from Definition 7.2.18 to derive the fact that Peter must sleep.

The third input formula contains a free variable, which is instantiated by all constants in the Herbrand base (two in our case). This gives rise to two models that correspond to the two readings of the discourse.

Let us now look at an example with more realistic background knowledge. Say we know that birds fly, if they are not penguins. Furthermore, eagles and penguins are birds, but eagles are not penguins. Then we can answer the classic question *Does Tweety fly?* by the following two tableaux.

Does Tweety fly? The everlasting Question in AI

▷ **Example 7.2.24.**

Tweety is a bird

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\mathsf{bird}(X) \Rightarrow (\mathsf{flies}(X) \vee \mathsf{penguin}(X)))^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\mathsf{penguin}(X) \Rightarrow \neg\mathsf{flies}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{\boxed{(\mathsf{bird}(\mathsf{tweety})^{\mathsf{T}})}} \\
\multicolumn{2}{c}{(\mathsf{flies}(\mathsf{tweety}) \vee \mathsf{penguin}(\mathsf{tweety}))^{\mathsf{T}}} \\
\mathsf{flies}(\mathsf{tweety})^{\mathsf{T}} & \mathsf{penguin}(\mathsf{tweety})^{\mathsf{T}} \\
 & \neg\mathsf{flies}(\mathsf{tweety})^{\mathsf{T}} \\
 & \mathsf{flies}(\mathsf{tweety})^{\mathsf{F}}
\end{array}$$

Tweety is an eagle

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\mathsf{bird}(X) \Rightarrow (\mathsf{flies}(X) \vee \mathsf{penguin}(X)))^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\mathsf{eagle}(X) \Rightarrow \mathsf{bird}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\mathsf{penguin}(X) \Rightarrow \neg\mathsf{eagle}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\mathsf{penguin}(X) \Rightarrow \neg\mathsf{flies}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{\boxed{(\mathsf{eagle}(\mathsf{tweety})^{\mathsf{T}})}} \\
\multicolumn{2}{c}{\mathsf{bird}(\mathsf{tweety})^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\mathsf{flies}(\mathsf{tweety}) \vee \mathsf{penguin}(\mathsf{tweety}))^{\mathsf{T}}} \\
\mathsf{flies}(\mathsf{tweety})^{\mathsf{T}} & \mathsf{penguin}(\mathsf{tweety})^{\mathsf{T}} \\
 & (\neg\mathsf{eagle}(\mathsf{tweety}))^{\mathsf{T}} \\
 & \mathsf{eagle}(\mathsf{tweety})^{\mathsf{F}} \\
 & \bot
\end{array}$$

▷ For the second we need to add more world knowledge.

7.2.3 Case Study: Peter loves Fido, even though he sometimes bites him

Let us now return to the motivating example from the introduction, and see how our system fares with it (this allows us to test our computational/linguistic theory). We will do this in a completely naive manner and see what comes out.
The first problem we run into immediately is that we do not know how to cope with *even though* and *sometimes*, so we simplify the discourse to *Peter loves Fido and he bites him.*.

Finally: *Peter loves Fido. He bites him.*

▷ Let's try it naively (worry about the problems later.)

$$\begin{array}{c|c|c|c}
\multicolumn{4}{c}{\boxed{(l(p,f)^{\mathsf{T}})}} \\
\multicolumn{4}{c}{\boxed{(b(X,Y)^{\mathsf{T}})}} \\
b(p,p)^{\mathsf{T}} & b(p,f)^{\mathsf{T}} & b(f,p)^{\mathsf{T}} & b(f,f)^{\mathsf{T}}
\end{array}$$

▷ **Problem:** We get four readings instead of one!

▷ **Idea:** We have not specified enough world knowledge

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 143 2024-01-20 SOME RIGHTS RESERVED

The next problem is obvious: We get four readings instead of one (or two)! What has happened? If we look at the models, we see that we did not even specify the background knowledge that was supposed filter out the one intended reading.
We try again with the additional knowledge that *Nobody bites himself* and *Humans do not bite dogs*.

Peter and Fido with World Knowledge

▷ Nobody bites himself, humans do not bite dogs.

$$\begin{array}{c|c|c|c}
\multicolumn{4}{c}{d(f)^{\mathsf{T}}} \\
\multicolumn{4}{c}{m(p)^{\mathsf{T}}} \\
\multicolumn{4}{c}{b(X,X)^{\mathsf{F}}} \\
\multicolumn{4}{c}{(d(X) \wedge m(Y) \Rightarrow \neg b(Y,X))^{\mathsf{T}}} \\
\multicolumn{4}{c}{\boxed{(l(p,f)^{\mathsf{T}})}} \\
\multicolumn{4}{c}{\boxed{(b(X,Y)^{\mathsf{T}})}} \\
b(p,p)^{\mathsf{T}} & b(p,f)^{\mathsf{T}} & b(f,p)^{\mathsf{T}} & b(f,f)^{\mathsf{T}} \\
b(p,p)^{\mathsf{F}} & (d(f) \wedge m(p) \Rightarrow \neg b(p,f))^{\mathsf{T}} & & b(f,f)^{\mathsf{F}} \\
\bot & b(p,f)^{\mathsf{F}} & & \bot \\
 & \bot & &
\end{array}$$

▷ **Observation:** Pronoun resolution introduces ambiguities.

▷ **Pragmatics:** Use world knowledge to filter out impossible readings.

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 144 2024-01-20 SOME RIGHTS RESERVED

We observe that our extended tableau calculus was indeed able to handle this example, if we only give it enough background knowledge to act upon.

But the world knowledge we can express in $\mathrm{PL}_{\mathrm{NQ}}^{\mathcal{V}}$ is very limited. We can say that humans do not bite dogs, but we cannot provide the background knowledge to understand a sentence like *Peter was late for class today, the car had a flat tire.*, which needs knowledge like *Every car has wheels, which have a tire.* and *if a tire is flat, the car breaks down.*, which is outside the realm of $\mathrm{PL}_{\mathrm{NQ}}^{\mathcal{V}}$.

7.2.4 The Computational Role of Ambiguities

In the case study, we have seen that pronoun resolution introduces ambiguities, and we can use world knowledge to filter out impossible readings. Generally in the traditional waterfall model of language processing – which posits that NL understanding is a process that analyzes the input in stages: syntax, semantics composition, pragmatics – every processing stage introduces ambiguities that need to be resolved in this stage or later.

The computational Role of Ambiguities

- ▷ **Observation:** (in the traditional waterfall model)
  Every processing stage introduces ambiguities that need to be resolved.
  - ▷ Syntax: e.g. *Peter chased the man in the red sports car* (attachment)
  - ▷ Semantics: e.g. *Peter went to the bank* (lexical)
  - ▷ Pragmatics: e.g. *Two men carried two bags* (collective vs. distributive)
- ▷ **Question:** Where does pronoun-ambiguity belong? (much less clear)
- ▷ **Answer:** we have freedom to choose
  1. resolve the pronouns in the syntax (generic waterfall model)
     $\rightsquigarrow$ multiple syntactic representations (pragmatics as filter)
  2. resolve the pronouns in the pragmatics (our model here)
     $\rightsquigarrow$ need underspecified syntactic representations (e.g. variables)
     $\rightsquigarrow$ pragmatics needs ambiguity treatment (e.g. tableaux)

Michael Kohlhase: LBS 145 2024-01-20

For pronoun ambiguities, this is much less clear. In a way we have the freedom to choose. We can

1. resolve the pronouns in the syntax as in the generic waterfall model, then we arrive at multiple syntactic representations, and can use pragmatics as filter to get rid of unwanted readings
2. resolve the pronouns in the pragmatics (our model here) then we need underspecified syntactic representations (e.g. variables) and pragmatics needs ambiguity treatment (in our case the tableaux).

We will continue to explore the second alternative in more detail, and refine the approach. One of the advantages of treating the anaphoric ambiguities in the syntax is that syntactic agreement information like gender can be used to disambiguate. Say that we vary the example from subsection 7.2.3 to *Peter loves Mary. She loves him.*.

Translation for $\mathcal{F}_2$ Reconsidered

- **Idea:** Pronouns are translated into new variables (so far)
- **Problem:** *Peter loves Mary. She loves him.*

$$\boxed{(\mathrm{loves}(\mathrm{peter},\mathrm{mary})^{\mathsf{T}})}$$
$$\boxed{(\mathrm{loves}(X,Y)^{\mathsf{T}})}$$
$$\mathrm{loves}(\mathrm{peter},\mathrm{peter})^{\mathsf{T}} \mid \mathrm{loves}(\mathrm{peter},\mathrm{mary})^{\mathsf{T}} \mid \mathrm{loves}(\mathrm{mary},\mathrm{peter})^{\mathsf{T}} \mid \mathrm{loves}(\mathrm{mary},\mathrm{mary})^{\mathsf{T}}$$

- **Idea:** attach world knowledge to pronouns (just as with Peter and Fido)
  - use the world knowledge to distinguish (linguistic) gender by predicates masc and fem
- **Idea:** attach world knowledge to pronouns (just as with Peter and Fido)
- **Problem:** properties of
  - proper names are given in the model,
  - pronouns must be given by the syntax/semantics interface
- **In particular:** How to generate $\mathrm{loves}(X,Y)\wedge\mathrm{masc}(X)\wedge\mathrm{fem}(Y)$ compositionally?

Michael Kohlhase: LBS 146 2024-01-20

The tableau (over)-generates the full set of pronoun readings. At first glance it seems that we can fix this just like we did in subsection 7.2.3 by attaching world knowledge to pronouns, just as with Peter and Fido. Then we could use the world knowledge to distinguish gender by predicates, say masc and fem.

But if we look at the whole picture of building a system, we can see that this idea will not work. The problem is that properties of proper names like Fido are given in the background knowledge, whereas the relevant properties of *pronouns* must be given by the syntax/semantics interface. Concretely, we would need to generate $\mathrm{loves}(X,Y)\wedge\mathrm{masc}(X)\wedge\mathrm{fem}(Y)$ for *She loves him*. How can we do such a thing compositionally?

Again we basically have two options, we can either design a clever syntax/semantics interface, or we can follow the lead of Montague semantics and extend the logic, so that compositionality becomes simpler to achieve. We will explore the latter option in the next section. The problem we stumbled across in the last section is how to associate certain properties (in this case agreement information) with variables compositionally. Fortunately, there is a ready-made logical theory for it. Sorted first-order logic. Actually there are various sorted first-order logics, but we will only need the simplest one for our application at the moment.
Sorted first-order logic extends the language with a set $\mathcal{S}$ of sorts $\mathbb{A},\mathbb{B},\mathbb{C},\ldots$, which are just special symbols that are attached to all terms in the language.

Syntactically, all constants, and variables are assigned sorts, which are annotated in the lower index, if they are not clear from the context. Semantically, the universe $\mathcal{D}_\iota$ is subdivided into subsets $\mathcal{D}_{\mathbb{A}}\subseteq\mathcal{D}_\iota$, which denote the objects of sort $\mathbb{A}$; furthermore, the interpretation function $\mathcal{I}$ and variable assignment φ have to be well sorted. Finally, on the calculus level, the only change we have to make is to restrict instantiation to well-sorted substitutions:

Sorts refine World Categories

- **Definition 7.2.25 (Sorted Logics).** (in our case $PL^1_{\mathcal{S}}$) assume a set of sorts $\mathcal{S}:=\{\mathbb{A},\mathbb{B},\mathbb{C},\ldots\}$, annotate every syntactic and semantic structure with them. Make all constructions and operations well worted:

- ▷ Syntax: variables and constants are sorted $X_{\mathbb{A}}, Y_{\mathbb{B}}, Z^1_{\mathbb{C}_1} \ldots, a_{\mathbb{A}}, b_{\mathbb{A}}, \ldots$
- ▷ Semantics: subdivide the Universe $\mathcal{D}_\iota$ into subsets $\mathcal{D}_{\mathbb{A}} \subseteq \mathcal{D}_\iota$
  Interpretation $\mathcal{I}$ and variable assignment φ have to be well-sorted. $\mathcal{I}(a_{\mathbb{A}}), \varphi(X_{\mathbb{A}}) \in \mathcal{D}_{\mathbb{A}}$.
- ▷ calculus: substitutions must be well sorted $[a_{\mathbb{A}}/X_{\mathbb{A}}]$ OK, $[a_{\mathbb{A}}/X_{\mathbb{B}}]$ not.

▷ **Observation:** Sorts do not add expressivity in principle (just practically) For every sort $\mathbb{A}$, we introduce a first-order predicate $\mathcal{R}_{\mathbb{A}}$ and

- ▷ Translate $R(X_{\mathbb{A}}) \wedge \neg P(Z_{\mathbb{C}})$ to $\mathcal{R}_{\mathbb{A}}(X) \wedge \mathcal{R}_{\mathbb{C}}(Z) \Rightarrow R(X) \wedge \neg P(Z)$ in world knowledge.
- ▷ Translate $R(X_{\mathbb{A}}) \wedge \neg P(Z_{\mathbb{C}})$ to $\mathcal{R}_{\mathbb{A}}(X) \wedge \mathcal{R}_{\mathbb{C}}(Z) \wedge R(X,Y) \wedge \neg P(Z)$ in input.
- ▷ Meaning is preserved, but translation is non-compositional!

7.3 Tableaux and Model Generation

7.3.1 Tableau Branches and Herbrand Models

We have claimed above that the set of literals in open saturated tableau branches corresponds to a model. To gain an intuition, we will study our example above,

Model Generation and Interpretation

▷ **Example 7.3.1 (from above).** In Example 7.2.16 we claimed that

$$\mathcal{H} := \{\mathrm{loves}(\mathrm{john}, \mathrm{mary})^{\mathsf{F}}, \mathrm{loves}(\mathrm{mary}, \mathrm{bill})^{\mathsf{T}}\}$$

constitutes a model

$$\begin{array}{c|c} \multicolumn{2}{c}{(\mathrm{loves}(\mathrm{mary}, \mathrm{bill}) \vee \mathrm{loves}(\mathrm{john}, \mathrm{mary}))^{\mathsf{T}}} \\ \multicolumn{2}{c}{\mathrm{loves}(\mathrm{john}, \mathrm{mary})^{\mathsf{F}}} \\ \mathrm{loves}(\mathrm{mary}, \mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john}, \mathrm{mary})^{\mathsf{T}} \\ & \bot \end{array}$$

▷ **Recap:** A model $\mathcal{M}$ is a pair $\langle U, \mathcal{I} \rangle$, where $\mathcal{D}$ is a set of individuals, and $\mathcal{I}$ is an interpretation function.

▷ **Problem:** Find U and $\mathcal{I}$

So the first task is to find a domain $\mathcal{D}$ of interpretation. Our formula mentions *Mary*, *John*, and *Bill*, which we assume to refer to distinct individuals so we need (at least) three individuals in the domain; so let us take $U := \{A, B, C\}$ and fix $\mathcal{I}(\mathrm{mary}) = A$, $\mathcal{I}(\mathrm{bill}) = B$, $\mathcal{I}(\mathrm{john}) = C$.

So the only task is to find a suitable interpretation for the predicate loves that makes $\mathrm{loves}(\mathrm{john}, \mathrm{mary})$ false and $\mathrm{loves}(\mathrm{mary}, \mathrm{bill})$ true. This is simple: we just take $\mathcal{I}(\mathrm{loves}) = \{\langle A, B \rangle\}$. Indeed we have

$$\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{mary}, \mathrm{bill}) \vee \mathrm{loves}(\mathrm{john}, \mathrm{mary})) = \mathsf{T}$$

but $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{john}, \mathrm{mary})) = \mathsf{F}$ according to the rules in[5].

[5] EdNote: crossref

Model Generation and Models

- ▷ **Idea:** Choose the universe U as the set Σ_0^f of constants, choose $\mathcal{I}(=)\mathrm{Id}_{\Sigma_0^f}$, interpret $p\in\Sigma_k^p$ via $\mathcal{I}(p):=\{\langle a_1,\ldots,a_k\rangle|p(a_1,\ldots,a_k)\in\mathcal{H}\}$.
- ▷ **Definition 7.3.2.** We call a model a Herbrand model, iff $U=\Sigma_0^f$ and $\mathcal{I}=\mathrm{Id}_{\Sigma_0^f}$.
- ▷ **Lemma 7.3.3.**
  Let $\mathcal{H}$ be a set of atomic propositions, then setting
  $$\mathcal{I}(p):=\{\langle a_1,\ldots,a_k\rangle|p(a_1,\ldots,a_k)\in\mathcal{H}\}$$
  yields a Herbrand Model that satisfies $\mathcal{H}$. (*proof trivial*)
- ▷ **Corollary 7.3.4.** *Let $\mathcal{H}$ be a consistent (i.e. ∇_c holds) set of atomic propositions, then there is a Herbrand Model that satisfies $\mathcal{H}$.* (*take $\mathcal{H}^\top$*)

In particular, the literals of an open saturated tableau branch $\mathcal{B}$ are a Herbrand model $\mathcal{H}$, as we have convinced ourselves above. By inspection of the inference rules above, we can further convince ourselves, that $\mathcal{H}$ satisfies all formulae on $\mathcal{B}$. We must only check that if $\mathcal{H}$ satisfies the succedents of the rule, then it satisfies the antecedent (which is immediate from the semantics of the principal connectives).

In particular, $\mathcal{H}$ is a model for the root formula of the tableau, which is on $\mathcal{B}$ by construction. So the tableau procedure is also a procedure that generates explicit (Herbrand) models for the root literal of the tableau. Every branch of the tableau corresponds to a (possibly) different Herbrand model. We will use this observation in the next section in an application to natural language semantics.

7.3.2 Using Model Generation for Interpretation

We will now use model generation directly as a tool for discourse interpretation.

Using Model Generation for Interpretation

- ▷ **Definition 7.3.5.** Mental model theory [JL83; JLB91] posits that humans form mental models of the world, i.e. (neural) representations of possible states of the world that are consistent with the perceptions up to date and use them to reason about the world.
- ▷ **So** communication by natural language is a process of transporting parts of the mental model of the speaker into the mental model of the hearer.
- ▷ **Therefore** the NL interpretation process on the part of the hearer is a process of integrating the meaning of the utterances of the speaker into his mental model.
- ▷ **Idea:** We can model discourse understanding as a process of generating Herbrand models for the logical form of an utterance in a discourse by a tableau based model generation procedure.
- ▷ **Advantage:** Capturing ambiguity by generating multiple models for input logical forms.

Tableau Machine

▷ **Definition 7.3.6.** The tableau machine is an inferential cognitive model for incremental natural language understanding that implements mental model theory via tableau based model generation over a sequence of input sentences.

It iterates the following process for every input sentence staring with the empty tableau:

1. add the logical form of the input sentence S_i to the selected branch,
2. perform tableau inferences below S_i until saturated or some resource criterion is met
3. if there are open branches choose a "preferred branch", otherwise backtrack to previous tableau for S_j with $j < i$ and open branches, then re-process $S_{j+1}, \ldots, S_i$ if possible, else fail.

The output is application dependent; some choices are

- ▷ the Herbrand model for the preferred branch $\rightsquigarrow$ preferred interpretation;
- ▷ the literals augmented with all non expanded formulae (from the discourse); (resource-bound was reached)
- ▷ machine answers user queries (preferred model $\models$ query?)

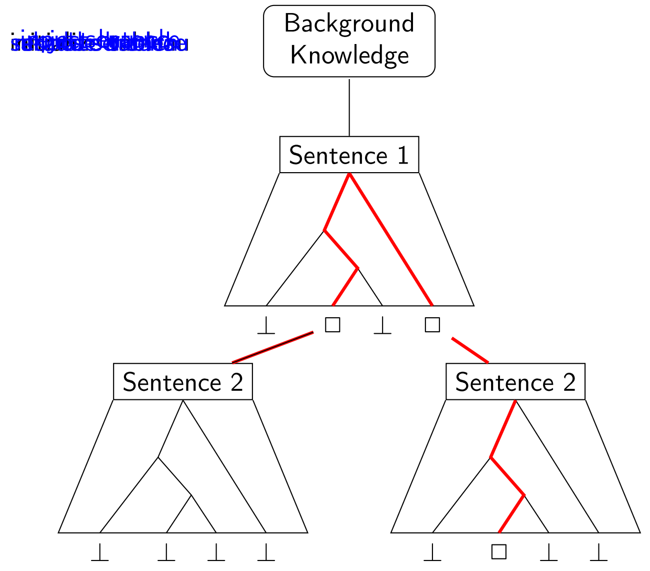
▷ model generation mode (guided by resources and strategies)

▷ theorem proving mode ($\square$ for side conditions; using tableau rules)

Concretely, we treat discourse understanding as an online process that receives as input the logical forms of the sentences of the discourse one by one, and maintains a tableau that represents the current set of alternative models for the discourse. Since we are interested in the internal state of the machine (the current tableau), we do not specify the output of the tableau machine. We also assume that the tableau machine has a mechanism for choosing a preferred model from a set of open branches and that it maintains a set of deferred branches that can be re-visited, if extension of the preferred model fails.
Upon input, the tableau machine will append the given logical form as a leaf to the preferred branch. (We will mark input logical forms in our tableaux by enclosing them in a box.) The machine then saturates the current tableau branch, exploring the set of possible models for the sequence of input sentences. If the subtableau generated by this saturation process contains open branches, then the machine chooses one of them as the preferred model, marks some of the other open branches as deferred, and waits for further input. If the saturation yields a closed sub-tableau, then the machine backtracks, i.e. selects a new preferred branch from the deferred ones, appends the input logical form to it, saturates, and tries to choose a preferred branch. Backtracking is repeated until successful, or until some termination criterion is met, in which case discourse processing fails altogether.

The Tableau Machine in Action

▷ **Example 7.3.7.** The tableau machine in action on two sentences.



Two (Syntactical) Readings

▷ **Example 7.3.8.** *Peter loves Mary and Mary sleeps or Peter snores*(syntactically ambiguous)

Reading 1 loves(peter, mary) ∧ (sleeps(mary) ∨ snores(peter))

Reading 2 loves(peter, mary) ∧ sleeps(mary) ∨ snores(peter)

▷ Let us first consider the first reading in Example 7.3.8. Let us furthermore assume that we start out with the empty tableau, even though this is cognitively implausible, since it simplifies the presentation.

$$\boxed{\text{loves}(\text{peter},\text{mary}) \wedge (\text{sleeps}(\text{mary}) \vee \text{snores}(\text{peter}))}$$
$$\text{loves}(\text{peter},\text{mary})^{T}$$
$$(\text{sleeps}(\text{mary}) \vee \text{snores}(\text{peter}))^{T}$$
$$\text{sleeps}(\text{mary})^{T} \mid \text{snores}(\text{peter})^{T}$$

▷ **Observation:** We have two models, so we have a case of semantical ambiguity.

We see that model generation gives us two models; in both Peter loves Mary, in the first, Mary sleeps, and in the second one Peter snores. If we get a logically different input, e.g. the second reading in Example 7.3.8, then we obtain different models.

The other (Syntactical) Reading

$$\begin{array}{c|c} \multicolumn{2}{c}{\boxed{\mathsf{loves}(\mathsf{peter},\mathsf{mary}) \wedge \mathsf{sleeps}(\mathsf{mary}) \vee \mathsf{snores}(\mathsf{peter})}} \\ (\mathsf{loves}(\mathsf{peter},\mathsf{mary}) \wedge \mathsf{sleeps}(\mathsf{mary}))^{T} & \mathsf{snores}(\mathsf{peter})^{T} \\ \mathsf{loves}(\mathsf{peter},\mathsf{mary})^{T} & \\ \mathsf{sleeps}(\mathsf{mary})^{T} & \end{array}$$

In a discourse understanding system, both readings have to considered in parallel, since they pertain to a genuine ambiguity. The strength of our tableau-based procedure is that it keeps the different readings around, so they can be acted upon later.
Note furthermore, that the overall (syntactical and semantic) ambiguity is not as bad as it looks: the left models of both readings are identical, so we only have three semantic readings not four.

Continuing the Discourse

- **Example 7.3.9.** *Peter does not love Mary*
  then the second tableau would be extended to

$$\begin{array}{c|c} \multicolumn{2}{c}{\boxed{\mathsf{loves}(\mathsf{peter},\mathsf{mary}) \wedge \mathsf{sleeps}(\mathsf{mary}) \vee \mathsf{snores}(\mathsf{peter})}} \\ (\mathsf{loves}(\mathsf{peter},\mathsf{mary}) \wedge \mathsf{sleeps}(\mathsf{mary}))^{T} & \mathsf{snores}(\mathsf{peter})^{T} \\ \mathsf{loves}(\mathsf{peter},\mathsf{mary})^{T} & \boxed{\neg\mathsf{loves}(\mathsf{peter},\mathsf{mary})} \\ \mathsf{sleeps}(\mathsf{mary})^{T} & \\ \boxed{\neg\mathsf{loves}(\mathsf{peter},\mathsf{mary})} & \\ \mathsf{loves}(\mathsf{peter},\mathsf{mary})^{F} & \\ \bot & \end{array}$$

  and the first tableau closes altogether.
- In effect the choice of models has been reduced to one, which constitutes the intuitively correct reading of the discourse

Model Generation models Discourse Understanding

- Conforms with psycholinguistic findings:
  - Zwaan& Radvansky [ZR98]: listeners not only represent logical form, but also models containing referents.
  - deVega [de 95]: online, incremental process.
  - Singer [Sin94]: enriched by background knowledge.
  - Glenberg et al. [GML87]: major function is to provide basis for anaphor resolution.

Towards a Performance Model for NLU

▷ **Problem:** The tableau machine is only a competence model.

▷ **Definition 7.3.10.** A competence model is a meaning theory that delineates a space of possible discourses. A performance model delineates the discourses actually used in communication. (after [Cho65a])

▷ **Idea:** We need to guide the tableau machine in which inferences and branch choices it performs.

▷ **Idea:** Each tableau rule comes with rule costs.

- ▷ **Here**: each sentence in the discourse has a fixed inference budget. Expansion until budget used up.
- ▷ **Ultimately** we want bounded optimization regime [Rus91]: Expansion as long as expected gain in model quality outweighs proof costs

▷ **Effect:** Expensive rules are rarely applied. (only if the promise great rewards)

▷ ⚠ Finding appropriate values for rule costs and model quality is an open problem.

7.3.3 Adding Equality to PLNQ or Fragment 1

We will now extend $\mathrm{PL}^{\mathrm{nq}}$ by equality, which is a very important relation in natural language. Generally, extending a logic with a new logical constant equality is counted as a logical constant, since it semantics is fixed in all models involves extending all three components of the logical system: the language, semantics, and the calculus.

$\mathrm{PL_{NQ}}^{\hat{=}}$: Adding Equality to $\mathrm{PL}^{\mathrm{nq}}$

▷ **Syntax:** Just another binary predicate constant $=$

▷ **Semantics:** Fixed as $\mathcal{I}_\varphi(a=b)=\mathsf{T}$, iff $\mathcal{I}_\varphi(a)=\mathcal{I}_\varphi(b)$. (logical constant)

▷ **Definition 7.3.11 (Tableau Calculus $\mathcal{T}_{\mathsf{NQ}}^{=}$).** Add two additional inference rules (a positive and a negative) to $\mathcal{T}_0$

$$\frac{a\in\mathcal{H}}{a=a^{\mathsf{T}}}\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{sym} \qquad \frac{\begin{array}{c}a=b^{\mathsf{T}}\\ \mathbf{A}\left[a\right]_p{}^{\alpha}\end{array}}{[b/p]\mathbf{A}^{\alpha}}\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{rep}$$

where

- ▷ $\mathcal{H} \mathrel{\hat{=}}$ the Herbrand Base, i.e. the set of constants occurring on the branch
- ▷ we write $\mathbf{C}\left[\mathbf{A}\right]_p$ to indicate that $\mathbf{C}|_p=\mathbf{A}$ ($\mathbf{C}$ has subterm $\mathbf{A}$ at position p).
- ▷ $[\mathbf{A}/p]\mathbf{C}$ is obtained from $\mathbf{C}$ by replacing the subterm at position p with $\mathbf{A}$.

▷ **Note:** We could have equivalently written $\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{sym}$ as $\frac{a=a^{\mathsf{F}}}{\perp}$: With $\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{sym}$ we can conjure a $a=a^{\mathsf{T}}$ from thin air which can then be used to close the $a=a^{\mathsf{F}}$.

▷ **So, . . .** $\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{sym}$ and $\mathcal{T}_{\mathsf{NQ}}^{=}\mathsf{rep}$ follow the pattern of having a T and a F rule per logical constant.

If we simplify the translation of definite descriptions, so that the phrase *the teacher* translates to a concrete individual constant, then we can interpret (**??**) as (**??**).

Reading Comprehension Example: Mini TOEFL test

- ▷ **Example 7.3.12 (Reading Comprehension).** If you hear/read *Mary is the teacher. Peter likes the teacher.*, do you know whether *Peter likes Mary*?
- ▷ **Idea:** Interpret via tableau machine (interpretation mode) and test entailment in theorem proving mode.
- ▷ **Interpretation:** Feed $\Phi_1{:=}\mathrm{mary} = \mathrm{the\_teacher}$ and $\Phi_2{:=}\mathrm{likes}(\mathrm{peter}, \mathrm{the\_teacher})$ to the tableau machine in turn.
  Model generation tableau (nothing to do on these inputs)

$$\boxed{\mathrm{mary} = \mathrm{the\_teacher}^{\mathsf{T}}}$$
$$\boxed{\mathrm{likes}(\mathrm{peter}, \mathrm{the\_teacher})^{\mathsf{T}}}$$

- ▷ **Entailment Test:** label $\varphi{:=}\mathrm{likes}(\mathrm{peter}, \mathrm{mary})$ with F and saturate the tableau.

$$\boxed{\mathrm{mary} = \mathrm{the\_teacher}^{\mathsf{T}}}$$
$$\boxed{\mathrm{likes}(\mathrm{peter}, \mathrm{the\_teacher})^{\mathsf{T}}}$$
$$\mathrm{likes}(\mathrm{peter}, \mathrm{mary})^{\mathsf{F}}$$
$$\mathrm{likes}(\mathrm{peter}, \mathrm{the\_teacher})^{\mathsf{F}}$$
$$\bot$$

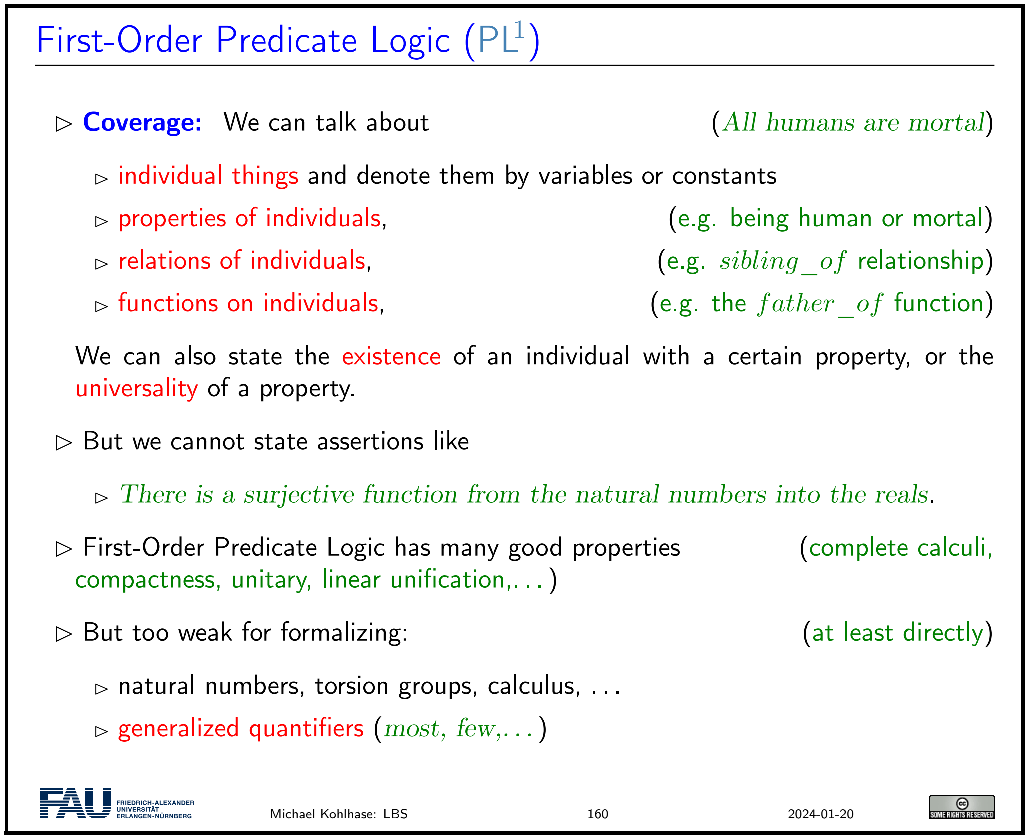
Indeed, it closes, so $\Phi_1, \Phi_2 \models \varphi$.

Chapter 8

Pronouns and World Knowledge in First-Order Logic

8.1 First-Order Logic

First-order logic is the most widely used formal systems for modelling knowledge and inference processes. It strikes a very good bargain in the trade-off between expressivity and conceptual and computational complexity. To many people first-order logic is “the logic”, i.e. the only logic worth considering, its applications range from the foundations of mathematics to natural language semantics.



First-Order Predicate Logic (PL^1)

- ▷ **Coverage:** We can talk about (*All humans are mortal*)
  - ▷ individual things and denote them by variables or constants
  - ▷ properties of individuals, (e.g. being human or mortal)
  - ▷ relations of individuals, (e.g. $sibling\_of$ relationship)
  - ▷ functions on individuals, (e.g. the $father\_of$ function)

  We can also state the existence of an individual with a certain property, or the universality of a property.
- ▷ But we cannot state assertions like
  - ▷ *There is a surjective function from the natural numbers into the reals*.
- ▷ First-Order Predicate Logic has many good properties (complete calculi, compactness, unitary, linear unification,...)
- ▷ But too weak for formalizing: (at least directly)
  - ▷ natural numbers, torsion groups, calculus, ...
  - ▷ generalized quantifiers (*most*, *few*,...)

Michael Kohlhase: LBS 160 2024-01-20

We will now introduce the syntax and semantics of first-order logic. This introduction differs from what we commonly see in undergraduate textbooks on logic in the treatment of substitutions in the presence of bound variables. These treatments are non syntactic, in that they take the renaming of bound variables (α equivalence) as a basic concept and directly introduce capture-avoiding substitutions based on this. But there is a conceptual and technical circularity in this approach, since a careful definition of α-equivalence needs substitutions.

In this section we follow Peter Andrews' lead from [And02] and break the circularity by introducing syntactic substitutions, show a substitution value lemma with a substitutability condition, use that for a soundness proof of α renaming, and only then introduce capture-avoiding substitutions on this basis. This can be done for any logic with bound variables, we go through the details for first-order logic here as an example.

8.1.1 First-Order Logic: Syntax and Semantics

The syntax and semantics of first-order logic is systematically organized in two distinct layers: one for truth values (like in propositional logic) and one for individuals (the new, distinctive feature of first-order logic).

The first step of defining a formal language is to specify the alphabet, here the first-order signatures and their components.

PL^1 Syntax (Signature and Variables)

▷ **Definition 8.1.1.** First-order logic (PL^1), is a formal system extensively used in mathematics, philosophy, linguistics, and computer science. It combines propositional logic with the ability to quantify over individuals.

▷ PL^1 talks about two kinds of objects: (so we have two kinds of symbols)

- ▹ truth values by reusing PL^0
- ▹ individuals, e.g. numbers, foxes, Pokémon,. . .

▷ **Definition 8.1.2.** A first-order signature consists of (all disjoint; $k\in\mathbb{N}$)

- ▹ connectives: $\Sigma_0 = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$ (functions on truth values)
- ▹ function constants: $\Sigma^f_k = \{f, g, h, \ldots\}$ (k-ary functions on individuals)
- ▹ predicate constants: $\Sigma^p_k = \{p, q, r, \ldots\}$ (k-ary relations among individuals.)
- ▹ (Skolem constants: $\Sigma^{sk}_k = \{f^1_k, f^2_k, \ldots\}$) (witness constructors; countably ∞)
- ▹ We take Σ_1 to be all of these together: $\Sigma_1 := \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$ and define $\Sigma := \Sigma_1 \cup \Sigma_0$.

▷ **Definition 8.1.3.** We assume a set of individual variables: $\mathcal{V}_\iota := \{X, Y, Z, \ldots\}$. (countably ∞)

We make the deliberate, but non-standard design choice here to include Skolem constants into the signature from the start. These are used in inference systems to give names to objects and construct witnesses. Other than the fact that they are usually introduced by need, they work exactly like regular constants, which makes the inclusion rather painless. As we can never predict how many Skolem constants we are going to need, we give ourselves countably infinitely many for every arity. Our supply of individual variables is countably infinite for the same reason.

The formulae of first-order logic are built up from the signature and variables as terms (to represent individuals) and propositions (to represent proposition). The latter include the connectives from PL^0, but also quantifiers.

PL^1 Syntax (Formulae)

- ▷ **Definition 8.1.4.** Terms: $\mathbf{A}\in wff_\iota(\Sigma_1,\mathcal{V}_\iota)$ (denote individuals)
  - ▷ $\mathcal{V}_\iota\subseteq wff_\iota(\Sigma_1,\mathcal{V}_\iota)$,
  - ▷ if $f\in\Sigma_k^f$ and $\mathbf{A}^i\in wff_\iota(\Sigma_1,\mathcal{V}_\iota)$ for $i\leq k$, then $f(\mathbf{A}^1,\ldots,\mathbf{A}^k)\in wff_\iota(\Sigma_1,\mathcal{V}_\iota)$.
- ▷ **Definition 8.1.5.** Propositions: $\mathbf{A}\in wff_o(\Sigma_1,\mathcal{V}_\iota)$: (denote truth values)
  - ▷ if $p\in\Sigma_k^p$ and $\mathbf{A}^i\in wff_\iota(\Sigma_1,\mathcal{V}_\iota)$ for $i\leq k$, then $p(\mathbf{A}^1,\ldots,\mathbf{A}^k)\in wff_o(\Sigma_1,\mathcal{V}_\iota)$,
  - ▷ if $\mathbf{A},\mathbf{B}\in wff_o(\Sigma_1,\mathcal{V}_\iota)$ and $X\in\mathcal{V}_\iota$, then $T,\mathbf{A}\wedge\mathbf{B},\neg\mathbf{A},\forall X.\mathbf{A}\in wff_o(\Sigma_1,\mathcal{V}_\iota)$.
    $\forall$ is a binding operator called the universal quantifier.
- ▷ **Definition 8.1.6.** We define the connectives $F,\vee,\Rightarrow,\Leftrightarrow$ via the abbreviations $\mathbf{A}\vee\mathbf{B}:=\neg(\neg\mathbf{A}\wedge\neg\mathbf{B})$, $\mathbf{A}\Rightarrow\mathbf{B}:=\neg\mathbf{A}\vee\mathbf{B}$, $\mathbf{A}\Leftrightarrow\mathbf{B}:=(\mathbf{A}\Rightarrow\mathbf{B})\wedge(\mathbf{B}\Rightarrow\mathbf{A})$, and $F:=\neg T$. We will use them like the primary connectives $\wedge$ and $\neg$
- ▷ **Definition 8.1.7.** We use $\exists X.\mathbf{A}$ as an abbreviation for $\neg(\forall X.\neg\mathbf{A})$. $\exists$ is a binding operator called the existential quantifier.
- ▷ **Definition 8.1.8.** Call formulae without connectives or quantifiers atomic else complex.

Note: We only need e.g. conjunction, negation, and universal quantifier, all other logical constants can be defined from them (as we will see when we have fixed their interpretations).

Alternative Notations for Quantifiers

Here	Elsewhere	
$\forall x.\mathbf{A}$	$\bigwedge x.\mathbf{A}$	$(x)\mathbf{A}$
$\exists x.\mathbf{A}$	$\bigvee x.\mathbf{A}$	

The introduction of quantifiers to first-order logic brings a new phenomenon: variables that are under the scope of a quantifiers will behave very differently from the ones that are not. Therefore we build up a vocabulary that distinguishes the two.

Free and Bound Variables

- ▷ **Definition 8.1.9.** We call an occurrence of a variable X bound in a formula $\mathbf{A}$, iff it occurs in a sub-formula $\forall X.\mathbf{B}$ of $\mathbf{A}$. We call a variable occurrence free otherwise.

  For a formula $\mathbf{A}$, we will use $\mathrm{BVar}(\mathbf{A})$ (and $\mathrm{free}(\mathbf{A})$) for the set of bound (free) variables of $\mathbf{A}$, i.e. variables that have a free/bound occurrence in $\mathbf{A}$.
- ▷ **Definition 8.1.10.** We define the set $\mathrm{free}(\mathbf{A})$ of frees variable of a formula $\mathbf{A}$:

$$\begin{array}{l}
\mathrm{free}(X):=\{X\}\\
\mathrm{free}(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)):=\bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
\mathrm{free}(p(\mathbf{A}_1,\ldots,\mathbf{A}_n)):=\bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
\mathrm{free}(\neg\mathbf{A}):=\mathrm{free}(\mathbf{A})\\
\mathrm{free}(\mathbf{A}\wedge\mathbf{B}):=\mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathbf{B})\\
\mathrm{free}(\forall X.\mathbf{A}):=\mathrm{free}(\mathbf{A})\backslash\{X\}
\end{array}$$

▷ **Definition 8.1.11.** We call a formula $\mathbf{A}$ closed or ground, iff $\mathrm{free}(\mathbf{A})=\emptyset$. We call a closed proposition a sentence, and denote the set of all ground terms with $cwff_\iota(\Sigma_1)$ and the set of sentences with $cwff_o(\Sigma_1)$.

We will be mainly interested in (sets of) sentences – i.e. closed propositions – as the representations of meaningful statements about individuals. Indeed, we will see below that free variables do not gives us expressivity, since they behave like constants and could be replaced by them in all situations, except the recursive definition of quantified formulae. Indeed in all situations where variables occur freely, they have the character of meta variables, i.e. syntactic placeholders that can be instantiated with terms when needed in an inference calculus.
The semantics of first-order logic is a Tarski-style set-theoretic semantics where the atomic syntactic entities are interpreted by mapping them into a well-understood structure, a first-order universe that is just an arbitrary set.

Semantics of PL^1 (Models)

▷ **Definition 8.1.12.** We inherit the domain $\mathcal{D}_0=\{\mathsf{T},\mathsf{F}\}$ of truth values from PL^0 and assume an arbitrary domain $\mathcal{D}_\iota\neq\emptyset$ of individuals. (this choice is a parameter to the semantics)

▷ **Definition 8.1.13.** An interpretation $\mathcal{I}$ assigns values to constants, e.g.

- ▷ $\mathcal{I}(\neg)\colon\mathcal{D}_0\to\mathcal{D}_0$ with $\mathsf{T}\mapsto\mathsf{F}$, $\mathsf{F}\mapsto\mathsf{T}$, and $\mathcal{I}(\wedge)=\ldots$ (as in PL^0)
- ▷ $\mathcal{I}\colon\Sigma_k^f\to{\mathcal{D}_\iota}^k\to\mathcal{D}_\iota$ (interpret function symbols as arbitrary functions)
- ▷ $\mathcal{I}\colon\Sigma_k^p\to\mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicates as arbitrary relations)

▷ **Definition 8.1.14.** A variable assignment $\varphi\colon\mathcal{V}_\iota\to\mathcal{D}_\iota$ maps variables into the domain.

▷ **Definition 8.1.15.** A model $\mathcal{M}=\langle\mathcal{D}_\iota,\mathcal{I}\rangle$ of PL^1 consists of a domain $\mathcal{D}_\iota$ and an interpretation $\mathcal{I}$.

We do not have to make the domain of truth values part of the model, since it is always the same; we determine the model by choosing a domain and an interpretation functiong.
Given a first-order model, we can define the evaluation function as a homomorphism over the construction of formulae.

Semantics of PL^1 (Evaluation)

▷ **Definition 8.1.16.** Given a model $\langle\mathcal{D},\mathcal{I}\rangle$, the value function $\mathcal{I}_\varphi$ is recursively defined: (two parts: terms & propositions)

- ▷ $\mathcal{I}_\varphi\colon \textit{wff}_\iota(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_\iota$ assigns values to terms.
  - ▷ $\mathcal{I}_\varphi(X):=\varphi(X)$ and
  - ▷ $\mathcal{I}_\varphi(f(\mathbf{A}_1,\ldots,\mathbf{A}_k)):=\mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1),\ldots,\mathcal{I}_\varphi(\mathbf{A}_k))$
- ▷ $\mathcal{I}_\varphi\colon \textit{wff}_o(\Sigma_1,\mathcal{V}_\iota)\to\mathcal{D}_0$ assigns values to formulae:
  - ▷ $\mathcal{I}_\varphi(T)=\mathcal{I}(T)=\mathsf{T}$,
  - ▷ $\mathcal{I}_\varphi(\neg\mathbf{A})=\mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$
  - ▷ $\mathcal{I}_\varphi(\mathbf{A}\wedge\mathbf{B})=\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}),\mathcal{I}_\varphi(\mathbf{B}))$ (just as in PL^0)
  - ▷ $\mathcal{I}_\varphi(p(\mathbf{A}_1,\ldots,\mathbf{A}_k)):=\mathsf{T}$, iff $\langle\mathcal{I}_\varphi(\mathbf{A}_1),\ldots,\mathcal{I}_\varphi(\mathbf{A}_k)\rangle\in\mathcal{I}(p)$
  - ▷ $\mathcal{I}_\varphi(\forall X.\mathbf{A}):=\mathsf{T}$, iff $\mathcal{I}_{(\varphi,[\mathsf{a}/X])}(\mathbf{A})=\mathsf{T}$ for all $\mathsf{a}\in\mathcal{D}_\iota$.

▷ **Definition 8.1.17 (Assignment Extension).** Let φ be a variable assignment into D and $a\in D$, then $\varphi,[a/X]$ is called the extension of φ with $[a/X]$ and is defined as $\{(Y,a)\in\varphi|Y\neq X\}\cup\{(X,a)\}$: $\varphi,[a/X]$ coincides with φ off X, and gives the result a there.

The only new (and interesting) case in this definition is the quantifier case, there we define the value of a quantified formula by the value of its scope – *but with an extension of the incoming variable assignment*. Note that by passing to the scope $\mathbf{A}$ of $\forall x.\mathbf{A}$, the occurrences of the variable x in $\mathbf{A}$ that were bound in $\forall x.\mathbf{A}$ become free and are amenable to evaluation by the variable assignment $\psi:=\varphi,[\mathsf{a}/X]$. Note that as an extension of φ, the assignment ψ supplies exactly the right value for x in $\mathbf{A}$. This variability of the variable assignment in the definition of the value function justifies the somewhat complex setup of first-order evaluation, where we have the (static) interpretation function for the symbols from the signature and the (dynamic) variable assignment for the variables.

Note furthermore, that the value $\mathcal{I}_\varphi(\exists x.\mathbf{A})$ of $\exists x.\mathbf{A}$, which we have defined to be $\neg(\forall x.\neg\mathbf{A})$ is true, iff it is not the case that $\mathcal{I}_\varphi(\forall x.\neg\mathbf{A})=\mathcal{I}_\psi(\neg\mathbf{A})=\mathsf{F}$ for all $\mathsf{a}\in\mathcal{D}_\iota$ and $\psi:=\varphi,[\mathsf{a}/X]$. This is the case, iff $\mathcal{I}_\psi(\mathbf{A})=\mathsf{T}$ for some $\mathsf{a}\in\mathcal{D}_\iota$. So our definition of the existential quantifier yields the appropriate semantics.

Semantics Computation: Example

▷ **Example 8.1.18.** We define an instance of first-order logic:

- ▷ Signature: Let $\Sigma_0^f:=\{j,m\}$, $\Sigma_1^f:=\{f\}$, and $\Sigma_2^p:=\{o\}$
- ▷ Universe: $\mathcal{D}_\iota:=\{J,M\}$
- ▷ Interpretation: $\mathcal{I}(j):=J$, $\mathcal{I}(m):=M$, $\mathcal{I}(f)(J):=M$, $\mathcal{I}(f)(M):=M$, and $\mathcal{I}(o):=\{(M,J)\}$.

Then $\forall X.o(f(X),X)$ is a sentence and with $\psi:=\varphi,[\mathsf{a}/X]$ for $\mathsf{a}\in\mathcal{D}_\iota$ we have

$$\begin{array}{rll}\mathcal{I}_\varphi(\forall X.o(f(X),X))=\mathsf{T} & \text{iff} & \mathcal{I}_\psi(o(f(X),X))=\mathsf{T}\text{ for all }\mathsf{a}\in\mathcal{D}_\iota\\ & \text{iff} & (\mathcal{I}_\psi(f(X)),\mathcal{I}_\psi(X))\in\mathcal{I}(o)\text{ for all }\mathsf{a}\in\{J,M\}\\ & \text{iff} & (\mathcal{I}(f)(\mathcal{I}_\psi(X)),\psi(X))\in\{(M,J)\}\text{ for all }\mathsf{a}\in\{J,M\}\\ & \text{iff} & (\mathcal{I}(f)(\psi(X)),a)=(M,J)\text{ for all }\mathsf{a}\in\{J,M\}\\ & \text{iff} & \mathcal{I}(f)(a)=M\text{ and }a=J\text{ for all }\mathsf{a}\in\{J,M\}\end{array}$$

But $\mathsf{a}\neq J$ for $\mathsf{a}=M$, so $\mathcal{I}_\varphi(\forall X.o(f(X),X))=\mathsf{F}$ in the model $\langle\mathcal{D}_\iota,\mathcal{I}\rangle$.

8.1.2 First-Order Substitutions

We will now turn our attention to substitutions, special formula-to-formula mappings that operationalize the intuition that (individual) variables stand for arbitrary terms.

Substitutions on Terms

- ▷ **Intuition:** If $\mathbf{B}$ is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term $\mathbf{A}$ by $\mathbf{B}$ with $[\mathbf{B}/X](\mathbf{A})$.
- ▷ **Problem:** What about $[Z/Y],[Y/X](X)$, is that Y or Z?
- ▷ **Folklore:** $[Z/Y],[Y/X](X) = Y$, but $[Z/Y]([Y/X](X)) = Z$ of course. (Parallel application)
- ▷ **Definition 8.1.19.** Let $wfe(\Sigma,\mathcal{V})$ be an expression language, then we call $\sigma\colon \mathcal{V}\to wfe(\Sigma,\mathcal{V})$ a substitution, iff the support $\mathrm{supp}(\sigma){:=}\{X|(X,\mathbf{A})\in\sigma, X \neq \mathbf{A}\}$ of σ is finite. We denote the empty substitution with ϵ.
- ▷ **Definition 8.1.20 (Substitution Application).** We define substitution application by
  - ▷ $\sigma(c) = c$ for $c\in\Sigma$
  - ▷ $\sigma(X) = \mathbf{A}$, iff $\mathbf{A}\in\mathcal{V}$ and $(X,\mathbf{A})\in\sigma$.
  - ▷ $\sigma(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)) = f(\sigma(\mathbf{A}_1),\ldots,\sigma(\mathbf{A}_n))$,
  - ▷ $\sigma(\beta X.\mathbf{A}) = \beta X.\sigma_{-X}(\mathbf{A})$.
- ▷ **Example 8.1.21.** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.
- ▷ **Definition 8.1.22.** Let σ be a substitution then we call $\mathrm{intro}(\sigma){:=}\bigcup_{X\in\mathrm{supp}(\sigma)}\mathrm{free}(\sigma(X))$ the set of variables introduced by σ.

Michael Kohlhase: LBS 168 2024-01-20

The extension of a substitution is an important operation, which you will run into from time to time. Given a substitution σ, a variable x, and an expression $\mathbf{A}$, $\sigma,[\mathbf{A}/x]$ extends σ with a new value for x. The intuition is that the values right of the comma overwrite the pairs in the substitution on the left, which already has a value for x, even though the representation of σ may not show it.

Substitution Extension

- ▷ **Definition 8.1.23 (Substitution Extension).**

  Let σ be a substitution, then we denote the extension of σ with $[\mathbf{A}/X]$ by $\sigma,[\mathbf{A}/X]$ and define it as $\{(Y,\mathbf{B})\in\sigma|Y \neq X\}\cup\{(X,\mathbf{A})\}$: $\sigma,[\mathbf{A}/X]$ coincides with σ off X, and gives the result $\mathbf{A}$ there.
- ▷ **Note:** If σ is a substitution, then $\sigma,[\mathbf{A}/X]$ is also a substitution.
- ▷ We also need the dual operation: removing a variable from the support:
- ▷ **Definition 8.1.24.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X}{:=}\sigma,[X/X]$.

Note that the use of the comma notation for substitutions defined in **??** is consistent with substitution extension. We can view a substitution $[a/x], [f(b)/y]$ as the extension of the empty substitution (the identity function on variables) by $[f(b)/y]$ and then by $[a/x]$. Note furthermore, that substitution extension is not commutative in general.
For PL^1 substitutions we need to extend the substitutions defined on terms to act on propositions. This is technically more involved, since we have to take care of bound variables.

Substitutions on Propositions

- ▷ **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.\mathbf{A})$?
- ▷ **Idea:** σ should not instantiate bound variables. ($[\mathbf{A}/X](\forall X.\mathbf{B}) = \forall \mathbf{A}.\mathbf{B}'$ ill-formed)
- ▷ **Definition 8.1.25.** $\sigma(\forall X.\mathbf{A}){:=}(\forall X.\sigma_{-X}(\mathbf{A}))$.
- ▷ **Problem:** This can lead to variable capture: $[f(X)/Y](\forall X.p(X,Y))$ would evaluate to $\forall X.p(X,f(X))$, where the second occurrence of X is bound after instantiation, whereas it was free before.
- ▷ **Definition 8.1.26.** Let $\mathbf{B}{\in}\mathit{wff}_\iota(\Sigma_\iota,\mathcal{V}_\iota)$ and $\mathbf{A}{\in}\mathit{wff}_o(\Sigma_\iota,\mathcal{V}_\iota)$, then we call $\mathbf{B}$ substitutable for X in $\mathbf{A}$, iff $\mathbf{A}$ has no occurrence of X in a subterm $\forall Y.\mathbf{C}$ with $Y{\in}\mathrm{free}(\mathbf{B})$.
- ▷ **Solution:** Forbid substitution $[\mathbf{B}/X]\mathbf{A}$, when $\mathbf{B}$ is not substitutablex for X in $\mathbf{A}$.
- ▷ **Better Solution:** Rename away the bound variable X in $\forall X.p(X,Y)$ before applying the substitution. (see alphabetic renaming later.)

Here we come to a conceptual problem of most introductions to first-order logic: they directly define substitutions to be by stipulating that bound variables are renamed in the to ensure subsitutability. But at this time, we have not even defined alphabetic renaming yet, and cannot formally do that without having a notion of substitution. So we will refrain from introducing capture-avoiding substitutions until we have done our homework.

We now introduce a central tool for reasoning about the semantics of substitutions: the "substitution value Lemma", which relates the process of instantiation to (semantic) evaluation. This result will be the motor of all soundness proofs on axioms and inference rules acting on variables via substitutions. In fact, any logic with variables and substitutions will have (to have) some form of a substitution value Lemma to get the meta-theory going, so it is usually the first target in any development of such a logic. We establish the substitution-value Lemma for first-order logic in two steps, first on terms, where it is very simple, and then on propositions, where we have to take special care of substitutability.

Substitution Value Lemma for Terms

- ▷ **Lemma 8.1.27.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be terms, then* $\mathcal{I}_\varphi([\mathbf{B}/X]\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$, *where* $\psi = \varphi, [\mathcal{I}_\varphi(\mathbf{B})/X]$.
- ▷ *Proof:* by induction on the depth of $\mathbf{A}$:

1. depth=0 *Then $\mathbf{A}$ is a variable (say Y), or constant, so we have three cases*
   1.1. $\mathbf{A} = Y = X$
      1.1.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.
   1.2. $\mathbf{A} = Y \neq X$
      1.2.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.
   1.3. $\mathbf{A}$ is a constant
      1.3.1. Analogous to the preceding case ($Y \neq X$).
   1.4. This completes the base case (depth = 0).
2. depth> 0
   2.1. then $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ and we have

$$\begin{array}{rcl}\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) & = & \mathcal{I}(f)(\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}_1)), \ldots, \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}_n))) \\ & = & \mathcal{I}(f)(\mathcal{I}_\psi(\mathbf{A}_1), \ldots, \mathcal{I}_\psi(\mathbf{A}_n)) \\ & = & \mathcal{I}_\psi(\mathbf{A}).\end{array}$$

      by induction hypothesis
   2.2. This completes the induction step, and we have proven the assertion.

We now come to the case of propositions. Note that we have the additional assumption of substitutability here.

Substitution Value Lemma for Propositions

▷ **Lemma 8.1.28.** *Let $\mathbf{B} \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$ be substitutable for X in $\mathbf{A} \in \mathit{wff}_o(\Sigma_\iota, \mathcal{V}_\iota)$, then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\psi(\mathbf{A})$, where $\psi = \varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$.*

▷ *Proof:* by induction on the number n of connectives and quantifiers in $\mathbf{A}$

1. $n = 0$
   1.1. then $\mathbf{A}$ is an atomic proposition, and we can argue like in the induction step of the substitution value lemma for terms.
2. $n>0$ and $\mathbf{A} = \neg\mathbf{B}$ or $\mathbf{A} = \mathbf{C} \circ \mathbf{D}$
   2.1. Here we argue like in the induction step of the term lemma as well.
3. $n>0$ and $\mathbf{A} = \forall X.\mathbf{C}$
   3.1. then $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall X.\mathbf{C}) = \mathsf{T}$, iff $\mathcal{I}_{(\psi,[a/X])}(\mathbf{C}) = \mathcal{I}_{(\varphi,[a/X])}(\mathbf{C}) = \mathsf{T}$, for all $a \in \mathcal{D}_\iota$, which is the case, iff $\mathcal{I}_\varphi(\forall X.\mathbf{C}) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathsf{T}$.
4. $n>0$ and $\mathbf{A} = \forall Y.\mathbf{C}$ where $X \neq Y$
   4.1. then $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall Y.\mathbf{C}) = \mathsf{T}$, iff $\mathcal{I}_{(\psi,[a/Y])}(\mathbf{C}) = \mathcal{I}_{(\varphi,[a/Y])}([\mathbf{B}/X](\mathbf{C})) = \mathsf{T}$, by induction hypothesis.
   4.2. So $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\varphi(\forall Y.[\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\forall Y.\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))$

To understand the proof fully, you should look out where the substitutability is actually used.

Armed with the substitution value lemma, we can now define alphabetic renaming and show it to be sound with respect to the semantics we defined above. And this soundness result will justify the definition of capture-avoiding substitution we will use in the rest of the course.

8.1.3 Alpha-Renaming for First-Order Logic

Armed with the substitution value lemma we can now prove one of the main representational facts for first-order logic: the names of bound variables do not matter; they can be renamed at liberty without changing the meaning of a formula.

Alphabetic Renaming

▷ **Lemma 8.1.29.** *Bound variables can be renamed: If Y is substitutable for X in $\mathbf{A}$, then $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathcal{I}_\varphi(\forall Y.[Y/X](\mathbf{A}))$.*

▷ *Proof:* by the definitions:

1. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{T}$, iff
2. $\mathcal{I}_{(\varphi,[a/X])}(\mathbf{A}) = \mathsf{T}$ for all $a\in\mathcal{D}_\iota$, iff
3. $\mathcal{I}_{(\varphi,[a/Y])}([Y/X](\mathbf{A})) = \mathsf{T}$ for all $a\in\mathcal{D}_\iota$, iff (by substitution value lemma)
4. $\mathcal{I}_\varphi(\forall Y.[Y/X](\mathbf{A})) = \mathsf{T}$.

▷ **Definition 8.1.30.** We call two formulae $\mathbf{A}$ and $\mathbf{B}$ alphabetic variants (or α-equal; write $\mathbf{A} =_\alpha \mathbf{B}$), iff $\mathbf{A} = \forall X.\mathbf{C}$ and $\mathbf{B} = \forall Y.[Y/X](\mathbf{C})$ for some variables X and Y.

We have seen that naive substitutions can lead to variable capture. As a consequence, we always have to presuppose that all instantiations respect a substitutability condition, which is quite tedious. We will now come up with an improved definition of substitution application for first-order logic that does not have this problem.

Avoiding Variable Capture by Built-in α-renaming

▷ **Idea:** Given alphabetic renaming, consider alphabetic variants as identical!

▷ **So:** Bound variable names in formulae are just a representational device. (we rename bound variables wherever necessary)

▷ **Formally:** Take $\mathit{cwff}_o(\Sigma_\iota)$ (new) to be the quotient space of $\mathit{cwff}_o(\Sigma_\iota)$ (old) modulo $=_\alpha$. (formulae as syntactic representatives of equivalence classes)

▷ **Definition 8.1.31 (Capture-Avoiding Substitution Application).** Let σ be a substitution, $\mathbf{A}$ a formula, and $\mathbf{A}'$ an alphabetic variant of $\mathbf{A}$, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(\mathbf{A}) = \emptyset$. Then $[\mathbf{A}]_{=_\alpha} = [\mathbf{A}']_{=_\alpha}$ and we can define $\sigma([\mathbf{A}]_{=_\alpha}){:=}[(\sigma(\mathbf{A}'))]_{=_\alpha}$.

▷ **Notation:** After we have understood the quotient construction, we will neglect making it explicit and write formulae and substitutions with the understanding that they act on quotients.

▷ **Alternative:** Replace variables with numbers in formulae (de Bruijn indices).

Undecidability of First-Order Logic

▷ **Theorem 8.1.32.** *Validity in first-order logic is undecidable.*

▷ *Proof:* We prove this by contradiction

1. Let us assume that there is a

8.2 First-Order Inference with Tableaux

We will now extend the propositional tableau techniques to first-order logic. We only have to add two new rules for the universal quantifier (in positive and negative polarity).

First-Order Standard Tableaux ($\mathcal{T}_1$)

▷ **Definition 8.2.1.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad \mathbf{C}\in cwff_\iota(\Sigma_\iota)}{([\mathbf{C}/X](\mathbf{A}))^{\mathsf{T}}}\mathcal{T}_1\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad c\in\Sigma_0^{sk} \text{ new}}{([c/X](\mathbf{A}))^{\mathsf{F}}}\mathcal{T}_1\exists$$

▷ **Problem:** The rule $\mathcal{T}_1\forall$ displays a case of "don't know indeterminism": to find a refutation we have to guess a formula $\mathbf{C}$ from the (usually infinite) set $cwff_\iota(\Sigma_\iota)$.

For proof search, this means that we have to systematically try all, so $\mathcal{T}_1\forall$ is infinitely branching in general.

The rule $\mathcal{T}_1\forall$ operationalizes the intuition that a universally quantified formula is true, iff all of the instances of the scope are. To understand the $\mathcal{T}_1\exists$ rule, we have to keep in mind that $\exists X.\mathbf{A}$ abbreviates $\neg(\forall X.\neg\mathbf{A})$, so that we have to read $(\forall X.\mathbf{A})^{\mathsf{F}}$ existentially — i.e. as $(\exists X.\neg\mathbf{A})^{\mathsf{T}}$, stating that there is an object with property $\neg\mathbf{A}$. In this situation, we can simply give this object a name: c, which we take from our (infinite) set of witness constants Σ_0^{sk}, which we have given ourselves expressly for this purpose when we defined first-order syntax. In other words $([c/X](\neg\mathbf{A}))^{\mathsf{T}} = ([c/X](\mathbf{A}))^{\mathsf{F}}$ holds, and this is just the conclusion of the $\mathcal{T}_1\exists$ rule.
Note that the $\mathcal{T}_1\forall$ rule is computationally extremely inefficient: we have to guess an (i.e. in a search setting to systematically consider all) instance $\mathbf{C}\in wff_\iota(\Sigma_\iota,\mathcal{V}_\iota)$ for X. This makes the rule infinitely branching.

8.2.1 Free Variable Tableaux

In the next calculus we will try to remedy the computational inefficiency of the $\mathcal{T}_1\forall$ rule. We do this by delaying the choice in the universal rule.

Free variable Tableaux ($\mathcal{T}_1^f$)

▷ **Definition 8.2.2.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad Y \text{ new}}{([Y/X](\mathbf{A}))^{\mathsf{T}}}\mathcal{T}_1^f\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\} \quad f\in\Sigma_k^{sk} \text{ new}}{([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{F}}}\mathcal{T}_1^f\exists$$

and generalizes its cut rule $\mathcal{T}_0\bot$ to:

$$\frac{\begin{array}{l}\mathbf{A}^\alpha\\ \mathbf{B}^\beta\end{array}\quad \alpha\neq\beta\quad \sigma(\mathbf{A})=\sigma(\mathbf{B})}{\bot:\sigma}\mathcal{T}_1^f\bot$$

$\mathcal{T}_1^f\bot$ instantiates the whole tableau by σ.

- ▷ **Advantage:** No guessing necessary in $\mathcal{T}_1^f\forall$-rule!
- ▷ **New Problem:** find suitable substitution (most general unifier) (later)

Metavariables: Instead of guessing a concrete instance for the universally quantified variable as in the $\mathcal{T}_1\forall$ rule, $\mathcal{T}_1^f\forall$ instantiates it with a new meta-variable Y, which will be instantiated by need in the course of the derivation.
Skolem terms as witnesses: The introduction of meta-variables makes is necessary to extend the treatment of witnesses in the existential rule. Intuitively, we cannot simply invent a new name, since the meaning of the body $\mathbf{A}$ may contain meta-variables introduced by the $\mathcal{T}_1^f\forall$ rule. As we do not know their values yet, the witness for the existential statement in the antecedent of the $\mathcal{T}_1^f\exists$ rule needs to depend on that. So witness it using a witness term, concretely by applying a Skolem function to the meta-variables in $\mathbf{A}$.
Instantiating Metavariables: Finally, the $\mathcal{T}_1^f\bot$ rule completes the treatment of meta-variables, it allows to instantiate the whole tableau in a way that the current branch closes. This leaves us with the problem of finding substitutions that make two terms equal.

Free variable Tableaux ($\mathcal{T}_1^f$): Derivable Rules

▷ **Definition 8.2.3.** Derivable quantifier rules in $\mathcal{T}_1^f$:

$$\frac{(\exists X.\mathbf{A})^{\mathsf{T}}\quad \mathrm{free}(\forall X.\mathbf{A})=\{X^1,\ldots,X^k\}\quad f\in\Sigma_k^{sk}\ \text{new}}{([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{T}}}$$

$$\frac{(\exists X.\mathbf{A})^{\mathsf{F}}\quad Y\ \text{new}}{([Y/X](\mathbf{A}))^{\mathsf{F}}}$$

We now come to some issues (and clarifications) pertaining to implementing proof search for free variable tableaux. They all have to do with the – often overlooked – fact that $\mathcal{T}_1^f\bot$ instantiates the whole tableau.
The first question one may ask for implementation is whether we expect a terminating proof search; after all, $\mathcal{T}_0$ terminated. We will see that the situation for $\mathcal{T}_1^f$ is different.

Termination and Multiplicity in Tableaux

▷ **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
$\leadsto$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.

▷ **Observation 8.2.4.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*

▷ **Example 8.2.5.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.

Start, close left branch	use $\mathcal{T}_1^f\forall$ again (right branch)
$\begin{array}{c} ((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}} \\ (p(a) \vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(y)^{\mathsf{T}} \\ p(y)^{\mathsf{F}} \\ \begin{array}{cc} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot : [a/y] & \end{array} \end{array}$	$\begin{array}{c} ((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}} \\ (p(a) \vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(a)^{\mathsf{T}} \\ p(a)^{\mathsf{F}} \\ \begin{array}{cc} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot : [a/y] & \neg p(z)^{\mathsf{T}} \\ & p(z)^{\mathsf{F}} \\ & \bot : [b/z] \end{array} \end{array}$

After we have used up $p(y)^{\mathsf{F}}$ by applying $[a/y]$ in $\mathcal{T}_1^f\bot$, we have to get a new instance $p(z)^{\mathsf{F}}$ via $\mathcal{T}_1^f\forall$.

▷ **Definition 8.2.6.** Let $\mathcal{T}$ be a tableau for $\mathbf{A}$, and a positive occurrence of $\forall x.\mathbf{B}$ in $\mathbf{A}$, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.\mathbf{B}$ its multiplicity.

▷ **Observation 8.2.7.** *Given a prescribed multiplicity for each positive $\forall$, saturation with $\mathcal{T}_1^f$ terminates.*

▷ *Proof sketch:* All $\mathcal{T}_1^f$ rules reduce the number of connectives and negative $\forall$ or the multiplicity of positive $\forall$.

▷ **Theorem 8.2.8.** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*

▷ *Proof sketch:* Replace $p(a) \vee p(b)$ with $p(a_1) \vee \ldots \vee p(a_n)$ in Example 8.2.5.

▷ **Remark:** Otherwise validity in PL^1 would be decidable.

▷ **Implementation:** We need an iterative multiplicity deepening process.

The other thing we need to realize is that there may be multiple ways we can use $\mathcal{T}_1^f\bot$ to close a branch in a tableau, and – as $\mathcal{T}_1^f\bot$ instantiates the whole tableau and not just the branch itself – this choice matters.

Treating $\mathcal{T}_1^f\bot$

▷ **Recall:** The $\mathcal{T}_1^f\bot$ rule instantiates the whole tableau.

▷ **Problem:** There may be more than one $\mathcal{T}_1^f\bot$ opportunity on a branch.

▷ **Example 8.2.9.** Choosing which matters – this tableau does not close!

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\exists x.(p(a) \wedge p(b) \Rightarrow p()) \wedge (q(b) \Rightarrow q(x)))^{\mathsf{F}}} \\
\multicolumn{2}{c}{((p(a) \wedge p(b) \Rightarrow p()) \wedge (q(b) \Rightarrow q(y)))^{\mathsf{F}}} \\
(p(a) \Rightarrow p(b) \Rightarrow p())^{\mathsf{F}} & (q(b) \Rightarrow q(y))^{\mathsf{F}} \\
p(a)^{\mathsf{T}} & q(b)^{\mathsf{T}} \\
p(b)^{\mathsf{T}} & q(y)^{\mathsf{F}} \\
p(y)^{\mathsf{F}} & \\
\bot : [a/y] &
\end{array}$$

choosing the other $\mathcal{T}_1^f\bot$ in the left branch allows closure.

▷ **Idea:** Two ways of systematic proof search in $\mathcal{T}_1^f$:

- ▷ backtracking search over $\mathcal{T}_1^f\bot$ opportunities
- ▷ saturate without $\mathcal{T}_1^f\bot$ and find spanning matings (next slide)

The method of spanning matings follows the intuition that if we do not have good information on how to decide for a pair of opposite literals on a branch to use in $\mathcal{T}_1^f\bot$, we delay the choice by initially disregarding the rule altogether during saturation and then – in a later phase– looking for a configuration of cuts that have a joint overall unifier. The big advantage of this is that we only need to know that one exists, we do not need to compute or apply it, which would lead to exponential blow-up as we have seen above.

Spanning Matings for $\mathcal{T}_1^f\bot$

▷ **Observation 8.2.10.** *$\mathcal{T}_1^f$ without $\mathcal{T}_1^f\bot$ is terminating and confluent for given multiplicities.*

▷ **Idea:** Saturate without $\mathcal{T}_1^f\bot$ and treat all cuts at the same time (later).

▷ **Definition 8.2.11.**

Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E}:=\mathbf{A}_1=^?\mathbf{B}_1 \wedge \ldots \wedge \mathbf{A}_n=^?\mathbf{B}_n$ a mating for $\mathcal{T}$, iff $\mathbf{A}_i{}^{\mathsf{T}}$ and $\mathbf{B}_i{}^{\mathsf{F}}$ occur in the same branch in $\mathcal{T}$.

We say that $\mathcal{E}$ is a spanning mating, if $\mathcal{E}$ is unifiable and every branch $\mathcal{B}$ of $\mathcal{T}$ contains $\mathbf{A}_i{}^{\mathsf{T}}$ and $\mathbf{B}_i{}^{\mathsf{F}}$ for some i.

▷ **Theorem 8.2.12.** *A $\mathcal{T}_1^f$-tableau with a spanning mating induces a closed $\mathcal{T}_1$ tableau.*

▷ *Proof sketch:* Just apply the unifier of the spanning mating.

▷ **Idea:** Existence is sufficient, we do not need to compute the unifier.

▷ **Implementation:** Saturate without $\mathcal{T}_1^f\bot$, backtracking search for spanning matings with $\mathcal{DU}$, adding pairs incrementally.

Excursion: We will cover first-order unification in??.

The method of spanning matings follows the intuition that if we do not have good information on how to decide for a pair of opposite literals on a branch to use in $\mathcal{T}_1^f\bot$, we delay the choice by initially disregarding the rule altogether during saturation and then – in a later phase– looking for a configuration of cuts that have a joint overall unifier. The big advantage of this is that we only need to know that one exists, we do not need to compute or apply it, which would lead to exponential blow-up as we have seen above.

Spanning Matings for $\mathcal{T}_1^f\bot$

- **Observation 8.2.13.** *$\mathcal{T}_1^f$ without $\mathcal{T}_1^f\bot$ is terminating and confluent for given multiplicities.*
- **Idea:** Saturate without $\mathcal{T}_1^f\bot$ and treat all cuts at the same time (later).
- **Definition 8.2.14.**

  Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E}:=\mathbf{A}_1=^?\mathbf{B}_1 \wedge \ldots \wedge \mathbf{A}_n=^?\mathbf{B}_n$ a mating for $\mathcal{T}$, iff ${\mathbf{A}_i}^{\mathsf{T}}$ and ${\mathbf{B}_i}^{\mathsf{F}}$ occur in the same branch in $\mathcal{T}$.

  We say that $\mathcal{E}$ is a spanning mating, if $\mathcal{E}$ is unifiable and every branch $\mathcal{B}$ of $\mathcal{T}$ contains ${\mathbf{A}_i}^{\mathsf{T}}$ and ${\mathbf{B}_i}^{\mathsf{F}}$ for some i.
- **Theorem 8.2.15.** *A $\mathcal{T}_1^f$-tableau with a spanning mating induces a closed $\mathcal{T}_1$ tableau.*
- *Proof sketch:* Just apply the unifier of the spanning mating.
- **Idea:** Existence is sufficient, we do not need to compute the unifier.
- **Implementation:** Saturate without $\mathcal{T}_1^f\bot$, backtracking search for spanning matings with $\mathcal{DU}$, adding pairs incrementally.

Excursion: We discuss soundness and completenss of first-order tableaux in??.

8.3 Model Generation with Quantifiers

Since we have introduced new logical constants, we have to extend the model generation calculus by rules for these. To keep the calculus simple, we will treat $\exists X.\mathbf{A}$ as an abbreviation of $\neg(\forall X.\neg\mathbf{A})$. Thus we only have to treat the universal quantifier in the rules.

Model Generation (The RM Calculus [Kon04])

- **Idea:** Try to generate domain-minimal (i.e. fewest individuals) models (for NL interpretation)
- **Problem:** Even one function constant makes Herbrand base infinite (solution: leave them out)
- **Definition 8.3.1.** RM adds ground quantifier rules to propositional tableau calculus

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad c\in\mathcal{H}}{([c/X](\mathbf{A}))^{\mathsf{T}}}RM\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad \mathcal{H}=\{a_1,\ldots,a_n\} \quad w\notin\mathcal{H} \text{ new}}{([a_1/X](\mathbf{A}))^{\mathsf{F}} \mid \ldots \mid ([a_n/X](\mathbf{A}))^{\mathsf{F}} \mid ([w/X](\mathbf{A}))^{\mathsf{F}}}RM\exists$$

  - $RM\exists$ rule introduces new witness constant w to Herbrand base $\mathcal{H}$ of branch

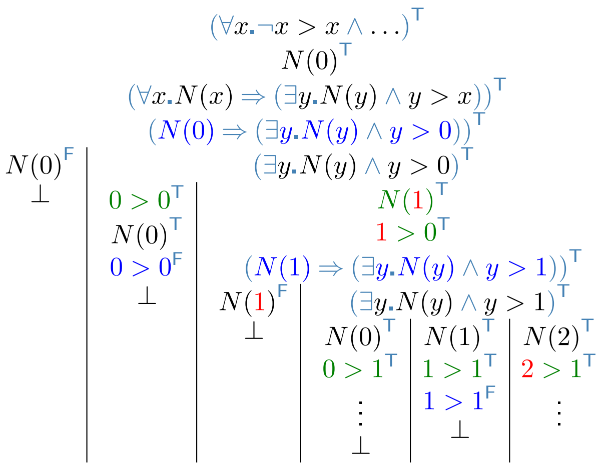
▷ Apply $RM\,\forall$ exhaustively (for new w reapply all $RM\,\forall$ rules on branch!)

The rule $RM\,\forall$ allows to instantiate the scope of the quantifier with all the instances of the Herbrand base, whereas the rule $RM\,\exists$ makes a case distinction between the cases that the scope holds for one of the already known individuals (those in the Herbrand base) or a currently unknown one (for which it introduces a witness constant $w \in \Sigma_0^{sk}$).
Note that in order to have a complete calculus, it is necessary to apply the $RM\,\forall$ rule to all universal formulae in the tree with the new constant w. With this strategy, we arrive at a complete calculus for (finite) satisfiability in first-order logic, i.e. if a formula has a (finite) Model, then this calculus will find it. Note that this calculus (in this simple form) does not necessarily find minimal models.

Generating infinite models (Natural Numbers)

▷ We have to re-apply the $RM\,\forall$ rule for any new constant

▷ **Example 8.3.2.** This leads to the generation of infinite models


$(\forall x.\neg x > x \wedge \ldots)^T$
$N(0)^T$
$(\forall x.N(x) \Rightarrow (\exists y.N(y) \wedge y > x))^T$
$(N(0) \Rightarrow (\exists y.N(y) \wedge y > 0))^T$
$N(0)^F$
$\bot$
$(\exists y.N(y) \wedge y > 0)^T$
$0 > 0^T$
$N(0)^T$
$0 > 0^F$
$\bot$
$N(1)^T$
$1 > 0^T$
$(N(1) \Rightarrow (\exists y.N(y) \wedge y > 1))^T$
$N(1)^F$
$\bot$
$(\exists y.N(y) \wedge y > 1)^T$
$N(0)^T$
$0 > 1^T$
$\vdots$
$\bot$
$N(1)^T$
$1 > 1^T$
$1 > 1^F$
$\bot$
$N(2)^T$
$2 > 1^T$
$\vdots$

The rules $RM\,\forall$ and $RM\,\exists$ may remind you of the rules we introduced for $\mathrm{PL}_{\mathrm{NQ}}^{\mathcal{V}}$. In fact the rules mainly differ in their scoping behavior. We will use $RM\,\forall$ as a drop-in replacement for the world-knowledge rule $\mathcal{T}_{\mathcal{V}}^{p}\,WK$, and express world knowledge as universally quantified sentences. The rules $\mathcal{T}_{\mathcal{V}}^{p}\,Ana$ and $RM\,\exists$ differ in that the first may only be applied to input formulae and does not introduce a witness constant. (It should not, since variables here are anaphoric). We need the rule $RM\,\exists$ to deal with rule-like world knowledge.

Example: *Peter is a man. No man walks*

without sorts

$$\begin{array}{c}
\boxed{\mathsf{man}(\mathsf{peter})}\\
\boxed{\neg(\exists X.\mathsf{man}(X)\wedge\mathsf{walks}(X))}\\
(\exists X.\mathsf{man}(X)\wedge\mathsf{walks}(X))^{\mathsf{F}}\\
(\mathsf{man}(\mathsf{peter})\wedge\mathsf{walks}(\mathsf{peter}))^{\mathsf{F}}\\
\begin{array}{c|c}
\mathsf{man}(\mathsf{peter})^{\mathsf{F}} & \mathsf{walks}(\mathsf{peter})^{\mathsf{F}}\\
\bot &
\end{array}
\end{array}$$

problem: 1000 women
$\Rightarrow$
1000 closed branches

with sort $\mathbb{Male}$

$$\begin{array}{c}
\boxed{\mathsf{man}(\mathsf{peter})}\\
\boxed{\neg(\exists X_{\mathbb{Male}}.\mathsf{walks}(X))}\\
(\exists X_{\mathbb{Male}}.\mathsf{walks}(X))^{\mathsf{F}}\\
\mathsf{walks}(\mathsf{peter})^{\mathsf{F}}
\end{array}$$

▷ Herbrand-model

$$\{\mathsf{man}(\mathsf{peter})^{\mathsf{T}},\mathsf{walks}(\mathsf{peter})^{\mathsf{F}}\}$$

Anaphor Resolution *A man sleeps. He snores*

$$\begin{tikzcd}[column sep=small]
& \boxed{\exists X.\mathsf{sleeps}(X)} & \\
& \mathsf{sleeps}(c^1_{\mathbb{Man}})^{\mathsf{T}} & \\
& \boxed{\exists Y_{\mathbb{Man}}.\mathsf{snores}(Y)} \arrow[dl] \arrow[dr] & \\
\mathsf{snores}(c^1_{\mathbb{Man}})^{\mathsf{T}} & & \mathsf{snores}(c^2_{\mathbb{Man}})^{\mathsf{T}}
\end{tikzcd}$$

▷ minimal deictic

In a situation without men (but maybe thousands of women)

Anaphora with World Knowledge

▷ **Example 8.3.3.** *Mary is married to Jeff. Her husband is not in town.* (slightly outside $\mathcal{F}_2$) In PL^1: $\mathsf{married}(\mathsf{mary},\mathsf{jeff})$, and

$$\exists W_{\mathbb{Male}},W'_{\mathbb{Female}}.\mathsf{husband}(W,W')\wedge\neg\mathsf{intown}(W)$$

▷ World knowledge

- ▷ If woman X is married to man Y, then Y is the only husband of X.
- ▷ $\forall X_{\mathbb{Female}},Y_{\mathbb{Male}}.\mathsf{married}(X,Y)\Rightarrow\mathsf{husband}(Y,X)\wedge(\forall Z.\mathsf{husband}(Z,X)\Rightarrow(Z=Y))$

▷ Model generation gives tableau where all open branches contain

$$\{\mathsf{married}(\mathsf{mary},\mathsf{jeff})^{\mathsf{T}},\mathsf{husband}(\mathsf{jeff},\mathsf{mary})^{\mathsf{T}},\mathsf{intown}(\mathsf{jeff})^{\mathsf{F}}\}$$

▷ **Differences:** Additional negative facts e.g. $\mathsf{married}(\mathsf{mary},\mathsf{mary})^F$.

A branch without world knowledge

$$\begin{array}{c}
\mathsf{married}(\mathsf{mary},\mathsf{jeff})^T \\
(\exists Z_{\mathtt{Male}}, Z'_{\mathtt{Female}}.\mathsf{husband}(Z,Z') \land \neg\mathsf{intown}(Z))^T \\
(\exists Z'.\mathsf{husband}(c^1_{\mathtt{Male}}, Z') \land \neg\mathsf{intown}(c^1_{\mathtt{Male}}))^T \\
(\mathsf{husband}(c^1_{\mathtt{Male}}, \mathsf{mary}) \land \neg\mathsf{intown}(c^1_{\mathtt{Male}}))^T \\
\mathsf{husband}(c^1_{\mathtt{Male}}, \mathsf{mary})^T \\
\neg\mathsf{intown}(c^1_{\mathtt{Male}})^T \\
\mathsf{intown}(c^1_{\mathtt{Male}})^F
\end{array}$$

▷ **Problem:** Bigamy: $c^1_{\mathtt{Male}}$ and jeff are husbands of $Mary$!

Chapter 9

Fragment 3: Complex Verb Phrases

9.1 Fragment 3 (Handling Verb Phrases)

New Data (Verb Phrases)

▷ *Ethel howled and screamed.*

▷ *Ethel kicked the dog and poisoned the cat.*

▷ *Fiona liked Jo and loathed Ethel and tolerated Prudence.*

▷ *Fiona kicked the cat and laughed.*

▷ *Prudence kicked and scratched Ethel.*

▷ *Bertie didn't laugh.*

▷ *Bertie didn't laugh and didn't scream.*

▷ *Bertie didn't laugh or scream.*

▷ *Bertie didn't laugh or kick the dog.*

▷ * *Bertie didn't didn't laugh.*

New Grammar in Fragment 3 (Verb Phrases)

▷ To account for the syntax we come up with the concept of a verb-phrase (VP)

▷ **Definition 9.1.1.** $\mathcal{F}_3$ has the following rules:

S1.	S	$\to$	NPVP_{+fin}
S2.	S	$\to$	Sconj$S$
V1.	$VP_{\pm fin}$	$\to$	$V^i_{\pm fin}$
V2.	$VP_{\pm fin}$	$\to$	$V^t_{\pm fin}$, NP
V3.	$VP_{\pm fin}$	$\to$	$VP_{\pm fin}$, conj, $VP_{\pm fin}$
V4.	VP_{+fin}	$\to$	$BE_{=}$, NP
V5.	VP_{+fin}	$\to$	BE_{pred}, Adj.
V6.	VP_{+fin}	$\to$	didn't VP_{-fin}
N1.	NP	$\to$	N_{pr}
N2.	NP	$\to$	Pron
N3.	NP	$\to$	the N

L8.	$BE_{=}$	$\to$	is
L9.	BE_{pred}	$\to$	is
L10.	V^i_{-fin}	$\to$	run, laugh, sing,...
L11.	V^t_{-fin}	$\to$	read, poison,eat,...

▷ **Limitations of $\mathcal{F}_3$:**

- ▷ The rule for *didn't* over-generates: * *John didn't didn't run* (need tense for that)
- ▷ $\mathcal{F}_3$ does not allow coordination of transitive verbs (problematic anyways)

The main extension of the fragment is the introduction of the new phrasal category *VP*, we have to interpret. Intuitively, *VP*s denote functions that can be applied to the NP meanings (rule 1). Complex *VP* functions can be constructed from simpler ones by NL connectives acting as functional operators.
Given the discussion above, we have to deal with various kinds of functions in the semantics. NP meanings are individuals, *VP* meanings are functions from individuals to individuals, and conj meanings are functionals that map functions to functions. It is a tradition in logic to distinguish such objects (individuals and functions of various kinds) by assigning them types.

Implementing Fragment 3 in GF

▷ The grammar of Fragment 3 only differs from that of Fragment 2 by

- ▷ Verb phrases: `cat VP; VPf;` infinite and finite verb phrases
- ▷ Verb Form: to distinguish *howl* and *howled* in English

```
param VForm = VInf | VPast;
oper VerbType : Type = {s : VForm => Str };
```

- ▷ English Paradigms to deal with verb forms.

```
mkVP = overload {
  mkVP : (v : VForm => Str) -> VP = \v -> lin VP {s = v};
  mkVP : (v : VForm => Str) -> Str -> VP =
  \v,str -> lin VP {s = table{VInf => v!VInf ++ str; VPast => v!VPast ++ str}};
  mkVP : (v : VForm => Str) -> Str -> (v : VForm => Str) -> VP =
  \v1,str,v2 -> lin VP {s = table{VInf => v1!VInf ++ str ++ v2!VInf;
      VPast => v1!VPast ++ str ++ v2!VPast}};};
mkVPf : Str -> VPf = \str -> lin VPf {s = str};
```

9.2 Dealing with Functions in Logic and Language

So we need to have a logic that can deal with functions and functionals (i.e. functions that construct new functions from existing ones) natively. This goes beyond the realm of first-order logic we have studied so far. We need two things from this logic:

1. a way of distinguishing the respective individuals, functions and functionals, and
2. a way of constructing functions from individuals and other functions.

There are standard ways of achieving both, which we will combine in the following to get the "simply typed lambda calculus" which will be the workhorse logic for $\mathcal{F}_3$.
The standard way for distinguishing objects of different levels is by introducing types, here we can get by with a very simple type system that only distinguishes functions from their arguments

Types

▷ Types are semantic annotations for terms that prevent antinomies

▷ **Definition 9.2.1.** Given a set $\mathcal{BT}$ of base types, construct function types: $\alpha \to \beta$ is the type of functions with domain type α and range type β. We call the closure $\mathcal{T}$ of $\mathcal{BT}$ under function types the set of types over $\mathcal{BT}$.

▷ **Definition 9.2.2.**

We will use ι for the type ofindividuals and prop for the type of truth values.

▷ **Right Associativity:** The type constructor is used as a right-associative operator, i.e. we use $\alpha \to \beta \to \gamma$ as an abbreviation for $\alpha \to \beta \to \gamma$

▷ **Vector Notation:**

We will use a kind of vector notation for function types, abbreviating $\alpha_1 \to \ldots \to \alpha_n \to \beta$ with $\overline{\alpha}_n \to \beta$.

Syntactical Categories and Types

▷ Now, we can assign types to phrasial categories.

Cat	Type	Intuition
S	prop	truth value
NP	ι	individual
N_{pr}	ι	individuals
VP	$\iota \to$ prop	property
V^i	$\iota \to$ prop	unary predicate
V^t	$\iota \to \iota \to$ prop	binary relation

▷ For the category conj, we cannot get by with a single type. Depending on where it is used, we need the types

- ▷ prop $\to$ prop $\to$ prop for S-coordination in rule $S2$: $S \to S$, conj, S
- ▷ $\iota \to$ prop$\to \iota \to$ prop $\to \iota \to$ prop for VP-coordination in $V3$: $VP \to VP$, conj, VP.
- ▷ **Note:** Computational Linguistics, often uses a different notation for types: e (entiry) for ι, t (truth value) for prop, and $\langle\alpha,\beta\rangle$ for $\alpha \to \beta$ (no bracket elision convention).
  So the type for VP-coordination has the form $\langle\langle\iota,[\rangle ling]t,\langle\langle\iota,[\rangle ling]t,\langle\iota,[\rangle ling]t\rangle\rangle$

For a logic which can really deal with functions, we have to have two properties, which we can already read off the language of mathematics (as the discipine that deals with functions and funcitonals professionally): We

1. need to be able to construct functions from expressions with variables, as in $f(x) = 3x^2+7x+5$, and
2. consider two functions the same, iff they return the same values on the same arguments.

In a logical system (let us for the moment assume a first-order logic with types that can quantify over functions) this gives rise to the following axioms:

Comprehension $\exists F_{\alpha\to\beta}.\forall X_\alpha.F\ X = \mathbf{A}_\beta$

Extensionality $\forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX = GX) \Rightarrow F = G$

The comprehension axioms are computationally very problematic. First, we observe that they are equality axioms, and thus are needed to show that two objects of $\mathrm{PL}\Omega$ are equal. Second we observe that there are countably infinitely many of them (they are parametric in the term $\mathbf{A}$, the type α and the variable name), which makes dealing with them difficult in practice. Finally, axioms with both existential and universal quantifiers are always difficult to reason with.
Therefore we would like to have a formulation of higher-order logic without comprehension axioms. In the next slide we take a close look at the comprehension axioms and transform them into a form without quantifiers, which will turn out useful.

From Comprehension to β-Conversion

▷ $\exists F_{\alpha\to\beta}.\forall X_\alpha.FX = \mathbf{A}_\beta$ for arbitrary variable X_α and term $\mathbf{A}\in wff_\beta(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})$ (for each term $\mathbf{A}$ and each variable X there is a function $f\in\mathcal{D}_{(\alpha\to\beta)}$, with $f(\varphi(X)) = \mathcal{I}_\varphi(\mathbf{A})$)
  - ▷ schematic in α, β, X_α and $\mathbf{A}_\beta$, very inconvenient for deduction

▷ Transformation in $\mathcal{H}_\Omega$
  - ▷ $\exists F_{\alpha\to\beta}.\forall X_\alpha.FX = \mathbf{A}_\beta$
  - ▷ $\forall X_\alpha.(\lambda X_\alpha.\mathbf{A})X = \mathbf{A}_\beta$ $(\exists E)$
    Call the function F whose existence is guaranteed "$(\lambda X_\alpha.\mathbf{A})$"
  - ▷ $(\lambda X_\alpha.\mathbf{A})\mathbf{B} = [\mathbf{B}/X]\mathbf{A}_\beta$ $(\forall E)$, in particular for $\mathbf{B}\in wff_\alpha(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})$.

▷ **Definition 9.2.3.** Axiom of β equality: $(\lambda X_\alpha.\mathbf{A})\ \mathbf{B} = [\mathbf{B}/X](\mathbf{A}_\beta)$

▷ **Idea:** Introduce a new class of formulae (λ-calculus [Chu40])

In a similar way we can treat (functional) extensionality.

From Extensionality to η-Conversion

▷ **Definition 9.2.4.** Extensionality Axiom: $\forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX = GX) \Rightarrow F = G$

▷ **Idea:** Maybe we can get by with a simplified equality schema here as well.

▷ **Definition 9.2.5.** We say that $\mathbf{A}$ and $\lambda X_\alpha.\mathbf{A}\ X$ are η-equal, (write $\mathbf{A}_{\alpha\to\beta} =_\eta (\lambda X_\alpha.\mathbf{A}\ X)$), iff $X \notin \mathrm{free}(\mathbf{A})$.

▷ **Theorem 9.2.6.** *η-equality and Extensionality are equivalent*

▷ *Proof:* We show that η-equality is special case of extensionality; the converse direction is trivial

1. Let $\forall X_\alpha.\mathbf{A}X = \mathbf{B}X$, thus $\mathbf{A}X = \mathbf{B}X$ with $\forall E$
2. $\lambda X_\alpha.\mathbf{A}X = \lambda X_\alpha.\mathbf{B}X$, therefore $\mathbf{A} = \mathbf{B}$ with η
3. Hence $\forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX = GX) \Rightarrow F = G$ by twice $\forall I$.

▷ Axiom of truth values: $\forall F_{\mathrm{prop}}.\forall G_{\mathrm{prop}}.FG \Leftrightarrow F = G$ unsolved.

The price to pay is that we need to pay for getting rid of the comprehension and extensionality axioms is that we need a logic that systematically includes the λ-generated names we used in the transformation as (generic) witnesses for the existential quantifier. Alonzo Church did just that with his "simply typed λ-calculus" which we will introduce next.
This is all very nice, but what do we actually translate into?

9.3 Translation for Fragment 3

Translations for Fragment 3

▷ We will look at the new translation rules (the rest stay the same).

$$T1\colon [X_{\mathrm{NP}}, Y_{VP}]_S \rightsquigarrow VP'(\mathrm{NP}'),\ T3\colon [X_{VP}, Y_{\mathrm{conj}}, Z_{VP}]_{VP} \rightsquigarrow \mathrm{conj}'(VP', VP'),$$
$$T4\colon [X_{V^t}, Y_{\mathrm{NP}}]_{VP} \rightsquigarrow V^{t\prime}(\mathrm{NP}')$$

▷ The lexical insertion rules will give us two items each for *is*, *and*, and *or*, corresponding to the two types we have given them.

word	type	term	case
BE$_{pred}$	$\iota \to \mathrm{prop} \to \iota \to \mathrm{prop}$	$\lambda P_{\iota\to\mathrm{prop}}.P$	adjective
BE$_{eq}$	$\iota \to \iota \to \mathrm{prop}$	$\lambda X_\iota Y_\iota.X = Y$	verb
and	$\mathrm{prop} \to \mathrm{prop} \to \mathrm{prop}$	$V!$	S-coord.
and	$\iota \to \mathrm{prop} \to \iota \to \mathrm{prop} \to \iota \to \mathrm{prop}$	$\lambda F_{\iota\to\mathrm{prop}} G_{\iota\to\mathrm{prop}} X_\iota.F(X) \wedge G(X)$	VP-coord.
or	$\mathrm{prop} \to \mathrm{prop} \to \mathrm{prop}$	$\vee$	S-coord.
or	$\iota \to \mathrm{prop} \to \iota \to \mathrm{prop} \to \iota \to \mathrm{prop}$	$\lambda F_{\iota\to\mathrm{prop}} G_{\iota\to\mathrm{prop}} X_\iota.F(X) \vee G(X)$	VP-coord.
didn't	$\iota \to \mathrm{prop} \to \iota \to \mathrm{prop}$	$\lambda P_{\iota\to\mathrm{prop}} X_\iota.\neg P\ X$	

Need to assume the logical connectives as constants of the λ-calculus.

▷ **Note:** With these definitions, it is easy to restrict ourselves to binary branching in the syntax of the fragment.

- **Definition 9.3.1 (Translation of non-branching nodes).** If φ is a non-branching node with daughter ψ, then the translation φ' of φ is given by the translation ψ' of ψ.

- **Definition 9.3.2 (Translation of branching nodes (Function Application)).** If φ is a branching node with daughters ψ and θ, where ψ' is an expression of type $\alpha \to \beta$ and θ' is an expression of type α, then $\varphi' = \psi'\ \theta'$.

- **Note on notation:** We now have higher-order constants formed using words from the fragment, which are not (or are not always) translations of the words from which they are formed. We thus need some new notation to represent the translation of an expression from the fragment. We will use the notation introduced above, i.e. *john′* is the translation of the word *John*. We will continue to use primes to indicate that something is an expression (e.g. john). Words of the fragment of English should be either underlined or italicized.

Translation Example

▷ **Example 9.3.3.** *Ethel howled and screamed* to

$$\begin{array}{ll} & (\lambda F_{\iota\to\mathsf{prop}} G_{\iota\to\mathsf{prop}} X_\iota . F(X) \wedge G(X))\ \mathsf{howls}\ \mathsf{screams}\ \mathsf{ethel} \\ \to_\beta & (\lambda G_{\iota\to\mathsf{prop}} X_\iota . \mathsf{howls}(X) \wedge G(X))\ \mathsf{screams}\ \mathsf{ethel} \\ \to_\beta & (\lambda X_\iota . \mathsf{howls}(X) \wedge \mathsf{screams}(X))\ \mathsf{ethel} \\ \to_\beta & \mathsf{howls}(\mathsf{ethel}) \wedge \mathsf{screams}(\mathsf{ethel}) \end{array}$$

Higher-Order Logic without Quantifiers (HOL_{NQ})

▷ **Problem:** Need a logic like $\mathrm{PL}^{\mathrm{nq}}$, but with λ-terms to interpret $\mathcal{F}_3$ into.

▷ **Idea:** Re-use the syntactical framework of $\Lambda^\to$.

▷ **Definition 9.3.4.** Let HOL_{NQ} be an instance of $\Lambda^\to$, with $\mathcal{BT} = \{\iota, \mathsf{prop}\}$, $\wedge \in \Sigma_{\mathsf{prop}\to\mathsf{prop}\to\mathsf{prop}}$, $\neg \in \Sigma_{\mathsf{prop}\to\mathsf{prop}}$, and $= \in \Sigma_{\alpha\to\alpha\to\mathsf{prop}}$ for all types α.

▷ **Idea:** To extend this to a semantics for HOL_{NQ}, we only have to say something about the base type prop, and the logical constants $\neg_{\mathsf{prop}\to\mathsf{prop}}$, $\wedge_{\mathsf{prop}\to\mathsf{prop}\to\mathsf{prop}}$, and $=_{\alpha\to\alpha\to\mathsf{prop}}$.

▷ **Definition 9.3.5.** We define the semantics of HOL_{NQ} by setting

1. $\mathcal{D}_{\mathsf{prop}} = \{\mathsf{T}, \mathsf{F}\}$; the set of truth values
2. $\mathcal{I}(\neg) \in \mathcal{D}_{(\mathsf{prop}\to\mathsf{prop})}$, is the function $\{\mathsf{F} \mapsto \mathsf{T}, \mathsf{T} \mapsto \mathsf{F}\}$
3. $\mathcal{I}(\wedge) \in \mathcal{D}_{(\mathsf{prop}\to\mathsf{prop}\to\mathsf{prop})}$ is the function with $\mathcal{I}(\wedge)@\langle \mathsf{a}, \mathsf{b}\rangle = \mathsf{T}$, iff $\mathsf{a} = \mathsf{T}$ and $\mathsf{b} = \mathsf{T}$.
4. $\mathcal{I}(=) \in \mathcal{D}_{(\alpha\to\alpha\to\mathsf{prop})}$ is the identity relation on $\mathcal{D}_\alpha$.

You may be worrying that we have changed our assumptions about the denotations of predicates. When we were working with $\mathrm{PL}^{\mathrm{nq}}$ as our translation language, we assumed that one-place predicates denote sets of individuals, that two-place predicates denote sets of pairs of individuals, and so on. Now, we have adopted a new translation language, HOL_{NQ}, which interprets all predicates as functions of one kind or another.

The reason we can do this is that there is a systematic relation between the functions we now assume as denotations, and the sets we used to assume as denotations. The functions in question are the *characteristic functions* of the old sets, or are curried versions of such functions.

Recall that we have characterized sets extensionally, i.e. by saying what their members are. A characteristic function of a set A is a function which "says" which objects are members of A. It does this by giving one value (for our purposes, the value 1) for any argument which is a member of A, and another value, (for our purposes, the value 0), for anything which is not a member of the set.

Definition 9.3.6 (Characteristic function of a set). f_S is the characteristic function of the set S iff $f_S(a) = \mathsf{T}$ if $\mathsf{a} \in S$ and $f_S(a) = \mathsf{F}$ if $\mathsf{a} \notin S$.

Thus any function in $\mathcal{D}_{(\iota\to\mathrm{prop})}$ will be the characteristic function of some set of individuals. So, for example, the function we assign as denotation to the predicate *run* will return the value T for some arguments and F for the rest. Those for which it returns T correspond exactly to the individuals which belonged to the set *run* in our old way of doing things.

Now, consider functions in $\mathcal{D}_{(\iota\to\iota\to\mathrm{prop})}$. Recall that these functions are equivalent to two-place relations, i.e. functions from pairs of entities to truth values. So functions of this kind are characteristic functions of sets of pairs of individuals.

In fact, any function which ultimately maps an argument to $\mathcal{D}_{\mathrm{prop}}$ is a characteristic function of some set. The fact that many of the denotations we are concerned with turn out to be characteristic functions of sets will be very useful for us, as it will allow us to go backwards and forwards between "set talk" and "function talk," depending on which is easier to use for what we want to say.

9.4 Simply Typed λ-Calculus

In this section we will present a logic that can deal with functions – the simply typed λ-calculus. It is a typed logic, so everything we write down is typed (even if we do not always write the types down).

Simply typed λ-Calculus (Syntax)

- ▷ **Definition 9.4.1.** Signature $\Sigma_\mathcal{T} = \bigcup_{\alpha\in\mathcal{T}} \Sigma_\alpha$ (includes countably infinite signatures Σ_α^{Sk} of Skolem contants).
- ▷ $\mathcal{V}_\mathcal{T} = \bigcup_{\alpha\in\mathcal{T}} \mathcal{V}_\alpha$, such that $\mathcal{V}_\alpha$ are countably infinite.
- ▷ **Definition 9.4.2.** We call the set $\mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ defined by the rules
  - ▷ $\mathcal{V}_\alpha \cup \Sigma_\alpha \subseteq \mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$
  - ▷ If $\mathbf{C} \in \mathit{wff}_{\alpha\to\beta}(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$, then $\mathbf{C}\ \mathbf{A} \in \mathit{wff}_\beta(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$
  - ▷ If $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$, then $\lambda X_\beta.\mathbf{A} \in \mathit{wff}_{\beta\to\alpha}(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$

  the set of well typed formulae of type α over the signature $\Sigma_\mathcal{T}$ and use $\mathit{wff}_\mathcal{T}(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T}) := \bigcup_{\alpha\in\mathcal{T}} \mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ for the set of all well-typed formulae.
- ▷ **Definition 9.4.3.** We will call all occurrences of the variable X in $\mathbf{A}$ bound in $\lambda X.\mathbf{A}$. Variables that are not bound in $\mathbf{B}$ are called free in $\mathbf{B}$.
- ▷ Substitutions are well typed, i.e. $\sigma(X_\alpha) \in \mathit{wff}_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ and capture-avoiding.
- ▷ **Definition 9.4.4 (Simply Typed λ-Calculus).** The simply typed λ calculus $\Lambda^\to$ over a signature $\Sigma_\mathcal{T}$ has the formulae $\mathit{wff}_\mathcal{T}(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ (they are called λ-terms) and the following equalities:
  - ▷ α conversion: $(\lambda X.\mathbf{A}) =_\alpha (\lambda Y.[Y/X](\mathbf{A}))$.
  - ▷ β conversion: $(\lambda X.\mathbf{A})\ \mathbf{B} =_\beta [\mathbf{B}/X](\mathbf{A})$.

▷ η conversion: $(\lambda X.\mathbf{A}\ X) =_{\eta} \mathbf{A}$ if $X \notin \mathrm{free}(\mathbf{A})$.

The intuitions about functional structure of λ-terms and about free and bound variables are encoded into three transformation rules $\Lambda^{\to}$: The first rule (α-conversion) just says that we can rename bound variables as we like. β conversion codifies the intuition behind function application by replacing bound variables with argument. The equality relation induced by the η-reduction is a special case of the extensionality principle for functions ($f = g$ iff $f(a) = g(a)$ for all possible arguments a): If we apply both sides of the transformation to the same argument – say $\mathbf{B}$ and then we arrive at the right hand side, since $(\lambda X_{\alpha}.\mathbf{A}\ X)\mathbf{B} =_{\beta} \mathbf{A}\ \mathbf{B}$.
We will use a set of bracket elision rules that make the syntax of $\Lambda^{\to}$ more palatable. This makes $\Lambda^{\to}$ expressions look much more like regular mathematical notation, but hides the internal structure. Readers should make sure that they can always reconstruct the brackets to make sense of the syntactic notions below.

Simply typed λ-Calculus (Notations)

▷ **Application is left-associative:**
We abbreviate $\mathbf{F}\ \mathbf{A}^1\ \mathbf{A}^2\ \ldots\ \mathbf{A}^n$ with $\mathbf{F}(\mathbf{A}^1,\ldots,\mathbf{A}^n)$ eliding the brackets and further with $\mathbf{F}\ \overline{\mathbf{A}^n}$ in a kind of vector notation.

▷ **Andrews' dot Notation:** A **.** stands for a left bracket whose partner is as far right as is consistent with existing brackets; i.e. $\mathbf{A}\ .\mathbf{B}\ \mathbf{C}$ abbreviates $\mathbf{A}\ (\mathbf{B}\ \mathbf{C})$.

▷ **Abstraction is right-associative:**
We abbreviate $\lambda X^1.\lambda X^2.\cdots\lambda X^n.\mathbf{A}\cdots$ with $\lambda X^1\ldots X^n.\mathbf{A}$ eliding brackets, and further to $\lambda\overline{X^n}.\mathbf{A}$ in a kind of vector notation.

▷ **Outer brackets:** Finally, we allow ourselves to elide outer brackets where they can be inferred.

Intuitively, $\lambda X.\mathbf{A}$ is the function f, such that $f(\mathbf{B})$ will yield $\mathbf{A}$, where all occurrences of the formal parameter X are replaced by $\mathbf{B}$.[6] In this presentation of the simply typed λ-calculus we build-in $=_{\alpha}$-equality and use capture-avoiding substitution directly. A clean introduction would followed the steps in **??** by introducing substitutions with a substitutability condition like the one in Definition 8.1.26, then establishing the soundness of $=_{\alpha}$ conversion, and only then postulating defining capture-avoiding substitution application as in **??**. The development for $\Lambda^{\to}$ is directly parallel to the one for PL^1, so we leave it as an exercise to the reader and turn to the computational properties of the λ-calculus.
Computationally, the λ-calculus obtains much of its power from the fact that two of its three equalities can be oriented into a reduction system. Intuitively, we only use the equalities in one direction, i.e. in one that makes the terms "simpler". If this terminates (and is confluent), then we can establish equality of two λ-terms by reducing them to normal forms and comparing them structurally. This gives us a decision procedure for equality. Indeed, we have these properties in $\Lambda^{\to}$ as we will see below.

$=_{\alpha\beta\eta}$-Equality (Overview)

[6] EdNote: rationalize the semantic macros for syntax!

▷ **Definition 9.4.5.** Reduction with $\begin{cases} =_\beta : (\lambda X.\mathbf{A})\,\mathbf{B} \rightarrow_\beta [\mathbf{B}/X](\mathbf{A}) \\ =_\eta : (\lambda X.\mathbf{A}\; X) \rightarrow_\eta \mathbf{A} \end{cases}$ under $=_\alpha$: $\begin{array}{c}\lambda X.\mathbf{A} \\ =_\alpha \\ \lambda Y.[Y/X](\mathbf{A})\end{array}$

▷ **Theorem 9.4.6.** *β-reduction is well-typed, terminating and confluent in the presence of α-conversion.*

▷ **Definition 9.4.7 (Normal Form).** We call a λ-term $\mathbf{A}$ a normal form (in a reduction system $\mathcal{E}$), iff no rule (from $\mathcal{E}$) can be applied to $\mathbf{A}$.

▷ **Corollary 9.4.8.** *$=_{\beta\eta}$-reduction yields unique normal forms (up to $=_\alpha$-equivalence).*

We will now introduce some terminology to be able to talk about λ terms and their parts.

Syntactic Parts of λ-Terms

▷ **Definition 9.4.9 (Parts of λ-Terms).** We can always write a λ-term in the form $\mathbf{T} = \lambda X^1 \ldots X^k.\mathbf{H}\mathbf{A}^1 \ldots \mathbf{A}^n$, where $\mathbf{H}$ is not an application. We call

- ▷ $\mathbf{H}$ the syntactic head of $\mathbf{T}$
- ▷ $\mathbf{H}(\mathbf{A}^1, \ldots, \mathbf{A}^n)$ the matrix of $\mathbf{T}$, and
- ▷ $\lambda X^1 \ldots X^k.$ (or the sequence $X^1, \ldots, X^k$) the binder of $\mathbf{T}$

▷ **Definition 9.4.10.**

Head reduction always has a unique β redex

$$(\lambda \overline{X^n}.\lambda Y.\mathbf{A}(\mathbf{B}^2, \ldots, \mathbf{B}^n)) \rightarrow^h_\beta (\lambda \overline{X^n}.[\mathbf{B}^1/Y](\mathbf{A})(\mathbf{B}^2, \ldots, \mathbf{B}^n))$$

▷ **Theorem 9.4.11.** *The syntactic heads of β-normal forms are constant or variables.*

▷ **Definition 9.4.12.** Let $\mathbf{A}$ be a λ-term, then the syntactic head of the β-normal form of $\mathbf{A}$ is called the head symbol of $\mathbf{A}$ and written as $\mathrm{head}(\mathbf{A})$. We call a λ-term a j-projection, iff its head is the jth bound variable.

▷ **Definition 9.4.13.** We call a λ-term a η long form, iff its matrix has base type.

▷ **Definition 9.4.14.** η Expansion makes η long forms

$$\eta\left[(\lambda X^1 \ldots X^n.\mathbf{A})\right] := (\lambda X^1 \ldots X^n.\lambda Y^1 \ldots Y^m.\mathbf{A}(Y^1, \ldots, Y^m))$$

▷ **Definition 9.4.15.** Long $\beta\eta$ normal form, iff it is β normal and η-long.

η long forms are structurally convenient since for them, the structure of the term is isomorphic to the structure of its type (argument types correspond to binders): if we have a term $\mathbf{A}$ of type $\overline{\alpha}_n \rightarrow \beta$ in η-long form, where $\beta \in \mathcal{BT}$, then $\mathbf{A}$ must be of the form $\lambda \overline{X^n_\alpha}.\mathbf{B}$, where $\mathbf{B}$ has type β. Furthermore, the set of η-long forms is closed under β-equality, which allows us to treat the two equality theories of $\Lambda^\rightarrow$ separately and thus reduce argumentational complexity.

The semantics of $\Lambda^\rightarrow$ is structured around the types. Like the models we discussed before, a model (we call them "algebras", since we do not have truth values in $\Lambda^\rightarrow$) is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$, where $\mathcal{D}$ is the universe of discourse and $\mathcal{I}$ is the interpretation of constants.

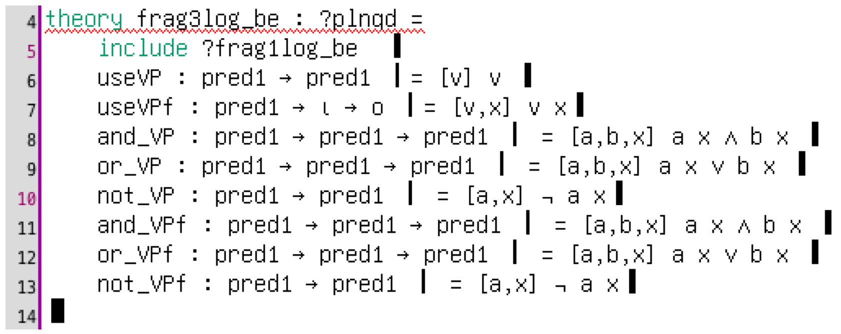
Semantics of $\Lambda^{\rightarrow}$

- ▷ **Definition 9.4.16.** We call a collection $\mathcal{D}_{\mathcal{T}}{:=}\{\mathcal{D}_\alpha|\alpha{\in}\mathcal{T}\}$ a typed collection (of sets) and a collection $f_{\mathcal{T}}\colon\mathcal{D}_{\mathcal{T}}{\rightarrow}\mathcal{E}_{\mathcal{T}}$, a typed function, iff $f_\alpha\colon\mathcal{D}_\alpha{\rightarrow}\mathcal{E}_\alpha$.
- ▷ **Definition 9.4.17.** A typed collection $\mathcal{D}_{\mathcal{T}}$ is called a frame, iff $\mathcal{D}_{(\alpha\rightarrow\beta)}\subseteq\mathcal{D}_\alpha{\rightarrow}\mathcal{D}_\beta$
- ▷ **Definition 9.4.18.** Given a frame $\mathcal{D}_{\mathcal{T}}$, and a typed function $\mathcal{I}\colon\Sigma{\rightarrow}\mathcal{D}$, then we call $\mathcal{I}_\varphi\colon wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}}){\rightarrow}\mathcal{D}$ the value function induced by $\mathcal{I}$, iff
  - ▷ $\mathcal{I}_\varphi|_{\mathcal{V}_{\mathcal{T}}}=\varphi$, $\qquad\mathcal{I}_\varphi|_{\Sigma_{\mathcal{T}}}=\mathcal{I}$
  - ▷ $\mathcal{I}_\varphi(\mathbf{A}\ \mathbf{B})=\mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
  - ▷ $\mathcal{I}_\varphi(\lambda X_\alpha.\mathbf{A})$ is that function $f{\in}\mathcal{D}_{(\alpha\rightarrow\beta)}$, such that $f(a)=\mathcal{I}_{(\varphi,[a/X])}(\mathbf{A})$ for all $a{\in}\mathcal{D}_\alpha$
- ▷ **Definition 9.4.19.** We call a frame $\langle\mathcal{D},\mathcal{I}\rangle$ comprehension closed or a $\Sigma_{\mathcal{T}}$-algebra, iff $\mathcal{I}_\varphi\colon wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}}){\rightarrow}\mathcal{D}$ is total. (every λ-term has a value)

Excursion: We will discuss the semantics, computational properties, and a more modern presentation of the λ calculus in**??**.

Domain Theory for $\mathcal{F}_3$

- ▷ **Observation 1:** We we can reuse the lexicon theories from $\mathcal{F}_1$
- ▷ **Observation 2:** We we can even reuse the grammar theory from $\mathcal{F}_1$, if we extend it in the obvious way (MMT has all we need)



```
theory frag3log_be : ?plnqd =
    include ?frag1log_be  ❙
    useVP : pred1 → pred1 ❘= [v] v  ❙
    useVPf : pred1 → ι → o ❘= [v,x] v x❙
    and_VP : pred1 → pred1 → pred1 ❘ = [a,b,x] a x ∧ b x ❙
    or_VP : pred1 → pred1 → pred1 ❘ = [a,b,x] a x ∨ b x ❙
    not_VP : pred1 → pred1 ❘ = [a,x] ¬ a x❙
    and_VPf : pred1 → pred1 → pred1 ❘ = [a,b,x] a x ∧ b x ❙
    or_VPf : pred1 → pred1 → pred1 ❘ = [a,b,x] a x ∨ b x ❙
    not_VPf : pred1 → pred1 ❘ = [a,x] ¬ a x❙
❚
```

Chapter 10

Fragment 4: Noun Phrases and Quantification

10.1 Overview/Summary so far

Where we started: A *VP*-less fragment and $\mathrm{PL}^{\mathrm{nq}}$.:

$\mathrm{PL}^{\mathrm{nq}}$	Fragment of English
Syntax: Definition of wffs	Syntax: Definition of allowable sentences
Semantics: Model theory	SEMANTICS BY TRANSLATION

What we did:

- Tested the translation by testing predictions: semantic tests of entailment.
- More testing: syntactic tests of entailment. For this, we introduced the model generation calculus. We can make this move from semantic proofs to syntactic ones safely, because we know that $\mathrm{PL}^{\mathrm{nq}}$ is sound and complete.
- Moving beyond semantics: Used model generation to predict interpretations of semantically under-determined sentence types.

Where we are now: A fragment with a *VP* and HOL_{NQ}.: We expanded the fragment and began to consider data which demonstrate the need for a *VP* in any adequate syntax of English, and the need for connectives which connect *VP*s and other expression types. At this point, the resources of $\mathrm{PL}^{\mathrm{nq}}$ no longer sufficed to provide adequate compositional translations of the fragment. So we introduced a new translation language, HOL_{NQ}. However, the general picture of the table above does not change; only the translation language itself changes.
Some discoveries:

- The task of giving a semantics via translation for natural language includes as a subtask the task of finding an adequate translation language.
- Given a typed language, function application is a powerful and very useful tool for modeling the derivation of the interpretation of a complex expression from the interpretations of its parts and their syntactic arrangement. To maintain a transparent interface between syntax and semantics, binary branching is preferable. Happily, this is supported by syntactic evidence.
- Syntax and semantics interact: Syntax forces us to introduce *VP*. The assumption of compositionality then forces us to translate and interpret this new category.

- We discovered that the "logical operators" of natural language can't always be translated directly by their formal counterparts. Their formal counterparts are all sentence connectives; but English has versions of these connectives for other types of expressions. However, we can use the familiar sentential connectives to derive appropriate translations for the differently-typed variants.

Some issues about translations: HOL_{NQ} provides multiple syntactically and semantically equivalent versions of many of its expressions. For example:

1. Let runs be an HOL_{NQ} constant of type $\iota \to \mathrm{prop}$. Then $\mathrm{runs} = \lambda X.\mathrm{runs}(X)$
2. Let loves be an HOL_{NQ} constant of type $\iota \to \iota \to \mathrm{prop}$. Then $\mathrm{loves} = \lambda X.\lambda Y.\mathrm{loves}(X, Y)$
3. Similarly, $\mathrm{loves}(a) = \lambda Y.\mathrm{loves}(a, Y)$
4. And $\mathrm{loves}(\mathrm{jane}, \mathrm{george}) = (\lambda X.\lambda Y.\mathrm{loves}(X, Y))\ \mathrm{jane}(\mathrm{george})$

Logically, both sides of the equations are considered equal, since $=_\eta$-equality (remember $(\lambda X.\mathbf{A}\ X) \to_\eta \mathbf{A}$, if $X \notin \mathrm{free}(\mathbf{A})$) is built into HOL_{NQ}. In fact all the right-hand sides are $=_\eta$-expansions of the left-hand sides. So you can use both, as you choose in principle.

But practically, you like to know which to give when you are asked for a translation? The answer depends on what you are using it for. Let's introduce a distinction between *reduced translations* and *unreduced translations*. An unreduced translation makes completely explicit the type assignment of each expression and the mode of composition of the translations of complex expressions, i.e. how the translation is derived from the translations of the parts. So, for example, if you have just offered a translation for a lexical item (say, *and* as a V^t connective), and now want to demonstrate how this lexical item works in a sentence, give the unreduced translation of the sentence in question and then demonstrate that it reduces to the desired reduced version.

The reduced translations have forms to which the deduction rules apply. So always use reduced translations for input in model generation: here, we are assuming that we have got the translation right, and that we know how to get it, and are interested in seeing what further deductions can be performed.

Where we are going: We will continue to enhance the fragment both by introducing additional types of expressions and by improving the syntactic analysis of the sentences we are dealing with. This will require further enrichments of the translation language. Next steps:

- Analysis of NP.
- Treatment of adjectives.
- Quantification

10.2 Fragment 4

New Data (more Noun Phrases)

▷ We want to be able to deal with the following sentences (without the "the-NP" trick)

1. *Peter loved the cat.*, but not * *Peter loved the the cat.*
2. *John killed a cat with a white tail.*
3. *Peter chased the gangster in the car.*
4. *Peter loves every cat.*
5. *Every man loves a woman.*

The first example suggests that we need a full and uniform treatment of determiners like *the*, *a*, and *every*. The second and third introduce a new phenomenon: prepositional phrases like *with a hammer/mouse*; these are essentially nominal phrases that modify the meaning of other phrases via a preposition like *with*, *in*, *on*, *at*. These two show that the prepositional phrase can modify the verb or the object.

New Grammar in Fragment 4 (Common Noun Phrases)

▷ To account for the syntax we extend the functionality of noun phrases.

▷ **Definition 10.2.1.** $\mathcal{F}_4$ adds the rules on the right to $\mathcal{F}_3$ (on the left):

$S1\colon\ S \to \mathrm{NP}, VP_{+fin},$ $\quad S2\colon\ S \to S, \mathrm{Sconj},$	
$V1\colon\ VP_{\pm fin} \to V^i_{\pm fin},$	$N3\colon\ \mathrm{NP} \to \mathrm{DetCNP},$ $\quad N4\colon\ \mathrm{CNP} \to N,$
$V2\colon\ VP_{\pm fin} \to V^t_{\pm fin}, \mathrm{CNP},$	$N5\colon\ \mathrm{CNP} \to \mathrm{PP},$ $\quad N6\colon\ \mathrm{CNP} \to \mathrm{Adj},$
$V3\colon\ VP_{\pm fin} \to VP_{\pm fin}, \mathrm{VPconj}_{\pm fin},$	$P1\colon\ \mathrm{PP} \to P, \mathrm{NP},$ $\quad S3\colon\ \mathrm{Sconj} \to \mathrm{conj}, S,$
$V4\colon\ VP_{+fin} \to BE_{=}, \mathrm{NP},$	$V4\colon\ \mathrm{VPconj}_{\pm fin} \to \mathrm{conj}, VP_{\pm fin},$
$V5\colon\ VP_{+fin} \to BE_{pred}, \mathrm{Adj},$	$L1\colon\ P \to \mathrm{with} \mid \mathrm{of} \mid \ldots$
$V6\colon\ VP_{+fin} \to \mathrm{didn't}, VP_{-fin},$ $\quad N1\colon\ \mathrm{NP} \to N_{\mathrm{pr}},$	
$N2\colon\ \mathrm{NP} \to \mathrm{Pron}$	

▷ **Definition 10.2.2.** A common noun is a noun that describes a type, for example *woman*, or *philosophy* rather than an individual, such as *Amelia Earhart* (proper name).

Note: Parentheses indicate optionality of a constituent. We assume appropriate lexical insertion rules without specification.

Implementing Fragment 4 in GF (Grammar)

▷ The grammar of Fragment 4 only differs from that of Fragment 4 by

- ▷ common noun phrases: `cat CNP; Npr; lincat CNP = NounPhraeType;`
- ▷ prepositional phrases : `cat PP; Det; Prep; lincat Npr, Det, Prep, PP = {s: Str}`
- ▷ new grammar rules

```
useDet : Det -> CNP -> NP; -- every book
useNpr : Npr -> NP; -- Bertie
useN : N -> CNP; -- book
usePrep : Prep -> NP -> PP; -- with a book
usePP : PP -> CNP -> CNP; -- teacher with a book
```

- ▷ grammar rules for "special" words that might not belong into the lexicon

Abstract

```
with_Prep
: Prep;
of_Prep : Prep;
the_Det : Det;
every_Det : Det;
a_Det : Det;
```

English

```
with_Prep
= mkPrep "with";
of_Prep = mkPrep "of";
the_Det = mkDet "the";
every_Det
= mkDet "every";
a_Det = mkDet "a";
```

Implementing Fragment 4 in GF (Grammar)

▷ English Paradigms to deal with (common) noun phrases

- ▷ Another case for `mkNP`

```
mkNP : Str −> (Case => Str) −> NP
     = \prefix,t −> lin NP { s = table { nom => prefix ++ t!nom;
                                         acc => prefix ++ t!acc}};

mkNpr : Str −> Npr = \name −> lin Npr { s = name };
mkDet : Str −> Det = \every −> lin Det { s = every };
mkPrep : Str −> Prep = \p −> lin Prep { s = p };
mkPP : Str −> PP = \s −> lin PP { s = s };
mkCNP = overload {
  mkCNP : Str −> CNP
        = \book −> lin CNP { s = table { nom => book; acc => book } };
  mkCNP : (Case => Str) −> Str −> CNP
        = \t,suffix −> lin CNP { s = table { nom => (t!nom) ++ suffix;
                                             acc => (t!acc) ++ suffix}};};
```

If we assume that $\forall X.\mathrm{boy}(X) \Rightarrow \mathrm{runs}(X)$ is an adequate translation of *Every boy runs*, and $\exists X.\mathrm{boy}(X) \wedge \mathrm{runs}(X)$ one for *Some boy runs*, then we obtain the translations of the determiners by straightforward $=_\beta$-expansion.

▷ Translation of Determiners and Quantifiers

▷ **Idea:** We establish the semantics of quantifying determiners by $=_\beta$-expansion.

1. assume that we are translating into a λ-calculus with quantifiers and that $\forall X.\mathrm{boy}(X) \Rightarrow \mathrm{runs}(X)$ translates *Every boy runs*, and $\exists X.\mathrm{boy}(X) \wedge \mathrm{runs}(X)$ for *Some boy runs*
2. $\mathbb{\forall}{:=}(\lambda P_{\iota\to\mathrm{prop}} Q_{\iota\to\mathrm{prop}}.(\forall.P(X) \Rightarrow Q(X)))$ for *every*. (subset relation)
3. $\mathbb{\exists}{:=}(\lambda P_{\iota\to\mathrm{prop}} Q_{\iota\to\mathrm{prop}}.(\exists.P(X) \wedge Q(X)))$ for *some*. (nonempty intersection)

▷ **Problem:** Linguistic Quantifiers take two arguments (restriction and scope), logical ones only one! (in logics, restriction is the universal set)

▷ We cannot treat *the* with regular quantifiers (new logical constant; see below)

▷ **Definition 10.2.3.** We translate the to $\tau{:=}(\lambda P_{\iota\to\mathrm{prop}} Q_{\iota\to\mathrm{prop}}.Q\ \iota\ P)$, where ι is a new operator that given a set returns its (unique) member.

▷ **Example 10.2.4.** This translates *The pope spoke* to $\tau(\mathrm{pope},\mathrm{speaks})$, which $=_\beta$-reduces to $\mathrm{speaks}(\iota\ \mathrm{pope})$.

Note that if we interpret objects of type $\iota \to \mathrm{prop}$ as sets, then the denotations of *boy* and *run* are sets (of boys and running individuals). Then the denotation of *every* is a relation between sets; more specifically the subset relation. As a consequence, *All boys run* is true if the set of boys is a subset of the set of running individuals. For *some* the relation is the non-empty intersection relation, *some boy runs* is true if the intersection of set of boys and the set of running individuals is non-empty.

Note that there is a mismatch in the "arity" of linguistic and logical notions of quantifiers here. Linguistic quantifiers take two arguments, the restriction (in our example *boy*) and the predication (*run*). The logical quantifiers only take one argument, the predication $\mathbf{A}$ in $\forall X.\mathbf{A}$. In a way, the restriction is always the universal set. In our model, we have modeled the linguistic quantifiers by adding the restriction with a connective (implication for the universal quantifier and conjunction for the existential one).

10.3 Inference for Fragment 4

10.3.1 Quantifiers and Equality in Higher-Order Logic

There is a more elegant way to treat quantifiers in $\mathrm{HOL}^{\rightarrow}$. It builds on the realization that the λ-abstraction is the only variable binding operator we need, quantifiers are then modeled as second-order logical constants. Note that we do not have to change the syntax of $\mathrm{HOL}^{\rightarrow}$ to introduce quantifiers; only the "lexicon", i.e. the set of logical constants. Since Π^α and σ^α are logical constants, we need to fix their semantics.

Higher-Order Abstract Syntax

▷ **Idea:** In $\mathrm{HOL}^{\rightarrow}$, we already have variable binder: λ, use that to treat quantification.

▷ **Definition 10.3.1.** We assume logical constants Π^α and σ^α of type $\alpha \rightarrow \mathrm{prop} \rightarrow \mathrm{prop}$.

Regain quantifiers as abbreviations:

$$(\forall X_\alpha.\mathbf{A}):=\Pi^\alpha\ (\lambda X_\alpha.\mathbf{A}) \qquad (\exists X_\alpha.\mathbf{A}):=\sigma^\alpha\ (\lambda X_\alpha.\mathbf{A})$$

▷ **Definition 10.3.2.** We must fix the semantics of logical constants:

1. $\mathcal{I}(\Pi^\alpha)(p) = \mathsf{T}$, iff $p(a) = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$ (i.e. if p is the universal set)
2. $\mathcal{I}(\sigma^\alpha)(p) = \mathsf{T}$, iff $p(a) = \mathsf{T}$ for some $a \in \mathcal{D}_\alpha$ (i.e. iff p is non-empty)

▷ With this, we re-obtain the semantics we have given for quantifiers above:

$$\mathcal{I}_\varphi(\forall X_\iota.\mathbf{A}) = \mathcal{I}_\varphi(\Pi^\iota\ (\lambda X_\iota.\mathbf{A})) = \mathcal{I}(\Pi^\iota)(\mathcal{I}_\varphi(\lambda X_\iota.\mathbf{A})) = \mathsf{T}$$

iff $\mathcal{I}_\varphi(\lambda X_\iota.\mathbf{A})(a) = \mathcal{I}_{([a/X],\varphi)}(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$

Equality

▷ **Definition 10.3.3 (Leibniz equality).** $\mathbf{Q}^\alpha\mathbf{A}_\alpha\mathbf{B}_\alpha = \forall P_{\alpha\rightarrow\mathrm{prop}}.P\mathbf{A} \Leftrightarrow P\mathbf{B}$ (indiscernability)

▷ **Note:** $\forall P_{\alpha\rightarrow\mathrm{prop}}.P\mathbf{A} \Rightarrow P\mathbf{B}$ (get the other direction by instantiating P with Q, where $QX \Leftrightarrow (\neg PX)$)

▷ **Theorem 10.3.4.** *If $\mathcal{M} = \langle\mathcal{D},\mathcal{I}\rangle$ is a standard model, then $\mathcal{I}_\varphi(\mathbf{Q}^\alpha)$ is the identity relation on $\mathcal{D}_\alpha$.*

▷ **Definition 10.3.5 (Notation).** We write $\mathbf{A} = \mathbf{B}$ for $\mathbf{Q}\mathbf{A}\mathbf{B}$ ($\mathbf{A}$ and $\mathbf{B}$ are equal, iff there is no property P that can tell them apart.)

▷ *Proof:*

1. $\mathcal{I}_\varphi(\mathbf{QAB}) = \mathcal{I}_\varphi(\forall P.P\mathbf{A} \Rightarrow P\mathbf{B}) = \mathsf{T}$, iff $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{A} \Rightarrow P\mathbf{B}) = \mathsf{T}$ for all $r \in \mathcal{D}_{(\alpha\to\mathsf{prop})}$.
2. For $\mathbf{A} = \mathbf{B}$ we have $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{A}) = r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{F}$ or $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{B}) = r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{T}$.
3. Thus $\mathcal{I}_\varphi(\mathbf{QAB}) = \mathsf{T}$.
4. Let $\mathcal{I}_\varphi(\mathbf{A}) \neq \mathcal{I}_\varphi(\mathbf{B})$ and $r = \{\mathcal{I}_\varphi(\mathbf{A})\} \in \mathcal{D}_{(\alpha\to\mathsf{prop})}$ (exists in a standard model)
5. so $r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{T}$ and $r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{F}$
6. $\mathcal{I}_\varphi(\mathbf{QAB}) = \mathsf{F}$, as $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{A} \Rightarrow P\mathbf{B}) = \mathsf{F}$, since $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{A}) = r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{T}$ and $\mathcal{I}_{(\varphi,[r/P])}(P\mathbf{B}) = r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{F}$.

Alternative: HOL$^\infty$

▷ **Definition 10.3.6.** There is only one logical constant in HOL$^\infty$: $q^\alpha \in \Sigma_{\alpha\to\alpha\to\mathsf{prop}}$ with $\mathcal{I}(q^\alpha)(a,b) = \mathsf{T}$, iff $a = b$.

We define the rest as below: Definitions (D) and Notations (N)

N	$\mathbf{A}_\alpha = \mathbf{B}_\alpha$	for	$q^\alpha \mathbf{A}_\alpha \mathbf{B}_\alpha$
D	T	for	$q^{\mathsf{prop}} = q^{\mathsf{prop}}$
D	F	for	$\lambda X_{\mathsf{prop}}.T = \lambda X_{\mathsf{prop}}.X_{\mathsf{prop}}$
D	Π^α	for	$q^{\alpha\to\mathsf{prop}}\ (\lambda X_\alpha.T)$
N	$\forall X_\alpha.\mathbf{A}$	for	$\Pi^\alpha\ (\lambda X_\alpha.\mathbf{A})$
D	$\wedge$	for	$\lambda X_{\mathsf{prop}}.\lambda Y_{\mathsf{prop}}.(\lambda G_{\mathsf{prop}\to\mathsf{prop}\to\mathsf{prop}}.GTT = \lambda G_{\mathsf{prop}\to\mathsf{prop}\to\mathsf{prop}}.GXY)$
N	$\mathbf{A} \wedge \mathbf{B}$	for	$\wedge\ (\mathbf{A}_{\mathsf{prop}})\ (\mathbf{B}_{\mathsf{prop}})$
D	$\Rightarrow$	for	$\lambda X_{\mathsf{prop}}.\lambda Y_{\mathsf{prop}}.(X = X \wedge Y)$
N	$\mathbf{A} \Rightarrow \mathbf{B}$	for	$\Rightarrow\ (\mathbf{A}_{\mathsf{prop}})\ (\mathbf{B}_{\mathsf{prop}})$
D	$\neg$	for	$q^{\mathsf{prop}}\ F$
D	$\vee$	for	$\lambda X_{\mathsf{prop}}.\lambda Y_{\mathsf{prop}}.\neg(\neg X \wedge \neg Y)$
N	$\mathbf{A} \vee \mathbf{B}$	for	$\vee\ (\mathbf{A}_{\mathsf{prop}})\ (\mathbf{B}_{\mathsf{prop}})$
D	$\exists X_\alpha.\mathbf{A}_{\mathsf{prop}}$	for	$\neg(\forall X_\alpha.\neg\mathbf{A})$
N	$\mathbf{A}_\alpha \neq \mathbf{B}_\alpha$	for	$\neg q^\alpha\ (\mathbf{A}_\alpha)\ (\mathbf{B}_\alpha)$

▷ yield the intuitive meanings for connectives and quantifiers.

In a way, this development of higher-order logic is more foundational, especially in the context of Henkin semantics. There, **??** does not hold (see [And72] for details). Indeed the proof of **??** needs the existence of singletons, which can be shown to be equivalent to the existence of the identity relation. In other words, Leibniz equality only denotes the equality relation, if we have an equality relation in the models. However, the only way of enforcing this (remember that Henkin models only guarantee functions that can be explicitly written down as λ-terms) is to add a logical constant for equality to the signature.

We have managed to deal with the determiners *every* and *some* in a compositional fashion, using the familiar first-order quantifiers. However, most natural language determiners cannot be treated so straightforwardly. Consider the determiner *most*, as in:

1. *Most boys run.*

There is clearly no simple way to translate this using ∀ or ∃ in any way familiar from first-order logic. As we have no translation at hand, then, let us consider what the truth conditions of this sentence are.

Generalized Quantifiers

▷ **Problem:** What about *Most boys run.*: linguistically *most* behaves exactly like *every* or *some*.

▷ **Idea:** *Most boys run* is true just in case the number of boys who run is greater than the number of boys who do not run.

$$\#(\mathcal{I}_\varphi(\mathsf{boy})\cap\mathcal{I}_\varphi(\mathsf{runs})) > \#(\mathcal{I}_\varphi(\mathsf{boy})\backslash\mathcal{I}_\varphi(\mathsf{runs}))$$

▷ **Definition 10.3.7.** $\#(A){>}\#(B)$, iff there is no surjective function from B to A, so we can define

$$most'{:=}(\lambda AB.\neg(\exists F.\forall X.A(X)\wedge\neg B(X)\Rightarrow(\exists.A(Y)\wedge B(Y)\wedge X=F(Y))))$$

The NP *most boys* thus must denote something which, combined with the denotation of a VP, gives this statement. In other words, it is a function from sets (or, equivalently, from functions in $\mathcal{D}_{(\iota\rightarrow\mathrm{prop})}$) to truth values which gives true just in case the argument stands in the relevant relation to the denotation of *boy*. This function is itself a characteristic function of a set of sets, namely:

$$\{X|\#(\mathcal{I}_\varphi(\mathrm{boy}),X){>}\#(\mathcal{I}_\varphi(\mathrm{boy})\backslash X)\}$$

Note that this is just the same kind of object (a set of sets) as we postulated above for the denotation of *every boy*.

Now we want to go a step further, and determine the contribution of the determiner *most* itself. *most* must denote a function which combines with a CNP denotation (i.e. a set of individuals or, equivalently, its characteristic function) to return a set of sets: just those sets which stand in the appropriate relation to the argument.

The function $most'$ is the characteristic function of a set of pairs:

$$\{\langle X,Y\rangle|\#(X\cap Y){>}\#(X\backslash Y)\}$$

Conclusion: *most* denotes a relation between sets, just as *every* and *some* do. In fact, all natural language determiners have such a denotation. (The treatment of the definite article along these lines raises some issues to which we will return.)

Back to *every* and *some* (set characterization)

▷ We can now give an explicit set characterization of *every* and *some*:

1. *every* denotes $\{\langle X,Y\rangle|X\subseteq Y\}$
2. *some* denotes $\{\langle X,Y\rangle|X\cap Y\neq\emptyset\}$

▷ The denotations can be given in equivalent function terms, as demonstrated above with the denotation of *most*.

10.3.2 Model Generation with Definite Descriptions

Semantics of Definite Descriptions

▷ **Problem:** We need a semantics for the determiner *the*, as in *the boy runs*

▷ **Idea (Type):** *the boy* behaves like a proper name (e.g. *Peter*), i.e. has type ι. Applying *the* to a noun (type $\iota \to \text{prop}$) yields ι. So *the* has type $\alpha \to \text{prop} \to \alpha$, i.e. it takes a set as argument.

▷ **Idea (Semantics):** *the* has the fixed semantics that this function returns the single member of its argument if the argument is a singleton, and is otherwise undefined. (new logical constant)

▷ **Definition 10.3.8.** We introduce a new logical constant ι. $\mathcal{I}(\iota)$ is the function $f{\in}\mathcal{D}_{(\alpha\to\text{prop}\to\alpha)}$, such that $f(s) = a$, iff $s{\in}\mathcal{D}_{(\alpha\to\text{prop})}$ is the singleton $\{a\}$, and is otherwise undefined. (remember that we can interpret predicates as sets)

▷ **Axioms for ι:**

$$\forall X_\alpha . X = \iota = X$$
$$\forall P, Q . Q(\iota\ P) \wedge (\forall X, Y . P(X) \wedge P(Y) \Rightarrow X = Y) \Rightarrow (\forall . P(Z) \Rightarrow Q(Z))$$

Note: The first axiom is an equational characterization of ι. It uses the fact that the singleton with member X can be written as $= X$ (or $\lambda Y . = XY$, which is $=_\eta$-equivalent). The second axiom says that if we have $Q\ \iota\ P$ and P is a singleton (i.e. all $X, Y{\in}P$ are identical), then Q holds on any member of P. Surprisingly, these two axioms are equivalent in $\text{HOL}^{\to}$.

More Operators and Axioms for $\text{HOL}^{\to}$

▷ **Definition 10.3.9.** The unary conditional $\mathbf{w}^\alpha{\in}\Sigma_{\text{prop}\to\alpha\to\alpha}$
$\mathbf{w}\ (\mathbf{A}_{\text{prop}})\mathbf{B}_\alpha$ means: "If $\mathbf{A}$, then $\mathbf{B}$".

▷ **Definition 10.3.10.** The binary conditional $\mathbf{if}^\alpha{\in}\Sigma_{\text{prop}\to\alpha\to\alpha\to\alpha}$
$\mathbf{if}\ (\mathbf{A}_{\text{prop}})\ (\mathbf{B}_\alpha)\ (\mathbf{C}_\alpha)$ means: "if $\mathbf{A}$, then $\mathbf{B}$ else $\mathbf{C}$".

▷ **Definition 10.3.11.** The description operator $\iota^\alpha{\in}\Sigma_{\alpha\to\text{prop}\to\alpha}$
if $\mathbf{P}$ is a singleton set, then $\iota\ (\mathbf{P}_{\alpha\to\text{prop}})$ is the (unique) element in $\mathbf{P}$.

▷ **Definition 10.3.12.** The choice operator $\gamma^\alpha{\in}\Sigma_{\alpha\to\text{prop}\to\alpha}$
if $\mathbf{P}$ is non-empty, then $\gamma\ (\mathbf{P}_{\alpha\to\text{prop}})$ is an arbitrary element from $\mathbf{P}$.

▷ **Definition 10.3.13 (Axioms for these Operators).**

- ▷ unary conditional: $\forall \varphi_{\text{prop}} . \forall X_\alpha . \varphi \Rightarrow \mathbf{w}\ \varphi X = X$
- ▷ binary conditional: $\forall \varphi_{\text{prop}} . \forall X_\alpha, Y_\alpha, Z_\alpha . (\varphi \Rightarrow \mathbf{if}\ \varphi\ X\ Y = X) \wedge (\neg\varphi \Rightarrow \mathbf{if}\ \varphi\ Z\ X = X)$
- ▷ description operator $\forall P_{\alpha\to\text{prop}} . (\exists^1 X_\alpha . PX) \Rightarrow (\forall Y_\alpha . PY \Rightarrow \iota\ P = Y)$
- ▷ choice operator $\forall P_{\alpha\to\text{prop}} . (\exists X_\alpha . PX) \Rightarrow (\forall Y_\alpha . PY \Rightarrow \gamma\ P = Y)$

▷ **Idea:** These operators ensure a much larger supply of functions in Henkin models.

More on the Description Operator

▷ ι is a weak form of the choice operator. (only works on singletons)

▷ Alternative Axiom of Descriptions: $\forall X_\alpha.\iota^\alpha \; = \; X = X$.

- ▷ use that $\mathcal{I}_{[\mathsf{a}/X]}(= \; X) = \{\mathsf{a}\}$
- ▷ we only need this for base types $\neq$ prop
- ▷ Define $\iota^{\mathsf{prop}} := = \; (\lambda X_{\mathsf{prop}}.X)$ or $\iota^{\mathsf{prop}} := (\lambda G_{\mathsf{prop}\to\mathsf{prop}}.G\; T)$ or $\iota^{\mathsf{prop}} := = = \; T$
- ▷ $\iota^{(\alpha\to\beta)} := (\lambda H_{\alpha\to\beta\to\mathsf{prop}} X_\alpha.\iota^\beta \; (\lambda Z_\beta.(\exists F_{\alpha\to\beta}.H\; F \wedge F\; X = Z)))$

To obtain a model generation calculus for HOL_{NQ} with descriptions, we could in principle add one of these axioms to the world knowledge, and work with that. It is better to have a dedicated inference rule, which we present here.

A Model Generation Rule for ι

▷ **Definition 10.3.14.**

$$\frac{\begin{array}{c} P(c)^{\mathsf{T}} \\ Q(\iota\; P)^\alpha \end{array} \quad \mathcal{H} = \{c, a_1, \ldots, a_n\}}{\begin{array}{c} Q(c)^\alpha \\ (P(a_1) \Rightarrow c = a_1)^{\mathsf{T}} \\ \vdots \\ (P(a_n) \Rightarrow c = a_n)^{\mathsf{T}} \end{array}} \; RM\,\iota$$

▷ **Intuition:** If we have a member c of P and $Q(\iota\; P)$ is defined (it has truth value $\alpha \in \{\mathsf{T}, \mathsf{F}\}$), then P must be a singleton (i.e. all other members X of P are identical to c) and Q must hold on c. So the rule $RM\,\iota$ forces it to be by making all other members of P equal to c.

Mary owned a lousy computer. The hard drive crashed.

$$\begin{array}{c}
(\forall X.\mathsf{computer}(X) \Rightarrow (\exists Y.\mathsf{harddrive}(Y) \land \mathsf{partof}(Y,X)))^{\mathsf{T}} \\
\boxed{(\exists X.\mathsf{computer}(X) \land \mathsf{lousy}(X) \land \mathsf{own}(\mathsf{mary},X))^{\mathsf{T}}} \\
\mathsf{computer}(c)^{\mathsf{T}} \\
\mathsf{lousy}(c)^{\mathsf{T}} \\
\mathsf{own}(\mathsf{mary},c)^{\mathsf{T}} \\
\begin{array}{c|c}
\begin{array}{c}
\mathsf{harddrive}(c)^{\mathsf{T}} \\
\mathsf{partof}(c,c)^{\mathsf{T}} \\
\vdots \\
\bot
\end{array}
&
\begin{array}{c}
\mathsf{harddrive}(d)^{\mathsf{T}} \\
\mathsf{partof}(d,c)^{\mathsf{T}} \\
\boxed{\mathsf{crashes}(\iota\ \mathsf{harddrive})^{\mathsf{T}}} \\
\mathsf{crashes}(d)^{\mathsf{T}} \\
(\mathsf{harddrive}(\mathsf{mary}) \Rightarrow \mathsf{mary} = d)^{\mathsf{T}} \\
(\mathsf{harddrive}(c) \Rightarrow c = d)^{\mathsf{T}}
\end{array}
\end{array}
\end{array}$$

Definition 10.3.15. In this example, we have a case of what is called a bridging reference, following H. Clark (1977): intuitively, we build an inferential bridge from the computer whose existence is asserted in the first sentence to the hard drive invoked in the second.

By incorporating world knowledge into the tableau, we are able to model this kind of inference, and provide the antecedent needed for interpreting the definite.

Now let us use the *RM* ι rule for interpreting *The dog barks* in a situation where there are two dogs: Fido and Chester. Intuitively, this should lead to a closed tableau, since the uniqueness presupposition is violated. Applying the rules, we get the following tableau.

Another Example *The dog barks*

- ▷ In a situation, where there are two dogs: Fido and Chester

$$\begin{array}{c}
\mathsf{dog}(\mathsf{fido})^{\mathsf{T}} \\
\mathsf{dog}(\mathsf{chester})^{\mathsf{T}} \\
\boxed{\mathsf{bark}(\iota\ \mathsf{dog})} \\
\mathsf{bark}(\mathsf{fido})^{\mathsf{T}} \\
(\mathsf{dog}(\mathsf{chester}) \Rightarrow \mathsf{chester} = \mathsf{fido})^{\mathsf{T}} \\
\begin{array}{c|c}
\begin{array}{c}\mathsf{dog}(\mathsf{chester})^{\mathsf{F}} \\ \bot\end{array} & \begin{array}{c}\mathsf{chester} = \mathsf{fido}^{\mathsf{T}} \\ \ \end{array}
\end{array}
\end{array} \tag{10.1}$$

- ▷ Note that none of our rules allows us to close the right branch, since we do not know that Fido and Chester are distinct. Indeed, they could be the same dog (with two different names). But we can eliminate this possibility by adopting a new assumption.

10.3.3 Model Generation with Unique Name Assumptions

Normally (i.e. in natural languages) we have the default assumption that names are unique. In principle, we could do this by adding axioms of the form $n = m^{\mathsf{F}}$ to the world knowledge for all pairs of names n and m. Of course the cognitive plausibility of this approach is very questionable. As a remedy, we can build a Unique-Name-Assumption (UNA) into the calculus itself.

Model Generation with Unique Name Assumption (UNA)

- ▷ **Problem:** Names are unique usually in natural language
- ▷ **Definition 10.3.16.** The unique name assumption (UNA) makes the assumption that names are unique (in the respective context)
- ▷ **Idea:** Add background knowledge of the form $n=m^{\mathsf{F}}$ (n and m names)
- ▷ **Better Idea:** Build UNA into the calculus: partition the Herbrand base $\mathcal{H}=\mathcal{U}\cup\mathcal{W}$ into subsets $\mathcal{U}$ for constants with a UNA, and $\mathcal{W}$ without. (treat them differently)
- ▷ **Definition 10.3.17 (Model Generation with UNA).** We add the following two rules to the $\mathcal{RM}$ calculus to deal with the unique name assumption.

$$\frac{\begin{array}{c} a=b^{\mathsf{T}} \\ \mathbf{A}^{\alpha} \end{array} \quad a\in\mathcal{W} \quad b\in\mathcal{H}}{([b/a](\mathbf{A}))^{\alpha}}\mathcal{RM}\text{ subst} \qquad \frac{a=b^{\mathsf{T}} \quad a,b\in\mathcal{U}}{\bot}\mathcal{RM}\text{ una}$$

In effect we make the equality replacement rule directional; it only allows the substitution for a constant without the unique name assumption. Finally, $\mathcal{RM}$ una mechanizes the unique name assumption by allowing a branch to close if two different constants with unique names are claimed to be equal. All the other rules in our model generation calculus stay the same. Note that with $\mathcal{RM}$ una, we can close the right branch of tableau (10.1), in accord with our intuition about the discourse.

Solving a Crime with Unique Names

- ▷ **Example 10.3.18.** Tony has observed (at most) two people. Tony observed a murderer that had black hair. It turns out that Bill and Bob were the two people Tony observed. Bill is blond, and Bob has black hair. (Who was the murderer.)
Let $\mathcal{U}=\{\text{Bill},\text{Bob}\}$ and $\mathcal{W}=\{\text{murderer}\}$:

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\forall z.\text{observes}(\text{Tony},z)\Rightarrow(z=\text{Bill}\vee z=\text{Bob}))^{\mathsf{T}}} \\
\multicolumn{2}{c}{\text{observes}(\text{Tony},\text{Bill})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\text{observes}(\text{Tony},\text{Bob})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\text{observes}(\text{Tony},\text{murderer})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\text{black\_hair}(\text{murderer})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\neg\text{black\_hair}(\text{Bill})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\text{black\_hair}(\text{Bill})^{\mathsf{F}}} \\
\multicolumn{2}{c}{\text{black\_hair}(\text{Bob})^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\text{observes}(\text{Tony},\text{murderer})\Rightarrow(\text{murderer}=\text{Bill}\vee\text{murderer}=\text{Bob}))^{\mathsf{T}}} \\
\multicolumn{2}{c}{(\text{murderer}=\text{Bill}\vee\text{murderer}=\text{Bob})^{\mathsf{T}}} \\
\text{murderer}=\text{Bill}^{\mathsf{T}} & \text{murderer}=\text{Bob}^{\mathsf{T}} \\
\text{black\_hair}(\text{Bill})^{\mathsf{T}} & \\
\bot &
\end{array}$$

Rabbits [Gardent & Konrad '99]

- ▷ Interpret "the" as $\lambda PQ.Q\iota\ P \wedge \mathsf{uniq}(P)$
  where uniq:=$(\lambda P.(\exists X.P(X) \wedge (\forall Y.P(Y) \Rightarrow X = Y)))$
  and $\mathbb{\forall}$:=$(\lambda PQ.(\forall X.P(X) \Rightarrow Q(X)))$.
- ▷ "the rabbit is cute", has logical form $\mathsf{uniq}(\mathsf{rabbit}) \wedge (\mathsf{rabbit} \subseteq \mathsf{cute})$.
- ▷ $\mathcal{RM}$ generates $\{\ldots, \mathsf{rabbit}(c), \mathsf{cute}(c)\}$ in situations with at most 1 rabbit.
  (special $\mathcal{RM}\ \exists$ rule yields identification and accommodation (c^{new}))
- + At last an approach that takes world knowledge into account!
- – tractable only for toy discourses/ontologies
  The world cup final was watched on TV by 7 million people.
  A rabbit is in the garden.
  $\forall X.\mathsf{human}(x)\exists Y.\mathsf{human}(X) \wedge \mathsf{father}(X,Y) \qquad \forall X,Y.\mathsf{father}(X,Y) \Rightarrow X \neq Y$

More than one Rabbit

- ▷ **Problem:** What about two rabbits?
  Bugs and Bunny are rabbits. Bugs is in the hat. Jon removes the rabbit from the hat.
- ▷ **Idea: Uniqueness under Scope [Gardent & Konrad '99]:**
  - ▹ refine *the* to $\lambda PRQ.\mathsf{uniq}(P \cap R \wedge \mathbb{\forall}(P \cap R, Q))$
    where R is an "identifying property" (identified from the context and passed as an arbument to *the*)
  - ▹ here R is "being in the hat" (by world knowledge about removing)
  - ▹ makes Bugs unique (in $P \cap R$) and the discourse acceptable.
- ▷ **Idea:** [Hobbs & Stickel&...]:
  - ▹ use generic relation rel for "relatedness to context" for P^2.

  ?? Is there a general theory of relatedness?

10.4 Davidsonian Semantics: Treating Verb Modifiers

Event semantics: Davidsonian Systems

- ▷ **Problem:** How to deal with argument structure of (action verbs) and their modifiers
  - ▹ *John killed a cat with a hammer.*

▷ **Idea:** Just add an argument to kills for express the means

▷ **Problem:** But there may be more modifiers

1. *Peter killed the cat in the bathroom with a hammer.*
2. *Peter killed the cat in the bathroom with a hammer at midnight.*

So we would need a lot of different predicates for the verb *killed*. (impractical)

▷ **Definition 10.4.1.** In event semantics we extend the argument structure of (action) verbs contains a 'hidden' argument, the event argument, then treat modifiers as predicates (often called roles) over events [Dav67a].

▷ **Example 10.4.2.**

1. $\exists e.\exists x,y.\mathsf{bathroom}(x) \wedge \mathsf{hammer}(y) \wedge \mathsf{kill}(e,\mathsf{peter},\iota\,\mathsf{cat}) \wedge \mathsf{in}(e,x) \wedge \mathsf{with}(e,y)$
2. $\exists e.\exists x,y.\mathsf{bathroom}(x) \wedge \mathsf{hammer}(y) \wedge \mathsf{kill}(e,\mathsf{peter},\iota\,\mathsf{cat}) \wedge \mathsf{in}(e,x) \wedge \mathsf{with}(e,y) \wedge \mathsf{at}(e,24:00)$

Event semantics: Neo-Davidsonian Systems

▷ **Idea:** Take apart the Davidsonian predicates even further, add event participants via thematic roles (from [Par90]).

▷ **Definition 10.4.3.** Neo-Davisonian semantics extends event semantics by adding two standardized roles: the agent $\mathsf{ag}(e,s)$ and the patient $\mathsf{pat}(e,o)$ for the subject s and direct object d of the event e.

▷ **Example 10.4.4.** Translate *John killed a cat with a hammer.* as $\exists e.\exists x.\mathsf{hammer}(x) \wedge \mathsf{killing}(e) \wedge \mathsf{ag}(e,\mathsf{peter}) \wedge \mathsf{pat}(e,\iota\,\mathsf{cat}) \wedge \mathsf{with}(e,x)$

▷ **Further Elaboration:** Events can be broken down into sub-events and modifiers can predicate over sub-events.

▷ **Example 10.4.5.** The "process" of climbing Mt. Everest starts with the "event" of (optimistically) leaving the base camp and culminates with the "achievement" of reaching the summit (being completely exhausted).

▷ **Note:** This system can get by without functions, and only needs unary and binary predicates. (well-suited for model generation)

Event types and properties of events

▷ **Example 10.4.6 (Problem).** Some (temporal) modifiers are incompatible with some events, e.g. in English progressive:

1. *He is eating a sandwich* and *He is pushing the cart.*, but not
2. * *He is being tall.* or * *He is finding a coin.*

▷ **Definition 10.4.7 (Types of Events).** There are different types of events that go with different temporal modifiers. [Ven57] distinguishes

1. states: e.g. *know the answer*, *stand in the corner*
2. processes: e.g. *run*, *eat*, *eat apples*, *eat soup*
3. accomplishments: e.g. *run a mile*, *eat an apple*, and
4. achievements: e.g. *reach the summit*

▷ **Observations:**

1. processes and accomplishments appear in the progressive (1),
2. states and achievements do not (2).

▷ **Definition 10.4.8.** The in test

1. states and activities, but not accomplishments and achievements are compatible with *for*-adverbials
2. whereas the opposite holds for in-adverbials (5).

▷ **Example 10.4.9.**

1. *run a mile in an hour* vs. * *run a mile for an hour*, but
2. * *reach the summit for an hour* vs *reach the summit in an hour*

Chapter 11

Davidsonian Semantics: Treating Verb Modifiers

Event semantics: Davidsonian Systems

▷ **Problem:** How to deal with argument structure of (action verbs) and their modifiers

▷ *John killed a cat with a hammer.*

▷ **Idea:** Just add an argument to kills for express the means

▷ **Problem:** But there may be more modifiers

1. *Peter killed the cat in the bathroom with a hammer.*
2. *Peter killed the cat in the bathroom with a hammer at midnight.*

So we would need a lot of different predicates for the verb *killed*. (impractical)

▷ **Definition 11.0.1.** In event semantics we extend the argument structure of (action) verbs contains a 'hidden' argument, the event argument, then treat modifiers as predicates (often called roles) over events [Dav67a].

▷ **Example 11.0.2.**

1. $\exists e.\exists x, y.\mathsf{bathroom}(x) \wedge \mathsf{hammer}(y) \wedge \mathsf{kill}(e, \mathsf{peter}, \iota\ \mathsf{cat}) \wedge \mathsf{in}(e, x) \wedge \mathsf{with}(e, y)$
2. $\exists e.\exists x, y.\mathsf{bathroom}(x) \wedge \mathsf{hammer}(y) \wedge \mathsf{kill}(e, \mathsf{peter}, \iota\ \mathsf{cat}) \wedge \mathsf{in}(e, x) \wedge \mathsf{with}(e, y) \wedge \mathsf{at}(e, 24:00)$

Event semantics: Neo-Davidsonian Systems

▷ **Idea:** Take apart the Davidsonian predicates even further, add event participants via thematic roles (from [Par90]).

▷ **Definition 11.0.3.** Neo-Davisonian semantics extends event semantics by adding two standardized roles: the agent $\mathsf{ag}(e, s)$ and the patient $\mathsf{pat}(e, o)$ for the subject s and direct object d of the event e.

▷ **Example 11.0.4.** Translate *John killed a cat with a hammer.* as $\exists e.\exists x.\mathsf{hammer}(x) \wedge \mathsf{killing}(e) \wedge \mathsf{ag}(e, \mathsf{peter}) \wedge \mathsf{pat}(e, \iota\ \mathsf{cat}) \wedge \mathsf{with}(e, x)$

▷ **Further Elaboration:** Events can be broken down into sub-events and modifiers can predicate over sub-events.

▷ **Example 11.0.5.** The "process" of climbing Mt. Everest starts with the "event" of (optimistically) leaving the base camp and culminates with the "achievement" of reaching the summit (being completely exhausted).

▷ **Note:** This system can get by without functions, and only needs unary and binary predicates. (well-suited for model generation)

Event types and properties of events

▷ **Example 11.0.6 (Problem).** Some (temporal) modifiers are incompatible with some events, e.g. in English progressive:

1. *He is eating a sandwich* and *He is pushing the cart.*, but not
2. * *He is being tall.* or * *He is finding a coin.*

▷ **Definition 11.0.7 (Types of Events).** There are different types of events that go with different temporal modifiers. [Ven57] distinguishes

1. states: e.g. *know the answer*, *stand in the corner*
2. processes: e.g. *run*, *eat*, *eat apples*, *eat soup*
3. accomplishments: e.g. *run a mile*, *eat an apple*, and
4. achievements: e.g. *reach the summit*

▷ **Observations:**

1. processes and accomplishments appear in the progressive (1),
2. states and achievements do not (2).

▷ **Definition 11.0.8.** The in test

1. states and activities, but not accomplishments and achievements are compatible with *for*-adverbials
2. whereas the opposite holds for in-adverbials (5).

▷ **Example 11.0.9.**

1. *run a mile in an hour* vs. * *run a mile for an hour*, but
2. * *reach the summit for an hour* vs *reach the summit in an hour*

Part II

Topics in Semantics

Chapter 12

Dynamic Approaches to NL Semantics

In this chapter we tackle another level of language, the discourse level, where we look especially at the role of cross-sentential anaphora. This is an aspect of natural language that cannot (compositionally) be modeled in first-order logic, due to the strict scoping behavior of quantifiers. This has led to the developments of dynamic variants of first-order logic: the "file change semantics" [Hei82] by Irene Heim and (independently) "discourse representation theory" (DRT [Kam81]) by Hans Kamp, which solve the problem by re-interpreting indefinites to introduce representational objects – called discourse referents in DRT – that are not bound variables and can therefore have a different scoping behavior. These approaches have been very influential in the representation of discourse – i.e. multi-sentence – phenomena.

In this chapter, we will introduce dynamic logics taking DRT as a starting point since it was adopted more widely than file change semantics and the later "dynamic predicate logics" (DPL [GS91]). section 12.1 gives an introduction to dynamic language phenomena and how they can be modeled in DRT. section 13.4 relates the linguistically motivated logics to modal logics used for modeling imperative programs and draws conclusions about the role of language in cognition. **??** extends our primary inference system – model generation – to DRT and relates the concept of discourse referents to Skolem constants. Dynamic model generation also establishes a natural system of "direct deduction" for dynamic semantics. Finally, Appendix E discusses how dynamic approaches to NL semantics can be combined with ideas Montague Semantics to arrive at a fully compositional approach to discourse semantics.

12.1 Discourse Representation Theory

In this section we introduce Discourse Representation Theory as the most influential framework for aproaching dynamic phenomena in natural language. We will only cover the basic ideas here and leave the coverage of larger fragments of natural language to [KR93].
Let us look at some data about effects in natural languages that we cannot really explain with our treatment of indefinite descriptions in Fragment 4 (see **??**).

Anaphora and Indefinites revisited (Data)

- ▷ **Observation:** We have concentrated on single sentences so far; let's do better.
- ▷ **Definition 12.1.1.** A discourse is a a unit of natural language longer than a single sentence.

▷ **New Data:** discourses interact with anaphora.:

- ▹ *Peter[1] is sleeping. He$_1$ is snoring.* (normal anaphoric reference)
- ▹ *A man[1] is sleeping. He$_1$ is snoring.* (Scope of existential?)
- ▹ *Peter has a car[1]. It$_1$ is parked outside.* (even if this worked)
- ▹ * *Peter has no car[1]. It$_1$ is parked outside.* (what about negation?)
- ▹ *There is a book[1] that Peter does not own. It$_1$ is a novel.* (OK)
- ▹ * *Peter does not own every book[1]. It$_1$ is a novel.* (equivalent in PL^1)
- ▹ *If a farmer[1] owns a donkey$_2$, he$_1$ beats it$_2$.* (even inside sentences)

In the first example, we can pick up the subject *Peter* of the first sentence with the anaphoric reference *He* in the second. We gloss the intended anaphoric reference with the labels in upper and lower indices. And indeed, we can resolve the anaphoric reference in the semantic representation by translating *He* to (the translation of) *Peter*. Alternatively we can follow the lead of fragment 2 (see **??**) and introduce variables for anaphora and adding a conjunct that equates the respective variable with the translation of *Peter*. This is the general idea of anaphor resolution we will adopt in this section.

Dynamic Effects in Natural Language

▷ **Problem:** E.g. Quantifier Scope

- ▹ * *A man sleeps. He snores.*
- ▹ $(\exists X.\mathsf{man}(X) \wedge \mathsf{sleeps}(X)) \wedge \mathsf{snores}(X)$
- ▹ X is bound in the first conjunct, and free in the second.

▷ **Problem:** donkey sentence: *If a farmer owns a donkey, he beats it.*
$\forall X, Y.\mathsf{farmer}(X) \wedge \mathsf{donkey}(Y) \wedge \mathsf{own}(X, Y) \Rightarrow \mathsf{beat}(X, Y)$

▷ **Ideas:**

- ▹ Composition of sentences by conjunction inside the scope of existential quantifiers (non-compositional, . . .)
- ▹ Extend the scope of quantifiers dynamically (DPL)
- ▹ Replace existential quantifiers by something else (DRT)

Intuitively, the second example should work exactly the same – it should not matter, whether the subject NP is given as a proper name or an indefinite description. The problem with the indefinite descriptions is that they are translated into existential quantifiers and we cannot refer to the bound variables see below. Note that this is not a failure of our envisioned treatment of anaphora, but of our treatment of indefinite descriptions; they just do not generate the objects that can be referred back to by anaphoric references (we will call them discourse referents). We will speak of the "anaphoric potential" for this the set of referents that can be anaphorically referred to.

The second pair of examples is peculiar in the sense that if we had a solution for the indefinite description in *Peter has a car*, we would need a solution that accounts for the fact that even though *Peter has a car* puts a car referent into the anaphoric potential *Peter has no car* – which

we analyze compositionally as *It is not the case that Peter has a car* does not. The interesting effect is that the negation closes the anaphoric potential and excludes the car referent that *Peter has a car* introduced.
The third pair of sentences shows that we need more than PL^1 to represent the meaning of quantification in natural language while the sentence *There is a book that peter does not own.* induces a book referent in the anaphoric potential, but the sentence *Peter does not own every book* does not, even though their translations $\exists x.\mathrm{book}(x) \wedge \neg\mathrm{own}(\mathrm{peter}, x)$ and $\neg(\forall x.\mathrm{book}(x) \Rightarrow \mathrm{own}(\mathrm{peter}, x))$ are logically equivalent.
The last sentence is the famous donkey sentence that shows that the dynamic phenomena we have seen above are not limited to inter-sentential anaphora.
The central idea of Discourse Representation Theory (DRT), is to eschew the first-order quantification and the bound variables it induces altogether and introduce a new representational device: discourse referents, and manage their visibility (called accessibility in DRT) explicitly.
We will introduce the traditional, visual "box notation" by example now before we turn to a systematic definition based on a symbolic notation later.

Discourse Representation Theory (DRT)

▷ **Definition 12.1.2.** Discourse Representation Theory (DRT) is a logical system, which uses discourse referents to model quantification and pronouns. DRT formulae are called discourse representation structure (DRS); these introduce a set of discourse referents and specify their meaning by conditions:
- ▷ atomic propositions,
- ▷ dynamic negations $\neg D$,
- ▷ dynamic implications $D \Rrightarrow E$, and
- ▷ dynamic disjunctions $D \vee\!\!\!\vee E$.

▷ Discourse referents e.g. in *A student owns a book.*
- ▷ are kept in a dynamic context (⇝ accessibility)
- ▷ are declared e.g. in indefinite nominals
- ▷ specified in conditions via predicates

$$\begin{array}{|l|}\hline X,Y\\\hline \mathrm{student}(X)\\ \mathrm{book}(Y)\\ \mathrm{own}(X,Y)\\\hline\end{array}$$

▷ Discourse representation structures (DRS)
A student owns a book. He reads it. *If a farmer owns a donkey, he beats it.*

$$\begin{array}{|l|}\hline X,Y,R,S\\\hline \mathrm{student}(X)\\ \mathrm{book}(Y)\\ \mathrm{own}(X,Y)\\ \mathrm{read}(R,S)\\ X=R\\ Y=S\\\hline\end{array}$$

$$\begin{array}{|l|}\hline \\\hline \begin{array}{|l|}\hline X,Y\\\hline \mathrm{farmer}(X)\\ \mathrm{donkey}(Y)\\ \mathrm{own}(X,Y)\\\hline\end{array} \Rrightarrow \begin{array}{|l|}\hline \\\hline \mathrm{beat}(X,Y)\\\hline\end{array}\\\hline\end{array}$$

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 234 2024-01-20 SOME RIGHTS RESERVED

These examples already show that there are three kinds of objects in DRT: The meaning of sentences is given as DRSes, which are denoted as "file cards" that list the discourse referents (the participants in the situation described in the DRS) at the top of the "card" and state a couple of conditions on the discourse referents. The conditions can contain DRSes themselves, e.g. in conditional conditions.
With this representational infrastructure in place we can now look at how we can construct discourse DRSes i.e. DRSes for whole discourses. The sentence composition problem was – after all

– the problem that led to the development of DRT since we could not compositionally solve it in first-order logic.

Discourse DRS Construction

▷ **Problem:** How do we construct DRSes for multi-sentence discourses?

▷ **Solution:** We construct sentence DRSes individually and merge them (DRSes and conditions separately)

▷ **Example 12.1.3.** A three-sentence discourse. (not quite Shakespeare)

Mary sees John.	*John kills a cat.*	*Mary calls a cop.*	merge
$\begin{array}{\|l\|}\hline \\ \hline \text{see}(\text{mary},\text{john}) \\ \hline\end{array}$	$\begin{array}{\|l\|}\hline U \\ \hline \text{cat}(U) \\ \text{kills}(\text{john},U) \\ \hline\end{array}$	$\begin{array}{\|l\|}\hline V \\ \hline \text{policeman}(V) \\ \text{calls}(\text{mary},V) \\ \hline\end{array}$	$\begin{array}{\|l\|}\hline U,V \\ \hline \text{see}(\text{mary},\text{john}) \\ \text{cat}(U) \\ \text{kills}(\text{john},U) \\ \text{policeman}(V) \\ \text{calls}(\text{mary},V) \\ \hline\end{array}$

▷ Sentence composition via the DRT Merge Operator $\otimes$. (acts on DRSes)

Note that – in contrast to the “smuggling-in”-type solutions we would have to dream up for first-order logic – sentence composition in DRT is compositional: We construct sentence DRSes[1] and merge them. We can even introduce a “logic operator” for this: the merge operator $\otimes$, which can be thought of as the “full stop” punctuation operator.
Now we can have a look at anaphor resolution in DRT. This is usually considered as a separate process – part of semantic-pragmatic analysis.

Anaphor Resolution in DRT

▷ **Problem:** How do we resolve anaphora in DRT?

▷ **Solution:** Two phases

- ▷ translate pronouns into discourse referents (semantics construction)
- ▷ identify (equate) coreferring discourse referents, (maybe) simplify (semantic/pragmatic analysis)

▷ **Example 12.1.4.** *A student owns a book. He reads it.*

A student^1 owns a book^2. He$_1$ reads it$_2$	resolution	simplify
$\begin{array}{\|l\|}\hline X,Y,R,S \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \text{read}(R,S) \\ \hline\end{array}$	$\begin{array}{\|l\|}\hline X,Y,R,S \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \text{read}(R,S) \\ X=R \\ Y=S \\ \hline\end{array}$	$\begin{array}{\|l\|}\hline X,Y \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \text{read}(X,Y) \\ \hline\end{array}$

[1]We will not go into the sentence semantics construction process here

We will sometime abbreviate the anaphor resolution process and directly use the simplified version of the DRSes for brevity.
Using these examples, we can now give a more systematic introduction of DRT using a more symbolic notation. Note that the grammar below over-generates, we still need to specify the visibility of discourse referents.

DRT (Syntax)

▷ **Definition 12.1.5.** Given a set $\mathcal{DR}$ of discourse referents, discourse representation structure (DRSes) are given by the following grammar:

conditions	$\mathcal{C}::=p(a_1,\ldots,a_n) \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \neg\mathcal{D} \mid \mathcal{D}_1 \vee \mathcal{D}_2 \mid \mathcal{D}_1 \Rrightarrow \mathcal{D}_2$
DRSes	$\mathcal{D}::=\delta U^1,\ldots,U^n.\mathcal{C} \mid \mathcal{D}_1 \otimes \mathcal{D}_2 \mid \mathcal{D}_1 ;; \mathcal{D}_2$

▷ $\otimes$ and $;;$ are for sentence composition ($\otimes$ from DRT, $;;$ from DPL)

▷ **Example 12.1.6.** $\delta U, V.\text{farmer}(U) \wedge \text{donkey}(V) \wedge \text{own}(U,V) \wedge \text{beat}(U,V)$

▷ **Definition 12.1.7.** The meaning of $\otimes$ and $;;$ is given operationally by $=_\tau$ Equality:

$$\begin{aligned}\delta\mathcal{X}.\mathcal{C}_1 \otimes \delta\mathcal{Y}.\mathcal{C}_2 &\to_\tau \delta\mathcal{X},\mathcal{Y}.\mathcal{C}_1 \wedge \mathcal{C}_2\\ \delta\mathcal{X}.\mathcal{C}_1 ;; \delta\mathcal{Y}.\mathcal{C}_2 &\to_\tau \delta\mathcal{X},\mathcal{Y}.\mathcal{C}_1 \wedge \mathcal{C}_2\end{aligned}$$

▷ Discourse referents used instead of bound variables.(specify scoping independently of logic)

▷ **Idea:** Semantics inherited from first-order logic by a translation mapping.

We can now define the notion of accessibility in DRT, which in turn determines the (predicted) dynamic potential of a DRS: A discourse referent has to be accessible to be picked up by an anaphoric reference.
We will follow the classical exposition and introduce accessibility as a derived concept induced by a non-structural notion of sub-DRS.

Sub DRSes and Accessibility

▷ **Problem:** How can we formally define accessibility. (to make predictions)

▷ **Idea:** Make use of the structural properties of DRT.

▷ **Definition 12.1.8.** A referent is accessible in all sub DRS of the declaring DRS.

- ▹ If $\mathcal{D} = \delta U^1,\ldots,U^n.\mathcal{C}$, then any sub DRS of $\mathcal{C}$ is a sub DRS of $\mathcal{D}$.
- ▹ If $\mathcal{D} = \mathcal{D}^1 \otimes \mathcal{D}^2$, then $\mathcal{D}^1$ is a sub DRS of $\mathcal{D}^2$ and vice versa.
- ▹ If $\mathcal{D} = \mathcal{D}^1 ;; \mathcal{D}^2$, then $\mathcal{D}^2$ is a sub DRS of $\mathcal{D}^1$.
- ▹ If $\mathcal{C}$ is of the form $\mathcal{C}^1 \wedge \mathcal{C}^2$, or $\neg\mathcal{D}$, or $\mathcal{D}^1 \vee \mathcal{D}^2$, or $\mathcal{D}^1 \Rrightarrow \mathcal{D}^2$, then any sub DRS of the $\mathcal{C}^i$, and the $\mathcal{D}^i$ is a sub DRS of $\mathcal{C}$.
- ▹ If $\mathcal{D} = \mathcal{D}^1 \Rrightarrow \mathcal{D}^2$, then $\mathcal{D}^2$ is a sub DRS of $\mathcal{D}^1$

- **Definition 12.1.9 (Dynamic Potential).** (which referents can be picked up?) A referent U is in the dynamic potential of a DRS $\mathcal{D}$, iff it is accessible in $\mathcal{D} \otimes \begin{array}{|c|}\hline \\ \hline p(U) \\ \hline \end{array}$
- **Definition 12.1.10.** We call a DRS static, iff the dynamic potential is empty, and dynamic, if it is not.

Michael Kohlhase: LBS 238 2024-01-20

Sub DRSes and Accessibility

- **Observation:** Accessibility gives DRSes the flavor of binding structures. (with non-standard scoping!)
- **Idea:** Apply the usual binding heuristics to DRT, e.g.
  - reject DRSes with unbound discourse referents.
- **Questions:** if view of discourse referents as "nonstandard bound variables"
  - what about renaming referents?

Michael Kohlhase: LBS 239 2024-01-20

The meaning of DRSes is (initially) given by a translation to PL¹. This is a convenient way to specify meaning, but as we will see, it has its costs, as we will see.

Translation from DRT to FOL

- **Definition 12.1.11.** For $=_\tau$-normal (fully merged) DRSes use the translation $\overline{\cdot}$:

$$\begin{array}{rcl}
\overline{\delta U^1,\ldots,U^n.\mathcal{C}} & = & \exists U^1,\ldots,U^n.\overline{\mathcal{C}} \\
\overline{\neg\mathcal{D}} & = & \neg\overline{\mathcal{D}} \\
\overline{\mathcal{D} \mathbb{V} \mathcal{E}} & = & \overline{\mathcal{D}} \vee \overline{\mathcal{E}} \\
\overline{\mathcal{D} \wedge \mathcal{E}} & = & \overline{\mathcal{D}} \wedge \overline{\mathcal{E}} \\
\overline{(\delta U^1,\ldots,U^n.\mathcal{C}_1) \Rrightarrow (\delta V^1,\ldots,V^n.\mathcal{C}_2)} & = & \forall U^1,\ldots,U^n.\overline{\mathcal{C}_1} \Rightarrow (\exists V^1,\ldots,V^n.\overline{\mathcal{C}_2})
\end{array}$$

- **Example 12.1.12.** $\overline{\begin{array}{|l|}\hline X, Y \\ \hline \mathsf{student}(X) \\ \mathsf{book}(Y) \\ \mathsf{own}(X,Y) \\ \hline \end{array}} = \exists X.\exists Y.\mathsf{student}(X) \wedge \mathsf{book}(Y) \wedge \mathsf{own}(X,Y)$.
- **Example 12.1.13.**

$$\overline{(\delta U,V.\mathsf{farmer}(U) \wedge \mathsf{donkey}(V) \wedge \mathsf{own}(U,V)) \Rrightarrow (\delta W.\mathsf{stick}(W) \wedge \mathsf{beatwith}(U,V,W))}$$
$$= \forall X,Y.\mathsf{farmer}(X) \wedge \mathsf{donkey}(X) \wedge \mathsf{own}(X,Y) \Rightarrow (\exists.\mathsf{stick}(Z) \wedge \mathsf{beatwith}(Z,X,Y))$$

- **Consequence:** Validity of DRSes can be checked by translation.
- **Question:** Why not use first-order logic directly?
- **Answer:** Only translate at the end of a discourse (translation closes all dynamic contexts: frequent re-translation).

We can now test DRT as a logical system on the data and see whether it makes the right predictions about the dynamic effects identified at the beginning of the section.

Properties of Dynamic Scope

- **Idea:** Test DRT on the data above for the dynamic phenomena
- **Example 12.1.14 (Negation Closes Dynamic Potential).**
  Peter has no^1 car. * *It$_1$ is parked outside.*

$$\begin{array}{|c|}\hline \\ \hline \neg \begin{array}{|l|}\hline U\\\hline \mathrm{acar}(U)\\ \mathrm{own}(\mathrm{peter},U)\\\hline\end{array}\\\hline\end{array} \otimes \begin{array}{|l|}\hline \\\hline \mathrm{parked}(U)\\\hline\end{array} \qquad \neg(\exists U.\mathrm{acar}(U)\land\mathrm{own}(\mathrm{peter},U))\ldots$$

- **Example 12.1.15 (Universal Quantification is Static).**
  Peter does not own every book^1. * *It$_1$ is a novel.*

$$\begin{array}{|c|}\hline \\ \hline \neg \begin{array}{|c|}\hline \\\hline \begin{array}{|l|}\hline U\\\hline \mathrm{book}(U)\\\hline\end{array} \Rrightarrow \begin{array}{|l|}\hline \\\hline \mathrm{own}(\mathrm{peter},U)\\\hline\end{array}\\\hline\end{array}\\\hline\end{array} \otimes \begin{array}{|l|}\hline \\\hline \mathrm{novel}(U)\\\hline\end{array} \qquad \neg(\forall U.\mathrm{book}(U)\Rightarrow\mathrm{own}(\mathrm{peter},U))\ldots$$

- **Example 12.1.16 (Existential Quantification is Dynamic).**
  There is a book^1 that Peter does not own. *It$_1$ is a novel.*

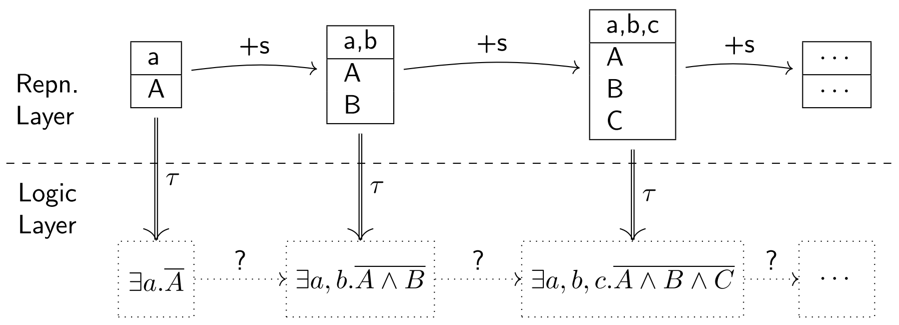
$$\begin{array}{|l|}\hline V\\\hline \mathrm{book}(V)\\ (\neg\mathrm{own}(\mathrm{peter},V))\\\hline\end{array} \otimes \begin{array}{|l|}\hline \\\hline \mathrm{novel}(V)\\\hline\end{array} \qquad \exists U.\mathrm{book}(U)\land\neg\mathrm{own}(\mathrm{peter},U)\land\mathrm{novel}(U)$$

Example 12.1.14 shows that dynamic negation closes off the dynamic potential. Indeed, the referent U is not accessible in the second argument of $\otimes$. Example 12.1.15 predicts the inaccessibility of U for the same reason. In contrast to that, U is accessible in Example 12.1.16, since it is not under the scope of a dynamic negation.

The examples above, and in particular the difference between Example 12.1.15 and Example 12.1.16 show that DRT forms a representational level above recall that we can translate down – PL^1, which serves as the semantic target language. Indeed DRT@ makes finer distinctions than PL^1, and supports an incremental process of semantics construction: DRS construction for sentences followed by DRS merging via $=_\tau$ reduction.

DRT as a Representational Level

- DRT adds a level to the knowledge representation which provides anchors (the discourse referents) for anaphora and the like.
- Propositional semantics by translation into PL^1. ("+s" adds a sentence)


Repn. Layer
Logic Layer
a
A
a,b
A
B
a,b,c
A
B
C
+s
τ
$\exists a.\overline{A}$
$\exists a,b.\overline{A\wedge B}$
$\exists a,b,c.\overline{A\wedge B\wedge C}$
?
...

▷ Anaphor resolution works incrementally on the representational level.

We will now introduce a "direct semantics" for DRT: a notion of "model" and an evaluation mapping that interprets DRSes directly – i.e. not via a translation of first-order logic. The main idea is that atomic conditions and conjunctions are interpreted largely like first-order formulae, while DRSes are interpreted as sets of states that satisfy the conditions. A DRS is satisfied by a model, if that set is non-empty.

A Direct Semantics for DRT (Dyn. Interpretation $\mathcal{I}^{\delta}_{\varphi}$)

▷ **Definition 12.1.17.** Let $\mathcal{M} = \langle\mathcal{D},\mathcal{I}\rangle$ be a first-order model, then a state is an assignment from discourse referents into $\mathcal{D}$.

▷ **Definition 12.1.18.** Let $\varphi,\psi\colon \mathcal{DR}\to\mathcal{U}$ be states, then we say that ψ extends φ on $\mathcal{X}\subseteq\mathcal{DR}$ (write $\varphi[\mathcal{X}]\psi$), if $\varphi(U)=\psi(U)$ for all $U\notin\mathcal{X}$.

▷ **Idea:** Conditions as truth values; DRSes as pairs $(\mathcal{X},\mathcal{S})$ ($\mathcal{S}$ set of states)

▷ **Definition 12.1.19 (Meaning of complex formulae).** The value function $\mathcal{I}_{\varphi}$ for DRT is defined with the help of a dynamic value function $\mathcal{I}^{\delta}_{\varphi}$ on DRSs: For conditions:

- ▷ $\mathcal{I}_{\varphi}(\neg\mathcal{D})=\mathsf{T}$, if $\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^2=\emptyset$.
- ▷ $\mathcal{I}_{\varphi}(\mathcal{D}\vee\mathcal{E})=\mathsf{T}$, if $\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^2\neq\emptyset$ or $\mathcal{I}^{\delta}_{\varphi}(\mathcal{E})^2\neq\emptyset$.
- ▷ $\mathcal{I}_{\varphi}(\mathcal{D}\Rrightarrow\mathcal{E})=\mathsf{T}$, if for all $\psi\in\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^2$ there is a $\tau\in\mathcal{I}^{\delta}_{\varphi}(\mathcal{E})^2$ with $\psi[\mathcal{I}^{\delta}_{\varphi}(\mathcal{E})^1]\tau$.

For DRSs $\mathcal{D}$ we set $\mathcal{I}_{\varphi}(\mathcal{D})=\mathsf{T}$, iff $\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^2\neq\emptyset$, and define

- ▷ $\mathcal{I}^{\delta}_{\varphi}(\delta\mathcal{X}.\mathbf{C})=(\mathcal{X},\{\psi|\varphi[\mathcal{X}]\psi$ and $\mathcal{I}_{\psi}(\mathbf{C})=\mathsf{T}\})$.
- ▷ $\mathcal{I}^{\delta}_{\varphi}(\mathcal{D}\otimes\mathcal{E})=\mathcal{I}^{\delta}_{\varphi}(\mathcal{D}\,;;\,\mathcal{E})=(\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^1\cup\mathcal{I}^{\delta}_{\varphi}(\mathcal{E})^1,\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})^2\cap\mathcal{I}^{\delta}_{\varphi}(\mathcal{E})^2)$

We use the dynamic value function $\mathcal{I}^{\delta}_{\varphi}(\mathcal{D})$ for DRSs $\mathcal{D}$ that might be continued and (the static0 $\mathcal{I}_{\varphi}(\mathcal{D})$ for ones that are already final.

We can now fortify our intuition by computing the direct semantics of two sentences, which differ in their dynamic potential. We start out with the simple *Peter owns a car* and then progress to *Peter owns no car*.

Examples (Computing Direct Semantics)

▷ **Example 12.1.20.** *Peter owns a car*

$$\begin{aligned} & \mathcal{I}^{\delta}_{\varphi}(\delta U.\mathsf{acar}(U) \wedge \mathsf{own}(\mathsf{peter},U)) \\ = \; & (\{U\},\{\psi|\varphi[U]\psi \text{ and } \mathcal{I}_{\psi}(\mathsf{acar}(U) \wedge \mathsf{own}(\mathsf{peter},U)) = \mathsf{T}\}) \\ = \; & (\{U\},\{\psi|\varphi[U]\psi \text{ and } \mathcal{I}_{\psi}(\mathsf{acar}(U)) = \mathsf{T} \text{ and } \mathcal{I}_{\psi}(\mathsf{own}(\mathsf{peter},U)) = \mathsf{T}\}) \\ = \; & (\{U\},\{\psi|\varphi[U]\psi \text{ and } \psi(U)\in\mathcal{I}(\mathsf{acar}) \text{ and } (\psi(U),\mathsf{peter})\in\mathcal{I}(\mathsf{own})\}) \end{aligned}$$

The set of states $[a/U]$, such that a is a car and is owned by Peter

▷ **Example 12.1.21.** For *Peter owns no car* we look at the condition:

$$\begin{aligned} & \mathcal{I}_{\varphi}(\neg(\delta U.\mathsf{acar}(U) \wedge \mathsf{own}(\mathsf{peter},U))) = \mathsf{T} \\ \Leftrightarrow \; & \mathcal{I}^{\delta}_{\varphi}(\delta U.\mathsf{acar}(U) \wedge \mathsf{own}(\mathsf{peter},U))^2 = \emptyset \\ \Leftrightarrow \; & (\{U\},\{\psi|\varphi[\mathcal{X}]\psi \text{ and } \psi(U)\in\mathcal{I}(\mathsf{acar}) \text{ and } (\psi(U),\mathsf{peter})\in\mathcal{I}(\mathsf{own})\})^2 = \emptyset \\ \Leftrightarrow \; & \{\psi|\varphi[\mathcal{X}]\psi \text{ and } \psi(U)\in\mathcal{I}(\mathsf{acar}) \text{ and } (\psi(U),\mathsf{peter})\in\mathcal{I}(\mathsf{own})\} = \emptyset \end{aligned}$$

i.e. iff there are no a, that are cars and that are owned by Peter.

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 244 2024-01-20 SOME RIGHTS RESERVED

The first thing we see in Example 12.1.20 is that the dynamic potential can directly be read off the direct interpretation of a DRS: it is the domain of the states in the first component. In Example 12.1.21, the interpretation is of the form $(\emptyset,\mathcal{I}^{\delta}_{\varphi}(\mathcal{C}))$, where $\mathcal{C}$ is the condition we compute the truth value of in Example 12.1.21.

12.2 Dynamic Model Generation

We will now establish a method for direct deduction on DRT, i.e. deduction at the representational level of DRT, without having to translate – and retranslate – before deduction.

Deduction in Dynamic Logics

▷ Mechanize the dynamic entailment relation (with anaphora)

▷ Use dynamic deduction theorem to reduce (dynamic) entailment to (dynamic) satisfiability

▷ Direct Deduction on DRT (or DPL) [Sau93; RG94; MR98]

(++) Specialized Calculi for dynamic representations

(– –) Needs lots of development until we have efficient implementations

▷ Translation approach (used in our experiment)

(–) Translate to FOL

(++) Use off-the-shelf theorem prover (in this case `MathWeb`)

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 245 2024-01-20 SOME RIGHTS RESERVED

An Opportunity for Off-The-Shelf ATP?

- ▷ **Pro:** ATP is mature enough to tackle applications
  - ▷ Current ATP are highly efficient reasoning tools.
  - ▷ Full automation is needed for NLP. (ATP as an oracle)
  - ▷ ATP as logic engines is one of the initial promises of the field.
- ▷ **contra:** ATP are general logic systems

1. NLP uses other representation formalisms (DRT, Feature Logic,. . .)
2. ATP optimized for mathematical (combinatorially complex) proofs.
3. ATP (often) do not terminate.

- ▷ **Experiment:** Use translation approach for 1. to test 2. and 3. [Bla+01] (Wow, it works!)

Excursion: Incrementality in Dynamic Calculi

- ▷ For applications, we need to be able to check for
  - ▷ satisfiability ($\exists\mathcal{M}.\mathcal{M}\models\mathbf{A}$), validity ($\forall\mathcal{M}.\mathcal{M}\models\mathbf{A}$) and
  - ▷ entailment ($\mathcal{H}\models\mathbf{A}$, iff $\mathcal{M}\models\mathcal{H}$ implies $\mathcal{M}\models\mathbf{A}$ for all $\mathcal{M}$)
- ▷ **Theorem 12.2.1 (Entailment Theorem).** *$\mathcal{H},\mathbf{A}\models\mathbf{B}$, iff $\mathcal{H}\models\mathbf{A}\Rightarrow\mathbf{B}$. (e.g. for first-order logic and DPL)*
- ▷ **Theorem 12.2.2 (Deduction Theorem).** *For most calculi $\mathcal{C}$ we have $\mathcal{H},\mathbf{A}\vdash_{\mathcal{C}}\mathbf{B}$, iff $\mathcal{H}\vdash_{\mathcal{C}}\mathbf{A}\Rightarrow\mathbf{B}$. (e.g. for $\mathcal{ND}^1$)*
- ▷ **Problem:** Analogue $\mathbf{H}_1\otimes\cdots\otimes\mathbf{H}_n\models\mathbf{A}$ is not equivalent to $\models(\mathbf{H}_1\otimes\cdots\otimes\mathbf{H}_n)\Rrightarrow\mathbf{A}$ in DRT, as $\otimes$ symmetric.
- ▷ **Thus** the validity check cannot be used for entailment in DRT.
- ▷ **Solution:** Use sequential merge $;;$ (from DPL) for sentence composition.

Model Generation for Dynamic Logics

- ▷ **Problem:** Translation approach is not incremental!
  - ▷ For each check, the DRS for the whole discourse has to be translated.
  - ▷ Can become infeasible, once discourses get large (e.g. novel).
  - ▷ This applies for all other approaches for dynamic deduction too.
- ▷ **Idea:** Extend model generation techniques instead!

- **Remember**: A DRS $\mathcal{D}$ is valid in $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$, iff $\mathcal{I}_\emptyset^\delta(\mathcal{D})^2 \neq \emptyset$.
- Find a model $\mathcal{M}$ and state φ, such that $\varphi \in \mathcal{I}_\emptyset^\delta(\mathcal{D})^2$.
- Adapt first-order model generation technology for that.

Dynamic Herbrand Interpretation

- **Definition 12.2.3.** We call a model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I}, \mathcal{I}^\delta_\cdot \rangle$ a dynamic Herbrand interpretation, if $\langle \mathcal{U}, \mathcal{I} \rangle$ is a Herbrand model.
- Can represent $\mathcal{M}$ as a triple $\langle \mathcal{X}, \mathcal{S}, \mathcal{B} \rangle$, where $\mathcal{B}$ is the Herbrand base for $\langle \mathcal{U}, \mathcal{I} \rangle$.
- **Definition 12.2.4.** $\mathcal{M}$ is called finite, iff $\mathcal{U}$ is finite.
- **Definition 12.2.5.** $\mathcal{M}$ is minimal, iff for all $\mathcal{M}'$ the following holds: $(\mathcal{B}(\mathcal{M})' \subseteq \mathcal{B}(\mathcal{M})) \Rightarrow \mathcal{M} = \mathcal{M}'$.
- **Definition 12.2.6.** $\mathcal{M}$ is domain minimal if for all $\mathcal{M}'$ the following holds:

$$\#(\mathcal{U}(\mathcal{M})) \leq \#(\mathcal{U}(\mathcal{M})')$$

Dynamic Model Generation Calculus

- **Definition 12.2.7.** We use a tableau framework, extend by state information, and rules for DRSes.
- $$\frac{(\delta U_{\mathbb{A}}.\mathbf{A})^{\mathsf{T}} \quad \mathcal{H} = \{a_1, \ldots, a_n\} \quad w \notin \mathcal{H} \text{ new}}{\begin{array}{c}[a_1/U] \\ ([a_1/U](\mathbf{A}))^{\mathsf{T}}\end{array} \Bigg| \ldots \Bigg| \begin{array}{c}[a_n/U] \\ ([a_n/U](\mathbf{A}))^{\mathsf{T}}\end{array} \Bigg| \begin{array}{c}[w/U] \\ ([w/U](\mathbf{A}))^{\mathsf{T}}\end{array}} RM\,\delta$$
- Mechanize ;; by adding representation of the second DRS at all leaves. ($\leadsto$ tableau machine)
- Treat conditions by DRT translation

$$\frac{\neg \mathcal{D}}{\overline{\neg \mathcal{D}}} \qquad \frac{\mathcal{D} \Rrightarrow \mathcal{D}'}{\overline{\mathcal{D} \Rrightarrow \mathcal{D}'}} \qquad \frac{\mathcal{D} \vee \mathcal{D}'}{\overline{\mathcal{D} \vee \mathcal{D}'}}$$

Example: *Peter is a man. No man walks*

- **Example 12.2.8 (Model Generation).** *Peter is a man. No man walks*

$$\begin{array}{c}
\boxed{\mathsf{man}(\mathsf{peter})}\\
\boxed{\neg(\delta U.\mathsf{man}(U) \wedge \mathsf{walks}(U))}\\
(\mathsf{man}(U) \wedge \mathsf{walks}(U))^{\mathsf{T}}\\
(\forall X.\mathsf{man}(X) \wedge \mathsf{walks}(X))^{\mathsf{F}}\\
(\mathsf{man}(\mathsf{peter}) \wedge \mathsf{walks}(\mathsf{peter}))^{\mathsf{F}}\\
\begin{array}{c|c}
\mathsf{man}(\mathsf{peter})^{\mathsf{F}} & \mathsf{walks}(\mathsf{peter})^{\mathsf{F}}\\
\bot &
\end{array}
\end{array}$$

Dynamic Herbrand interpretation: $\langle\emptyset, \emptyset, \{\mathsf{man}(\mathsf{peter})^{\mathsf{T}}, \mathsf{walks}(\mathsf{peter})^{\mathsf{F}}\}\rangle$

Example: Anaphor Resolution *A man sleeps. He snores*

▷ **Example 12.2.9 (Anaphor Resolution).** *A man sleeps. He snores*

$$\begin{array}{c}
\boxed{\delta U_{\mathrm{Man}}.\mathsf{man}(U) \wedge \mathsf{sleeps}(U)}\\
[c^1_{\mathrm{Man}}/U_{\mathrm{Man}}]\\
\mathsf{man}(c^1_{\mathrm{Man}})^{\mathsf{T}}\\
\mathsf{sleeps}(c^1_{\mathrm{Man}})^{\mathsf{T}}\\
\boxed{\delta V_{\mathrm{Man}}.\mathsf{snores}(V)}\\
\begin{array}{c|c}
[c^1_{\mathrm{Man}}/V_{\mathrm{Man}}] & [c^2_{\mathrm{Man}}/V_{\mathrm{Man}}]\\
\mathsf{snores}(c^1_{\mathrm{Man}})^{\mathsf{T}} & \mathsf{snores}(c^2_{\mathrm{Man}})^{\mathsf{T}}\\
\text{minimal} & \text{deictic}
\end{array}
\end{array}$$

Anaphora with World Knowledge

▷ **Example 12.2.10 (Anaphora with World Knowledge).**

- ▷ *Mary is married to Jeff. Her husband is not in town.*
- ▷ $\delta U_{\mathbb{F}}, V_{\mathbb{M}}.U = \mathsf{mary} \wedge \mathsf{married}(U,V) \wedge V = \mathsf{jeff} \,;;\, \delta W_{\mathbb{M}}, W'_{\mathbb{F}}.\mathsf{husband}(W,W') \wedge \neg\mathsf{intown}(W)$
- ▷ World knowledge
  - ▷ if a female X is married to a male Y, then Y is X's only husband
  - ▷ $\forall X_{\mathbb{F}}, Y_{\mathbb{M}}.\mathsf{married}(X,Y) \Rightarrow \mathsf{husband}(Y,X) \wedge (\forall Z.\mathsf{husband}(Z,X) \Rightarrow Z = Y)$
- ▷ Model generation yields tableau, all branches contain

$$\langle\{U, V, W, W'\}, \{[\mathsf{mary}/U], [\mathsf{jeff}/V], [\mathsf{jeff}/W], [\mathsf{mary}/W']\}, \mathcal{H}\rangle$$

with

$$\mathcal{H} = \{\mathsf{married}(\mathsf{mary},\mathsf{jeff})^{\mathsf{T}}, \mathsf{husband}(\mathsf{jeff},\mathsf{mary})^{\mathsf{T}}, \neg\mathsf{intown}(\mathsf{jeff})^{\mathsf{T}}\}$$

  - ▹ they only differ in additional negative facts, e.g. $\mathrm{married}(\mathrm{mary},\mathrm{mary})^{\mathsf{F}}$.

Model Generation models Discourse Understanding

- ▷ Conforms with psycholinguistic findings:
  - ▹ Zwaan& Radvansky [ZR98]: listeners not only represent logical form, but also models containing referents.
  - ▹ deVega [de 95]: online, incremental process.
  - ▹ Singer [Sin94]: enriched by background knowledge.
  - ▹ Glenberg et al. [GML87]: major function is to provide basis for anaphor resolution.

The cost we had to pay for being able to deal with discourse phenomena is that we had to abandon the compositional treatment of natural language we worked so hard to establish in fragments 3 and 4. To have this, we would have to have a dynamic λ calculus that would allow us to raise the respective operators to the functional level. Such a logical system is non-trivial, since the interaction of structurally scoped λ-bound variables and dynamically bound discourse referents is non-trivial. **Excursion:** We will discuss such a dynamic λ calculus in**??**.

Chapter 13

Propositional Attitudes and Modalities

13.1 Introduction

Modalities and Propositional Attitudes

▷ **Definition 13.1.1.** Modality is a feature of language that allows for communicating things about, or based on, situations which need not be actual.

▷ **Definition 13.1.2.** Modality is signaled by grammatical expressions (called moods) that express a speaker's general intentions and commitment to how believable, obligatory, desirable, or actual an expressed proposition is.

▷ **Example 13.1.3.** Data on modalities (moods in red)

- ▷ *$\mathbf{A}$ probably holds,* (possibilistic)
- ▷ *it has always been the case that $\mathbf{A}$,* (temporal)
- ▷ *it is well-known that $\mathbf{A}$,* (epistemic)
- ▷ *$\mathbf{A}$ is allowed/prohibited,* (deontic)
- ▷ *$\mathbf{A}$ is provable,* (provability)
- ▷ *$\mathbf{A}$ holds after the program P terminates,* (program)
- ▷ *$\mathbf{A}$ hods during the execution of P.* (dito)
- ▷ *it is necessary that $\mathbf{A}$,* (aletic)
- ▷ *it is possible that $\mathbf{A}$,* (dito)

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 255 2024-01-20 SOME RIGHTS RESERVED

Modeling Modalities and Propositional Attitudes

▷ **Example 13.1.4.** Again, the pattern from above:

- ▷ *it is necessary that Peter knows logic* ($\mathbf{A}$ = Peter knows logic)
- ▷ *it is possible that John loves logic,* ($\mathbf{A}$ = John loves logic)

- ▷ **Observation:** All of the red parts above modify the clause/sentence $\mathbf{A}$. We call them modalities.
- ▷ **Definition 13.1.5 (A related Concept from Philosophy).** A propositional attitude is a mental state held by an agent toward a proposition.
- ▷ **Question:** But how to model this in logic?
- ▷ **Idea:** New sentence-to-sentence operators for *necessary* and *possible*. (extend existing logics with them.)
- ▷ **Observation:** $\mathbf{A}$ *is necessary*, iff $\neg\mathbf{A}$ *is impossible*.
- ▷ **Definition 13.1.6.** A modal logic is a logical system that has logical constants that model modalities.

Various logicians and philosophers looked at ways to use possible worlds, or similar theoretical entities, to give a semantics for modal sentences (specifically, for a modal logic), including Descartes and Leibniz. In the modern era, Carnap, Montague and Hintikka pursued formal developments of this idea. But the semantics for modal logic which became the basis of all following work on the topic was developed by Kripke 1963. This kind of semantics is often referred to as Kripke semantics.

History of Modal Logic

- ▷ Aristoteles studies the logic of necessity and possibility
- ▷ Diodorus: temporal modalities
  - ▷ possible: *is true or will be*
  - ▷ necessary: *is true and will never be false*
- ▷ Clarence Irving Lewis 1918 [Lew18] (Systems $S1, \ldots, S5$)
  - ▷ strict implication $I(\mathbf{A} \wedge \mathbf{B})$ (I for "impossible")
- ▷ Kurt Gödel 1932: Modal logic of provability (S4) [Göd32]
- ▷ Saul Kripke 1959-63: Possible worlds semantics [Kri63]
- ▷ Vaugham Pratt 1976: Dynamic Program Logic [Pra76]
- ▷ $\vdots$

Basic Modal Logics (ML^0 and ML^1)

- ▷ **Definition 13.1.7.** Propositional modal logic ML^0 extends propositional logic with two new logical constants: $\Box$ for necessity and $\Diamond$ for possibility. ($\Diamond\mathbf{A} = \neg(\Box\neg\mathbf{A})$)
- ▷ **Observation:** Nothing hinges on the fact that we use propositional logic!

▷ **Definition 13.1.8.** First-order modal logic ML^1 extends first-order logic with two new logical constants: $\Box$ for necessity and $\Diamond$ for possibility.

▷ **Example 13.1.9.** We interpret

1. *Necessarily, every mortal will die.* as $\Box(\forall X.\mathsf{mortal}(X) \Rightarrow \mathsf{willdie}(X))$
2. *Possibly, something is immortal.* as $\Diamond(\exists X.\neg\mathsf{mortal}(X))$

▷ **Questions:** What do $\Box$ and $\Diamond$ mean? How do they behave?

Epistemic and Doxastic Modality

▷ **Definition 13.1.10.** Modal sentences can convey information about the speaker's state of knowledge (epistemic state) or belief (doxastic state).

▷ **Example 13.1.11.** We might paraphrase sentence (epposs) as (3):

1. A: *Where's John?*
2. B: *He might be in the library.*
3. B': *It is consistent with the speaker's knowledge that John is in the library.*

▷ **Definition 13.1.12.** We way that a world w is an epistemic possibility for an agent B if it could be consistent with B's knowledge.

▷ **Definition 13.1.13.** An epistemic logic is one that models the epistemic state of a speaker. Doxastic logic does the same for the doxastic state.

▷ **Definition 13.1.14.** In deontic modal logic, we interpret the accessibility relation $\mathcal{R}$ as epistemic accessibility:

- ▹ With this $\mathcal{R}$, represent B's utterance as $\Diamond\mathsf{inlib}(j)$.
- ▹ Similarly, represent *John must be in the library*. as $\Box\mathsf{inlib}(j)$.

▷ **Question:** If $\mathcal{R}$ is epistemic accessibility, what properties should it have?

To determine the properties of epistemic accessibility we ask ourselves, what statements involving $\Box$ and $\Diamond$ should be valid on the epistemic interpretation of the operators, and how do we fix the accessibility relation to guarantee this?

Deontic modality

▷ **Definition 13.1.15.** Deontic modality is a modality that indicates how the world ought to be according to certain norms, expectations, speaker desire, etc.

▷ **Definition 13.1.16.** Deontic modality has the following subcategories

- ▹ Commissive modality (the speaker's commitment to do something, like a promise or threat): e.g. *I shall help you*.

- ▷ Directive modality (commands, requests, etc.): e.g. *Come!*, *Let's go!*, *You've got to taste this curry!*
- ▷ Volitive modality (wishes, desires, etc.): *If only I were rich!*

▷ **Question:** If we want to interpret $\Box\text{runs}(j)$ as *It is required that John runs* (or, more idiomatically, as *John must run*), what formulae should be valid on this interpretation of the operators? (This is for homework!)

13.2 Semantics for Modal Logics

Basic Ideas: The fundamental intuition underlying the semantics for modality is that modal statements are statements about *how things might be*, statements about possible states of affairs. According to this intuition, sentence (Example 13.1.9.1) in Example 13.1.9 says that in every possible state of affairs – every way that things might be – every mortal will die, while sentence (Example 13.1.9.2) says that there is some possible state of affairs – some way that things might be – in which something is mortal[1]. What is needed in order to express this intuition in a model theory is some kind of entity which will stand for possible states of affairs, or ways things might be. The entity which serves this purpose is the infamous possible world.

Semantics of ML^0

▷ **Definition 13.2.1.** We use a set $\mathcal{W}$ of possible worlds, and a accessibility relation $\mathcal{R}\subseteq\mathcal{W}\times\mathcal{W}$: if $\mathcal{R}(v,w)$, then we say that w is accessible from v.

▷ **Example 13.2.2.** $\mathcal{W}=\mathbb{N}$ with $\mathcal{R}=\{\langle n,n+1\rangle|n\in\mathbb{N}\}$. (temporal logic)

▷ **Definition 13.2.3.** Variable assignment $\varphi\colon\mathcal{V}_0\times\mathcal{W}\to\mathcal{D}_0$ assigns values to variables in a given possible world.

▷ **Definition 13.2.4.** Value function $\mathcal{I}_\cdot^\cdot\colon\mathcal{W}\times wf\!f_0()\to\mathcal{D}_0$ (assigns values to formulae in a possible world)

- ▷ $\mathcal{I}_\varphi^w(V)=\varphi(w,V)$
- ▷ $\mathcal{I}_\varphi^w(\neg\mathbf{A})=\mathsf{T}$, iff $\mathcal{I}_\varphi^w(\mathbf{A})=\mathsf{F}$ ($\wedge$ analogous)
- ▷ $\mathcal{I}_\varphi^w(\Box\mathbf{A})=\mathsf{T}$, iff $\mathcal{I}_\varphi^{w'}(\mathbf{A})=\mathsf{T}$ for all $w'\in\mathcal{W}$ with $w\mathcal{R}w'$.

▷ **Definition 13.2.5.** We call a triple $\mathcal{M}:=\langle\mathcal{W},\mathcal{R},\mathcal{I}\rangle$ a Kripke model.

In Kripke semantics, the intuitions about the truth conditions of modals sentences are expressed as follows:

- A sentence of the form $\Box\mathbf{A}$, where $\mathbf{A}$ is a proposition, is true at w iff $\mathbf{A}$ is true at every possible world accessible from w.
- A sentence of the form $\Diamond\mathbf{A}$, where $\mathbf{A}$ is a proposition, is true at w iff $\mathbf{A}$ is true at some possible world accessible from w.

You might notice that these truth conditions are parallel in certain ways to the truth conditions for tensed sentence. In fact, the semantics of tense is itself a modal semantics which was developed on analogy to Kripke's modal semantics. Here are the relevant similarities:

[1]Note the impossibility of avoiding modal language in the paraphrase!

1. **Relativization of evaluation** A tensed sentence must be evaluated for truth relative to a given time. A tensed sentence may be true at one time butg false at another. Similarly, we must evaluate modal sentences relative to a possible world, for a modal sentence may be true at one world (i.e. relative to one possible state of affairs) but false at another.

2. **Truth depends on value of embedded formula at another world** The truth of a tensed sentence at a time t depends on the truth of the formula embedded under the tense operator at some relevant time (possibly) different from t. Similarly, the truth of a modal sentence at w depends on the truth of the formula embedded under the modal operator at some world or worlds possibly different from w.

3. **Accessibility** You will notice that the world at which the embedded formula is to be evaluated is required to be accessible from the world of evaluation. The accessibility relation on possible worlds is a generalization of the ordering relation on times that we introduced in our temporal semantics. (We will return to this momentarily).

It will be helpful to start by thinking again about the ordering relation on times introduced in temporal models. This ordering relation is in fact one sort of accessibility relation.
Why did we need the ordering relation? We needed it in order to ensure that our temporal semantics makes intuitively correct predictions about the truth conditions of tensed sentences and about entailment relations between them. Here are two illustrative examples:

Accessibility Relations. E.g. for Temporal Modalities

▷ **Example 13.2.6 (Temporal Worlds with Ordering).** Let $\langle\mathcal{W},\circ,<,\subseteq\rangle$ an interval time structure, then we can use $\langle\mathcal{W},<\rangle$ as a Kripke models. Then PAST becomes a modal operator.

▷ **Example 13.2.7.** Suppose we have $i<j$ and $j<k$. Then intuitively, if *Jane is laughing* is true at i, then *Jane laughed* should be true at j and at k, i.e. $\mathcal{I}_\varphi^w(j)\mathsf{PAST}(\mathsf{laughs}(j))$ and $\mathcal{I}_\varphi^w(k)\mathsf{PAST}(\mathsf{laughs}(j))$.
But this holds only if "$<$" is transitive. (which it is!)

▷ **Example 13.2.8.** Here is a clearly counter-intuitive claim: For any time i and any sentence $\mathbf{A}$, if $\mathcal{I}_\varphi^w(i)\mathsf{PRES}(\mathbf{A})$ then $\mathcal{I}_\varphi^w(i)\mathsf{PAST}(\mathbf{A})$.
(For example, the truth of *Jane is at the finish line* at i implies the truth of *Jane was at the finish line* at i.)
But we would get this result if we allowed $<$ to be reflexive. ($<$ is irreflexive)

▷ Treating tense modally, we obtain reasonable truth conditions.

Thus, by ordering the times in our model in accord with our intuitions about time, we can ensure correct predictions about truth conditions and entailment relations for tensed sentences.

In the modal domain, we do not have intuitions about how possible worlds should be ordered. But we do have intuitions about truth conditions and entailment relations among modal sentences. So we need to set up an accessibility relation on the set of possible worlds in our model which, in combination with the truth conditions for $\Box$ and $\Diamond$ given above, will produce intuitively correct claims about entailment.
One of the prime occupations of modal logicians is to look at the sets of validities which are obtained by imposing various different constraints on the accessibility relation. We will here consider just two examples.
What must be, is:

1. It seems intuitively correct that if it is necessarily the case that $\mathbf{A}$, then $\mathbf{A}$ is true, i.e. that $w_g(\Box\mathbf{A}) = \mathsf{T}$ implies that $w_g(\mathbf{A}) = \mathsf{T}$ or, more simply, that the following formula is valid:
$$\Box\mathbf{A} \Rightarrow \mathbf{A}$$
2. To guarantee this implication, we must ensure that any world w is among the world accessible from w, i.e. we must make $\mathcal{R}$ reflexive.
3. Note that this also guarantees, among other things, that the following is valid: $\mathbf{A} \Rightarrow \Diamond\mathbf{A}$

Whatever is, is necessarily possible:

1. This also seems like a reasonable slogan. Hence, we want to guarantee the validity of:
$$\mathbf{A} \Rightarrow \Box\Diamond\mathbf{A}$$
2. To do this, we must guarantee that if $\mathbf{A}$ is true at a some world w, then for every world w' accessible from w, there is at least one $\mathbf{A}$ world accessible from w'. To do this, we can guarantee that every world w is accessible from every world which is accessible from it, i.e. make $\mathcal{R}$ symmetric.

Modal Axioms (Propositional Logic)

▷ **Definition 13.2.9.** Necessitation: $\dfrac{\mathbf{A}}{\Box\mathbf{A}}N$

▷ **Definition 13.2.10 (Normal Modal Logics).**

System	Axioms	Accessibility Relation
$\mathbb{K}$	$\Box(\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\Box\mathbf{A} \Rightarrow \Box\mathbf{B})$	general
$\mathbb{T}$	$\mathbb{K} + \Box\mathbf{A} \Rightarrow \mathbf{A}$	reflexive
$\mathbb{S}4$	$\mathbb{T} + \Box\mathbf{A} \Rightarrow \Box\Box\mathbf{A}$	reflexive + transitive
$\mathbb{B}$	$\mathbb{T} + \Diamond\Box\mathbf{A} \Rightarrow \mathbf{A}$	reflexive + symmetric
$\mathbb{S}5$	$\mathbb{S}4 + \Diamond\mathbf{A} \Rightarrow \Box\Diamond\mathbf{A}$	equivalence relation

$\mathbb{K}$ Theorems

▷ **Observation 13.2.11.** $\Box(\mathbf{A} \wedge \mathbf{B}) \models \Box\mathbf{A} \wedge \Box\mathbf{B}$ *in* $\mathbb{K}$.

▷ **Observation 13.2.12.** $\mathbf{A} \Rightarrow \mathbf{B} \models \Box\mathbf{A} \Rightarrow \Box\mathbf{B}$ *in* $\mathbb{K}$.

▷ **Observation 13.2.13.** $\mathbf{A} \Rightarrow \mathbf{B} \models \Diamond\mathbf{A} \Rightarrow \Diamond\mathbf{B}$ *in* $\mathbb{K}$.

Translation to First-Order Logic

▷ **Question:** Is modal logic more expressive than predicate logic?

▷ **Answer:** Very rarely! (usually can be translated)

▷ **Definition 13.2.14.** Translation τ from ML into PL^1, (so that the diagram commutes)

$$\begin{tikzcd}
\text{Kripke-Sem.} \arrow[r, "\overline{\tau}"] & \text{Tarski-Sem.} \\
\text{modal logic} \arrow[u, "\mathcal{I}_\varphi^w"] \arrow[r, "\tau"] & \text{predicate logic} \arrow[u, "\mathcal{I}_\varphi"']
\end{tikzcd}$$

▷ **Idea:** Axiomatize Kripke models in PL^1. (diagram is simple consequence)

▷ **Definition 13.2.15.** A logic morphism $\Theta\colon \mathcal{L} \to \mathcal{L}'$ is called

- ▷ correct, iff $\exists \mathcal{M}.\mathcal{M} \models \Phi$ implies $\exists \mathcal{M}'.\mathcal{M}' \models' \Theta(\Phi)$.
- ▷ complete, iff $\exists \mathcal{M}'.\mathcal{M}' \models' \Theta(\Phi)$ implies $\exists \mathcal{M}.\mathcal{M} \models \Phi$.

Modal Logic Translation (formal)

▷ **Definition 13.2.16.** The standard translation τ_w from modal logics to first-order logic is given by the following process:

- ▷ Extend all function constants by a "world argument": $\overline{f} \in \Sigma^f_{k+1}$ for every $f \in \Sigma^f_k$
- ▷ for predicate constants accordingly.
- ▷ insert the "translation world" there: e.g. $\tau_w(f(a,b)) = \overline{f}(w, \overline{a}(w), \overline{b}(w))$.
- ▷ New predicate constant $\mathcal{R}$ for the accessibility relation.
- ▷ New constant s for the "start world".
- ▷ $\tau_w(\Box \mathbf{A}) = \forall w'.w \mathcal{R} w' \Rightarrow \tau_{w'}(\mathbf{A})$.
- ▷ Use all axioms from the respective correspondence theory.

▷ **Definition 13.2.17 (Alternative).** Functional translations, if $\mathcal{R}$ associative:

- ▷ New function constant $f_\mathcal{R}$ for the accessibility relation.
- ▷ Revise the standard translation by one of the following
  - ▷ $\tau_w(\Box \mathbf{A}) = \forall w'.w = f_\mathcal{R}(w') \Rightarrow \tau_w(\mathbf{A})$. (naive solution)
  - ▷ $\tau_{f_\mathcal{R}(w)}(\Box \mathbf{A}) = \tau_w(\mathbf{A})$ (better for mechanizing [Ohl88])

Translation (continued)

▷ **Theorem 13.2.18.** *$\tau_s\colon ML^0 \to PL^0$ is correct and complete.*

▷ *Proof:* show that $\exists \mathcal{M}.\mathcal{M} \models \Phi$ iff $\exists \mathcal{M}'.\mathcal{M}' \models \tau_s(\Phi)$

1. Let $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \varphi \rangle$ with $\mathcal{M} \models \mathbf{A}$
2. chose $\mathcal{M} = \langle \mathcal{W}, \mathcal{I}' \rangle$, such that $\mathcal{I}(\overline{p}) = \varphi(p)\colon \mathcal{W} \to \{\mathsf{T}, \mathsf{F}\}$ and $\mathcal{I}(r) = \mathcal{R}$.

we prove $\mathcal{M} \models_\psi \tau_w(\mathbf{A})'$ for $\psi = Id_\mathcal{W}$ by structural induction over $\mathbf{A}$.

3. $\mathbf{A} = P$
   3.1. $\mathcal{I}_\psi(\tau_w(\mathbf{A})) = \mathcal{I}_\psi(\overline{p}(w)) = \mathcal{I}(\overline{p}(w)) = \varphi(P, w) = \mathsf{T}$
4. $\mathbf{A} = \neg\mathbf{B}$, $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$ *trivial by IH.*
5. $\mathbf{A} = \Box\mathbf{B}$
   5.1. $\mathcal{I}_\psi(\tau_w(\mathbf{A})) = \mathcal{I}_\psi(\forall w.r(w, v) \Rightarrow \tau_v(\mathbf{B})) = \mathsf{T}$, if $\mathcal{I}_\psi(r(w, v)) = \mathsf{F}$ or $\mathcal{I}_\psi(\tau_v(\mathbf{B})) = \mathsf{T}$ for all $v \in \mathcal{W}$.
   5.2. $\mathcal{M} \models_\psi \tau_{v'}(\mathbf{B})$ so by IH $\mathcal{M} \models^v \mathbf{B}$.
   5.3. so $\mathcal{M} \models_\psi \tau_w(\mathbf{A})'$.

Modal Logic (References)

▷ G. E. Hughes und M. M. Cresswell: *A companion to Modal Logic*, University Paperbacks, Methuen (1984) [HC84].

▷ David Harel: *Dynamic Logic*, Handbook of Philosophical Logic, D. Gabbay, Hrsg. Reidel (1984) [Har84].

▷ Johan van Benthem: *Language in Action, Categories, Lambdas and Dynamic Logic*, North Holland (1991) [Ben91].

▷ Reinhard Muskens, Johan van Benthem, Albert Visser, *Dynamics*, in *Handbook of Logic and Language*, Elsevier, (1995) [MBV95].

▷ Blackburn, DeRijke, Vedema: *Modal Logic*; 2001 [BRV01]. look at the chapter "Guide to the literature" in the end.

Excursion: We discuss a model existence theorem that can be the basis of completeness proofs for modal logics in??.

13.3 A Multiplicity of Modalities $\leadsto$ Multimodal Logic

The epistemic and deontic modalities differ from alethic, or logical, modality in that they must be relativized to an individual. Although we can choose to abstract away from this, it is clear that what is possible relative to John's set of beliefs may not be possible relative to Jane's, or that what is obligatory for Jane may not be obligatory for John. A theory of modality for natural language must have a means of representing this relativization.

A Multiplicity of Modalities

▷ Epistemic (knowledge and belief) modalities must be relativized to an individual

- ▷ *Peter knows that Trump is lying habitually.*
- ▷ *John believes that Peter knows that Trump is lying habitually.*
- ▷ *You must take the written drivers' exam to be admitted to the practical test.*

▷ Similarly, we find in natural language expressions of necessity and possibility relative to many different kinds of things.

- ▷ Consider the deontic (obligatory/permissible) modalities
  - ▷ *[Given the university's rules] Jane can take that class.*
  - ▷ *[Given her intellectual ability] Jane can take that class.*
  - ▷ *[Given her schedule] Jane can take that class.*
  - ▷ *[Given my desires] I must meet Henry.*
  - ▷ *[Given the requirements of our plan] I must meet Henry.*
  - ▷ *[Given the way things are] I must meet Henry [every day and not know it].*
- ▷ Many different sorts of modality, sentences are multiply ambiguous towards which one.

Michael Kohlhase: LBS 269 2024-01-20

In a series of papers beginning with her 1978 dissertation (in German), Angelika Kratzer proposed an account of the semantics of natural language which accommodates this ambiguity. (The ambiguity is treated not as a semantic ambiguity, but as context dependency.) Kratzer's account, which is now the standard view in semantics and (well-informed) philosophy of language, adopts central ingredients from Kripke semantics – the basic possible world framework and the notion of an accessibility relation – but puts these together in a novel way. Kratzer's account of modals incorporates an account of natural language conditionals; this account has been influenced by, and been influential for, the accounts of conditionals developed by David Lewis and Robert Stalnaker. These also are now standardly accepted (at least by those who accept the possible worlds framework).

Some references: [Kra12; Lew73; Sta68].

Multimodal Logics

- ▷ **Definition 13.3.1.** A multi modal logic provides operators for multiple modalities: $[1], [2], [3], \ldots, \langle 1\rangle, \langle 2\rangle, \ldots$
- ▷ **Definition 13.3.2.** Multi modal Kripke models provide multiple accessibility relations $\mathcal{R}_1, \mathcal{R}_2, \ldots \subseteq \mathcal{W} \times \mathcal{W}$.
- ▷ **Definition 13.3.3.** The value function in logic generalizes the clause for $\Box$ in ML^0 to
  - ▷ $\mathcal{I}_\varphi^w([i]\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\varphi^{w'}(\mathbf{A}) = \mathsf{T}$ for all $w' \in \mathcal{W}$ with $w \mathcal{R}_i w'$.
- ▷ **Example 13.3.4 (Epistemic Logic: talking about knowing/believing).** $[peter]\langle klaus\rangle\mathbf{A}$ (Peter knows that Klaus considers $\mathbf{A}$ possible)
- ▷ **Example 13.3.5 (Program Logic: talking about programs).**
  $[X{:=}\mathbf{A}][Y{:=}\mathbf{A}]X = Y$ (after assignments, the values of X and Y are equal)

Michael Kohlhase: LBS 270 2024-01-20

We will now contrast DRT (see section 12.1) with a modal logic for modeling imperative programs – incidentally also called "dynamic logic". This will give us new insights into the nature of dynamic phenomena in natural language.

13.4 Dynamic Logic for Imperative Programs

Dynamic Program Logic (DL)

- Modal logics for argumentation about imperative, non-deterministic programs.
- **Idea:** Formalize the traditional argumentation about program correctness: tracing the variable assignments (state) across program statements.
- **Example 13.4.1 (Fibonacci).**
  Consider the following (imperative) program that computes Fib(X) as the value of Z:
  $\alpha{:=}\langle Y,Z\rangle{:=}\langle 1,1\rangle\ ;\ \mathbf{while}\ X \neq 0\ \mathbf{do}\ \langle X,Y,Z\rangle{:=}\langle X-1,Z,Y+Z\rangle\ \mathbf{end}$
  - **States** for the "input" $X = 4$: $\langle 4,\_,\_\rangle, \langle 4,1,1\rangle, \langle 3,1,2\rangle, \langle 2,2,3\rangle, \langle 1,3,5\rangle, \langle 0,5,8\rangle$
  - **Correctness?** For positive X, running α with input $\langle X,\_,\_\rangle$ we end with $\langle 0,F_{(X-1)},F_X\rangle$
  - **Termination?** α does not terminate on input $\langle -1,\_,\_\rangle$.

Multi-Modal Logic fits well

- **Observation:** Multi modal logic fits well
  - States as possible worlds, program statements as accessibility relations.
  - Two syntactic categories: programs α and formulae $\mathbf{A}$.
  - Interpret $[\alpha]\mathbf{A}$ as *If α terminates, then $\mathbf{A}$ holds afterwards*
  - Interpret $\langle\alpha\rangle\mathbf{A}$ as *α terminates and $\mathbf{A}$ holds afterwards*.
- **Example 13.4.2.** Assertions about Fibonacci number (α)
  - $\forall X,Y.[\alpha]Z = \mathrm{Fib}(X)$
  - $\forall X,Y.(X{\geq}0) \Rightarrow \langle\alpha\rangle Z = \mathrm{Fib}(X)$

Levels of Description in Dynamic Logic

- Propositional dynamic logic (DL^0) (independent of variable assignments)
  - $\models ([\alpha]\mathbf{A} \wedge [\alpha]\mathbf{B}) \Leftrightarrow ([\alpha](\mathbf{A} \wedge \mathbf{B}))$
  - $\models ([\mathbf{while}\ \mathbf{A} \vee \mathbf{B}\ \mathbf{do}\ \alpha\ \mathbf{end}]\mathbf{C}) \Leftrightarrow ([\mathbf{while}\ \mathbf{A}\ \mathbf{do}\ \alpha\ \mathbf{end}\ ;\ \mathbf{while}\ \mathbf{B}\ \mathbf{do}\ \alpha\ ;\ \mathbf{while}\ \mathbf{A}\ \mathbf{do}\ \alpha\ \mathbf{end}\ \mathbf{end}]\mathbf{C})$
- First-order program logic (DL^1) (function, predicates uninterpreted)
  - $\models p(f(X)) \Rightarrow g(Y,f(X)) \Rightarrow \langle(Z{:=}f(X))\rangle p(Z,g(Y,Z))$
  - $\models Z = Y \wedge (\forall X.f(g(X)) = X) \Rightarrow [\mathbf{while}\ p(Y)\ \mathbf{do}\ Y{:=}g(Y)\ \mathbf{end}]\langle\mathbf{while}\ Y \neq Z\ \mathbf{do}\ Y{:=}f(Y)\ \mathbf{end}\rangle T$
- DL^1 with interpreted functions, predicates (maybe some other time)

▷ $\forall X.\langle \mathbf{while}\ X \neq 1\ \mathbf{do\ if}\ even(X)\ \mathbf{then} X{:=}\frac{X}{2}\ \mathbf{else}\ X{:=}3X+1\ \mathbf{end}\rangle T$

DL^0 Syntax

▷ **Definition 13.4.3.** Propositional dynamic logic (DL^0) is PL^0 extended by

- ▷ program variables $\mathcal{V}_\pi = \{\alpha, \beta, \gamma, \ldots\}$,
- ▷ modalities $[\alpha], \langle\alpha\rangle$.
- ▷ program constructors $\Sigma^\pi = \{;, \cup, *, ?\}$ (minimal set)

$\alpha \,;\, \beta$	execute first α, then β	sequence
$\alpha \cup \beta$	execute (non-deterministically) either α or β	distribution
$*\alpha$	(non-deterministically) repeat α finitely often	iteration
$\mathbf{A}?$	proceed if $\models \mathbf{A}$, else error	test

▷ **Idea:** Standard program primitives as derived concepts

Construct	as
$\mathbf{if}\ \mathbf{A}\ \mathbf{then}\alpha\ \mathbf{else}\ \beta$	$(\mathbf{A}?\,;\,\alpha) \cup (\neg\mathbf{A}?\,;\,\beta)$
$\mathbf{while}\ \mathbf{A}\ \mathbf{do}\ \alpha\ \mathbf{end}$	$*(\mathbf{A}?\,;\,\alpha)\,;\,\neg\mathbf{A}?$
$\mathbf{repeat}\ \alpha\ \mathbf{until}\ \mathbf{A}\ \mathbf{end}$	$*(\alpha\,;\,\neg\mathbf{A}?)\,;\,\mathbf{A}?$

DL^0 Semantics

▷ **Definition 13.4.4.** A model for DL^0 consists of a set $\mathcal{W}$ of possible worlds called states for DL^0.

▷ **Definition 13.4.5.** DL^0 variable assignments come in two parts:

- ▷ $\varphi\colon \mathcal{V}_0 \times \mathcal{W} \to \mathcal{D}_0$ (for propositional variables)
- ▷ $\pi\colon \mathcal{V}_\pi \to \mathcal{P}(\mathcal{W} \times \mathcal{W})$ (maps program variables to accessibility relations)

▷ **Definition 13.4.6.** The meaning of complex formulae is given by the following value function $\mathcal{I}^w_{\varphi,\pi}\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_0$:

- ▷ $\mathcal{I}^w_{\varphi,\pi}(V) = \varphi(w, V)$ for $V \in \mathcal{V}_0$ and $\mathcal{I}^w_{\varphi,\pi}(\alpha) = \pi(\alpha)$ for $\alpha \in \mathcal{V}_\pi$.
- ▷ $\mathcal{I}^w_{\varphi,\pi}(\neg\mathbf{A}) = \mathsf{T}$ iff $\mathcal{I}^w_{\varphi,\pi}(\mathbf{A}) = \mathsf{F}$
- ▷ $\mathcal{I}^w_{\varphi,\pi}([\alpha]\mathbf{A}) = \mathsf{T}$ iff $\mathcal{I}^{w'}_{\varphi,\pi}(\mathbf{A}) = \mathsf{T}$ for all $w' \in \mathcal{W}$ with $w\mathcal{I}^w_{\varphi,\pi}(\alpha)w'$.
- ▷ $\mathcal{I}^w_{\varphi,\pi}(\alpha) = \pi(\alpha)$. (program variable by assignment)
- ▷ $\mathcal{I}^w_{\varphi,\pi}(\alpha\,;\,\beta) = \mathcal{I}^w_{\varphi,\pi}(\beta) \circ \mathcal{I}^w_{\varphi,\pi}(\alpha)$ (sequence by composition)
- ▷ $\mathcal{I}^w_{\varphi,\pi}(\alpha \cup \beta) = \mathcal{I}^w_{\varphi,\pi}(\alpha) \cup \mathcal{I}^w_{\varphi,\pi}(\beta)$ (distribution by union)
- ▷ $\mathcal{I}^w_{\varphi,\pi}(*\alpha) = \mathcal{I}^w_{\varphi,\pi}(\alpha)^*$ (iteration by reflexive transitive closure)
- ▷ $\mathcal{I}^w_{\varphi,\pi}(\mathbf{A}?) = \{\langle w, w\rangle | \mathcal{I}^w_{\varphi,\pi}(\mathbf{A}) = \mathsf{T}\}$ (test by subset of identity relation)

First-Order Program Logic (DL^1)

- **Observation:** Imperative programs contain variables, constants, functions and predicates (uninterpreted), but no program variables. The main operation is variable assignment.
- **Idea:** Make a multi modal logic in the spirit of DL^0 that features all of these for a deeper understanding.
- **Definition 13.4.7.** First-order program logic (DL^1) combines the features of PL^1, DL^0 without program variables, with the following two assignment operators:
  - nondeterministic assignment $X{:=}?$
  - deterministic assignment $X{:=}\mathbf{A}$
- **Example 13.4.8.** $\models p(f(X)) \Rightarrow g(Y,f(X)) \Rightarrow \langle Z{:=}f(X)\rangle p(Z,g(Y,Z))$ in DL^1.
- **Example 13.4.9.** In DL^1 we have
  $\models Z = Y \wedge (\forall X.p(f(g(X)) = X)) \Rightarrow [\mathbf{while}\ p(Y)\ \mathbf{do}\ Y{:=}g(Y)\ \mathbf{end}]\langle\mathbf{while}\ Y \neq Z\ \mathbf{do}\ Y{:=}f(Y)\ \mathbf{end}\rangle T$

DL^1 Semantics

- **Definition 13.4.10.** Let $\mathcal{M} = \langle\mathcal{D},\mathcal{I}\rangle$ be a first-order model then the states (possible worlds) are variable assignments: $\mathcal{W} = \{\varphi | \varphi\colon \mathcal{V}_\iota \to \mathcal{D}\}$
- **Definition 13.4.11.** For a set $\mathcal{X}$ of variables, write $\varphi[\mathcal{X}]\psi$, iff $\varphi(X) = \psi(X)$ for all $X \notin \mathcal{X}$.
- **Definition 13.4.12.** The meaning of complex formulae is given by the following value function $\mathcal{I}_\varphi^w\colon \mathit{wff}_o(\Sigma,\mathcal{V}_\iota) \to \mathcal{D}_0$
  - $\mathcal{I}_\varphi^w(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{A})$ if $\mathbf{A}$ term or atom.
  - $\mathcal{I}_\varphi^w(\neg\mathbf{A}) = \mathsf{T}$ iff $\mathcal{I}_\varphi^w(\mathbf{A}) = \mathsf{F}$
  - $\vdots$
  - $\mathcal{I}_\varphi^w(X{:=}?) = \{\langle\varphi,\psi\rangle | \varphi[X]\psi\}$
  - $\mathcal{I}_\varphi^w(X{:=}\mathbf{A}) = \{\langle\varphi,\psi\rangle | \varphi[X]\psi$ and $\psi(X) = \mathcal{I}_\varphi(\mathbf{A})\}$.
- **Observation 13.4.13 (Substitution and Quantification).** *We have*
  - $\mathcal{I}_\varphi([X{:=}\mathbf{A}]\mathbf{B}) = \mathcal{I}_{(\varphi,[\mathcal{I}_\varphi(\mathbf{A})/X])}(\mathbf{B})$
  - $\forall X.\mathbf{A} = [X{:=}?]\mathbf{A}$.
- Thus substitutions and quantification are definable in DL^1.

Natural Language as Programming Languages

▷ **Question:** Why is dynamic program logic interesting in a natural language course?

▷ **Answer:** There are fundamental relations between dynamic (discourse) logics and dynamic program logics.

▷ **David Israel:** "*Natural languages are programming languages for mind*" [Isr93]

Chapter 14

Some Issues in the Semantics of Tense

Tense as a Deictic Element

▷ **Goal:** capturing the truth conditions and the logical form of sentences of English.

▷ **Clearly:** the following three sentences have different truth conditions.

1. *Jane saw George.*
2. *Jane sees George.*
3. *Jane will see George.*

▷ **Observation 14.0.1.** *Tense is a deictic element, i.e. its interpretation requires reference to something outside the sentence itself.*

▷ **Remark:** Often, in particular in the case of monoclausal sentences occurring in isolation, as in our examples, this "something" is the speech time.

▷ **Idea:** make use of the reference time *now*:

- ▹ *Jane saw George* is true at a time iff *Jane sees George* was true at some point in time before now.
- ▹ *Jane will see George* is true at a time iff *Jane sees George* will be true at some point in time after now.

A Simple Semantics for Tense

▷ **Problem:** the meaning of *Jane saw George* and *Jane will see George* is defined in terms of *Jane sees George*.

$\leadsto$ We need the truth conditions of the present tense sentence.

▷ **Idea:** *Jane sees George* is true at a time iff Jane sees George at that time.

▷ **Implementation:** Postulate tense operators as sentential operators (expressions of type prop $\to$ prop). Interpret

1. *Jane saw George* as $\mathsf{PAST}(\mathsf{see}(g,j))$,
2. *Jane sees George* as $\mathsf{PRES}(\mathsf{see}(g,j))$, and
3. *Jane wil see George* as $\mathsf{FUT}(\mathsf{see}(g,j))$.

Some notes:

- Most treatments of the semantics of tense invoke some notion of a tenseless proposition/formula for the base case, just like we do. The idea here is that markers of past, present and future all operate on an underlying un-tensed expression, which can be evaluated for truth at a time.
- Note that we have made no attempt to show how these translations would be derived from the natural language syntax. Giving a compositional semantics for tense is a complicated business – for one thing, it requires us to first establish the syntax of tense – so we set this goal aside in this brief presentation.
- Here, we have implicitly assumed that the English modal *will* is simply a tense marker. This is indeed assumed by some. But others consider that it is no accident that *will* has the syntax of other modals like *can* and *must*, and believe that *will* is also semantically a modal.

Models and Evaluation for a Tensed Language

▷ **Problem:** The interpretations of constants vary over time.

▷ **Idea:** Introduce times into our models, and let the interpretation function give values of constants at a time. Relativize the valuation function to times

▷ **Idea:** We will consider temporal structures, where denotations are constant on intervals.

▷ **Definition 14.0.2.** Let $I\subseteq\{[i,j]|i,j{\in}\mathbb{R}\}$ be a set of real intervals, then we call $\langle I,\circ,<,\subseteq\rangle$ an interval time structure, where for intervals $i{:=}[i_l,i_l]$ and $j{:=}[l_l,j_r]$ we say that

- ▹ i and j overlap (written $i\circ j$), iff $l_l{\leq}ir$,
- ▹ i precedes j (written $i<j$), iff $ir{\leq}l_l$, and
- ▹ i is contained in j (written $i\subseteq j$), iff $l_l{\leq}i_l$ and $ir{\leq}j_r$.

▷ **Definition 14.0.3.** A temporal model is a triple $\langle\mathcal{D},\mathbb{I},\mathcal{I}\rangle$, where

- ▹ $\mathcal{D}$ is a set called the domain,
- ▹ $\mathbb{I}$is a interval time structure, and
- ▹ $\mathcal{I}\colon\mathbb{I}\times\Sigma_{\mathcal{T}}{\to}\mathcal{D}$ an interpretation function.

The ordering relation: The ordering relation $<$ is needed to make sure that our models represent temporal relations in an intuitively correct way. Whatever the truth may be about time, as language users we have rather robust intuitions that time goes in one direction along a straight line, so that every moment of time is either before, after or identical to any other moment; and no moment of time is both before and after another moment. If we think of the set of times as the set of natural numbers, then the ordering relation $<$ is just the relation *less than* on that set.

Intervals: Although intuitively time is given by as a set of moments of time, we will adopt here (following Cann, who follows various others) an *interval semantics*, in which expressions are evaluated relative to intervals of time. Intervals are defined in terms of moments, as a continuous set of moments ordered by $<$.
The new interpretation function: In models without times, the interpretation function $\mathcal{I}$ assigned an extension to every constant. Now, we want it to assign an extension to each constant relative to each interval in our interval time structure. I.e. the interpretation function associates each constant with a pair consisting of an interval and an appropriate extension, interpreted as the extension at that interval. This set of pairs is, of course, equivalent to a function from intervals to extensions.

Interpretation rules for the temporal operators

▷ **Definition 14.0.4.** For the value function $\mathcal{I}_i(\varphi)\cdot$ we only redefine the clause for constants:

- ▷ $\mathcal{I}_i(\varphi)c{:=}\mathcal{I}(i,c)$
- ▷ $\mathcal{I}_i(\varphi)X{:=}\varphi(X)$
- ▷ $\mathcal{I}_i(\varphi)\mathbf{FA}{:=}\mathcal{I}_i(\varphi)\mathbf{F}(\mathcal{I}_i(\varphi)\mathbf{A})$.

▷ **Definition 14.0.5.** We define the meaning of the tense operators

1. $\mathcal{I}_i(\varphi)\mathsf{PRES}(\Phi) = \mathsf{T}$, iff $\mathcal{I}_i(\varphi)\Phi = \mathsf{T}$.
2. $\mathcal{I}_i(\varphi)\mathsf{PAST}(\Phi) = \mathsf{T}$ iff there is an interval $j{\in}I$ with $j < i$ and $\mathcal{I}_j(\varphi)\Phi = \mathsf{T}$.
3. $\mathcal{I}_i(\varphi)\mathsf{FUT}(\Phi) = \mathsf{T}$ iff there is an interval $j{\in}I$ with $i < j$ and $\mathcal{I}_j(\varphi)\Phi = \mathsf{T}$.

Complex tenses in English

▷ How do we use this machinery to deal with complex tenses in English?

- ▷ Past of past (pluperfect): *Jane had left (by the time I arrived)*.
- ▷ Future perfect: *Jane will have left (by the time I arrive)*.
- ▷ Past progressive: *Jane was going to leave (when I arrived)*.

▷ Perfective vs. imperfective

- ▷ *Jane left.*
- ▷ *Jane was leaving.*

▷ How do the truth conditions of these sentences differ? **Standard observation:** Perfective indicates a completed action, imperfective indicates an incomplete or ongoing action. This becomes clearer when we look at the "creation predicates" like *build a house* or *write a book*

- ▷ *Jane built a house.* entails: *There was a house that Jane built.*
- ▷ *Jane was building a house.* does not entail that *there was a house that Jane built.*

Future readings of present tense

▷ New Data;

1. *Jane leaves tomorrow.*
2. *Jane is leaving tomorrow.*
3. ?? *It rains tomorrow.*
4. ?? *It is raining tomorrow.*
5. ?? *The dog barks tomorrow.*
6. ??*The dog is barking tomorrow.*

▷ Future readings of present tense appear to arise only when the event described is planned, or planable, either by the subject of the sentence, the speaker, or a third party.

Sequence of Tense

▷ *George said that Jane was laughing.*

- ▷ Reading 1: George said "*Jane is laughing.*" I.e. saying and laughing co-occur. So past tense in subordinate clause is past of utterance time, but not of main clause reference time.
- ▷ Reading 2: George said "*Jane was laughing.*" I.e. laughing precedes saying. So past tense in subordinate clause is past of utterance time and of main clause reference time.

▷ *George saw the woman who was laughing.*

- ▷ How many readings?

▷ *George will say that Jane is laughing.*

- ▷ Reading 1: George will say "*Jane is laughing.*" Saying and laughing co-occur, but both saying and laughing are future of utterance time. So present tense in subordinate clause indicates futurity relative to utterance time, but not to main clause reference time.
- ▷ Reading 2: Laughing overlaps utterance time and saying (by George). So present tense in subordinate clause is present relative to utterance time *and* main clause reference time.

Sequence of Tense

▷ *George will see the woman who is laughing.*

- ▷ How many readings?

▷ Note that in all of the above cases, the predicate in the subordinate clause describes an event that is extensive in time. Consider readings when subordinate event is punctual.

▷ *George said that Mary fell.*

  ▷ Falling must precede George's saying.

▷ *George saw the woman who fell.*

  ▷ Same three readings as before: falling must be past of utterance time, but could be past, present or future relative to seeing (i.e main clause reference time).

▷ And just for fun, consider past under present... *George will claim that Mary hit Bill.*

  ▷ Reading 1: hitting is past of utterance time (therefore past of main clause reference time).

  ▷ Reading 2: hitting is future of utterance time, but past of main clause reference time.

▷ And finally...

1. *A week ago, John decided that in ten days at breakfast he would tell his mother that they were having their last meal together.* (Abusch 1988)
2. *John said a week ago that in ten days he would buy a fish that was still alive.* (Ogihara 1996)

Interpreting tense in Discourse

▷ **Example 14.0.6 (Ordering and Overlap).** *A man walked into the bar. He sat down and ordered a beer. He was wearing a nice jacket and expensive shoes, but he asked me if I could spare a buck.*

▷ **Example 14.0.7 (Tense as anaphora?).**

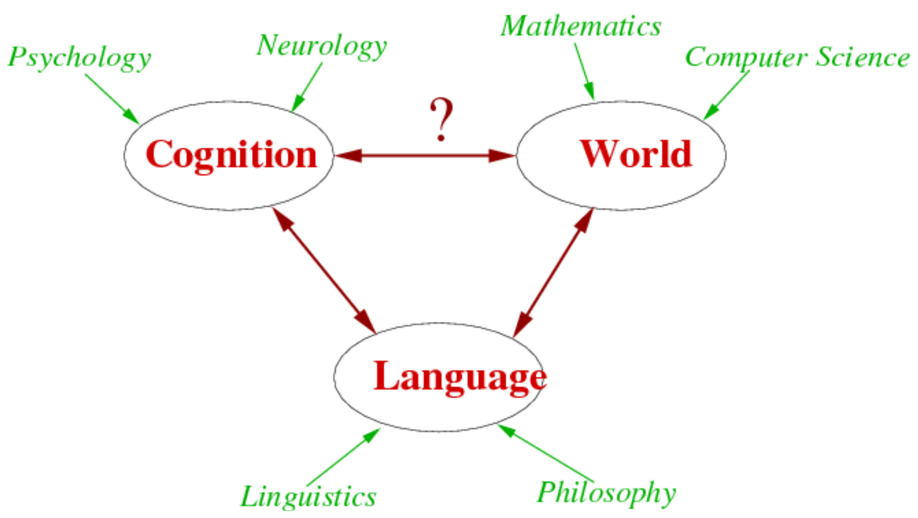
1. Said while driving down the NJ turnpike: *I forgot to turn off the stove.*
2. *I didn't turn off the stove.*

Chapter 15

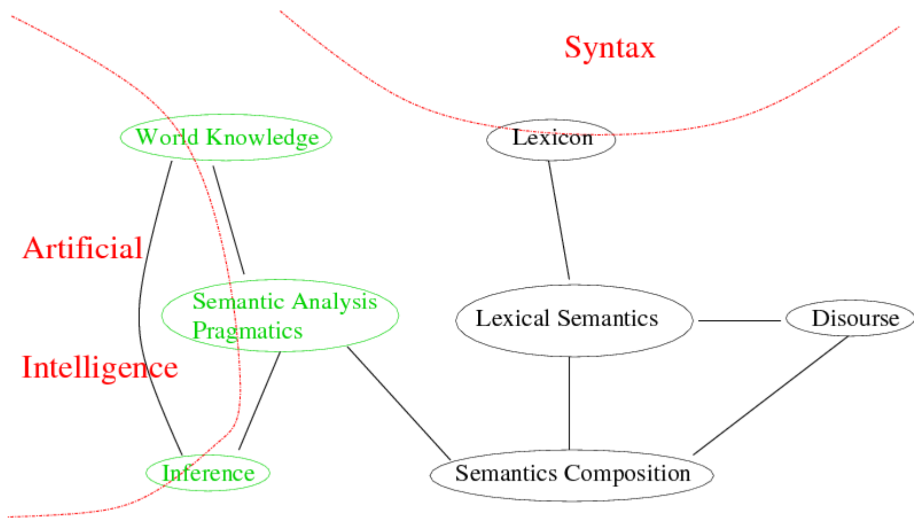
Conclusion

15.1 A Recap in Diagrams

NL Semantics as an Intersective Discipline



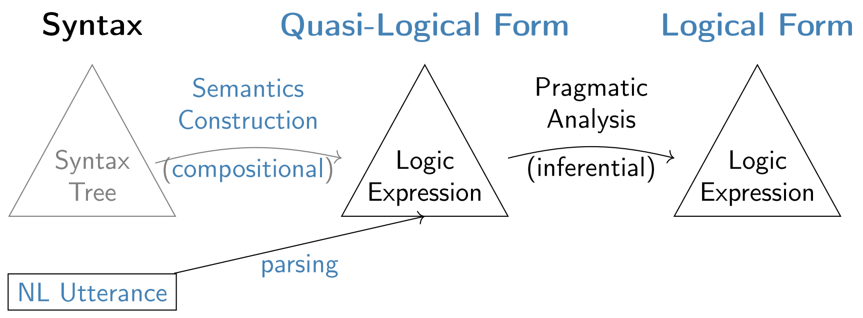
A landscape of formal semantics



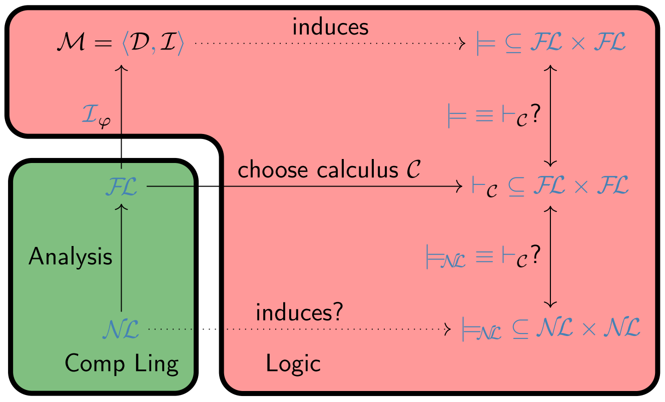
Modeling Natural Language Semantics

- **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.* $((\exists X.\mathsf{man}(X) \wedge \mathsf{sleeps}(X))) \wedge \mathsf{snores}(X)$
  - Various dynamic approaches (e.g. DRT, DPL)
    * *Most men wear black*
  - Higher-order Logic, e.g. generalized quantifiers
  - ...

A Semantic Processing Pipeline based on LF



Natural Language Semantics?


$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$
induces
$\models \subseteq \mathcal{FL} \times \mathcal{FL}$
$\mathcal{I}_\varphi$
$\models \equiv \vdash_\mathcal{C}$?
$\mathcal{FL}$
choose calculus $\mathcal{C}$
$\vdash_\mathcal{C} \subseteq \mathcal{FL} \times \mathcal{FL}$
Analysis
$\models_{\mathcal{NL}} \equiv \vdash_\mathcal{C}$?
induces?
$\mathcal{NL}$
$\models_{\mathcal{NL}} \subseteq \mathcal{NL} \times \mathcal{NL}$
Comp Ling
Logic

15.2 Where to From Here

Where to from here?

- ▷ We can continue the exploration of semantics in two different ways:
  - ▷ Look around for additional logical/formal systems and see how they can be applied to various linguistic problems. (the logician's approach)
  - ▷ Look around for additional linguistic forms and wonder about their truth conditions, their logical forms, and how to represent them. (the linguist's approach)
- ▷ Here are some possibilities...

Semantics of Plurals

1. *The dogs were barking.*
2. *Fido and Chester were barking.* (What kind of an object do the subject NPs denote?)
3. *Fido and Chester were barking. They were hungry.*
4. *Jane and George came to see me. She was upset.* (Sometimes we need to look inside a plural!)
5. *Jane and George have two children.* (Each? Or together?)
6. *Jane and George got married.* (To each other? Or to other people?)
7. *Jane and George met.* (The predicate makes a difference to how we interpret the plural)

Reciprocals

▷ What's required to make these true?

1. *The men all shook hands with one another.*
2. *The boys are all sitting next to one another on the fence.*
3. *The students all learn from each other.*

Presuppositional expressions

▷ What are presuppositions?

▷ What expressions give rise to presuppositions?

▷ Are all apparent presuppositions really the same thing?

1. *The window in that office is open.*
2. *The window in that office isn't open.*
3. *George knows that Jane is in town.*
4. *George doesn't know that Jane is in town.*
5. *It was / wasn't George who upset Jane.*
6. *Jane stopped / didn't stop laughing.*
7. *George is / isn't late.*

Presupposition projection

1. *George doesn't know that Jane is in town.*
2. *Either Jane isn't in town or George doesn't know that she is.*
3. *If Jane is in town, then George doesn't know that she is.*
4. *Henry believes that George knows that Jane is in town.*

Conditionals

▷ What are the truth conditions of conditionals?

1. *If Jane goes to the game, George will go.*
   - ▷ Intuitively, not made true by falsity of the antecedent or truth of consequent independent of antecedent.
2. *If John is late, he must have missed the bus.*

▷ Generally agreed that conditionals are modal in nature. Note presence of modal in consequent of each conditional above.

Counterfactual conditionals

▷ And what about these??

1. *If kangaroos didn't have tails, they'd topple over.* (David Lewis)
2. *If Woodward and Bernstein hadn't got on the Watergate trail, Nixon might never have been caught.*
3. *If Woodward and Bernstein hadn't got on the Watergate trail, Nixon would have been caught by someone else.*

▷ Counterfactuals undoubtedly modal, as their evaluation depends on which alternative world you put yourself in.

Before and after

▷ These seem easy. But modality creeps in again...

1. *Jane gave up linguistics after she finished her dissertation.* (Did she finish?)
2. *Jane gave up linguistics before she finished her dissertation.* (Did she finish? Did she start?)

Bibliography

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. second. Kluwer Academic Publishers, 2002.

[And72] Peter B. Andrews. “General Models and Extensionality”. In: *Journal of Symbolic Logic* 37.2 (1972), pp. 395–397.

[Ari10] Mira Ariel. *Defining Pragmatics*. Research Surveys in Linguistics. Cambridge University Press, 2010.

[BB05] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI, 2005.

[Ben91] Johan van Benthem. *Language in Action, Categories, Lambdas and Dynamic Logic*. Vol. 130. Studies in Logic and Foundation of Mathematics. North Holland, 1991.

[Bir13] Betty J. Birner. *Introduction to Pragmatics*. Wiley-Blackwell, 2013.

[Bla+01] Patrick Blackburn et al. “Inference and Computational Semantics”. In: *Computing Meaning (Volume 2)*. Ed. by Harry Bunt et al. Kluwer Academic Publishers, 2001, pp. 11–28.

[BRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. New York, NY, USA: Cambridge University Press, 2001. ISBN: 0-521-80200-8.

[Cho65a] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.

[Cho65b] Noam Chomsky. *Syntactic structures*. Den Haag: Mouton, 1965.

[Chu40] Alonzo Church. “A Formulation of the Simple Theory of Types”. In: *Journal of Symbolic Logic* 5 (1940), pp. 56–68.

[CKG09] Ronnie Cann, Ruth Kempson, and Eleni Gregoromichelaki. *Semantics – An Introduction to Meaning in Language*. Cambridge University Press, 2009. ISBN: 0521819628.

[Cre82] M. J. Cresswell. “The Autonomy of Semantics”. In: *Processes, Beliefs, and Questions: Essays on Formal Semantics of Natural Language and Natural Language Processing*. Ed. by Stanley Peters and Esa Saarinen. Springer, 1982, pp. 69–86. DOI: 10.1007/978-94-015-7668-0_2.

[Cru11] Alan Cruse. *Meaning in Language: An Introduction to Semantics and Pragmatics*. Oxford Textbooks in Linguistics. 2011.

[Dav67a] Donald Davidson. “The logical form of action sentences”. In: *The logic of decision and action*. Ed. by N. Rescher. Pittsburgh: Pittsburgh University Press, 1967, pp. 81–95.

[Dav67b] Donald Davidson. “Truth and Meaning”. In: *Synthese* 17 (1967).

[de 95] Manuel de Vega. “Backward updating of mental models during continuous reading of narratives”. In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 21 (1995), pp. 373–385.

[DSP91] Mary Dalrymple, Stuart Shieber, and Fernando Pereira. “Ellipsis and Higher-Order Unification”. In: *Linguistics & Philosophy* 14 (1991), pp. 399–452.

[Eij97] Jan van Eijck. “Type Logic with States”. In: *Logic Journal of the IGPL* 5.5 (Sept. 1997).

[EU10] Jan van Eijck and Christina Unger. *Computational Semantics with Functional Programming*. Cambridge University Press, 2010.

[Fre92] Gottlob Frege. “Über Sinn und Bedeutung”. In: *Zeitschrift für Philosophie und philosophische Kritik* 100 (1892), pp. 25–50.

[GF] *GF - Grammatical Framework*. URL: `http://www.grammaticalframework.org` (visited on 09/27/2017).

[GK96] Claire Gardent and Michael Kohlhase. “Focus and Higher–Order Unification”. In: *Proceedings of the 16th International Conference on Computational Linguistics*. Copenhagen, 1996, pp. 268–279. URL: `https://kwarc.info/kohlhase/papers/coling96.pdf`.

[GKL96] Claire Gardent, Michael Kohlhase, and Noor van Leusen. “Corrections and Higher-Order Unification”. In: *Proceedings of KONVENS’96*. Bielefeld, Germany: De Gruyter, 1996, pp. 268–279. URL: `https://kwarc.info/kohlhase/papers/konvens96.pdf`.

[GML87] A. M. Glenberg, M. Meyer, and K. Lindem. “Mental models contribute to foregrounding during text comprehension”. In: *Journal of Memory and Language* 26 (1987), pp. 69–83.

[Göd32] Kurt Gödel. “Zum Intuitionistischen Aussagenkalkül”. In: *Anzeiger der Akademie der Wissenschaften in Wien* 69 (1932), pp. 65–66.

[GS90] Jeroen Groenendijk and Martin Stokhof. “Dynamic Montague Grammar”. In: *Papers from the Second Symposium on Logic and Language*. Ed. by L. Kálmán and L. Pólos. Akadémiai Kiadó, Budapest, 1990, pp. 3–48.

[GS91] Jeroen Groenendijk and Martin Stokhof. “Dynamic Predicate Logic”. In: *Linguistics & Philosophy* 14 (1991), pp. 39–100.

[Har84] D. Harel. “Dynamic Logic”. In: *Handbook of Philosophical Logic*. Ed. by D. Gabbay and F. Günthner. Vol. 2. Reidel, Dordrecht, 1984, pp. 497–604.

[HC84] G. E. Hughes and M. M. Cresswell. *A companion to Modal Logic*. University Paperbacks. Methuen, 1984.

[Hei82] Irene Heim. “The Semantics of Definite and Indefinite Noun Phrases”. PhD thesis. University of Massachusetts, 1982.

[HHS07] James R. Hurford, Brendan Heasley, and Michael B. Smith. *Semantics: A coursebook*. 2nd. Cambridge University Press, 2007.

[Hue80] Gérard Huet. “Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems”. In: *Journal of the ACM (JACM)* 27.4 (1980), pp. 797–821.

[Isr93] David J. Israel. “The Very Idea of Dynamic Semantics”. In: *Proceedings of the Ninth Amsterdam Colloquium*. 1993. URL: `https://arxiv.org/pdf/cmp-lg/9406026.pdf`.

[Jac83] Ray Jackendoff. *Semantics and Cognition*. MIT Press, 1983.

[JL83] P. N. Johnson-Laird. *Mental Models*. Cambridge University Press, 1983.

[JLB91] P. N. Johnson-Laird and Ruth M. J. Byrne. *Deduction*. Lawrence Erlbaum Associates Publishers, 1991.

[Kam81] Hans Kamp. “A Theory of Truth and Semantic Representation”. In: *Formal Methods in the Study of Language*. Ed. by J. Groenendijk, Th. Janssen, and M. Stokhof. Amsterdam, Netherlands: Mathematisch Centrum Tracts, 1981, pp. 277–322.

[Kea11] Kate Kearns. *Semantics*. 2nd. Palgrave Macmillan, 2011.

[KKP96] Michael Kohlhase, Susanna Kuschert, and Manfred Pinkal. "A type-theoretic semantics for λ-DRT". In: *Proceedings of the 10th Amsterdam Colloquium*. Ed. by P. Dekker and M. Stokhof. ILLC. Amsterdam, 1996, pp. 479–498. URL: `https://kwarc.info/kohlhase/papers/amscoll95.pdf`.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: `https://kwarc.info/kohlhase/papers/mcs08-stex.pdf`.

[Kon04] Karsten Konrad. *Model Generation for Natural Language Interpretation and Analysis*. Vol. 2953. LNCS. Springer, 2004. ISBN: 3-540-21069-5. DOI: `10.1007/b95744`.

[Kow97] Robert Kowalski. "Algorithm = Logic + Control". In: *Communications of the Association for Computing Machinery* 22 (1997), pp. 424–436.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993.

[Kra12] Angelika Kratzer. *Modals and Conditionals. New and Revised Perspectives*. Oxford Studies in Theoretical Linguistics. Oxford University Press, 2012.

[Kri63] Saul Kripke. "Semantical Considerations on Modal Logic". In: *Acta Philosophica Fennica* (1963), pp. 83–94.

[Lee02] Lillian Lee. "Fast context-free grammar parsing requires fast Boolean matrix multiplication". In: *Journal of the ACM* 49.1 (2002), pp. 1–15.

[Lew18] Clarence Irving Lewis. *A Survey of Symbolic Logic*. University of California Press, 1918. URL: `http://hdl.handle.net/2027/hvd.32044014355028`.

[Lew73] David K. Lewis. *Counterfactuals*. Blackwell Publishers, 1973.

[MBV95] Reinhard Muskens, Johan van Benthem, and Albert Visser. "Dynamics". In: ed. by Johan van Benthem and Ter Meulen. Elsevier Science B.V., 1995.

[Mon70] R. Montague. "English as a Formal Language". In: Reprinted in [Tho74], 188–221. Edizioni di Communita, Milan, 1970, pp. 189–224.

[Mon74] Richard Montague. "The Proper Treatment of Quantification in Ordinary English". In: *Formal Philosophy. Selected Papers*. Ed. by R. Thomason. New Haven: Yale University Press, 1974.

[MR98] C. Monz and M. de Rijke. "A Resolution Calculus for Dynamic Semantics". In: *Logics in Artificial Intelligence. European Workshop JELIA '98*. LNAI 1489. Springer Verlag, 1998.

[Mus96] Reinhard Muskens. "Combining Montague Semantics and Discourse Representation". In: *Linguistics & Philosophy* 14 (1996), pp. 143 –186.

[Ohl88] Hans Jürgen Ohlbach. "A Resolution Calculus for Modal Logics". PhD thesis. Universität Kaiserslautern, 1988.

[OMT] Michael Kohlhase and Dennis Müller. *OMDoc/MMT Tutorial for Mathematicians*. URL: `https://gl.mathhub.info/Tutorials/Mathematicians/blob/master/tutorial/mmt-math-tutorial.pdf` (visited on 10/07/2017).

[Par90] Terence Parsons. *Events in the Semantics of English: A Study in Subatomic Semantics*. Vol. 19. Current Studies in Linguistics. MIT Press, 1990.

[Pin96] Manfred Pinkal. "Radical underspecification". In: *Proceedings of the 10th Amsterdam Colloquium*. Ed. by P. Dekker and M. Stokhof. ILLC. Amsterdam, 1996, pp. 587–606.

[Pop34] Karl Popper. *Logik der Forschung*. Springer Verlag, 1934.

[Pop59] Karl Popper. *Logic of Scientific Discovery*. Basic Books, 1959.

[Por04] Paul Portner. *What is Meaning? Fundamentals of Formal Semantics*. Blackwell, 2004.

[Pra76] V. Pratt. "Semantical considerations of Floyd-Hoare logic". In: *Proceedings of the 17th Symposium on Foundations of Computer Science*. 1976, pp. 109–121.

[Pul94] Stephen G. Pulman. *Higher Order Unification and the Interpretation of Focus*. Tech. rep. CRC-049. SRI Cambridge, UK, 1994.

[Ran04] Aarne Ranta. "Grammatical Framework — A Type-Theoretical Grammar Formalism". In: *Journal of Functional Programming* 14.2 (2004), pp. 145–189.

[Ran11] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, 2011. ISBN: 1-57586-626-9.

[RG94] Uwe Reyle and Dov M. Gabbay. "Direct Deductive computation on Discourse Representation Structures". In: *Linguistics & Philosophy* 17 (1994), pp. 343–390.

[Rie10] Nick Riemer. *Introducing Semantics*. Cambridge Introductions to Language and Linguistics. Cambridge University Press, 2010.

[Rus91] Stuart J. Russell. "An Architecture for Bounded Rationality". In: *SIGART Bulletin* 2.4 (1991), pp. 146–150.

[Sae03] John I. Saeed. *Semantics*. 2nd. Blackwell, 2003.

[Sau93] Werner Saurer. "A Natural Deduction System for Discourse Representation Theory". In: *Journal of Philosophical Logic* 22 (1993).

[Sch20] Jan Frederik Schaefer. "Prototyping NLU Pipelines – A Type-Theoretical Framework". Master's Thesis. Informatik, FAU Erlangen-Nürnberg, 2020. URL: `https://gl.kwarc.info/supervision/MSc-archive/blob/master/2020/Schaefer_Jan_Frederik.pdf`.

[Sin94] M. Singer. "Discourse Inference Processes". In: *Handbook of Psycholinguistics*. Ed. by M. A. Gernsbacher. Academic Press, 1994, pp. 479–515.

[Smu63] Raymond M. Smullyan. "A Unifying Principle for Quantification Theory". In: *Proc. Nat. Acad Sciences* 49 (1963), pp. 828–832.

[Spe17] Jeff Speaks. "Theories of Meaning". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2017. Metaphysics Research Lab, Stanford University, 2017. URL: `https://plato.stanford.edu/archives/fall2017/entries/meaning/`.

[Sta14] Robert Stalnaker. *Context*. Oxford University Press, 2014.

[Sta68] Robert C. Stalnaker. "A Theory of Conditionals". In: *Studies in Logical Theory, American Philosophical Quarterly*. Blackwell Publishers, 1968, pp. 98–112.

[Sta85] Rick Statman. "Logical relations and the typed lambda calculus". In: *Information and Computation* 65 (1985).

[sTeX] *sTeX: A semantic Extension of TeX/LaTeX*. URL: `https://github.com/sLaTeX/sTeX` (visited on 05/11/2020).

[Tho74] R. Thomason, ed. *Formal Philosophy: selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.

[Ven57] Zeno Vendler. "Verbs and times". In: *Philosophical Review* 56 (1957), pp. 143–160.

[Zee89] Henk Zeevat. "A Compositional Approach to DRT". In: *Linguistics & Philosophy* 12 (1989), pp. 95–131.

[ZR98] R. A. Zwaan and G. A. Radvansky. "Situation models in language comprehension and memory". In: *Psychological Bulletin* 123 (1998), pp. 162–185.

[ZS13] Thomas Ede Zimmermann and Wolfgang Sternefeld. *Introduction to Semantics*. de Gruyter Mouton, 2013.

Index

Part III

Excursions

As this course is predominantly about modeling natural language and not about the theoretical aspects of the logics themselves, we give the discussion about these as a “suggested readings” section part here.

Appendix A

Properties of Propositional Tableaux

A.1 Soundness and Termination of Tableaux

As always we need to convince ourselves that the calculus is sound, otherwise, tableau proofs do not guarantee validity, which we are after. Since we are now in a refutation setting we cannot just show that the inference rules preserve validity: we care about unsatisfiability (which is the dual notion to validity), as we want to show the initial labeled formula to be unsatisfiable. Before we can do this, we have to ask ourselves, what it means to be (un)-satisfiable for a labeled formula or a tableau.

Soundness (Tableau)

▷ **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.

▷ **Definition A.1.1.** A labeled formula $\mathbf{A}^{\alpha}$ is valid under φ, iff $\mathcal{I}_{\varphi}(\mathbf{A}) = \alpha$.

▷ **Definition A.1.2.** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae on $\mathcal{P}$ is satisfiable.

▷ **Lemma A.1.3.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*

▷ **Theorem A.1.4 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wff_0(\mathcal{V}_0)$ valid, if there is a closed tableau $\mathcal{T}$ for Φ^{F}.*

▷ *Proof:* by contradiction

1. Suppose Φ isfalsifiable $\widehat{=}$ not valid.
2. Then the initial tableau is satisfiable, (Φ^{F} satisfiable)
3. so $\mathcal{T}$ is satisfiable, by Lemma C.0.3.
4. Thus there is a satisfiable branch (by definition)
5. but all branches are closed ($\mathcal{T}$ closed)

▷ **Theorem A.1.5 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wff_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^{F}.*

Proof sketch: Proof difficult/interesting; see Corollary A.3.2

Thus we only have to prove Lemma C.0.3, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains $(\mathbf{A} \wedge \mathbf{B})^{\mathsf{T}}$ and is satisfiable, then it must have a satisfiable branch. If $(\mathbf{A} \wedge \mathbf{B})^{\mathsf{T}}$ is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$ for some variable assignment φ. Thus $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ and $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$, so after the extension (which adds the formulae $\mathbf{A}^{\mathsf{T}}$ and $\mathbf{B}^{\mathsf{T}}$ to the branch), the branch is still satisfiable. The cases for the other rules are similar.

The next result is a very important one, it shows that there is a procedure (the tableau procedure) that will always terminate and answer the question whether a given propositional formula is valid or not. This is very important, since other logics (like the often-studied first-order logic) does not enjoy this property.

▷ Termination for Tableaux

▷ **Lemma A.1.6.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*

▷ *Proof:* By examining the rules wrt. a measure μ

1. Let us call a labeled formulae $\mathbf{A}^\alpha$ worked off in a tableau $\mathcal{T}$, if a $\mathcal{T}_0$ rule has already been applied to it.
2. It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
3. Let $\mu(\mathcal{T})$ be the number of connectives in labeled formulae in $\mathcal{T}$ that are not worked off.
4. Then each rule application to a labeled formula in $\mathcal{T}$ that is not worked off reduces $\mu(\mathcal{T})$ by at least one. (inspect the rules)
5. At some point the tableau only contains worked off formulae and literals.
6. Since there are only finitely many literals in $\mathcal{T}$, so we can only apply $\mathcal{T}_0\bot$ a finite number of times.

▷ **Corollary A.1.7.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*

Proof: We combine the results so far

▷
1. By Lemma A.1.6 it is decidable whether $\vdash_{\mathcal{T}_0}\mathbf{A}$
2. By soundness (Theorem C.0.4) and completeness (Theorem C.0.5), $\vdash_{\mathcal{T}_0}\mathbf{A}$ iff $\mathbf{A}$ is valid.

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 302 2024-01-20 SOME RIGHTS RESERVED

Note: The proof above only works for the "base $\mathcal{T}_0$" because (only) there the rules do not "copy". A rule like

$$\begin{array}{c|c} \multicolumn{2}{c}{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{T}}} \\ \hline \mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}} \end{array}$$

does, and in particular the number of non-worked-off variables below the line is larger than above the line. For such rules, we would have a more intricate version of μ which – instead of returning a natural number – returns a more complex object; a multiset of numbers. would work here. In our proof we are just assuming that the defined connectives have already eliminated. The tableau calculus basically computes the disjunctive normal form: every branch is a disjunct that is a conjunction of literals. The method relies on the fact that a DNF is unsatisfiable, iff each literal is, i.e. iff each branch contains a contradiction in form of a pair of opposite literals.

For proving completeness of tableaux we will use the abstract consistency method introduced by Raymond Smullyan — a famous logician who also wrote many books on recreational mathematics and logic (most notably one is titled "What is the name of this book?") which you may like.

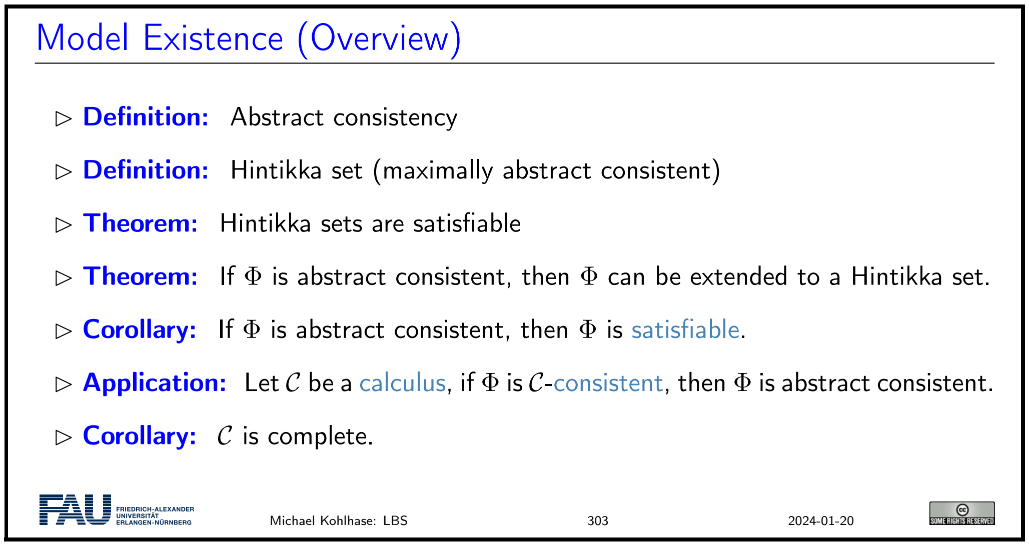
A.2 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the "abstract consistency"/"model existence" method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan's Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization "Smullyan's Unifying Principle".
The basic intuition for this method is the following: typically, a logical system $\mathcal{L} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus $\mathcal{C}$ for $\mathcal{S}$ typically comes in two parts: one analyzes $\mathcal{C}$-consistency (sets that cannot be refuted in $\mathcal{C}$), and the other construct $\mathcal{K}$-models for $\mathcal{C}$-consistent sets.

In this situtation the "abstract consistency"/"model existence" method encapsulates the model construction process into a meta-theorem: the "model existence" theorem. This provides a set of syntactic ("abstract consistency") conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that $\mathcal{C}$-consistency is an abstract consistency property (a purely syntactic task that can be done by a $\mathcal{C}$-proof transformation argument) to obtain a completeness result for $\mathcal{C}$.



Model Existence (Overview)

- ▷ **Definition:** Abstract consistency
- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable.
- ▷ **Application:** Let $\mathcal{C}$ be a calculus, if Φ is $\mathcal{C}$-consistent, then Φ is abstract consistent.
- ▷ **Corollary:** $\mathcal{C}$ is complete.

Michael Kohlhase: LBS 303 2024-01-20

The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus $\mathcal{C}$ and every $\mathcal{C}$-consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.
To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.
Consistency and refutability are very important notions when studying the completeness for

calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let $\mathcal{C}$ be a calculus,...
- ▷ **Definition A.2.1.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-, if there is a refutation, i.e. a derivation of a contradiction from Φ. The act of finding a refutation for Φ is called refuting Φ.
- ▷ **Definition A.2.2.** We call a pair of formulae $\mathbf{A}$ and $\neg\mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ canderive a contradiction from it.
- ▷ **Definition A.2.3.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition A.2.4.** We call a calculus $\mathcal{C}$ reasonable, iff implication elimination and conjunction introduction are admissible in $\mathcal{C}$ and $\mathbf{A} \wedge \neg\mathbf{A} \Rightarrow \mathbf{B}$ is a $\mathcal{C}$-theorem.
- ▷ **Theorem A.2.5.** *$\mathcal{C}$-inconsistency and $\mathcal{C}$-refutability coincide for reasonable calculi.*

It is very important to distinguish the syntactic $\mathcal{C}$-refutability and $\mathcal{C}$-consistency from satisfiability, which is a property of formulae that is at the heart of semantics. Note that the former have the calculus (a syntactic device) as a parameter, while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\langle\mathcal{L}, \mathcal{K}, \models\rangle$ is the current logical system.

Even the word "contradiction" has a syntactical flavor to it, it translates to "saying against each other" from its Latin root.

Abstract Consistency

- ▷ **Definition A.2.6.** Let ∇ be a collection of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇.
- ▷ **Definition A.2.7 (Notation).** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.
- ▷ **Definition A.2.8.** A collection ∇ of sets of propositional formulae is called an abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$

  ∇_c) $P \notin \Phi$ or $\neg P \notin \Phi$ for $P \in \mathcal{V}_0$

  $\nabla_\neg$) $\neg\neg\mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$

  $\nabla_\vee$) $\mathbf{A} \vee \mathbf{B} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$ or $\Phi * \mathbf{B} \in \nabla$

  $\nabla_\wedge$) $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \nabla$
- ▷ **Example A.2.9.** The empty set is an abstract consistency class
- ▷ **Example A.2.10.** The set $\{\emptyset, \{Q\}, \{P \vee Q\}, \{P \vee Q, Q\}\}$ is an abstract consistency class
- ▷ **Example A.2.11.** The family of satisfiable sets is an abstract consistency class.

So a family of sets (we call it a family, so that we do not have to say "set of sets" and we can distinguish the levels) is an abstract consistency class, iff it fulfills five simple conditions, of which the last three are closure conditions.

Think of an abstract consistency class as a family of "consistent" sets (e.g. $\mathcal{C}$-consistent for some calculus $\mathcal{C}$), then the properties make perfect sense: They are naturally closed under subsets — if we cannot derive a contradiction from a large set, we certainly cannot from a subset, furthermore,

∇_c) If both $P\in\Phi$ and $\neg P\in\Phi$, then Φ cannot be "consistent".

$\nabla_\neg$) If we cannot derive a contradiction from Φ with $\neg\neg\mathbf{A}\in\Phi$ then we cannot from $\Phi*\mathbf{A}$, since they are logically equivalent.

The other two conditions are motivated similarly. We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

▷ **Definition A.2.12.** We call a collection ∇ of sets compact, iff for any set Φ we have
$\Phi\in\nabla$, iff $\Psi\in\nabla$ for every finite subset Ψ of Φ.

▷ **Lemma A.2.13.** *If ∇ is compact, then ∇ is closed under subsets.*

▷ *Proof:*

1. Suppose $S\subseteq T$ and $T\in\nabla$.
2. Every finite subset A of S is a finite subset of T.
3. As ∇ is compact, we know that $A\in\nabla$.
4. Thus $S\in\nabla$.

The property of being closed under subsets is a "downwards-oriented" property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an "upwards-oriented" property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a collection ∇ by testing all their finite subsets (which is much simpler).

Compact Abstract Consistency Classes

▷ **Lemma A.2.14.** *Any abstract consistency class can be extended to a compact one.*

▷ *Proof:*

1. We choose $\nabla':=\{\Phi\subseteq wff_0(\mathcal{V}_0)|$every finite subset of Φ is in $\nabla\}$.
2. Now suppose that $\Phi\in\nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi\in\nabla'$. Hence $\nabla\subseteq\nabla'$.
3. Next let us show that each ∇ is compact.'
   3.1. Suppose $\Phi\in\nabla'$ and Ψ is an arbitrary finite subset of Φ.
   3.2. By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi\in\nabla'$.

3.3. Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.
3.4. On the other hand, suppose all finite subsets of Φ are in ∇'.
3.5. Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi\in\nabla'$. Thus ∇' is compact.
4. Note that ∇' is closed under subsets by the Lemma above.
5. Now we show that if ∇ satisfies ∇_*, then ∇ satisfies ∇_*.'
5.1. To show ∇_c, let $\Phi\in\nabla'$ and suppose there is an atom $\mathbf{A}$, such that $\{\mathbf{A},\neg\mathbf{A}\}\subseteq\Phi$. Then $\{\mathbf{A},\neg\mathbf{A}\}\in\nabla$ contradicting ∇_c.
5.2. To show $\nabla_\neg$, let $\Phi\in\nabla'$ and $\neg\neg\mathbf{A}\in\Phi$, then $\Phi*\mathbf{A}\in\nabla'$.
5.2.1. Let Ψ be any finite subset of $\Phi*\mathbf{A}$, and $\Theta:=(\Psi\backslash\{\mathbf{A}\})*\neg\neg\mathbf{A}$.
5.2.2. Θ is a finite subset of Φ, so $\Theta\in\nabla$.
5.2.3. Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A}\in\Theta$, we get $\Theta*\mathbf{A}\in\nabla$ by $\nabla_\neg$.
5.2.4. We know that $\Psi\subseteq\Theta*\mathbf{A}$ and ∇ is closed under subsets, so $\Psi\in\nabla$.
5.2.5. Thus every finite subset Ψ of $\Phi*\mathbf{A}$ is in ∇ and therefore by definition $\Phi*\mathbf{A}\in\nabla'$.
5.3. the other cases are analogous to $\nabla_\neg$.

Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇-Hintikka Set

▷ **Definition A.2.15.** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H}\in\nabla$ a ∇ Hintikka Set, iff $\mathcal{H}$ is maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H}*\mathbf{A}\in\nabla$ we already have $\mathbf{A}\in\mathcal{H}$.

▷ **Theorem A.2.16 (Hintikka Properties).** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

$\mathcal{H}_c$) *For all $\mathbf{A}\in wff_0(\mathcal{V}_0)$ we have $\mathbf{A}\notin\mathcal{H}$ or $\neg\mathbf{A}\notin\mathcal{H}$*

$\mathcal{H}_\neg$) *If $\neg\neg\mathbf{A}\in\mathcal{H}$ then $\mathbf{A}\in\mathcal{H}$*

$\mathcal{H}_\vee$) *If $\mathbf{A}\vee\mathbf{B}\in\mathcal{H}$ then $\mathbf{A}\in\mathcal{H}$ or $\mathbf{B}\in\mathcal{H}$*

$\mathcal{H}_\wedge$) *If $\neg(\mathbf{A}\vee\mathbf{B})\in\mathcal{H}$ then $\neg\mathbf{A},\neg\mathbf{B}\in\mathcal{H}$*

∇-Hintikka Set

▷ *Proof:*

We prove the properties in turn
1. $\mathcal{H}_c$ *by induction on the structure of* $\mathbf{A}$
1.1. $\mathbf{A}\in\mathcal{V}_0$ Then $\mathbf{A}\notin\mathcal{H}$ or $\neg\mathbf{A}\notin\mathcal{H}$ by ∇_c.
1.2. $\mathbf{A}=\neg\mathbf{B}$
1.2.1. Let us assume that $\neg\mathbf{B}\in\mathcal{H}$ and $\neg\neg\mathbf{B}\in\mathcal{H}$,
1.2.2. then $\mathcal{H}*\mathbf{B}\in\nabla$ by $\nabla_\neg$, and therefore $\mathbf{B}\in\mathcal{H}$ by maximality.

1.2.3. So both $\mathbf{B}$ and $\neg\mathbf{B}$ are in $\mathcal{H}$, which contradicts the induction hypothesis.

1.3. $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$ *similar to the previous case*

2. We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.

2.1. If $\neg\neg\mathbf{A}\in\mathcal{H}$, then $\mathcal{H}*\mathbf{A}\in\nabla$ by $\nabla_\neg$.

2.2. The maximality of $\mathcal{H}$ now gives us that $\mathbf{A}\in\mathcal{H}$.

Proof sketch: *other $\mathcal{H}_*$ are similar*

The following theorem is one of the main results in the “abstract consistency”/”model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set $\mathcal{H}$ with $\Phi\in\mathcal{H}$.

Extension Theorem

▷ **Theorem A.2.17.** *If ∇ is an abstract consistency class and $\Phi\in\nabla$, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$.*

▷ *Proof:*

1. Wlog. we assume that ∇ is compact (otherwise pass to compact extension)
2. We choose an enumeration $\mathbf{A}_1, \ldots$ of the set $\mathit{wff}_0(\mathcal{V}_0)$
3. and construct a sequence of sets $\mathbf{H}_i$ with $\mathbf{H}_0 := \Phi$ and

$$\mathbf{H}_{n+1} := \begin{cases} \mathbf{H}_n & \text{if } \mathbf{H}_n * \mathbf{A}_n \notin \nabla \\ \mathbf{H}_n * \mathbf{A}_n & \text{if } \mathbf{H}_n * \mathbf{A}_n \in \nabla \end{cases}$$

4. Note that all $\mathbf{H}_i\in\nabla$, choose $\mathcal{H} := \bigcup_{i\in\mathbb{N}} \mathbf{H}_i$
5. $\Psi \subseteq \mathcal{H}$ finite implies there is a $j\in\mathbb{N}$ such that $\Psi \subseteq \mathbf{H}_j$,
6. so $\Psi\in\nabla$ as ∇ is closed under subsets and $\mathcal{H}\in\nabla$ as ∇ is compact.
7. Let $\mathcal{H}*\mathbf{B}\in\nabla$, then there is a $j\in\mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$, so that $\mathbf{B}\in\mathbf{H}_{j+1}$ and $\mathbf{H}_{j+1} \subseteq \mathcal{H}$
8. Thus $\mathcal{H}$ is ∇-maximal

Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for $\mathcal{H}$ is not executed in our original abstract consistency class ∇, but in a suitably extended one to make it compact — the original would not have contained $\mathcal{H}$ in general. Second, the set $\mathcal{H}$ is not unique for Φ, but depends on the choice of the enumeration of $\mathit{wff}_0(\mathcal{V}_0)$. If we pick a different enumeration, we will end up with a different $\mathcal{H}$. Say if $\mathbf{A}$ and $\neg\mathbf{A}$ are both ∇-consistent[7] with Φ, then depending on which one is first in the enumeration $\mathcal{H}$, will contain that one; with all the consequences for subsequent choices in the construction process.

Valuation

▷ **Definition A.2.18.** A function $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is called a valuation, iff

▷ $\nu(\neg\mathbf{A}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{F}$

▷ $\nu(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{T}$ and $\nu(\mathbf{B}) = \mathsf{T}$

[7] EdNote: introduce this above

▷ **Lemma A.2.19.** *If $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation and $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*

▷ *Proof sketch:* $\nu|_{\mathcal{V}_0} : \mathcal{V}_0 \to \mathcal{D}_o$ is a satisfying variable assignment.

▷ **Lemma A.2.20.** *If $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ is a variable assignment, then $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation.*

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Lemma A.2.21 (Hintikka-Lemma).** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ *Proof:*

1. We define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$
2. then ν is a valuation by the Hintikka properties
3. and thus $\nu|_{\mathcal{V}_0}$ is a satisfying assignment.

▷ **Theorem A.2.22 (Model Existence).** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ 1. There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
2. We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)
3. In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

A.3 A Completeness Proof for Propositional Tableaux

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that Tableaux-consistency is an abstract consistency property.
We encapsulate all of the technical difficulties of the problem in a technical Lemma. From that, the completeness proof is just an application of the high-level theorems we have just proven.

Abstract Completeness for $\mathcal{T}_0$

▷ **Lemma A.3.1.** $\{\Phi | \Phi^{\mathsf{T}}$ *has no closed tableau*$\}$ *is an abstract consistency class.*

▷ *Proof:* Let's call the set above ∇

We have to convince ourselves of the abstract consistency properties

1. $\nabla_c P, \neg P \in \Phi$ *implies* $P^{\mathsf{F}}, P^{\mathsf{T}} \in \Phi^{\mathsf{T}}$.
2. $\nabla_\neg$ *Let* $\neg\neg\mathbf{A} \in \Phi$.
   2.1. For the proof of the contrapositive we assume that $\Phi * \mathbf{A}$ has a closed tableau $\mathcal{T}$ and show that already Φ has one:
   2.2. applying each of $\mathcal{T}_0\neg^{\mathsf{T}}$ and $\mathcal{T}_0\neg^{\mathsf{F}}$ once allows to extend any tableau with $\neg\neg\mathbf{B}^\alpha$ by $\mathbf{B}^\alpha$.

2.3. any path in $\mathcal{T}$ that is closed with $\neg\neg\mathbf{A}^{\alpha}$, can be closed by $\mathbf{A}^{\alpha}$.

3. $\nabla_{\vee}$ *Suppose* $\mathbf{A}\vee\mathbf{B}\in\Phi$ *and both* $\Phi{\ast}\mathbf{A}$ *and* $\Phi{\ast}\mathbf{B}$ *have closed tableaux*
   3.1. consider the tableaux:

$$\begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{A}^{\mathsf{T}}\\ Rest^1\end{array} \qquad \begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\\ Rest^2\end{array} \qquad \begin{array}{c}\Psi^{\mathsf{T}}\\ (\mathbf{A}\vee\mathbf{B})^{\mathsf{T}}\\ \begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}}\\ Rest^1 & Rest^2\end{array}\end{array}$$

4. $\nabla_{\wedge}$ *suppose,* $\neg(\mathbf{A}\vee\mathbf{B})\in\Phi$ *and* $\Phi\{\neg\mathbf{A},\neg\mathbf{B}\}$ *have closed tableau* $\mathcal{T}$.
   4.1. We consider

$$\begin{array}{c}\Phi^{\mathsf{T}}\\ \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\\ Rest\end{array} \qquad \begin{array}{c}\Psi^{\mathsf{T}}\\ (\mathbf{A}\vee\mathbf{B})^{\mathsf{F}}\\ \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\\ Rest\end{array}$$

   where $\Phi=\Psi{\ast}\neg(\mathbf{A}\vee\mathbf{B})$.

Observation: If we look at the completeness proof below, we see that the Lemma above is the only place where we had to deal with specific properties of the $\mathcal{T}_0$.

So if we want to prove completeness of any other calculus with respect to propositional logic, then we only need to prove an analogon to this lemma and can use the rest of the machinery we have already established "off the shelf".

This is one great advantage of the "abstract consistency method"; the other is that the method can be extended transparently to other logics.

Completeness of $\mathcal{T}_0$

▷ **Corollary A.3.2.** *$\mathcal{T}_0$ is complete.*

▷ *Proof:* by contradiction

1. We assume that $\mathbf{A}\in wf\!f_0(\mathcal{V}_0)$ is valid, but there is no closed tableau for $\mathbf{A}^{\mathsf{F}}$.
2. We have $\{\neg\mathbf{A}\}\in\nabla$ as $\neg\mathbf{A}^{\mathsf{T}}=\mathbf{A}^{\mathsf{F}}$.
3. so $\neg\mathbf{A}$ is satisfiable by the model existence theorem (which is applicable as ∇ is an abstract consistency class by our Lemma above)
4. this contradicts our assumption that $\mathbf{A}$ is valid.

Appendix B

First-Order Unification

We will now look into the problem of finding a substitution σ that make two terms equal (we say it unifies them) in more detail. The presentation of the unification algorithm we give here "transformation-based" this has been a very influential way to treat certain algorithms in theoretical computer science.

A transformation-based view of algorithms: The "transformation-based" view of algorithms divides two concerns in presenting and reasoning about algorithms according to Kowalski's slogan [Kow97]

algorithm = logic + control

The computational paradigm highlighted by this quote is that (many) algorithms can be thought of as manipulating representations of the problem at hand and transforming them into a form that makes it simple to read off solutions. Given this, we can simplify thinking and reasoning about such algorithms by separating out their "logical" part, which deals with is concerned with how the problem representations can be manipulated in principle from the "control" part, which is concerned with questions about when to apply which transformations.

It turns out that many questions about the algorithms can already be answered on the "logic" level, and that the "logical" analysis of the algorithm can already give strong hints as to how to optimize control.

In fact we will only concern ourselves with the "logical" analysis of unification here.

The first step towards a theory of unification is to take a closer look at the problem itself. A first set of examples show that we have multiple solutions to the problem of finding substitutions that make two terms equal. But we also see that these are related in a systematic way.

Unification (Definitions)

- ▷ **Definition B.0.1.** For given terms $\mathbf{A}$ and $\mathbf{B}$, unification is the problem of finding a substitution σ, such that $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$.
- ▷ **Notation:** We write term pairs as $\mathbf{A}=^?\mathbf{B}$ e.g. $f(X)=^?f(g(Y))$.
- ▷ **Definition B.0.2.** Solutions (e.g. $[g(a)/X],[a/Y]$, $[g(g(a))/X],[g(a)/Y]$, or $[g(Z)/X],[Z/Y]$) are called unifiers, $\mathbf{U}(\mathbf{A}=^?\mathbf{B}):=\{\sigma|\sigma(\mathbf{A}) = \sigma(\mathbf{B})\}$.
- ▷ **Idea:** Find representatives in $\mathbf{U}(\mathbf{A}=^?\mathbf{B})$, that generate the set of solutions.
- ▷ **Definition B.0.3.** Let σ and θ be substitutions and $W \subseteq \mathcal{V}_\iota$, we say that a substitution σ is more general than θ (on W; write $\sigma\leq\theta[W]$), iff there is a substitution ρ, such that $\theta=(\rho\circ\sigma)[W]$, where $\sigma=\rho[W]$, iff $\sigma(X) = \rho(X)$ for all $X\in W$.

▷ **Definition B.0.4.** σ is called a most general unifier (mgu) of $\mathbf{A}$ and $\mathbf{B}$, iff it is minimal in $\mathbf{U}(\mathbf{A}=^?\mathbf{B})$ wrt. $\leq[(\mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathbf{B}))]$.

The idea behind a most general unifier is that all other unifiers can be obtained from it by (further) instantiation. In an automated theorem proving setting, this means that using most general unifiers is the least committed choice — any other choice of unifiers (that would be necessary for completeness) can later be obtained by other substitutions.
Note that there is a subtlety in the definition of the ordering on substitutions: we only compare on a subset of the variables. The reason for this is that we have defined substitutions to be total on (the infinite set of) variables for flexibility, but in the applications (see the definition of most general unifiers), we are only interested in a subset of variables: the ones that occur in the initial problem formulation. Intuitively, we do not care what the unifiers do off that set. If we did not have the restriction to the set W of variables, the ordering relation on substitutions would become much too fine-grained to be useful (i.e. to guarantee unique most general unifiers in our case).

Now that we have defined the problem, we can turn to the unification algorithm itself. We will define it in a way that is very similar to logic programming: we first define a calculus that generates "solved forms" (formulae from which we can read off the solution) and reason about control later. In this case we will reason that control does not matter.

Unification Problems ($\widehat{=}$ Equational Systems)

▷ **Idea:** Unification is equation solving.

▷ **Definition B.0.5.** We call a formula $\mathbf{A}^1=^?\mathbf{B}^1\wedge\ldots\wedge\mathbf{A}^n=^?\mathbf{B}^n$ an unification problem iff $\mathbf{A}^i,\mathbf{B}^i\in\mathit{wff}_\iota(\Sigma_\iota,\mathcal{V}_\iota)$.

▷ **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).

▷ **Definition B.0.6.** A substitution is called a unifier for a unification problem $\mathcal{E}$ (and thus $\mathcal{D}$ unifiable), iff it is a (simultaneous) unifier for all pairs in $\mathcal{E}$.

In principle, unification problems are sets of equations, which we write as conjunctions, since all of them have to be solved for finding a unifier. Note that it is not a problem for the "logical view" that the representation as conjunctions induces an order, since we know that conjunction is associative, commutative and idempotent, i.e. that conjuncts do not have an intrinsic order or multiplicity, if we consider two equational problems as equal, if they are equivalent as propositional formulae. In the same way, we will abstract from the order in equations, since we know that the equality relation is symmetric. Of course we would have to deal with this somehow in the implementation (typically, we would implement equational problems as lists of pairs), but that belongs into the "control" aspect of the algorithm, which we are abstracting from at the moment.

Solved forms and Most General Unifiers

▷ **Definition B.0.7.** We call a pair $\mathbf{A}=^?\mathbf{B}$ solved in a unification problem $\mathcal{E}$, iff $\mathbf{A}=X$, $\mathcal{E}=X=^?\mathbf{A}\wedge\mathcal{E}$, and $X\notin(\mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathcal{E}))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.

- **Lemma B.0.8.** *Solved forms are of the form $X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ where the X^i are distinct and $X^i \notin \mathrm{free}(\mathbf{B}^j)$.*
- **Definition B.0.9.** Any substitution $\sigma = [\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma := (X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n)$.
- **Lemma B.0.10.** *If $\mathcal{E} = X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ is a solved form, then $\mathcal{E}$ has the unique most general unifier $\sigma_\mathcal{E} := [\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$.*
- *Proof:* Let $\theta \in \mathbf{U}(\mathcal{E})$
  1. then $\theta(X^i) = \theta(\mathbf{B}^i) = \theta \circ \sigma_\mathcal{E}(X^i)$
  2. and thus $\theta = (\theta \circ \sigma_\mathcal{E})[\mathrm{supp}(\sigma)]$.
- **Note:** We can rename the introduced variables in most general unifiers!

It is essential to our "logical" analysis of the unification algorithm that we arrive at unification problems whose unifiers we can read off easily. Solved forms serve that need perfectly as Lemma B.0.10 shows.
Given the idea that unification problems can be expressed as formulae, we can express the algorithm in three simple rules that transform unification problems into solved forms (or unsolvable ones).

Unification Algorithm

- **Definition B.0.11.** The inference system $\mathcal{U}$ consists of the following rules:

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1,\ldots,\mathbf{A}^n) =^? f(\mathbf{B}^1,\ldots,\mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 =^? \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^? \mathbf{B}^n}\mathcal{U}\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathbf{A} =^? \mathbf{A}}{\mathcal{E}}\mathcal{U}\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^? \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad X \in \mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =^? \mathbf{A}}\mathcal{U}\mathrm{elim}$$

- **Lemma B.0.12.** *$\mathcal{U}$ is correct: $\mathcal{E} \vdash_\mathcal{U} \mathcal{F}$ implies $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$.*
- **Lemma B.0.13.** *$\mathcal{U}$ is complete: $\mathcal{E} \vdash_\mathcal{U} \mathcal{F}$ implies $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$.*
- **Lemma B.0.14.** *$\mathcal{U}$ is confluent: the order of derivations does not matter.*
- **Corollary B.0.15.** *First-order unification is unitary: i.e. most general unifiers are unique up to renaming of introduced variables.*
- *Proof sketch:* $\mathcal{U}$ is trivially branching.

The decomposition rule $\mathcal{U}$dec is completely straightforward, but note that it transforms one unification pair into multiple argument pairs; this is the reason, why we have to directly use unification problems with multiple pairs in $\mathcal{U}$.

Note furthermore, that we could have restricted the $\mathcal{U}$triv rule to variable-variable pairs, since for any other pair, we can decompose until only variables are left. Here we observe, that constant-constant pairs can be decomposed with the $\mathcal{U}$dec rule in the somewhat degenerate case without arguments.

Finally, we observe that the first of the two variable conditions in $\mathcal{U}$elim (the "occurs-in-check") makes sure that we only apply the transformation to unifiable unification problems, whereas the second one is a termination condition that prevents the rule to be applied twice.

The notion of completeness and correctness is a bit different than that for calculi that we compare to the entailment relation. We can think of the "logical system of unifiability" with the model class of sets of substitutions, where a set satisfies an equational problem $\mathcal{E}$, iff all of its members are unifiers. This view induces the soundness and completeness notions presented above.
The three meta-properties above are relatively trivial, but somewhat tedious to prove, so we leave the proofs as an exercise to the reader.
We now fortify our intuition about the unification calculus by two examples. Note that we only need to pursue one possible $\mathcal{U}$ derivation since we have confluence.

Unification Examples

▷ **Example B.0.16.** Two similar unification problems:

$\begin{array}{ll} f(g(X,X),h(a))=^? f(g(a,Z),h(Z)) & \\ \hline g(X,X)=^? g(a,Z) \wedge h(a)=^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X=^? a \wedge X=^? Z \wedge h(a)=^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X=^? a \wedge X=^? Z \wedge a=^? Z & \mathcal{U}\mathrm{dec} \\ \hline X=^? a \wedge a=^? Z \wedge a=^? Z & \mathcal{U}\mathrm{elim} \\ \hline X=^? a \wedge Z=^? a \wedge a=^? a & \mathcal{U}\mathrm{elim} \\ \hline X=^? a \wedge Z=^? a & \mathcal{U}\mathrm{triv} \end{array}$	$\begin{array}{ll} f(g(X,X),h(a))=^? f(g(b,Z),h(Z)) & \\ \hline g(X,X)=^? g(b,Z) \wedge h(a)=^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X=^? b \wedge X=^? Z \wedge h(a)=^? h(Z) & \mathcal{U}\mathrm{dec} \\ \hline X=^? b \wedge X=^? Z \wedge a=^? Z & \mathcal{U}\mathrm{dec} \\ \hline X=^? b \wedge b=^? Z \wedge a=^? Z & \mathcal{U}\mathrm{elim} \\ \hline X=^? b \wedge Z=^? b \wedge a=^? b & \mathcal{U}\mathrm{elim} \end{array}$
MGU: $[a/X], [a/Z]$	$a=^? b$ not unifiable

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG Michael Kohlhase: LBS 321 2024-01-20

We will now convince ourselves that there cannot be any infinite sequences of transformations in $\mathcal{U}$. Termination is an important property for an algorithm.

The proof we present here is very typical for termination proofs. We map unification problems into a partially ordered set $\langle S, \prec\rangle$ where we know that there cannot be any infinitely descending sequences (we think of this as measuring the unification problems). Then we show that all transformations in $\mathcal{U}$ strictly decrease the measure of the unification problems and argue that if there were an infinite transformation in $\mathcal{U}$, then there would be an infinite descending chain in S, which contradicts our choice of $\langle S, \prec\rangle$.

The crucial step in coming up with such proofs is finding the right partially ordered set. Fortunately, there are some tools we can make use of. We know that $\langle \mathbb{N}, <\rangle$ is terminating, and there are some ways of lifting component orderings to complex structures. For instance it is well-known that the lexicographic ordering lifts a terminating ordering to a terminating ordering on finite dimensional Cartesian spaces. We show a similar, but less known construction with multisets for our proof.

Unification (Termination)

▷ **Definition B.0.17.** Let S and T be multisets and $\leq$ a partial ordering on $S \cup T$. Then we define $S \prec^m S$, iff $S = C \uplus T'$ and $T = C \uplus \{t\}$, where $s \leq t$ for all $s \in S'$.

We call $\leq^m$ the multiset ordering induced by $\leq$.

▷ **Definition B.0.18.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X=^?\mathbf{A}$.

▷ **Lemma B.0.19.** *If $\prec$ is linear/terminating on S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*

▷ **Lemma B.0.20.** *$\mathcal{U}$ is terminating.* *(any $\mathcal{U}$-derivation is finite)*

▷ *Proof:* We prove termination by mapping $\mathcal{U}$ transformation into a Noetherian space.

1. Let $\mu(\mathcal{E}):=\langle n,\mathcal{N}\rangle$, where
   - ▷ n is the number of unsolved variables in $\mathcal{E}$
   - ▷ $\mathcal{N}$ is the multiset of term depths in $\mathcal{E}$
2. The lexicographic order $\prec$ on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.
   - 2.1. $\mathcal{U}$dec and $\mathcal{U}$triv decrease the multiset of term depths without increasing the unsolved variables.
   - 2.2. $\mathcal{U}$elim decreases the number of unsolved variables (by one), but may increase term depths.

But it is very simple to create terminating calculi, e.g. by having no inference rules. So there is one more step to go to turn the termination result into a decidability result: we must make sure that we have enough inference rules so that any unification problem is transformed into solved form if it is unifiable.

First-Order Unification is Decidable

▷ **Definition B.0.21.** We call an equational problem $\mathcal{E}$ $\mathcal{U}$-reducible, iff there is a $\mathcal{U}$-step $\mathcal{E}\vdash_{\mathcal{U}}\mathcal{F}$ from $\mathcal{E}$.

▷ **Lemma B.0.22.** *If $\mathcal{E}$ is unifiable but not solved, then it is $\mathcal{U}$-reducible.*

▷ *Proof:* We assume that $\mathcal{E}$ is unifiable but unsolved and show the $\mathcal{U}$ rule that applies.

1. There is an unsolved pair $\mathbf{A}=^?\mathbf{B}$ in $\mathcal{E} = \mathcal{E} \wedge \mathbf{A}=^?\mathbf{B}'$.

   we have two cases
2. $\mathbf{A},\mathbf{B}\notin\mathcal{V}_\iota$
   - 2.1. then $\mathbf{A} = f(\mathbf{A}^1 \ldots \mathbf{A}^n)$ and $\mathbf{B} = f(\mathbf{B}^1 \ldots \mathbf{B}^n)$, and thus $\mathcal{U}$dec is applicable
3. $\mathbf{A} = X\in\text{free}(\mathcal{E})$
   - 3.1. then $\mathcal{U}$elim (if $\mathbf{B} \neq X$) or $\mathcal{U}$triv (if $\mathbf{B} = X$) is applicable.

▷ **Corollary B.0.23.** *First-order unification is decidable in PL^1.*

Proof:

▷ 1. $\mathcal{U}$-irreducible unification problems can be reached in finite time by Lemma B.0.20.
2. They are either solved or unsolvable by Lemma B.0.22, so they provide the answer.

Appendix C

Soundness and Completeness of First-Order Tableaux

For the soundness result, we recap the definition of soundness for test calculi from the propositional case.

Soundness (Tableau)

▷ **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.

▷ **Definition C.0.1.** A labeled formula $\mathbf{A}^\alpha$ is valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$.

▷ **Definition C.0.2.** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae on $\mathcal{P}$ is satisfiable.

▷ **Lemma C.0.3.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*

▷ **Theorem C.0.4 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wff_0(\mathcal{V}_0)$ valid, if there is a closed tableau $\mathcal{T}$ for Φ^F.*

▷ *Proof:* by contradiction

1. Suppose Φ isfalsifiable $\widehat{=}$ not valid.
2. Then the initial tableau is satisfiable, (Φ^F satisfiable)
3. so $\mathcal{T}$ is satisfiable, by Lemma C.0.3.
4. Thus there is a satisfiable branch (by definition)
5. but all branches are closed ($\mathcal{T}$ closed)

▷ **Theorem C.0.5 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wff_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^F.*

Proof sketch: Proof difficult/interesting; see Corollary A.3.2

Thus we only have to prove Lemma C.0.3, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ and is satisfiable, then it must have a satisfiable branch. If $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$ for some variable assignment φ. Thus $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ and $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$, so after the extension (which adds the formulae $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}^\mathsf{T}$ to the branch), the branch is still satisfiable. The cases for the other rules are similar. The soundness of the first-order free-variable tableaux calculus can be established a simple induction over the size of the tableau.

▷ Soundness of $\mathcal{T}_1^f$

▷ **Lemma C.0.6.** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ *Proof:*

we examine the tableau rules in turn

1. propositional rules *as in propositional tableaux*
2. $\mathcal{T}_1^f\exists$ *by* **??**
3. $\mathcal{T}_1^f\bot$ *by Lemma 8.1.28 (substitution value lemma)*
4. $\mathcal{T}_1^f\forall$
   - 4.1. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for all $a\in\mathcal{D}_\iota$
   - 4.2. so in particular for some $a\in\mathcal{D}_\iota \neq \emptyset$.

▷ **Corollary C.0.7.** $\mathcal{T}_1^f$ *is correct.*

The only interesting steps are the cut rule, which can be directly handled by the substitution value lemma, and the rule for the existential quantifier, which we do in a separate lemma.

Soundness of $\mathcal{T}_1^f\exists$

▷ **Lemma C.0.8.** $\mathcal{T}_1^f\exists$ *transforms satisfiable tableaux into satisfiable ones.*

▷ *Proof:* Let $\mathcal{T}'$ be obtained by applying $\mathcal{T}_1^f\exists$ to $(\forall X.\mathbf{A})^\mathsf{F}$ in $\mathcal{T}$, extending it with $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^\mathsf{F}$, where $W{:=}\mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\}$

1. Let $\mathcal{T}$ be satisfiable in $\mathcal{M}{:=}\langle\mathcal{D},\mathcal{I}\rangle$, then $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{F}$.
   We need to find a model $\mathcal{M}'$ that satisfies $\mathcal{T}'$ (*find interpretation for f*)
2. By definition $\mathcal{I}_{(\varphi,[a/X])}(\mathbf{A}) = \mathsf{F}$ for some $a\in\mathcal{D}$ (depends on $\varphi|_W$)
3. Let $g\colon \mathcal{D}^k\to\mathcal{D}$ be defined by $g(a_1,\ldots,a_k){:=}a$, if $\varphi(X^i) = a_i$
4. choose $\mathcal{M} = \langle\mathcal{D},\mathcal{I}'\rangle'$ with $\mathcal{I}'{:=}\mathcal{I},[g/f]$, then by subst. value lemma

$$
\begin{aligned}
\mathcal{I}'_\varphi([f(X^1,\ldots,X^k)/X](\mathbf{A})) &= \mathcal{I}'_{(\varphi,[\mathcal{I}'_\varphi(f(X^1,\ldots,X^k))/X])}(\mathbf{A})\\
&= \mathcal{I}'_{(\varphi,[a/X])}(\mathbf{A}) = \mathsf{F}
\end{aligned}
$$

5. So $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^\mathsf{F}$ satisfiable in $\mathcal{M}'$

This proof is paradigmatic for soundness proofs for calculi with Skolemization. We use the axiom of choice at the meta-level to choose a meaning for the Skolem function symbol. Armed with the Model Existence Theorem for first-order logic (**??**), the completeness of first-order tableaux is similarly straightforward. We just have to show that the collection of tableau-irrefutable sentences is an abstract consistency class, which is a simple proof-transformation exercise in all but the universal quantifier case, which we postpone to its own Lemma (Theorem C.0.10).

Completeness of $(\mathcal{T}_1^f)$

▷ **Theorem C.0.9.** *$\mathcal{T}_1^f$ is refutation complete.*

▷ *Proof:* We show that $\nabla{:=}\{\Phi|\Phi^{\mathsf{T}}$ has no closed Tableau$\}$ is an abstract consistency class

1. as for propositional case.
2. by the lifting lemma below
3. Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A})\in\Phi$ and $\Phi^{\mathsf{T}}*([c/X](\mathbf{A}))^{\mathsf{F}}\in\nabla$.

$$\begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([c/X](\mathbf{A}))^{\mathsf{F}} \\ Rest \end{array} \qquad\qquad \begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([f(X_1,\ldots,X_k)/X](\mathbf{A}))^{\mathsf{F}} \\ [f(X_1,\ldots,X_k)/c](Rest) \end{array}$$

So we only have to treat the case for the universal quantifier. This is what we usually call a “lifting argument”, since we have to transform (“lift”) a proof for a formula $\theta(\mathbf{A})$ to one for $\mathbf{A}$. In the case of tableaux we do that by an induction on the tableau refutation for $\theta(\mathbf{A})$ which creates a tableau-isomorphism to a tableau refutation for $\mathbf{A}$.

Tableau-Lifting

▷ **Theorem C.0.10.** *If $\mathcal{T}_\theta$ is a closed tableau for a set $\theta(\Phi)$ of formulae, then there is a closed tableau $\mathcal{T}$ for Φ.*

▷ *Proof:* by induction over the structure of $\mathcal{T}_\theta$ we build an isomorphic tableau $\mathcal{T}$, and a tableau-isomorphism $\omega\colon\mathcal{T}\to\mathcal{T}_\theta$, such that $\omega(\mathbf{A})=\theta(\mathbf{A})$.

only the tableau-substitution rule is interesting.

1. Let $(\theta(\mathbf{A}_i))^{\mathsf{T}}$ and $(\theta(\mathbf{B}_i))^{\mathsf{F}}$ cut formulae in the branch Θ_θ^i of $\mathcal{T}_\theta$
2. there is a joint unifier σ of $(\theta(\mathbf{A}_1))=^?(\theta(\mathbf{B}_1))\wedge\ldots\wedge(\theta(\mathbf{A}_n))=^?(\theta(\mathbf{B}_n))$
3. thus $\sigma\circ\theta$ is a unifier of $\mathbf{A}$ and $\mathbf{B}$
4. hence there is a most general unifier ρ of $\mathbf{A}_1=^?\mathbf{B}_1\wedge\ldots\wedge\mathbf{A}_n=^?\mathbf{B}_n$
5. so Θ is closed.

Again, the “lifting lemma for tableaux” is paradigmatic for lifting lemmata for other refutation calculi.

Appendix D

Properties of the Simply Typed λ Calculus

D.1 Computational Properties of λ-Calculus

As we have seen above, the main contribution of the λ-calculus is that it casts the comprehension and (functional) extensionality axioms in a way that is more amenable to automation in reasoning systems, since they can be oriented into a confluent and terminating reduction system. In this section we prove the respective properties. We start out with termination, since we will need it later in the proof of confluence.

D.1.1 Termination of β-reduction

We will use the termination of $=_\beta$ reduction to present a very powerful proof method, called the "logical relations method", which is one of the basic proof methods in the repertoire of a proof theorist, since it can be extended to many situations, where other proof methods have no chance of succeeding.
Before we start into the termination proof, we convince ourselves that a straightforward induction over the structure of expressions will not work, and we need something more powerful.

Termination of β-Reduction

- only holds for the typed case
  $(\lambda X.XX)\ (\lambda X.XX) \rightarrow_\beta (\lambda X.XX)\ (\lambda X.XX)$
- **Theorem D.1.1 (Typed β-Reduction terminates).** *For all $\mathbf{A} \in wff_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$, the chain of reductions from $\mathbf{A}$ is finite.*
- proof attempts:
  - Induction on the structure $\mathbf{A}$ must fail, since this would also work for the untyped case.
  - Induction on the type of $\mathbf{A}$ must fail, since β-reduction conserves types.
- combined induction on both: Logical Relations [Tait 1967]

The overall shape of the proof is that we reason about two relations: $\mathcal{SR}$ and $\mathcal{LR}$ between λ-terms and their types. The first is the one that we are interested in, $\mathcal{LR}(\mathbf{A},\alpha)$ essentially states the property that $=_{\beta\eta}$ reduction terminates at $\mathbf{A}$. Whenever the proof needs to argue by induction on types it uses the "logical relation" $\mathcal{LR}$, which is more "semantic" in flavor. It coincides with $\mathcal{SR}$ on base types, but is defined via a functionality property.

Relations $\mathcal{SR}$ and $\mathcal{LR}$

▷ **Definition D.1.2.** $\mathbf{A}$ is called strongly reducing at type α (write $\mathcal{SR}(\mathbf{A},\alpha)$), iff each chain β-reductions from $\mathbf{A}$ terminates.

▷ **Definition D.1.3.** We define a logical relation $\mathcal{LR}$ inductively on the structure of the type

- ▷ α base type: $\mathcal{LR}(\mathbf{A},\alpha)$, iff $\mathcal{SR}(\mathbf{A},\alpha)$
- ▷ $\mathcal{LR}(\mathbf{C},\alpha\to\beta)$, iff $\mathcal{LR}(\mathbf{C}\ \mathbf{A},\beta)$ for all $\mathbf{A}\in wff_\alpha(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})$ with $\mathcal{LR}(\mathbf{A},\alpha)$.

▷ *Proof:* Termination Proof

1. $\mathcal{LR}\subseteq\mathcal{SR}$ (?? b))
2. $\mathbf{A}\in wff_\alpha(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})$ implies $\mathcal{LR}(\mathbf{A},\alpha)$ (?? with $\sigma=\emptyset$)
3. thus $\mathcal{SR}(\mathbf{A},\alpha)$.

▷ **Lemma D.1.4 ($\mathcal{SR}$ is closed under subterms).** *If $\mathcal{SR}(\mathbf{A},\alpha)$ and $\mathbf{B}_\beta$ is a subterm of $\mathbf{A}$, then $\mathcal{SR}(\mathbf{B},\beta)$.*

Proof sketch: Every infinite β reduction from $\mathbf{B}$ would be one from $\mathbf{A}$.

The termination proof proceeds in two steps, the first one shows that $\mathcal{LR}$ is a sub-relation of $\mathcal{SR}$, and the second that $\mathcal{LR}$ is total on λ-terms. Togther they give the termination result.
The next result proves two important technical side results for the termination proofs in a joint induction over the structure of the types involved. The name "rollercoaster lemma" alludes to the fact that the argument starts with base type, where things are simple, and iterates through the two parts each leveraging the proof of the other to higher and higher types.

▷ $\mathcal{LR}\subseteq\mathcal{SR}$ (Rollercoaster Lemma)

▷ **Lemma D.1.5 (Rollercoaster Lemma).**

a) If h is a constant or variable of type $\overline{\alpha_n}\to\alpha$ and $\mathcal{SR}(\mathbf{A}^i,\alpha^i)$, then $\mathcal{LR}(h\ \overline{\mathbf{A}^n},\alpha)$.

b) $\mathcal{LR}(\mathbf{A},\alpha)$ implies $\mathcal{SR}(\mathbf{A},\alpha)$.

▷ *Proof:* we prove both assertions by simultaneous induction on α

1. α base type
   1.1. a)
      1.1.1. $h\ \overline{\mathbf{A}^n}$ is strongly reducing, since the $\mathbf{A}^i$ are (brackets!)
      1.1.2. so $\mathcal{LR}(h\ \overline{\mathbf{A}^n},\alpha)$ as α is a base type ($\mathcal{SR}=\mathcal{LR}$)
   1.2. b)*by definition*
2. $\alpha=\beta\to\gamma$
   2.1. a)

2.1.1. Let $\mathcal{LR}(\mathbf{B},\beta)$.
2.1.2. by IH b) we have $\mathcal{SR}(\mathbf{B},\beta)$, and $\mathcal{LR}((h\ \overline{\mathbf{A}^n})\ \mathbf{B},\gamma)$ by IH a)
2.1.3. so $\mathcal{LR}(h\ \overline{\mathbf{A}^n},\alpha)$ by definition.

2.2. b)
2.2.1. Let $\mathcal{LR}(\mathbf{A},\alpha)$ and $X_\beta \notin \mathrm{free}(\mathbf{A})$.
2.2.2. $\mathcal{LR}(X,\beta)$ by IH a) with $n=0$, thus $\mathcal{LR}(\mathbf{A}\ X,\gamma)$ by definition.
2.2.3. By IH b) we have $\mathcal{SR}(\mathbf{A}X,\gamma)$ and by **??** $\mathcal{SR}(\mathbf{A},\alpha)$.

The part of the rollercoaster lemma we are really interested in is part b). But part a) will become very important for the case where $n=0$; here it states that constants and variables are $\mathcal{LR}$.

The next step in the proof is to show that all well-formed formulae are $\mathcal{LR}$. For that we need to prove closure of $\mathcal{LR}$ under $=_\beta$ expansion

β-Expansion Lemma

▷ **Lemma D.1.6.** *If $\mathcal{LR}([\mathbf{B}/X](\mathbf{A}),\alpha)$ and $\mathcal{LR}(\mathbf{B},\beta)$ for $X_\beta \notin \mathrm{free}(\mathbf{B})$, then $\mathcal{LR}((\lambda X_\alpha.\mathbf{A})\ \mathbf{B},\alpha)$.*

▷ *Proof:*

1. Let $\alpha = \overline{\gamma}_i \to \delta$ where δ base type and $\mathcal{LR}(\mathrm{C}^i,\gamma^i)$
2. It is sufficient to show that $\mathcal{SR}((\lambda X.\mathbf{A})\ \mathbf{B}\ \overline{\mathbf{C}},\delta)$, as δ base type
3. We have $\mathcal{LR}(([\mathbf{B}/X](\mathbf{A}))\ \overline{\mathbf{C}},\delta)$ by hypothesis and definition of $\mathcal{LR}$.
4. thus $\mathcal{SR}(([\mathbf{B}/X](\mathbf{A}))\ \overline{\mathbf{C}},\delta)$, as δ base type.
5. in particular $\mathcal{SR}([\mathbf{B}/X](\mathbf{A}),\alpha)$ and $\mathcal{SR}(\mathrm{C}^i,\gamma^i)$ (subterms)
6. $\mathcal{SR}(\mathbf{B},\beta)$ by hypothesis and **??**
7. So an infinite reduction from $(\lambda X.\mathbf{A})\ \mathbf{B}\ \overline{\mathbf{C}}$ cannot solely consist of redexes from $[\mathbf{B}/X](\mathbf{A})$ and the C^i.
8. so an infinite reduction from $(\lambda X.\mathbf{A})\ \mathbf{B}\ \overline{\mathbf{C}}$ must have the form

$$\begin{aligned}(\lambda X.\mathbf{A})\ \mathbf{B}\ \overline{\mathbf{C}} &\to_\beta^* (\lambda X.\mathbf{A}')\ \mathbf{B}'\ \overline{\mathbf{C}'}\\ &\to_\beta^1 ([\mathbf{B}'/X](\mathbf{A}'))\ \overline{\mathbf{C}'}\\ &\to_\beta^* \ldots\end{aligned}$$

   where $\mathbf{A}\to_\beta^*\mathbf{A}'$, $\mathbf{B}\to_\beta^*\mathbf{B}'$ and $\mathrm{C}^i\to_\beta^*\mathrm{C}^{i'}$
9. so we have $[\mathbf{B}/X](\mathbf{A})\to_\beta^*[\mathbf{B}'/X](\mathbf{A}')$
10. so we have the infinite reduction

$$\begin{aligned}([\mathbf{B}/X](\mathbf{A}))\ \overline{\mathbf{C}} &\to_\beta^* ([\mathbf{B}'/X](\mathbf{A}'))\ \overline{\mathbf{C}'}\\ &\to_\beta^* \ldots\end{aligned}$$

   which contradicts our assumption

▷ **Lemma D.1.7 ($\mathcal{LR}$ is closed under β-expansion).** *If $\mathbf{C}\to_\beta\mathbf{D}$ and $\mathcal{LR}(\mathbf{D},\alpha)$, so is $\mathcal{LR}(\mathbf{C},\alpha)$.*

Note that this Lemma is one of the few places in the termination proof, where we actually look at the properties of β reduction.

We now prove that every well-formed formula is related to its type by $\mathcal{LR}$. But we cannot prove this by a direct induction. In this case we have to strengthen the statement of the theorem – and thus the induction hypothesis, so that we can make the step cases go through. This is common for non-trivial induction proofs. Here we show instead that *every instance* of a well-formed formula is related to its type by $\mathcal{LR}$; we will later only use this result for the cases of the empty substitution, but the stronger assertion allows a direct induction proof.

$\mathbf{A}\in wff_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$ implies $\mathcal{LR}(\mathbf{A}, \alpha)$

▷ **Definition D.1.8.** We write $\mathcal{LR}(\sigma)$ if $\mathcal{LR}(\sigma(X_\alpha), \alpha)$ for all $X\in\mathrm{supp}(\sigma)$.

▷ **Theorem D.1.9.** *If $\mathbf{A}\in wff_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$, then $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$ for any substitution σ with $\mathcal{LR}(\sigma)$.*

▷ *Proof:* by induction on the structure of $\mathbf{A}$

1. $\mathbf{A} = X_\alpha \in \mathrm{supp}(\sigma)$
   1.1. then $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$ by assumption
2. $\mathbf{A} = X \notin \mathrm{supp}(\sigma)$
   2.1. then $\sigma(\mathbf{A}) = \mathbf{A}$ and $\mathcal{LR}(\mathbf{A}, \alpha)$ by **??** with $n = 0$.
3. $\mathbf{A}\in\Sigma_\mathcal{T}$
   3.1. then $\sigma(\mathbf{A}) = \mathbf{A}$ as above
4. $\mathbf{A} = \mathbf{BC}$
   4.1. by IH $\mathcal{LR}(\sigma(\mathbf{B}), \gamma \to \alpha)$ and $\mathcal{LR}(\sigma(\mathbf{C}), \gamma)$
   4.2. so $\mathcal{LR}((\sigma(\mathbf{B}))\ (\sigma(\mathbf{C})), \alpha)$ by definition of $\mathcal{LR}$.
5. $\mathbf{A} = \lambda X_\beta.\mathbf{C}_\gamma$
   5.1. Let $\mathcal{LR}(\mathbf{B}, \beta)$ and $\theta{:=}\sigma,[\mathbf{B}/X]$, then θ meets the conditions of the IH.
   5.2. Moreover $(\sigma(\lambda X_\beta.\mathbf{C}_\gamma))\ \mathbf{B}{\to_\beta}\sigma,[\mathbf{B}/X](\mathbf{C}) = \theta(\mathbf{C})$.
   5.3. Now, $\mathcal{LR}(\theta(\mathbf{C}), \gamma)$ by IH and thus $\mathcal{LR}((\sigma(\mathbf{A}))\ \mathbf{B}, \gamma)$ by **??**.
   5.4. So $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$ by definition of $\mathcal{LR}$.

In contrast to the proof of the roller coaster Lemma above, we prove the assertion here by an induction on the structure of the λ-terms involved. For the base cases, we can directly argue with the first assertion from **??**, and the application case is immediate from the definition of $\mathcal{LR}$. Indeed, we defined the auxiliary relation $\mathcal{LR}$ exclusively that the application case – which cannot be proven by a direct structural induction; remember that we needed induction on types in **??**– becomes easy.

The last case on λ-abstraction reveals why we had to strengthen the induction hypothesis: β reduction introduces a substitution which may increase the size of the subterm, which in turn keeps us from applying the induction hypothesis. Formulating the assertion directly under all possible $\mathcal{LR}$ substitutions unblocks us here.

This was the last result we needed to complete the proof of termiation of $=_\beta$-reduction.

Remark:

If we are only interested in the termination of head reductions, we can get by with a much simpler version of this lemma, that basically relies on the uniqueness of head $=_\beta$ reduction.

Closure under Head β-Expansion (weakly reducing)

▷ **Lemma D.1.10 ($\mathcal{LR}$ is closed under head β-expansion).** *If $\mathbf{C}\to^h_\beta\mathbf{D}$ and $\mathcal{LR}(\mathbf{D}, \alpha)$, so is $\mathcal{LR}(\mathbf{C}, \alpha)$.*

▷ *Proof:* by induction over the structure of α

1. α base type
   1.1. we have $\mathcal{SR}(\mathbf{D},\alpha)$ by definition
   1.2. so $\mathcal{SR}(\mathbf{C},\alpha)$, since head reduction is unique
   1.3. and thus $\mathcal{LR}(\mathbf{C},\alpha)$.
2. $\alpha=\beta\to\gamma$
   2.1. Let $\mathcal{LR}(\mathbf{B},\beta)$, by definition we have $\mathcal{LR}(\mathbf{DB},\gamma)$.
   2.2. but $\mathbf{C}\ \mathbf{B}\to_{\beta}^{h}\mathbf{D}\ \mathbf{B}$, so $\mathcal{LR}(\mathbf{CB},\gamma)$ by IH
   2.3. and $\mathcal{LR}(\mathbf{C},\alpha)$ by definition.

▷ **Note:** This result only holds for weak reduction (any chain of β head reductions terminates) for strong reduction we need a stronger Lemma.

For the termination proof of head $=_{\beta}$-reduction we would just use the same proof as above, just for a variant of $\mathcal{SR}$, where $\mathcal{SR}(\mathbf{A},\alpha)$ that only requires that the head reduction sequence out of $\mathbf{A}$ terminates. Note that almost all of the proof except **??** (which holds by the same argument) is invariant under this change. Indeed Rick Statman uses this observation in [Sta85] to give a set of conditions when logical relations proofs work.

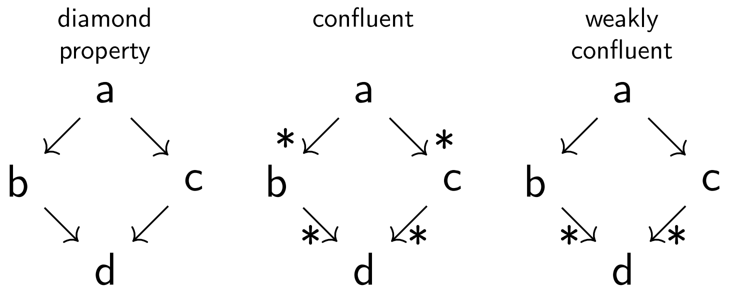
D.1.2 Confluence of $\beta\eta$ Conversion

We now turn to the confluence for $=_{\beta\eta}$, i.e. that the order of reductions is irrelevant. This entails the uniqueness of $=_{\beta\eta}$ normal forms, which is very useful.
Intuitively confluence of a relation R means that “anything that flows apart will come together again.” – and as a consequence normal forms are unique if they exist. But there is more than one way of formalizing that intuition.

Confluence

▷ **Definition D.1.11 (Confluence).** Let $R\subseteq A^2$ be a relation on a set A, then we say that

- ▷ has a diamond property, iff for every $a,b,c\in A$ with $a\to_R^1 b$ $a\to_R^1 c$ there is a $d\in A$ with $b\to_R^1 d$ and $c\to_R^1 d$.
- ▷ is confluent, iff for every $a,b,c\in A$ with $a\to_R^* b$ $a\to_R^* c$ there is a $d\in A$ with $b\to_R^* d$ and $c\to_R^* d$.
- ▷ weakly confluent iff for every $a,b,c\in A$ with $a\to_R^1 b$ $a\to_R^1 c$ there is a $d\in A$ with $b\to_R^* d$ and $c\to_R^* d$.



$$\begin{tikzcd} & a \arrow[dl] \arrow[dr] & \\ b \arrow[dr] & & c \arrow[dl] \\ & d & \end{tikzcd}
\qquad
\begin{tikzcd} & a \arrow[dl, "*"'] \arrow[dr, "*"] & \\ b \arrow[dr, "*"'] & & c \arrow[dl, "*"] \\ & d & \end{tikzcd}
\qquad
\begin{tikzcd} & a \arrow[dl] \arrow[dr] & \\ b \arrow[dr, "*"'] & & c \arrow[dl, "*"] \\ & d & \end{tikzcd}$$

The diamond property is very simple, but not many reduction relations enjoy it. Confluence is the notion that directly gives us unique normal forms, but is difficult to prove via a digram chase, while weak confluence is amenable to this, does not directly give us confluence.
We will now relate the three notions of confluence with each other: the diamond property (sometimes also called strong confluence) is stronger than confluence, which is stronger than weak confluence

Relating the notions of confluence

▷ **Observation D.1.12.** *If a rewrite relation has a diamond property, then it is weakly confluent.*

▷ **Theorem D.1.13.** *If a rewrite relation has a diamond property, then it is confluent.*

▷ *Proof sketch:* by a tiling argument, composing 1×1 diamonds to an $n\times m$ diamond.

▷ **Theorem D.1.14 (Newman's Lemma).** *If a rewrite relation is terminating and weakly confluent, then it is also confluent.*

Note that Newman's Lemma cannot be proven by a tiling argument since we cannot control the growth of the tiles. There is a nifty proof by Gérard Huet [Hue80] that is worth looking at.

After this excursion into the general theory of reduction relations, we come back to the case at hand: showing the confluence of $=_{\beta\eta}$-reduction.
$\rightarrow_\eta^*$ is very well-behaved – i.e. confluent and terminating

η-Reduction ist terminating and confluent

▷ **Lemma D.1.15.** *η-Reduction ist terminating*

▷ *Proof sketch:* by a simple counting argument

▷ **Lemma D.1.16.** *η-reduction is confluent.*

▷ *Proof sketch:* We show that η-reduction has the diamond property by diagram chase over

$$\begin{tikzcd}
 & \lambda X.\mathbf{A}\,X \arrow[dl] \arrow[dr] & \\
\mathbf{A} \arrow[dr] & & \lambda X.\mathbf{A}'\,X \arrow[dl] \\
 & \mathbf{A}' &
\end{tikzcd}$$

where $\mathbf{A}\rightarrow_\eta\mathbf{A}'$. Then the assertion follows by **??**.

For $=_\beta$-reduction the situation is a bit more involved, but a simple diagram chase is still sufficient to prove weak confluence, which gives us confluence via **??**

$=_\beta$ is confluent

▷ **Lemma D.1.17.** *$=_\beta$-Reduction is weakly confluent.*

▷ *Proof sketch:* by diagram chase over

$$\begin{tikzcd}
 & & (\lambda X.\mathbf{A})\,\mathbf{B} \arrow[dll] \arrow[d] \arrow[drr] & & \\
(\lambda X.\mathbf{A}')\,\mathbf{B} \arrow[dr] & & (\lambda X.\mathbf{A})\,\mathbf{B}' \arrow[dl] \arrow[dr] & & {[\mathbf{B}/X](\mathbf{A})} \arrow[dl, "*"] \\
 & (\lambda X.\mathbf{A}')\,\mathbf{B}' \arrow[dr] & & {[\mathbf{B}'/X](\mathbf{A})} \arrow[dl] & \\
 & & {[\mathbf{B}'/X](\mathbf{A}')} & &
\end{tikzcd}$$

▷ **Corollary D.1.18.** *$=_\beta$-Reduction is confluent.*

Proof sketch: by Newman's Lemma.

There is one reduction in the diagram in the proof of **??** which (note that $\mathbf{B}$ can occur multiple times in $[\mathbf{B}/X](\mathbf{A})$) is not necessary single-step. The diamond property is broken by the outer two reductions in the diagram as well.

We have shown that the $=_\beta$ and $=_\eta$ reduction relations are terminating and confluent and terminating individually, now, we have to show that $=_{\beta\eta}$ is a well. For that we introduce a new concept.

▷ Commuting Relations

▷ **Definition D.1.19.** Let A be a set, then we say that relations $\mathcal{R}\in A^2$ and $\mathcal{S}\in A^2$ commute, if $X\rightarrow_{\mathcal{R}}Y$ and $X\rightarrow_{\mathcal{S}}Z$ entail the existence of a $W\in A$ with $Y\rightarrow_{\mathcal{S}}W$ and $Z\rightarrow_{\mathcal{R}}W$.

$$\begin{tikzcd}
 & X \arrow[dl, "\mathcal{R}"'] \arrow[dr, "\mathcal{S}"] & \\
Y \arrow[dr, "\mathcal{S}"'] & & Z \arrow[dl, "\mathcal{R}"] \\
 & W &
\end{tikzcd}$$

▷ **Observation D.1.20.** *If $\mathcal{R}$ and $\mathcal{S}$ commute, then $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{S}}$ do as well.*

▷ **Observation D.1.21.** *$\mathcal{R}$ is confluent, if $\mathcal{R}$ commutes with itself.*

▷ **Lemma D.1.22.** *If $\mathcal{R}$ and $\mathcal{S}$ are terminating and confluent relations such that $\rightarrow^*_{\mathcal{R}}$ and $\rightarrow^*_{\mathcal{S}}$ commute, then $\rightarrow^*_{\mathcal{R}\cup\mathcal{S}}$ is confluent.*

▷ *Proof sketch:* As $\mathcal{R}$ and $\mathcal{S}$ commute, we can reorder any reduction sequence so that all $\mathcal{R}$-reductions precede all $\mathcal{S}$-reductions. As $\mathcal{R}$ is terminating and confluent, the $\mathcal{R}$-part ends in a unique normal form, and as $\mathcal{S}$ is normalizing it must lead to a unique normal form as well.

This directly gives us our goal.

$\rightarrow^*_{\beta\eta}$ is confluent

▷ **Lemma D.1.23.** *$\rightarrow_\beta^*$ and $\rightarrow_\eta^*$ commute.*

▷ *Proof sketch:* diagram chase

D.2 The Semantics of the Simply Typed λ-Calculus

The semantics of $\Lambda^\rightarrow$ is structured around the types. Like the models we discussed before, a model (we call them "algebras", since we do not have truth values in $\Lambda^\rightarrow$) is a pair $\langle\mathcal{D},\mathcal{I}\rangle$, where $\mathcal{D}$ is the universe of discourse and $\mathcal{I}$ is the interpretation of constants.

Semantics of $\Lambda^\rightarrow$

▷ **Definition D.2.1.** We call a collection $\mathcal{D}_\mathcal{T}{:=}\{\mathcal{D}_\alpha|\alpha{\in}\mathcal{T}\}$ a typed collection (of sets) and a collection $f_\mathcal{T}\colon\mathcal{D}_\mathcal{T}\rightarrow\mathcal{E}_\mathcal{T}$, a typed function, iff $f_\alpha\colon\mathcal{D}_\alpha\rightarrow\mathcal{E}_\alpha$.

▷ **Definition D.2.2.** A typed collection $\mathcal{D}_\mathcal{T}$ is called a frame, iff $\mathcal{D}_{(\alpha\rightarrow\beta)}\subseteq\mathcal{D}_\alpha\rightarrow\mathcal{D}_\beta$

▷ **Definition D.2.3.** Given a frame $\mathcal{D}_\mathcal{T}$, and a typed function $\mathcal{I}\colon\Sigma\rightarrow\mathcal{D}$, then we call $\mathcal{I}_\varphi\colon \mathit{wff}_\mathcal{T}(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})\rightarrow\mathcal{D}$ the value function induced by $\mathcal{I}$, iff

- ▷ $\mathcal{I}_\varphi|_{\mathcal{V}_\mathcal{T}}=\varphi$, $\quad\mathcal{I}_\varphi|_{\Sigma_\mathcal{T}}=\mathcal{I}$
- ▷ $\mathcal{I}_\varphi(\mathbf{A}\ \mathbf{B})=\mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
- ▷ $\mathcal{I}_\varphi(\lambda X_\alpha.\mathbf{A})$ is that function $f{\in}\mathcal{D}_{(\alpha\rightarrow\beta)}$, such that $f(a)=\mathcal{I}_{(\varphi,[a/X])}(\mathbf{A})$ for all $a{\in}\mathcal{D}_\alpha$

▷ **Definition D.2.4.** We call a frame $\langle\mathcal{D},\mathcal{I}\rangle$ comprehension closed or a $\Sigma_\mathcal{T}$-algebra, iff $\mathcal{I}_\varphi\colon \mathit{wff}_\mathcal{T}(\Sigma_\mathcal{T},\mathcal{V}_\mathcal{T})\rightarrow\mathcal{D}$ is total. (every λ-term has a value)

D.2.1 Soundness of the Simply Typed λ-Calculus

We will now show is that $=_{\alpha\beta\eta}$-reduction does not change the value of formulae, i.e. if $\mathbf{A}{=_{\alpha\beta\eta}}\mathbf{B}$, then $\mathcal{I}_\varphi(\mathbf{A})=\mathcal{I}_\varphi(\mathbf{B})$, for all $\mathcal{D}$ and φ. We say that the reductions are sound. As always, the main tool for proving soundess is a substitution value lemma. It works just as always and verifies that we the definitions are in our semantics plausible.

Substitution Value Lemma for λ-Terms

▷ **Lemma D.2.5 (Substitution Value Lemma).** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be terms, then* $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))=\mathcal{I}_\psi(\mathbf{A})$, *where* $\psi=\varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$

▷ *Proof:* by induction on the depth of $\mathbf{A}$

we have five cases

1. $\mathbf{A}=X$
   1.1. Then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))=\mathcal{I}_\varphi([\mathbf{B}/X](X))=\mathcal{I}_\varphi(\mathbf{B})=\psi(X)=\mathcal{I}_\psi(X)=\mathcal{I}_\psi(\mathbf{A})$.
2. $\mathbf{A}=Y\neq X$ and $Y{\in}\mathcal{V}_\mathcal{T}$
   2.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))=\mathcal{I}_\varphi([\mathbf{B}/X](Y))=\mathcal{I}_\varphi(Y)=\varphi(Y)=\psi(Y)=\mathcal{I}_\psi(Y)=\mathcal{I}_\psi(\mathbf{A})$.

3. $\mathbf{A}\in\Sigma_{\mathcal{T}}$
   3.1. This is analogous to the last case.
4. $\mathbf{A} = \mathbf{C}\;\mathbf{D}$
   4.1. then $\mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{C}\;\mathbf{D})) = \mathcal{I}_{\varphi}(([\mathbf{B}/X](\mathbf{C}))\;([\mathbf{B}/X](\mathbf{D}))) = \mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{C}))(\mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{D}))) = \mathcal{I}_{\psi}(\mathbf{C})(\mathcal{I}_{\psi}(\mathbf{D})) = \mathcal{I}_{\psi}(\mathbf{C}\;\mathbf{D}) = \mathcal{I}_{\psi}(\mathbf{A})$
5. $\mathbf{A} = \lambda Y_{\alpha}.\mathbf{C}$
   5.1. We can assume that $X \neq Y$ and $Y \notin \text{free}(\mathbf{B})$
   5.2. Thus for all $a\in\mathcal{D}_{\alpha}$ we have $\mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{A}))(a) = \mathcal{I}_{\varphi}([\mathbf{B}/X](\lambda Y.\mathbf{C}))(a) = \mathcal{I}_{\varphi}(\lambda Y.[\mathbf{B}/X](\mathbf{C}))(a) = \mathcal{I}_{(\varphi,[a/Y])}([\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_{(\psi,[a/Y])}(\mathbf{C}) = \mathcal{I}_{\psi}(\lambda Y.\mathbf{C})(a) = \mathcal{I}_{\psi}(\mathbf{A})(a)$

Soundness of $\alpha\beta\eta$-Equality

▷ **Theorem D.2.6.** *Let $\mathcal{A}:=\langle\mathcal{D},\mathcal{I}\rangle$ be a $\Sigma_{\mathcal{T}}$-algebra and $Y\notin\text{free}(\mathbf{A})$, then $\mathcal{I}_{\varphi}(\lambda X.\mathbf{A}) = \mathcal{I}_{\varphi}(\lambda Y.[Y/X]\mathbf{A})$ for all assignments φ.*

▷ *Proof:* by substitution value lemma

$$\begin{aligned}
\mathcal{I}_{\varphi}(\lambda Y.[Y/X]\mathbf{A})@a &= \mathcal{I}_{(\varphi,[a/Y])}([Y/X](\mathbf{A}))\\
&= \mathcal{I}_{(\varphi,[a/X])}(\mathbf{A})\\
&= \mathcal{I}_{\varphi}(\lambda X.\mathbf{A})@a
\end{aligned}$$

▷ **Theorem D.2.7.** *If $\mathcal{A}:=\langle\mathcal{D},\mathcal{I}\rangle$ is a $\Sigma_{\mathcal{T}}$-algebra and X not bound in $\mathbf{A}$, then $\mathcal{I}_{\varphi}((\lambda X.\mathbf{A})\;\mathbf{B}) = \mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{A}))$.*

Proof: by substitution value lemma again

▷

$$\begin{aligned}
\mathcal{I}_{\varphi}((\lambda X.\mathbf{A})\;\mathbf{B}) &= \mathcal{I}_{\varphi}(\lambda X.\mathbf{A})@\mathcal{I}_{\varphi}(\mathbf{B})\\
&= \mathcal{I}_{(\varphi,[\mathcal{I}_{\varphi}(\mathbf{B})/X])}(\mathbf{A})\\
&= \mathcal{I}_{\varphi}([\mathbf{B}/X](\mathbf{A}))
\end{aligned}$$

Soundness of $\alpha\beta\eta$ (continued)

▷ **Theorem D.2.8.** *If $X\notin\text{free}(\mathbf{A})$, then $\mathcal{I}_{\varphi}(\lambda X.\mathbf{A}\;X) = \mathcal{I}_{\varphi}(\mathbf{A})$ for all φ.*

▷ *Proof:* by calculation

$$\begin{aligned}
\mathcal{I}_{\varphi}(\lambda X.\mathbf{A}\;X)@a &= \mathcal{I}_{(\varphi,[a/X])}(\mathbf{A}\;X)\\
&= \mathcal{I}_{(\varphi,[a/X])}(\mathbf{A})@\mathcal{I}_{(\varphi,[a/X])}(X)\\
&= \mathcal{I}_{\varphi}(\mathbf{A})@\mathcal{I}_{(\varphi,[a/X])}(X) \qquad \text{as } X\notin\text{free}(\mathbf{A}).\\
&= \mathcal{I}_{\varphi}(\mathbf{A})@a
\end{aligned}$$

▷ **Theorem D.2.9.** *$\alpha\beta\eta$-equality is sound wrt. $\Sigma_{\mathcal{T}}$-algebras. (if $\mathbf{A}=_{\alpha\beta\eta}\mathbf{B}$, then $\mathcal{I}_{\varphi}(\mathbf{A})=\mathcal{I}_{\varphi}(\mathbf{B})$ for all assignments φ)*

D.2.2 Completeness of $\alpha\beta\eta$-Equality

We will now show is that $=_{\alpha\beta\eta}$-equality is complete for the semantics we defined, i.e. that whenever $\mathcal{I}_{\varphi}(\mathbf{A})=\mathcal{I}_{\varphi}(\mathbf{B})$ for all variable assignments φ, then $\mathbf{A}=_{\alpha\beta\eta}\mathbf{B}$. We will prove this by a model existence argument: we will construct a model $\mathcal{M}:=\langle\mathcal{D},\mathcal{I}\rangle$ such that if $\mathbf{A}\neq_{\alpha\beta\eta}\mathbf{B}$ then $\mathcal{I}_{\varphi}(\mathbf{A})\neq\mathcal{I}_{\varphi}(\mathbf{B})$ for some φ.

As in other completeness proofs, the model we will construct is a "ground term model", i.e. a model where the carrier (the frame in our case) consists of ground terms. But in the λ-calculus, we have to do more work, as we have a non-trivial built-in equality theory; we will construct the "ground term model" from sets of normal forms. So we first fix some notations for them.

Normal Forms in the simply typed λ-calculus

▷ **Definition D.2.10.** We call a term $\mathbf{A}\in wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}})$ a β normal form iff there is no $\mathbf{B}\in wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}})$ with $\mathbf{A}\to_{\beta}\mathbf{B}$.

We call $\mathbf{N}$ a β normal form of $\mathbf{A}$, iff $\mathbf{N}$ is a β-normal form and $\mathbf{A}\to_{\beta}\mathbf{N}$.

We denote the set of β-normal forms with $wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}})\downarrow_{\beta\eta}$.

▷ We have just proved that $\beta\eta$-reduction is terminating and confluent, so we have

▷ **Corollary D.2.11 (Normal Forms).** *Every $\mathbf{A}\in wff_{\mathcal{T}}(\Sigma_{\mathcal{T}},\mathcal{V}_{\mathcal{T}})$ has a unique β normal form ($\beta\eta$, long $\beta\eta$ normal form), which we denote by $\mathbf{A}\downarrow_{\beta}$ ($\mathbf{A}\downarrow_{\beta\eta}$ $\mathbf{A}\downarrow_{\beta\eta^{l}}$)*

The term frames will be a quotient spaces over the equality relations of the λ-calculus, so we introduce this construction generally.

Frames and Quotients

▷ **Definition D.2.12.** Let $\mathcal{D}$ be a frame and $\sim$ a typed equivalence relation on $\mathcal{D}$, then we call $\sim$ a congruence on $\mathcal{D}$, iff $f\sim f'$ and $g\sim g'$ imply $f(g)\sim f'(g')$.

▷ **Definition D.2.13.** We call a congruence $\sim$ functional, iff for all $f,g\in\mathcal{D}_{(\alpha\to\beta)}$ the fact that $f(a)\sim g(a)$ holds for all $a\in\mathcal{D}_{\alpha}$ implies that $f\sim g$.

▷ **Example D.2.14.** $=_{\beta}$ ($=_{\beta\eta}$) is a (functional) congruence on $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})$ by definition.

▷ **Theorem D.2.15.** *Let $\mathcal{DT}$ be a $\Sigma_{\mathcal{T}}$-frame and $\sim$ a functional congruence on $\mathcal{D}$, then the quotient space $\mathcal{D}/\sim$ is a $\Sigma_{\mathcal{T}}$-frame.*

▷ *Proof:*

1. $\mathcal{D}/\sim=\{[f]_{\sim}|f\in\mathcal{D}\}$, define $[f]_{\sim}([a]_{\sim}):=[f(a)]_{\sim}$.
2. This only depends on equivalence classes: Let $f'\in[f]_{\sim}$ and $a'\in[a]_{\sim}$.
3. Then $[f(a)]_{\sim}=[f'(a)]_{\sim}=[f'(a')]_{\sim}=[f(a')]_{\sim}$

4. To see that we have $[f]_\sim = [g]_\sim$, iff $f \sim g$, iff $f(a) = g(a)$ since $\sim$ is functional.
5. This is the case iff $[f(a)]_\sim = [g(a)]_\sim$, iff $[f]_\sim([a]_\sim) = [g]_\sim([a]_\sim)$ for all $a \in \mathcal{D}_\alpha$ and thus for all $[a]_\sim \in \mathcal{D}/\sim$.

To apply this result, we have to establish that $=_{\beta\eta}$-equality is a functional congruence. We first establish $=_{\beta\eta}$ as a functional congruence on $wff_{\mathcal{T}}(\Sigma_{\mathcal{T}}, \mathcal{V}_{\mathcal{T}})$ and then specialize this result to show that is is also functional on $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})$ by a grounding argument.

$\beta\eta$-Equivalence as a Functional Congruence

▷ **Lemma D.2.16.** *$\beta\eta$-equality is a functional congruence on $wff_{\mathcal{T}}(\Sigma_{\mathcal{T}}, \mathcal{V}_{\mathcal{T}})$.*

▷ *Proof:* Let $\mathbf{A}\ \mathbf{C} =_{\beta\eta} \mathbf{B}\ \mathbf{C}$ for all $\mathbf{C}$ and $X \in (\mathcal{V}_\gamma \backslash (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{B})))$.

1. then (in particular) $\mathbf{A}\ X =_{\beta\eta} \mathbf{B}\ X$, and
2. $(\lambda X.\mathbf{A}\ X) =_{\beta\eta} (\lambda X.\mathbf{B}\ X)$, since $\beta\eta$-equality acts on subterms.
3. By definition we have $\mathbf{A} =_\eta (\lambda X_\alpha.\mathbf{A}\ X) =_{\beta\eta} (\lambda X_\alpha.\mathbf{B}\ X) =_\eta \mathbf{B}$.

▷ **Definition D.2.17.** We call an injective substitution $\sigma\colon \mathrm{free}(\mathbf{C}) \to \Sigma_{\mathcal{T}}$ a grounding substitution for $\mathbf{C} \in wff_{\mathcal{T}}(\Sigma_{\mathcal{T}}, \mathcal{V}_{\mathcal{T}})$, iff no $\sigma(X)$ occurs in $\mathbf{C}$.

▷ **Observation:** They always exist, since all Σ_α are infinite and $\mathrm{free}(\mathbf{C})$ is finite.

▷ **Theorem D.2.18.** *$\beta\eta$-equality is a functional congruence on $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})$.*

▷ *Proof:* We use **??**

1. Let $\mathbf{A}, \mathbf{B} \in cwff_{(\alpha\to\beta)}(\Sigma_{\mathcal{T}})$, such that $\mathbf{A} \neq_{\beta\eta} \mathbf{B}$.
2. As $\beta\eta$ is functional on $wff_{\mathcal{T}}(\Sigma_{\mathcal{T}}, \mathcal{V}_{\mathcal{T}})$, there must be a $\mathbf{C}$ with $\mathbf{A}\ \mathbf{C} \neq_{\beta\eta} \mathbf{B}\ \mathbf{C}$.
3. Now let $\mathbf{C}' := \sigma(\mathbf{C})$, for a grounding substitution σ.
4. Any $\beta\eta$ conversion sequence for $\mathbf{A}\ \mathbf{C}' \neq_{\beta\eta} \mathbf{B}\ \mathbf{C}'$ induces one for $\mathbf{A}\ \mathbf{C} \neq_{\beta\eta} \mathbf{B}\ \mathbf{C}$.
5. Thus we have shown that $\mathbf{A} \neq_{\beta\eta} \mathbf{B}$ entails $\mathbf{A}\ \mathbf{C}' \neq_{\beta\eta} \mathbf{B}\ \mathbf{C}'$.

Note that: the result for $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})$ is sharp. For instance, if $\Sigma_{\mathcal{T}} = \{c_\iota\}$, then $(\lambda X.X) \neq_{\beta\eta} (\lambda X.c)$, but $(\lambda X.X)\ c =_{\beta\eta} c =_{\beta\eta} (\lambda X.c)\ c$, as $\{c\} = cwff_\iota(\Sigma_{\mathcal{T}})$ (it is a relatively simple exercise to extend this problem to more than one constant). The problem here is that we do not have a constant d_ι that would help distinguish the two functions. In $wff_{\mathcal{T}}(\Sigma_{\mathcal{T}}, \mathcal{V}_{\mathcal{T}})$ we could always have used a variable.

This completes the preparation and we can define the notion of a term algebra, i.e. a $\Sigma_{\mathcal{T}}$-algebra whose frame is made of $=_{\beta\eta}$-normal λ-terms.

A Herbrand Model for $\Lambda^\to$

▷ **Definition D.2.19.** We call $\mathcal{T}_{\beta\eta} := \langle cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})\downarrow_{\beta\eta}, \mathcal{I}^{\beta\eta}\rangle$ the Σ term algebra, if $\mathcal{I}^{\beta\eta} = \mathrm{Id}_{\Sigma_{\mathcal{T}}}$.

▷ The name "term algebra" in the previous definition is justified by the following

▷ **Theorem D.2.20.** *$\mathcal{T}_{\beta\eta}$ is a $\Sigma_{\mathcal{T}}$-algebra*

▷ *Proof:* We use the work we did above

1. Note that $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})\big\downarrow_{\beta\eta} = cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})/{=_{\beta\eta}}$ and thus a $\Sigma_{\mathcal{T}}$-frame by **??** and **??**.
2. So we only have to show that the value function $\mathcal{I}^{\beta\eta} = \mathrm{Id}_{\Sigma_{\mathcal{T}}}$ is total.
3. Let φ be an assignment into $cwff_{\mathcal{T}}(\Sigma_{\mathcal{T}})\big\downarrow_{\beta\eta}$.
4. Note that $\sigma := \varphi|_{\mathrm{free}(\mathbf{A})}$ is a substitution, since $\mathrm{free}(\mathbf{A})$ is finite.
5. A simple induction on the structure of $\mathbf{A}$ shows that $\mathcal{I}^{\beta\eta}{}_{\varphi}(\mathbf{A}) = (\sigma(\mathbf{A}))\big\downarrow_{\beta\eta}$.
6. So the value function is total since substitution application is.

And as always, once we have a term model, showing completeness is a rather simple exercise. We can see that $\alpha\beta\eta$-equality is complete for the class of $\Sigma_{\mathcal{T}}$-algebras, i.e. if the equation $\mathbf{A} = \mathbf{B}$ is valid, then $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$. Thus $\alpha\beta\eta$ equivalence fully characterizes equality in the class of all $\Sigma_{\mathcal{T}}$-algebras.

Completetness of $\alpha\beta\eta$-Equality

▷ **Theorem D.2.21.** *$\mathbf{A} = \mathbf{B}$ is valid in the class of $\Sigma_{\mathcal{T}}$-algebras, iff $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$.*

▷ *Proof:* For $\mathbf{A}$, $\mathbf{B}$ closed this is a simple consequence of the fact that $\mathcal{T}_{\beta\eta}$ is a $\Sigma_{\mathcal{T}}$-algebra.

1. If $\mathbf{A} = \mathbf{B}$ is valid in all $\Sigma_{\mathcal{T}}$-algebras, it must be in $\mathcal{T}_{\beta\eta}$ and in particular $\mathbf{A}\downarrow_{\beta\eta} = \mathcal{I}^{\beta\eta}(\mathbf{A}) = \mathcal{I}^{\beta\eta}(\mathbf{B}) = \mathbf{B}\downarrow_{\beta\eta}$ and therefore $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$.

If the equation has free variables, then the argument is more subtle.

2. Let σ be a grounding substitution for $\mathbf{A}$ and $\mathbf{B}$ and φ the induced variable assignment.
3. Thus $\mathcal{I}^{\beta\eta}{}_{\varphi}(\mathbf{A}) = \mathcal{I}^{\beta\eta}{}_{\varphi}(\mathbf{B})$ is the $\beta\eta$-normal form of $\sigma(\mathbf{A})$ and $\sigma(\mathbf{B})$.
4. Since φ is a structure preserving homomorphism on well-formed formulae, $\varphi^{-1}(\mathcal{I}^{\beta\eta}{}_{\varphi}(\mathbf{A}))$ is the is the $\beta\eta$-normal form of both $\mathbf{A}$ and $\mathbf{B}$ and thus $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$.

?? and **??** complete our study of the semantics of the simply-typed λ-calculus by showing that it is an adequate logic for modeling (the equality) of functions and their applications.

D.3 Simply Typed λ-Calculus via Inference Systems

Now, we will look at the simply typed λ-calculus again, but this time, we will present it as an inference system for well-typedness jugdments. This more modern way of developing type theories is known to scale better to new concepts.

Simply Typed λ-Calculus as an Inference System: Terms

▷ **Idea:** Develop the λ-calculus in two steps

- ▹ A context-free grammar for "raw λ-terms" (for the structure)
- ▹ Identify the well-typed λ-terms in that (cook them until well-typed)

▷ **Definition D.3.1.** A grammar for the raw terms of the simply typed λ-calculus:

$$\begin{array}{lll} \alpha & ::= & c \mid \alpha \to \alpha \\ \Sigma & ::= & \cdot \mid \Sigma,[c : \mathsf{type}] \mid \Sigma,[c{:}\alpha] \\ \Gamma & ::= & \cdot \mid \Gamma,[x{:}\alpha] \\ \mathbf{A} & ::= & c \mid X \mid \mathbf{A}^1\ \mathbf{A}^2 \mid \lambda X_\alpha.\mathbf{A} \end{array}$$

▷ **Then:** Define all the operations that are possible at the "raw terms level", e.g. realize that signatures and contexts are partial functions to types.

Simply Typed λ-Calculus as an Inference System: Judgments

▷ **Definition D.3.2.** Judgments make statements about complex properties of the syntactic entities defined by the grammar.

▷ **Definition D.3.3.** Judgments for the simply typed λ-calculus

$\vdash \Sigma : \mathsf{sig}$	Σ is a well-formed signature
$\Sigma \vdash \alpha : \mathsf{type}$	α is a well-formed type given the type assumptions in Σ
$\Sigma \vdash \Gamma : \mathsf{ctx}$	Γ is a well-formed context given the type assumptions in Σ
$\Gamma \vdash_\Sigma \mathbf{A} : \alpha$	$\mathbf{A}$ has type α given the type assumptions in Σ and Γ

Simply Typed λ-Calculus as an Inference System: Rules

▷ **Definition D.3.4.** $\mathbf{A} \in wff_\alpha(\Sigma_\mathcal{T}, \mathcal{V}_\mathcal{T})$, iff $\Gamma \vdash_\Sigma \mathbf{A} : \alpha$ derivable in

$$\frac{\Sigma \vdash \Gamma : \mathsf{ctx} \quad \Gamma(X) = \alpha}{\Gamma \vdash_\Sigma X : \alpha}\, wff\,var \qquad \frac{\Sigma \vdash \Gamma : \mathsf{ctx} \quad \Sigma(c) = \alpha}{\Gamma \vdash_\Sigma c : \alpha}\, wff\,const$$

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} : \beta \to \alpha \quad \Gamma \vdash_\Sigma \mathbf{B} : \beta}{\Gamma \vdash_\Sigma \mathbf{A}\ \mathbf{B} : \alpha}\, wff\,app \qquad \frac{\Gamma, [X{:}\beta] \vdash_\Sigma \mathbf{A} : \alpha}{\Gamma \vdash_\Sigma \lambda X_\beta.\mathbf{A} : \beta \to \alpha}\, wff\,abs$$

▷ **Oops:** this looks surprisingly like a natural deduction calculus. ($\leadsto$ Curry Howard Isomorphism)

▷ To be complete, we need rules for well-formed signatures, types and contexts

▷ **Definition D.3.5.**

$$\frac{}{\vdash \cdot : \mathsf{sig}}\, sig\,empty \qquad \frac{\vdash \Sigma : \mathsf{sig}}{\vdash (\Sigma,[\alpha : \mathsf{type}]) : \mathsf{sig}}\, sig\,type$$

$$\frac{\vdash \Sigma : \mathsf{sig} \quad \Sigma \vdash \alpha : \mathsf{type}}{\vdash (\Sigma,[c{:}\alpha]) : \mathsf{sig}}\, sig\,const$$

$$\frac{\Sigma \vdash \alpha : \mathsf{type} \quad \Sigma \vdash \beta : \mathsf{type}}{\Sigma \vdash (\alpha \to \beta) : \mathsf{type}}\, typ\,fn \qquad \frac{\vdash \Sigma : \mathsf{sig} \quad \Sigma(\alpha) = \mathsf{type}}{\Sigma \vdash \alpha : \mathsf{type}}\, typ\,start$$

$$\frac{\vdash \Sigma : \mathsf{sig}}{\Sigma \vdash \cdot : \mathsf{ctx}}\, ctx\,empty \qquad \frac{\Sigma \vdash \Gamma : \mathsf{ctx} \quad \Sigma \vdash \alpha : \mathsf{type}}{\Sigma \vdash (\Gamma,[X{:}\alpha]) : \mathsf{ctx}}\, ctx\,var$$

Example: A Well-Formed Signature

▷ Let $\Sigma{:=}[\alpha : \mathsf{type}],[f{:}\alpha \to \alpha \to \alpha]$, then Σ is a well-formed signature, since we have derivations $\mathcal{A}$ and $\mathcal{B}$

$$\dfrac{\vdash \cdot : \mathsf{sig}}{\vdash [\alpha : \mathsf{type}] : \mathsf{sig}}\,\mathit{sig\,type} \qquad \dfrac{\mathcal{A} \quad [\alpha : \mathsf{type}](\alpha) = \mathsf{type}}{[\alpha : \mathsf{type}] \vdash \alpha : \mathsf{type}}\,\mathit{typ\,start}$$

and with these we can construct the derivation $\mathcal{C}$

$$\dfrac{\mathcal{A} \quad \dfrac{\mathcal{B} \quad \dfrac{\mathcal{B} \quad \mathcal{B}}{[\alpha : \mathsf{type}] \vdash (\alpha \to \alpha) : \mathsf{type}}\,\mathit{typ\,fn}}{[\alpha : \mathsf{type}] \vdash (\alpha \to \alpha \to \alpha) : \mathsf{type}}\,\mathit{typ\,fn}}{\vdash \Sigma : \mathsf{sig}}\,\mathit{sig\,const}$$

Example: A Well-Formed λ-Term

▷ using Σ from above, we can show that $\Gamma{:=}[X{:}\alpha]$ is a well-formed context:

$$\dfrac{\dfrac{\mathcal{C}}{\Sigma \vdash \cdot : \mathsf{ctx}}\,\mathit{ctx\,empty} \quad \dfrac{\mathcal{C} \quad \Sigma(\alpha) = \mathsf{type}}{\Sigma \vdash \alpha : \mathsf{type}}\,\mathit{typ\,start}}{\Sigma \vdash \Gamma : \mathsf{ctx}}\,\mathit{ctx\,var}$$

We call this derivation $\mathcal{G}$ and use it to show that

▷ $\lambda X_\alpha.f\ X\ X$ is well-typed and has type $\alpha \to \alpha$ in Σ. This is witnessed by the type derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{C} \quad \Sigma(f) = \alpha \to \alpha \to \alpha}{\Gamma \vdash_\Sigma f : \alpha \to \alpha \to \alpha}\,\mathit{wff\,const} \quad \dfrac{\mathcal{G}}{\Gamma \vdash_\Sigma X : \alpha}\,\mathit{wff\,var}}{\Gamma \vdash_\Sigma f\ X : \alpha \to \alpha}\,\mathit{wff\,app} \quad \dfrac{\mathcal{G}}{\Gamma \vdash_\Sigma X : \alpha}\,\mathit{wff\,var}}{\Gamma \vdash_\Sigma f\ X\ X : \alpha}\,\mathit{wff\,app}}{\cdot \vdash_\Sigma \lambda X_\alpha.f\ X\ X : \alpha \to \alpha}\,\mathit{wff\,abs}$$

$\beta\eta$-Equality by Inference Rules: One-Step Reduction

▷ **Definition D.3.6.** One-step Reduction ($+\in\{\alpha,\beta,\eta\}$)

$$\frac{\Gamma,[X:\beta]\vdash_\Sigma\mathbf{A}:\alpha \quad \Gamma\vdash_\Sigma\mathbf{B}:\beta}{\Gamma\vdash_\Sigma(\lambda X.\mathbf{A})\,\mathbf{B}\rightarrow^1_\beta[\mathbf{B}/X](\mathbf{A})}\,wff\beta top$$

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}:\beta\rightarrow\alpha \quad X\notin\mathbf{dom}(\Gamma)}{\Gamma\vdash_\Sigma\lambda X.\mathbf{A}\,X\rightarrow^1_\eta\mathbf{A}}\,wff\eta top$$

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^1_+\mathbf{B} \quad \Gamma\vdash_\Sigma\mathbf{A}\,\mathbf{C}:\alpha}{\Gamma\vdash_\Sigma\mathbf{A}\,\mathbf{C}\rightarrow^1_+\mathbf{B}\,\mathbf{C}}\,tr\,appfn$$

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^1_+\mathbf{B} \quad \Gamma\vdash_\Sigma\mathbf{C}\,\mathbf{A}:\alpha}{\Gamma\vdash_\Sigma\mathbf{C}\,\mathbf{A}\rightarrow^1_+\mathbf{C}\,\mathbf{B}}\,tr\,apparg$$

$$\frac{\Gamma,[X:\alpha]\vdash_\Sigma\mathbf{A}\rightarrow^1_+\mathbf{B}}{\Gamma\vdash_\Sigma\lambda X.\mathbf{A}\rightarrow^1_+\lambda X.\mathbf{B}}\,tr\,abs$$

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG   Michael Kohlhase: LBS   356   2024-01-20

$\beta\eta$-Equality by Inference Rules: Multi-Step Reduction

▷ **Definition D.3.7.** Multi-Step-Reduction ($+\in\{\alpha,\beta,\eta\}$)

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^1_+\mathbf{B}}{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^*_+\mathbf{B}}\,ms\,start \qquad \frac{\Gamma\vdash_\Sigma\mathbf{A}:\alpha}{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^*_+\mathbf{A}}\,ms\,ref$$

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^*_+\mathbf{B} \quad \Gamma\vdash_\Sigma\mathbf{B}\rightarrow^*_+\mathbf{C}}{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^*_+\mathbf{C}}\,ms\,trans$$

▷ Congruence Relation

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}\rightarrow^*_+\mathbf{B}}{\Gamma\vdash_\Sigma\mathbf{A}=_+\mathbf{B}}\,eq\,start$$

$$\frac{\Gamma\vdash_\Sigma\mathbf{A}=_+\mathbf{B}}{\Gamma\vdash_\Sigma\mathbf{B}=_+\mathbf{A}}\,eq\,sym \qquad \frac{\Gamma\vdash_\Sigma\mathbf{A}=_+\mathbf{B} \quad \Gamma\vdash_\Sigma\mathbf{B}=_+\mathbf{C}}{\Gamma\vdash_\Sigma\mathbf{A}=_+\mathbf{C}}\,eq\,trans$$

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG   Michael Kohlhase: LBS   357   2024-01-20

Type Computation: Manage Types Algorithmically

▷ **Questions:**
- type check: Is $\Gamma\vdash_\Sigma\mathbf{A}:\alpha$?
- type inference: are there Γ, α, such that $\Gamma\vdash_\Sigma\mathbf{A}:\alpha$?
- type reconstruction the above without type annotations at bound variables?

▷ prenex fragment makes problems decidable (see Curry Howard)

▷ **Algorithm (Hindley & Milner):**

- ▷ invert inference rules

- first-order unification,
- universal generalization, minimization

Example Computation

rule tree	constraint
$\dfrac{\dfrac{\dfrac{[X:\alpha]}{\Gamma, [X:\beta]}}{\Gamma, [X:\beta] \vdash_\Sigma X:\alpha} \quad \Gamma \vdash_\Sigma \lambda X.X:\beta \to \alpha}{\dfrac{\dfrac{[W:\delta] \in \Gamma, [Z:\gamma]}{\Gamma, [Z:\gamma] \vdash_\Sigma W:\delta}}{\Gamma \vdash_\Sigma \lambda Z.W:\beta}} \; \Gamma \vdash_\Sigma \lambda X.X(\lambda Z.W):\alpha$	$\alpha = \beta$, $[W:\delta] \in \Gamma$, $\beta = \gamma \to \delta$

- unification: $\alpha = \beta = \gamma \to \delta$,
- minimization: $\Gamma = [W:\delta]$
- **Solution:** $[W:\delta]] \vdash_\Sigma \lambda X.X(\lambda Z.W) : \forall \gamma.\gamma \to \delta$

Appendix E

Higher-Order Dynamics

In this chapter we will develop a typed λ calculus that extend DRT-like dynamic logics like the simply typed λ calculus extends first-order logic.

E.1 Introduction

We start out our development of a Montague-like compositional treatment of dynamic semantics construction by naively “adding λs” to DRT and deriving requirements from that.

Making Montague Semantics Dynamic

▷ **Example E.1.1.** *A man sleeps.*

$$\mathsf{a\_man} = \lambda Q.\left(\begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\\hline\end{array} \otimes Q(U)\right)$$

$$\mathsf{sleep} = \lambda X.\begin{array}{|l|}\hline \\\hline \mathsf{sleeps}(X)\\\hline\end{array}$$

Application and β-reduction:

$$\begin{aligned}\mathsf{a\_man\_sleep} &= \mathsf{a\_man}(\mathsf{sleep})\\ &\rightarrow_\beta \begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\\hline\end{array} \otimes \begin{array}{|l|}\hline \\\hline \mathsf{sleeps}(U)\\\hline\end{array} \rightarrow_\tau \begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\ \mathsf{sleeps}(U)\\\hline\end{array}\end{aligned}$$

At the sentence level we just disregard that we have no idea how to interpret λ-abstractions over DRSes and just proceed as in the static (first-order) case. Somewhat surprisingly, this works rather well, so we just continue at the discourse level.

Coherent Text (Capturing Discourse Referents)

▷ **Example E.1.2.** *A man*^{1} *sleeps. He*$_1$ *snores.*

$(\lambda PQ.(P \otimes Q))$ a_man_sleep he_snore

$$\to_{=_\beta} \left(\lambda Q.\begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\ \mathsf{sleeps}(U)\\\hline\end{array} \otimes Q\right) \begin{array}{|l|}\hline \\\hline \mathsf{snores}(U)\\\hline\end{array}$$

$$\to_\tau \begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\ \mathsf{sleeps}(U)\\\hline\end{array} \otimes \begin{array}{|l|}\hline \\\hline \mathsf{snores}(U)\\\hline\end{array} \quad \to_\tau \begin{array}{|l|}\hline U\\\hline \mathsf{man}(U)\\ \mathsf{sleeps}(U)\\ \mathsf{snores}(U)\\\hline\end{array}$$

▷ **Example E.1.3 (Linear notation).** $(\lambda Q.(\delta U.\mathsf{man}(U) \wedge \mathsf{sleeps}(U) \wedge Q(U)))$ he_snore $\longrightarrow_{\beta\tau}$ $\delta U.\mathsf{man}(U) \wedge \mathsf{sleeps}(U) \wedge \mathsf{snores}(U)$

Here we have our first surprise: the second $=_\beta$ reduction seems to variable capturecapture the discourse referent U: intuitively it is "free" in $\delta U.\mathrm{snores}(U)$ and after $=_\beta$ reduction it is under the influence of a δ declaration. In the λ-calculus tradition variable capture is the great taboo, whereas in our example, referent capture seems to drive/enable anaphor resolution.
Considerations like the ones above have driven the development of many logical systems attempting the compositional treatment of dynamic logics. All were more or less severely flawed.

Compositional Discourse Representation Theories

▷ Many logical systems

- ▹ Compositional DRT (Zeevat, 1989 [Zee89])
- ▹ Dynamic Montague Grammar (DMG Gronendijk/Stokhof 1990 [GS90])
- ▹ CDRT (Muskens 1993/96 [Mus96])
- ▹ λ-DRT (Kohlhase/Kuschert/Pinkal 1995 [KKP96])
- ▹ TLS (van Eijck 1996 [Eij97])

▷ **Problem:** Difficult to tell the differences or make predictions!

▷ **One Answer:** Dynamic λ-calculus [Kohlhase&Kuschert&Müller'96,98]

- ▹ Augment type system by information on referents: a meta-logic that models different forms of accessibility as a parameter.

Here we will look at a system that makes the referent capture the central mechanism using an elaborate type system to describe referent visibility and thus accessibility. This generalization allows to understand and model the interplay of λ-bound variables and discourse referents without being distracted by linguistic modeling questions (which are relegated to giving appropriate types to the operators).
Another strong motivation for a higher-order treatment of dynamic logics is that maybe the computational semantic analysis methods based on higher-order features (mostly higher-order unification) can be analogously transferred to the dynamic setting.

Motivation for the Future

- ▷ Higher-Order Unification Analyses of
  - ▷ Ellipsis (Dalrymple/Shieber/Pereira 1991 [DSP91])
  - ▷ Focus (Pulman 1994 [Pul94], Gardent/Kohlhase 1996 [GK96])
  - ▷ Corrections (Gardent/Kohlhase/van Leusen 1996 [GKL96])
  - ▷ Underspecification (Pinkal 1995 [Pin96])
- ▷ are based on static type theory [Mon74]
- ▷ Higher-Order Dynamic Unification needed for dynamic variants of these

To set the stage for the development of a higher-order system for dynamic logic, let us remind ourselves of the setup of the static system

Recap: Simple Type Theory

- ▷ Structural layer: simply typed λ-calculus
  - ▷ types, well-formed formulae, λ-abstraction
  - ▷ Theory: $\alpha\beta\eta$-conversion, Operational: Higher-Order Unification
- ▷ Logical layer: higher-order logic
  - ▷ special types ι, prop
  - ▷ logical constants $\wedge_{\mathrm{prop}\to\mathrm{prop}\to\mathrm{prop}}, \Rightarrow, \forall, \ldots$ with fixed semantics
  - ▷ Theory: logical theory, Operational: higher-order theorem proving
- ▷ **Goal:** Develop two-layered approach to compositional discourse theories.
- ▷ **Application:** Dynamic Higher-Order Unification (DHOU) with structural layer only.

This separation of concerns: structural properties of functions vs. a propositional reasoning level has been very influential in modeling static, intra-sentential properties of natural language, therefore we want to have a similar system for dynamic logics as well. We will use this as a guiding intuition below.

E.2 Setting Up Higher-Order Dynamics

To understand what primitives a language for higher-order dynamics should provide, we will analyze one of the attempts – λ-DRT – to higher-order dynamics
λ-DRT is a relatively straightforward (and naive) attempt to "sprinkle λs over DRT" and give that a semantics. This is mirrored in the type system, which had a primitive types for DRSes and "intensions" (mappings from states to objects). To make this work we had to introduce "intensional closure", a semantic device akin to type raising that had been in the folklore for some time. We will not go into intensions and closure here, since this did not lead to a solution and refer the reader to [KKP96] and the references there.

Recap: λ-DRT (simplified)

- **Definition E.2.1 (Types).** ι (individuals), prop (conditions), t (DRSes), $\alpha \to \beta$ (functions), $s \to \alpha$ (intensions)
- **Syntax:** if U_ι a referent and $\mathbf{A}$ an expression of type prop, then $\delta U_\iota.\mathbf{A}$ a DRS (type t).
- **Definition E.2.2.** $=_{\alpha\beta\eta}$-reduction for the λ-calculus part, and further:
  - $(\delta\mathcal{X}.\mathbf{A} \otimes \delta\mathcal{Y}.\mathbf{B}) \to_\tau (\delta\mathcal{X} \cup \mathcal{Y}.\mathbf{A} \wedge \mathbf{B})$
  - ${}^{\vee\wedge}\mathbf{A} \to_\mu \mathbf{A}$
- **Observations:**
  - complex interaction of λ and δ
  - alphabetical change for δ-bound “variables” (referents)?
  - need intensional closure for $=_{\beta\eta}$-reduction to be correct

In hindsight, the contribution of λ-DRT was less the proposed semantics – this never quite worked beyond correctness of $=_{\alpha\beta\eta}$ equality – but the logical questions about types, reductions, and the role of states it raised, and which led to further investigations.

We will now look at the general framework of “a λ-calculus with discourse referents and δ-binding” from a logic-first perspective and try to answer the questions this raises. The questions of modeling dynamic phenomena of natural language take a back seat for the moment.

Finding the right Dynamic Primitives

- Need to understand merge reduction: ($\to_\tau$-reduction)
  - Why do we have $(\delta U.\mathbf{A} \otimes \mathbf{B}) \to_\tau (\delta U.\mathbf{A} \wedge \mathbf{B})$
  - but not $((\delta U.\mathbf{A}) \Rrightarrow \mathbf{B}) \to_\tau (\delta U.\mathbf{A} \Rrightarrow \mathbf{B})$
- and Referent Scoping: (ρ-equivalence)
  - When are the meanings of $\mathbf{C}\left[(\delta U.\mathbf{A})\right]_\pi$ and $\mathbf{C}\left[(\delta V.[V/U](\mathbf{A}))\right]_\pi$ equal?
  - OK for $\mathbf{C} = \neg$ and $\mathbf{C} = \lambda P.(\delta W.\mathbf{A} \Rrightarrow P)$
  - Not for $\mathbf{C} = \lambda P.P$ and $\mathbf{C} = \lambda P.P \wedge \neg P$.
- **Observation:** There must be a difference of $\otimes, \neg, \lambda P.(\delta W.\mathbf{A} \Rrightarrow P), \lambda P.P \wedge \neg P$ wrt. the behavior on referents
- **Intuitively:** $\otimes, \lambda P.(\delta W.\mathbf{A} \Rrightarrow P)$ transport U, while $\neg, \lambda P.P \wedge \neg P$ do not
- **Idea:** Model this in the types (rest of the talk/lecture)

A particularly interesting phenomenon is that of referent capture as the motor or anaphor resolution, which have already encountered Example 12.1.4.

Variable/Referent Capture

▷ **Example E.2.3 (Anaphor Resolution Revisited).** Let us revisit Example 12.1.4

A student[1] owns a book[2].
He[1] reads it[2]

anaphor resolution

simplify

$$\begin{array}{|l|}\hline X, Y \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \hline \end{array} \otimes \begin{array}{|l|}\hline R, S \\ \hline \text{read}(R \\ S) \\ \hline \end{array} \qquad \begin{array}{|l|}\hline X, Y \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \hline \end{array} \otimes \begin{array}{|l|}\hline R, S \\ \hline \text{read}(R \\ S) \\ R = X \\ S = Y \\ \hline \end{array} \qquad \begin{array}{|l|}\hline X, Y \\ \hline \text{student}(X) \\ \text{book}(Y) \\ \text{read}(X \\ Y) \\ \hline \end{array}$$

▷ **Example E.2.4.** $(\lambda P.\begin{array}{|l|}\hline U \\ \hline (\neg P) \\ \hline \end{array}) \begin{array}{|l|}\hline \\ \hline r(U) \\ \hline \end{array}$ (functor has dynamic binding power)

▷ **Definition E.2.5.** We call this referent capture.

▷ **Note:** Referent capture

- ▷ is the motor of dynamicity
- ▷ is a structural property

▷ **Idea:** Code the information for referent capture in the type system.

In Example E.2.3 we see that with the act of anaphor resolution, the discourse referents induced by the anaphoric pronouns get placed under the influence of the dynamic binding in the first DRS – which is OK from an accessibility point of view, but from a λ-calculus perspective this constitutes a capturing event, since the binding relation changes. This becomes especially obvious, if we look at the simplified form, where the discourse referents introduced in the translation of the pronouns have been eliminated altogether.
In Example E.2.4 we see that a capturing situation can occur even more explicitly, if we allow λs – and $=_{\alpha\beta\eta}$ equality – in the logic. We have to deal with this, and again, we choose to model it in the type system.
With the intuitions sharpened by the examples above, we will now start to design a type system that can take information about referents into account. In particular we are interested in the capturing behavior identified above. Therefore we introduce information about the "capturing status" of discourse referents in the respective expressions into the types.

Types in $\mathcal{DLC}$

▷ **Requirements:** In the types we need information about

- ▷ δ-bound referents (they do the capturing)
- ▷ free referents (they are liable to be captured)

▷ **Definition E.2.6.** New type (moded type) $\Gamma\#\alpha$ where

- ▷ mode $\Gamma = V^-, U^+, \ldots$ (V is a free and U a capturing referent)
- ▷ term type α (type in the old sense)

▷ What about functional types? (Look at example)

To see how our type system for $\mathcal{DLC}$ fares in real life, we see whether we can capture the referent dynamics of λ-DRT. Maybe this also tells us what we still need to improve.

Rational Reconstruction of λ-DRT (First Version)

▷ Two-level approach
  - ▷ model structural properties (e.g. accessibility relation) in the types
  - ▷ leave logical properties (e.g. negation flips truth values) for later

▷ Types: $\iota, \mathsf{prop}, \alpha \to \beta$ only. $\Gamma\#\mathsf{prop}$ is a DRS.

▷ **Idea:** Use mode constructors $\downarrow$ and $\uplus$ to describe the accessibility relation.

▷ **Definition E.2.7.** $\downarrow$ closes off the dynamic binding potential and makes the referents classically bound
($\downarrow U^+, V^+ = U^\circ, V^\circ$)

▷ **Definition E.2.8.** The prioritized union operator combines two modes by letting $+$ overwrite $-$. ($U^+, V^- \uplus U^-, V^+ = U^+, V^+$)

▷ **Example E.2.9 (DRT Operators).** Types of DRT connectives (indexed by Γ, Δ):
  - ▷ $\neg$ has type $\Gamma\#\mathsf{prop} \to \downarrow\Gamma\#\mathsf{prop}$ (intuitively like $t \to \mathsf{prop}$)
  - ▷ $\otimes$ has type $\Gamma\#\mathsf{prop} \to \Delta\#\mathsf{prop} \to \Gamma \uplus \Delta\#\mathsf{prop}$ (intuitively like $t \to t \to t$)
  - ▷ $\vee$ has type $\Gamma\#\mathsf{prop} \to \Delta\#\mathsf{prop} \to \downarrow\Gamma \uplus \downarrow\Delta\#\mathsf{prop}$
  - ▷ $\Rightarrow$ has type $\Gamma\#\mathsf{prop} \to \Delta\#\mathsf{prop} \to \downarrow(\Gamma \uplus \downarrow\Delta)\#\mathsf{prop}$

We can already see with the experiment of modeling the DRT operators that the envisioned type system gives us a way of specifying accessibility and how the dynamic operators handle discourse referents. So we indeed have the beginning of a structural level for higher-order dynamics, and at the same time a meta-logic flavor, since we can specify other dynamic logics in a λ-calculus.

E.3 A Type System for Referent Dynamics

We will now take the ideas above as the basis for a type system for $\mathcal{DLC}$.
The types above have the decided disadvantage that they mix mode information with information about the order of the operators. They also need free mode variables, which turns out to be a problem for designing the semantics. Instead, we will employ two-dimensional types, where the mode part is a function on modes and the other a normal simple type.

Types in $\mathcal{DLC}$ (Final Version)

▷ **Problem:** A type like $\Gamma\#\mathsf{prop} \to \Gamma^-\#\mathsf{prop}$ mixes mode information with simple type information.

- **Alternative formulation:** $\downarrow\#\mathsf{prop} \to \mathsf{prop}$ (use a mode operator for the mode part)
- **Definition E.3.1.** $\mathcal{DLC}$ types are pairs $\mathbf{A}\#\alpha$, where
  - $\mathbf{A}$ is a mode specifier, α is a simple type; $\mathbf{A}$ is functional, iff α is.
- **Idea:** Use the simply typed λ-calculus for mode specifiers
- Other connectives (new version)
  - $\neg$ gets type $\lambda F.\downarrow F\#\mathsf{prop} \to \mathsf{prop}$
  - $\otimes$ gets type $\uplus\#\mathsf{prop} \to \mathsf{prop} \to \mathsf{prop}$
  - $\mathbb{V}$ gets type $\lambda FG.(\downarrow F \uplus \downarrow G)\#\mathsf{prop} \to \mathsf{prop} \to \mathsf{prop}$
  - $\Rrightarrow$ gets type $\lambda FG.\downarrow(F \uplus \downarrow G)\#\mathsf{prop} \to \mathsf{prop} \to \mathsf{prop}$

With this idea, we can re-interpret the DRT types from Example E.2.9.

A λ-Calculus for Mode Specifiers

- **Definition E.3.2.** New base type μ for modes; $\widetilde{\alpha}$ is α with ι, prop replaced by μ.
- mode specifiers $\mathbb{A}, \mathbb{B}, \mathbb{C}$ are simply typed λ-terms built up from mode variables $F, G, F^1, \ldots$ and
- **Definition E.3.3 (Mode constants).**
  - the empty mode $\emptyset$ of type μ
  - the elementary modes U^+, U^- and U° of type μ for all referents $U \in \mathcal{R}$
  - the mode functions $\cdot^+$, $\cdot^-$, $\downarrow\cdot$, $+\cdot$, and $-\cdot$ of type $\mu \to \mu$, and
  - the mode function $\uplus$ of type $\mu \to \mu \to \mu$.
- **Definition E.3.4.** Theory of mode equality specifies the meaning of mode constants (e.g. $(U^+, V^-, W^- \uplus U^-, V^+) \to_\mu U^+, V^+, W^-$)

Summary: DLC Grammar

- We summarize the setup in the following context-free grammar

$\alpha ::= \iota \mid o \mid \alpha_1 \to \alpha_2$	simple types
$\gamma ::= \mu \mid \gamma_1 \to \gamma_2$	mode types
$\mathbb{B} ::= \emptyset \mid U^+ \mid U^- \mid U^\circ \mid \mathbb{B}_1, \mathbb{B}_2 \mid \mathbb{B}_1 \uplus \mathbb{B}_2 \mid \downarrow\mathbb{B}$	basic modes
$\mathbb{M} ::= \mathbb{B} \mid \mathbb{M}_1\mathbb{M}_2 \mid \lambda F_\gamma.\mathbb{M}$	modes (typed via mode types γ)
$\tau ::= \mathbb{M}\#\alpha$	DLC types
$\mathbf{M} ::= U \mid c \mid \mathbf{M}_1\mathbf{M}_2 \mid \lambda X_\tau.\mathbf{M} \mid \delta U.\mathbf{M}$	DLC terms (typed via DLC types τ)

- But not all of these raw terms can be given a meaning $\leadsto$ only use those that can be shown to be well-typed. (up next)

Type Inference for $\mathcal{DLC}$ (two dimensions)

▷ **Definition E.3.5.** The type inference system for $\mathcal{DLC}$ consists of the following rules:

$$\frac{c\in\Sigma_\alpha}{\mathcal{A}\vdash_\Sigma c\colon\alpha}\qquad\frac{\mathcal{A}(X)=F\#\alpha\quad\mathcal{A}(F)=\widetilde{\alpha}}{\mathcal{A}\vdash_\Sigma X\colon F\#\alpha}\qquad\frac{U\in\mathcal{R}_\alpha,\mathcal{A}(U)=\emptyset\#\alpha}{\mathcal{A}\vdash_\Sigma U\colon U^{-}\#\alpha}$$

$$\frac{\mathcal{A},[X{:}F\#\beta],[F{:}\widetilde{\beta}]\vdash_\Sigma\mathbf{A}\colon\mathbb{A}\#\alpha}{\mathcal{A}\vdash_\Sigma\lambda X_{F\#\beta}.\mathbf{A}\colon\lambda F.\mathbb{A}\#\beta\to\alpha}\qquad\frac{\mathcal{A}\vdash_\Sigma\mathbf{A}\colon\mathbb{A}\#\beta\to\gamma\quad\mathcal{A}\vdash_\Sigma\mathbf{B}\colon\mathbb{B}\#\beta}{\mathcal{A}\vdash_\Sigma\mathbf{A}\ \mathbf{B}\colon\mathbb{A}(\mathbb{B})\#\gamma}$$

$$\frac{\mathcal{A}\vdash_\Sigma\mathbf{A}\colon\mathbb{A}\#\alpha\quad\mathcal{A}\vdash_\Sigma\mathbb{A}=_{\beta\eta\mu}\mathbb{B}}{\mathcal{A}\vdash_\Sigma\mathbf{A}\colon\mathbb{B}\#\alpha}\qquad\frac{\mathcal{A}\vdash_\Sigma\mathbf{A}\colon\lambda F.\mathbb{A}\#\alpha\quad\mathcal{A}\vdash_\Sigma\mathbb{A}\colon\mu}{\mathcal{A}\vdash_\Sigma\delta U_\beta.\mathbf{A}\colon\lambda F.(U^{+}\uplus\mathbb{A})\#\alpha}$$

where $\mathcal{A}$ is a variable context mapping variables and referents to types

Example (Identity)

▷ We have the following type derivation for the identity.

$$\frac{\overline{[F{:}\widetilde{\alpha}],[X{:}F\#\alpha]\vdash_\Sigma X\colon F\#\alpha}}{\vdash_\Sigma\lambda X_{F\#\alpha}.X\colon\lambda F_{\widetilde{\alpha}}.F\#\alpha\to\alpha}$$

▷ $(\lambda X_{F\#\alpha\to\alpha}.X)\ (\lambda X_{G\#\alpha}.X)$ has type

$$\mathcal{A}\vdash_\Sigma(\lambda F_{\mu\to\mu}.F)\ (\lambda G_\mu.G)\#\alpha\to\alpha=_{\beta\eta\mu}\lambda G_\mu.G\#\alpha\to\alpha$$

▷ **Theorem E.3.6 (Principal Types).** *For any given variable context $\mathcal{A}$ and formula $\mathbf{A}$, there is at most one type $\mathbb{A}\#\alpha$ (up to mode $\beta\eta\mu$-equality) such that $\mathcal{A}\vdash_\Sigma\mathbf{A}\colon\mathbf{A}\#\alpha$ is derivable in $\mathcal{DLC}$.*

Linguistic Example

▷ **Example E.3.7.** *No man sleeps.*

Assume $U{\in}\mathcal{R}_\iota$ and $\mathsf{man}, \mathsf{sleeps}{\in}\mathcal{R}_{\lambda F.F\#\iota\to\mathsf{prop}}$.

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathcal{A}\vdash_\Sigma \mathsf{man}(U) : U^-\#\mathsf{prop}}}{\mathcal{A}\vdash_\Sigma \delta U.\mathsf{man}(U) : U^+\#\mathsf{prop}} \qquad \dfrac{\vdots}{\mathcal{A}\vdash_\Sigma \mathsf{sleeps}(U) : U^-\#\mathsf{prop}}}{\dfrac{\dfrac{\mathcal{A}\vdash_\Sigma \delta U.\mathsf{man}(U)\wedge \mathsf{sleeps}(U) : U^+\uplus U^-\#\mathsf{prop}}{\mathcal{A}\vdash_\Sigma \neg(\delta U.\mathsf{man}(U)\wedge \mathsf{sleeps}(U)) : \downharpoonleft(U^+\uplus U^-)\#\mathsf{prop}}}{\mathcal{A}\vdash_\Sigma \neg(\delta U.\mathsf{man}(U)\wedge \mathsf{sleeps}(U)) : U^\circ\#\mathsf{prop}}}$$

A Further (Tricky) Example: $\mathbf{A}_\neg{:=}(\lambda X.X\wedge\neg X)$

▷ a referent declaration in the argument of $\mathbf{A}_\neg$ will be copied, and the two occurrences will have a different status
$\mathbf{A}_\neg\ (\delta U.\mathsf{man}(U)){\to_\beta}(\delta U.\mathsf{man}(U)\wedge\neg(\delta U.\mathsf{man}(U)))$

▷ assuming $\mathcal{A}(X) = F\#\mathsf{prop}$ gives

$$\dfrac{\dfrac{\mathcal{A}\vdash_\Sigma X : F\#\mathsf{prop} \qquad \dfrac{\mathcal{A}\vdash_\Sigma X : F\#\mathsf{prop}}{\mathcal{A}\vdash_\Sigma \neg X : \downharpoonleft F\#\mathsf{prop}}}{\mathcal{A}\vdash_\Sigma X\wedge\neg X : F\uplus\downharpoonleft F\#\mathsf{prop}}}{\mathcal{A}\vdash_\Sigma \lambda X.X\wedge\neg X : \lambda F.(F\uplus\downharpoonleft F)\#\mathsf{prop}\to\mathsf{prop}}$$

▷ thus, assuming $\mathcal{A}\vdash_\Sigma\delta U.\mathsf{man}(U) : U^+\#\mathsf{prop}$, we derive

$$\mathcal{A}\vdash_\Sigma \mathbf{A}_\neg\ (\delta U.\mathsf{man}(U)) : U^+, U^\circ\#\mathsf{prop}$$

A Further Example: Generalized Coordination

▷ We may define a generalised *and*:
$\lambda R^1 \ldots R^n.\lambda X^1 \ldots X^m.(R^1\ X^1\ \ldots\ X^m \otimes \ldots \otimes R^n\ X^1\ \ldots\ X^m)$
with type
$\lambda F^1 \ldots F^n.(F^1 \uplus \ldots \uplus F^n)\#\overline{\overline{\beta}_m \to \mathsf{prop}} \to \overline{\beta}_m \to \mathsf{prop}$

▷ thus from $\mathsf{john}{:=}(\lambda P.(\delta U.U = j\otimes P(U)))$
and $\mathsf{mary}{:=}(\lambda P.(\delta V.V = m\otimes P(V)))$

▷ we get $\mathsf{john}\text{and}\mathsf{mary} = \lambda P.(\delta U.U = j\otimes P(U)\otimes \delta V.V = m\otimes P(V))$

▷ combine this with *own a donkey*:

$$\lambda X.(\delta W.\mathsf{donkey}(W)\otimes\mathsf{own}(W,X)\otimes\delta U.U = j\otimes\delta W.\mathsf{donkey}(W)\otimes\mathsf{own}(W,U)\otimes\delta V.V = m\otimes\delta W.\mathsf{donkey}(W)\otimes\mathsf{own}($$

E.4 Modeling Higher-Order Dynamics

Discourse Variants $=_\delta$

▷ **Definition E.4.1.** We capture “referent renaming” in an equality judgment $=_\delta$.

▷ The order and multiplicity of introduction of discourse referents is irrelevant

- ▷ $\delta U.\delta V.\mathbf{A}=_\delta\delta V.\delta U.\mathbf{A}$
- ▷ $\delta U.\delta U.\mathbf{A}=_\delta\delta U.\mathbf{A}$.
- ▷ This is needed to model DRT, where discourse referents appear in sets.

▷ functional and dynamic binding can be interchanged

- ▷ $\lambda X.(\delta U.\mathbf{A})=_\delta\delta U.\lambda X.\mathbf{A}$
- ▷ This is useful for convenient $=_\eta$-long-forms (DHOU).

Renaming of Discourse Referents?

▷ Consider $\mathbf{A}{:=}(\lambda XY.Y)\;(\delta U.U)$

- ▷ δU cannot have any effect on the environment, since it can be deleted by $=_\beta$-reduction.
- ▷ $\mathbf{A}$ has type $\lambda F.F\#\alpha\to\alpha$ (U does not occur in it).

▷ **Idea:** Allow to rename U in $\mathbf{A}$, if “$\mathbf{A}$ is independent of U”

▷ Similar effect for $\mathbf{B}{:=}\neg(\delta U.\mathsf{man}(U))$, this should equal $\neg(\delta V.\mathsf{man}(V))$

▷ **Definition E.4.2.** ρ renaming is induced by the following inference rule:

$$\frac{V\in\mathcal{R}_\beta \text{ fresh} \quad U_\beta\notin\mathcal{DP}(\mathbf{A})}{\mathbf{A}=_\rho\mathcal{C}_U^V(\mathbf{A})}$$

Where $\mathcal{C}_U^V(\mathbf{A})$ is the result of replacing all referents U by V.

Dynamic Potential

▷ The binding effect of an expression $\mathbf{A}$ can be read off its modality $\mathbf{A}$

▷ A modality $\mathbf{A}$ may be simplified by $\beta\eta\mu$-reduction (where μ-equality reflects the semantics of the mode functions, e.g. $U^+ \uplus U^- =_\mu U^+$).

▷ **Definition E.4.3.** The dynamic binding potential of $\mathbf{A}$: $\mathcal{DP}(\mathbf{A}) := \{U | U^+ \in \mathrm{occ}(\mathbf{A}') \text{ or } U^- \in \mathrm{occ}(\mathbf{A}')\}$, where $\mathbf{A}'$ is the $\beta\eta\mu$-normal form of $\mathbf{A}$.

▷ **Definition E.4.4.** If $U \notin \mathcal{DP}(\mathbf{A})$, then U is called independent of $\mathbf{A}$.

Some Examples for Dynamic Potential

▷ **Example E.4.5.**

Formula	Modality	$\mathcal{DP}$
$\delta U.P$	U^+	$\{U\}$
$\lambda P.(\delta U.P)$	$\lambda F.(U^+ \uplus F)$	$\{U\}$
$\neg(\delta U.\mathrm{man}(U))$	U°	$\emptyset$
$\lambda P.\neg(\delta U.P)$	$\lambda F.\Downarrow(U^+), F$	$\{U\}$
$\lambda X.U$	$\lambda F.U^-$	$\{U\}$
$(\lambda X.X)\ U$	$(\lambda F.F)\ U^-$	$\{U\}$
$\lambda P.\mathrm{man}(U) \wedge P$	$\lambda F.(F \uplus U^-)$	$\{U\}$
$\lambda P.P$	$\lambda F.F$	$\emptyset$
$\lambda XY.Y$	$\lambda FG.G$	$\emptyset$
$(\lambda XY.Y)\ (\delta U.U)$	$\lambda G.G$	$\emptyset$
$\lambda P.P\ (\lambda Q.\neg(\delta U.Q))\ (\lambda R.(\delta U.R))$		$\{U\}$

Reductions

▷ $\beta\eta$-reduction: $\dfrac{}{(\lambda X.\mathbf{A})\ \mathbf{B} \rightarrow_\beta [\mathbf{B}/X](\mathbf{A})}$ and $\dfrac{X \notin \mathrm{free}(\mathbf{A})}{(\lambda X.\mathbf{A}\ X) \rightarrow_\eta \mathbf{A}}$

▷ **Definition E.4.6.** Dynamic Reduction: $\dfrac{\mathcal{A} \vdash_\Sigma \mathbf{A} : \mathbb{A}\#\alpha \quad U^+ \in \mathbf{Trans}(\mathbb{A})}{\mathbf{A}\ (\delta U.\mathbf{B}) \rightarrow_\tau (\delta U.\mathbf{A}\ \mathbf{B})}$

▷ **Example E.4.7.** Merge-Reduction $(\delta U.\mathbf{A} \otimes \delta V.\mathbf{B}) \rightarrow_\tau (\delta U.\delta V.(\mathbf{A} \otimes \mathbf{B}))$

▷ **Intuition:** The merge operator is just dynamic conjunction!

▷ **Observation:** Sequential merge $;;$ of type $\overset{\rightarrow}{\uplus}\ \#\mathrm{prop} \rightarrow \mathrm{prop} \rightarrow \mathrm{prop}$ does not transport V in the second argument.

E.5 Direct Semantics for Dynamic λ Calculus

Higher-Order Dynamic Semantics (Static Model)

- Frame $\mathcal{D} = \{\mathcal{D}_\alpha | \alpha \in \mathcal{T}\}$
  - $\mathcal{D}_\mu$ is the set of modes (mappings from variables to signs)
  - $\mathcal{D}_{\mathsf{prop}}$ is the set of truth values $\{\mathsf{T}, \mathsf{F}\}$.
  - $\mathcal{D}_\iota$ is an arbitrary universe of individuals.
  - $\mathcal{D}_{(\alpha \to \beta)} \subseteq \mathcal{D}_\alpha \to \mathcal{D}_\beta$
- Interpretation $\mathcal{I}$ of constants, assignment φ of variables.
  - $\mathcal{I}_\varphi(c) = \mathcal{I}(c)$, for a constant c
  - $\mathcal{I}_\varphi(X) = \varphi(X)$, for a variable X
  - $\mathcal{I}_\varphi(\mathbf{A}\ \mathbf{B}) = \mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
  - $\mathcal{I}_\varphi(\lambda X.\mathbf{B})(\mathsf{a}) = \mathcal{I}_{(\varphi,[\mathsf{a}/X])}(\mathbf{B})$.

Dynamic Semantics (Frames)

- **Two approaches:** "Dynamic" (Amsterdam) and "Static" (Saarbrücken)
  - Will show that they are equivalent (later)
  - Use the static semantics for $\mathcal{DLC}$ now.
- What is the denotation of a dynamic object?
  - "Static Semantics": essentially a set of states (considers only type prop) (equivalently function from states to $\mathcal{D}_{\mathsf{prop}}$: characteristic function)
  - generalize this to arbitrary base type: $\mathcal{D}_\alpha^\Gamma = \mathcal{B}_\Gamma \to \mathcal{D}_\alpha$, where $\mathcal{B}_\Gamma$ is the set of Γ-states
- Γ-states: well-typed referent assignments $s\colon \mathbf{Dom}^\pm(\Gamma) \to \mathcal{D}$ $s|\Delta$ is s coerced into a Δ-state.
- For expressions of functional type: $\mathcal{D}_{(\alpha\to\beta)}^\Phi = \bigcup_{\Psi \in \mathcal{D}_\alpha^\sim} \mathcal{D}_\alpha^\Psi \to \mathcal{D}_\beta^{\Phi(\Psi)}$

Dynamic Semantics (Evaluation)

- **Standard Tool:** Intensionalization (guards variables by delaying evaluation of current state)
- **Idea:** Ideal for semantics of variable capture
  - guard all referents
  - make this part of the semantics (thus implicit in syntax)
- Evaluation of variables and referents

- If $X{\in}\mathcal{V}$, then $\mathcal{I}_\varphi(X) = \varphi(X)$
- If $U{\in}\mathcal{R}$, then $\mathcal{I}_\varphi(U) = \Lambda s{\in}\mathcal{B}_{U^-}.s(U)$ (implicit intensionalization!)
- $\mathcal{I}_\varphi(\delta U.\mathbf{B}_{\mathbb{B}\#\beta}) = \Lambda s{\in}\mathcal{B}_{(\mathcal{I}_\varphi(\mathbb{B}_\mu)\uplus U^+)}.\mathcal{I}_\varphi(\mathbf{B})s|\mathcal{I}_\varphi(\mathbb{B}_\mu)$.
- $\mathcal{I}_\varphi(\mathbf{B}\ \mathbf{C}) = \mathcal{I}_\varphi(\mathbf{B})(\mathcal{I}_\varphi(\mathbf{C}))$.
- $\mathcal{I}_\varphi(\lambda X_\gamma.\mathbf{B}) = \Lambda^\Phi \mathsf{a}{\in}\mathcal{D}_\gamma^\Phi.\mathcal{I}_{(\varphi,[\mathsf{a}/X])}(\mathbf{B})$

▷ Referent names crucial in dynamic objects

▷ **Well actually:** $\mathcal{I}_\varphi(\delta U.\mathcal{B}_{(\Lambda\overline{F_n}.\mathbb{B}_\mu\#\beta)}) = \Lambda\overline{\mathsf{a}_n}.\Lambda s{\in}\mathcal{B}_{(\mathcal{I}_\varphi(\mathbb{B}_\mu)\uplus U^+)}.\mathcal{I}_\varphi(\mathbf{B})s|\mathcal{I}_\varphi(\mathcal{B}_\mu)$.

Metatheoretic Results

▷ **Theorem E.5.1 (Normalization).** *$\beta\eta\tau$-Reduction is terminating and confluent (modulo $\alpha\rho\delta$).*

▷ **Theorem E.5.2 (Substitution is type-preserving).** *If $X\notin\mathbf{dom}(\mathcal{A})$, then $\mathcal{A},[X{:}F\#\beta]\vdash_\Sigma\mathbf{A}:\mathbb{A}\#\alpha$ and $\mathcal{A}\vdash_\Sigma\mathbf{B}:\mathbb{B}\#\beta$ imply*

$$\mathcal{A}\vdash_\Sigma[\mathbf{B}/X](\mathbf{A}):[\mathbf{B}/F](\mathbb{A})\#\alpha$$

▷ **Theorem E.5.3 (Subject Reduction).** *If $\mathcal{A}\vdash_\Sigma\mathbf{A}:\mathbb{A}\#\alpha$ and $\mathcal{A}\vdash_\Sigma\mathbf{A}{=}_{\beta\eta\tau}\mathbf{B}$, then $\mathcal{A}\vdash_\Sigma\mathbf{B}:\mathbb{A}\#\alpha$.*

▷ **Theorem E.5.4 (Soundness of Reduction).** *If $\mathcal{A}\vdash_\Sigma\mathbf{A}{=}_{\alpha\beta\delta\eta\tau\rho}\mathbf{B}$, then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$.*

▷ If $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$, then $\mathcal{A}\vdash_\Sigma\mathbf{A}{=}_{\alpha\beta\delta\eta\tau\rho}\mathbf{B}$ (just needs formalisation of equality of logical operators.)

E.6 Dynamic λ Calculus outside Linguistics

Conclusion

▷ Basis for compositional discourse theories

- two-layered approach (only use theorem proving where necessary)
- functional and dynamic information can be captured structurally
- comprehensive equality theory (interaction of func. and dyn. part)

▷ In particular

- new dynamic primitives (explain others)
- simple semantics (compared to other systems)

- This leads to
  - dynamification of existing linguistic analyses (DHOU)
  - rigorous comparison of different dynamic systems (Meta-Logic)

Future Directions

- Generalize $\mathcal{DLC}$ to a true mode calculus:
  - turn δ into a logical constant δ_U: (use type declaration and application)

$$\frac{\mathcal{A}\vdash_\Sigma \mathbf{A}\colon \mathbb{A}\#\alpha}{\mathcal{A}\vdash_\Sigma \delta U_\beta.\mathbf{A}\colon U^+ \uplus \mathbb{A}_\mu\#\alpha} \qquad \frac{\vdash_\Sigma \delta_U\colon \lambda F.(U^+ \uplus F)\#\alpha \to \alpha \quad \mathcal{A}\vdash_\Sigma \mathbf{A}\colon \mathbb{A}\#\alpha}{\mathcal{A}\vdash_\Sigma \delta_U\ \mathbf{A}\colon U^+ \uplus \mathbb{A}_\mu\#\alpha}$$

  - this allows for more than one δ-like operator
- Better still (?) go for a dependent type discipline (implement in LF?)
- δ of type $\lambda UF.(U^+ \uplus F)\#\alpha \to \alpha$ yields $\delta(U) \widehat{=} \delta_U$

Use $\mathcal{DLC}$ as a model for Programming

- Remember dynamic binding in `Lisp`? `((lambda (F) (let ((U 1)) (F 1)))(lambda (X) (+ X U))` $\to$ 2 `((lambda (F) (let ((U 0)) (F 1)))(lambda (X) (+ X U))` $\to$ 1
- Ever wanted to determine the \\`$PRINTER`environment variable in a `Java` applet? (sorry forbidden, since the semantics of dynamic binding are unclear.)
- $\mathcal{DLC}$ is ideal for that (about time too!)
- **Example E.6.1 (LISP).** give `let`$_U$ the type $\lambda F.F\Uparrow^\circ_U$, where $(\mathbb{A}, U^-)\Uparrow^\circ_U = \mathbb{A}, U^\circ$. (no need for U^+ in `Lisp`)
- **Example E.6.2 (Java).** If you want to keep your `$EDITOR` variable private (pirated?) only allow applets of type $\mathbb{A}\#\alpha$, where `$EDITOR`$\notin \mathcal{DP}(\mathbb{A})$.
- It is going to be a lot of fun!

Appendix F

Model Existence and Completeness for Modal Logic

Abstract Consistency for $\mathcal{ML}^0$

▷ **Definition F.0.1.** If Φ is a set of propositions, then

$$\Box^-(\Phi) := \{\mathbf{A} \mid \Box\mathbf{A} \in \Phi\}$$

▷ **Definition F.0.2.** A collection ∇ of sets of $\mathcal{ML}^0$-formulae is called abstract consistency class for $\mathcal{ML}^0$, it if is closed under subsets and for all $\Phi \in \nabla$ we have

∇_c) $P \notin \Phi$ or $\neg P \notin \Phi$ for $P \in \mathcal{V}_0$

$\vdots$

$\nabla_\wedge$) $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \nabla$

$\nabla_\Box$) $\Diamond\mathbf{A} \in \Phi$ implies $\Box^-(\Phi) * \mathbf{A} \in \nabla$

∇-Hintikka Set

▷ **Definition F.0.3.** If ∇ is an abstract consistency class for $\mathcal{ML}^0$, then we call $\mathcal{H}$ a ∇-Hintikka set, if $\mathcal{H}$ maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H} * \mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.

▷ **Theorem F.0.4 (Extension Theorem).** *If ∇ is an abstract consistency class for ML and $\Phi \in \nabla$, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$.*

Proof:

1. chose an enumeration $\mathbf{A}_1, \mathbf{A}_2 \ldots$ of $\mathit{wff}_0(\mathcal{V}_0)$
2. construct sequence of sets H_i with $H_0 := \Phi$ and
   - ▷ $H_{n+1} := H_n$, if $H_n * \mathbf{A}_n \notin \nabla$
   - ▷ $H_{n+1} := H_n * \mathbf{A}_n$, if $H_n * \mathbf{A}_n \in \nabla$
3. All $H_i \in \nabla$, so choose $\mathcal{H} := \bigcup_{i \in \mathbb{N}} H_i$

4. $\Psi \subseteq \mathcal{H}$ finite implies that there is a $j \in \mathbb{N}$ with $\Psi \subseteq H_j$, so $\Psi \in \nabla$ as ∇ closed under subsets.
5. $\mathcal{H} \in \nabla$ since ∇ compact.
6. let $\mathcal{H} * \mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$
7. $\mathbf{B} \in H_{j+1} \subseteq \mathcal{H}$, so $\mathcal{H}$ ∇-maximal.

Canonical ∇-Model

▷ **Definition F.0.5.** If ∇ is an abstract consistency class, for ML^0, then we call $\mathcal{M}_\nabla := \langle \mathcal{W}_\nabla, \mathcal{R}_\nabla, \varphi_\nabla \rangle$ the canonical ∇ model, iff

- ▷ $\mathcal{W}_\nabla = \{\mathcal{H} | \mathcal{H} \in \nabla \text{maximal}\}$
- ▷ $\mathcal{R}_\nabla(v, w)$ iff $\Box^-(v) \subseteq w$
- ▷ $\varphi(P, w) = \mathsf{T}$ iff $P \in w$

▷ **Lemma F.0.6.** *If $w \in \mathcal{W}_\nabla$ and $\Diamond \mathbf{A} \in w$, then there is a $w' \in \mathcal{W}_\nabla$ with $\mathcal{R}_\nabla(w, w')$ and $\mathbf{A} \in w'$.*

▷ *Proof:* Let $\Diamond \mathbf{A} \in w$

1. thus $\Box^-(w) * \mathbf{A} \in \nabla$
2. by the extension theorem there is a $w' \in \mathcal{W}_\nabla$ with $\Box^-(w) * \mathbf{A} \subseteq w'$
3. so $\Box^-(w) \subseteq w'$ and thus $\mathcal{R}_\nabla(w, w')$.
4. on the other and we have $\mathbf{A} \in w'$.

Model existence for ML^0

▷ **Lemma F.0.7.** *If $w \in \mathcal{W}_\nabla$, then $\mathcal{I}^w_{\varphi_\nabla}(\mathbf{A}) = \mathsf{T}$ iff $\mathbf{A} \in w$.*

▷ *Proof:* Induction on the structure of $\mathbf{A}$

1. If $\mathbf{A}$ is a variable *then we get the assertion by the definition of φ_∇.*
2. If $\mathbf{A} = \neg \mathbf{B}$ and $\mathbf{A} \in w$ *then* $\mathbf{B} \notin w$, *thus* $\mathcal{I}^w_{\varphi_\nabla}(\mathbf{B}) = \mathsf{F}$, *and thus* $\mathcal{I}^w_{\varphi_\nabla}(\mathbf{A}) = \mathsf{T}$.
3. $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$ *analog*
4. $\mathbf{A} = \Box \mathbf{B}$
   - 4.1. Let $\mathbf{A} \in w$ and $w \mathcal{R}_\nabla w'$
   - 4.2. thus $\Box^-(w) \subseteq w'$ and thus $\mathbf{B} \in w'$
   - 4.3. so (IH) $\mathcal{I}^{w'}_{\varphi_\nabla}(\mathbf{B}) = \mathsf{T}$ for any such w'.
   - 4.4. and finally $\mathcal{I}^w_{\varphi_\nabla}(\mathbf{A}) = \mathsf{T}$
5. $\mathbf{A} = \Diamond \mathbf{B}$
   - 5.1. Let $\mathbf{A} \notin w$
   - 5.2. so $\neg \mathbf{A} = \Diamond \neg \mathbf{B} \notin w$
   - 5.3. and thus $\neg \mathbf{B} \in w'$ for some $w \mathcal{R}_\nabla w'$ by (Lemma1)
   - 5.4. so $\mathbf{B} \in w'$ and thus $\mathcal{I}^{w'}_{\varphi_\nabla}(\mathbf{B}) = \mathsf{T}$ by IH and finally $\mathcal{I}^w_{\varphi_\nabla}(\mathbf{A}) = \mathsf{T}$.

▷ **Theorem F.0.8 (Model existence).** *If ∇ is an abstract consistency class for ML^0 and $\Phi\in\nabla$, then there is a world $w\in\mathcal{W}_\nabla$ with $\mathcal{M}_\nabla \models^w \Phi$.*

Proof:

▷ 1. there is a ∇-Hintikka set $\mathcal{H} = w$ with $w\in\mathcal{W}_\nabla$ and $\Phi \subseteq \mathcal{H}$.
2. by Lemma 2 we have $\mathcal{I}_\varphi^w(\mathbf{A}) = \mathsf{T}$ for all $\mathbf{A}\in\Phi$.

Completeness

▷ **Theorem F.0.9.** *$\mathbb{K}$-consistency is an abstract consistency class for ML^0*

▷ *Proof:* Let $\Diamond\mathbf{A}\in\Phi$

1. To show: $\Box^-(\Phi)*\mathbf{A}$ is $\mathbb{K}$-consistent if Φ is $\mathbb{K}$-consistent
2. converse: Φ is $\mathbb{K}$-inconsistent if $\Box^-(\Phi)*\mathbf{A}$ $\mathbb{K}$-inconsistent.
3. There is a finite subset $\Psi \subseteq \Box^-(\Phi)$ with $\Psi\vdash_{\mathbb{K}}(\neg\mathbf{A})$
4. $(\Box\Psi)\vdash_{\mathbb{K}}(\Box\neg\mathbf{A})$ (distributivity of $\Box$)
5. $\Phi\vdash_{\mathbb{K}}(\Box\neg\mathbf{A}) = \neg(\Diamond\mathbf{A})$ since $\Box\Psi \subseteq \Phi$
6. thus Φ is $\mathbf{K}$-inconsistent.

▷ **Corollary F.0.10.** *$\mathbb{K}$ is complete wrt. Kripke models*

Further Completeness Theorems

▷ **Theorem F.0.11.** *$\mathbb{T}$-consistency is an abstract consistency class for ML^0 and $\mathcal{R}_{\mathbb{T}}$ is reflexive.*

▷ *Proof:* Let $\mathbf{A}\in\Box^-(w)$

1. then $\Box\mathbf{A}\in w$ by definition
2. with $\mathbb{T}$ $(\Box\mathbf{A} \Rightarrow \mathbf{A})$ and Modus Ponens we have $\mathbf{A}\in w$.
3. Thus $\Box^-(w) \subseteq w$ and $w\mathcal{R}_{\mathbb{T}}w$ for all $w\in\mathcal{W}_{\mathbb{T}}$.

▷ **Theorem F.0.12.** *$\mathbb{S}4$-consistency is an abstract consistency class for ML^0 and $\mathcal{R}_{\mathbb{S}4}$ is transitive.*

Proof: Let $w_1\mathcal{R}_{\mathbb{S}4}w_2\mathcal{R}_{\mathbb{S}4}w_3$ and $\Box\mathbf{A}\in w$.

▷ 1. by $\mathbb{S}4$ $(\Box\mathbf{A} \Rightarrow \Box\Box\mathbf{A})$ and Modus Ponens we have $\Box\Box\mathbf{A}\in w_1$.
2. and thus $\Box\mathbf{A}\in w_2 = \Box^-(w_1)$ and $\mathbf{A}\in w_3 = \Box^-(w_2)$.
3. Thus $\Box^-(w_1) \subseteq w_3$ and $w_1\mathcal{R}_{\mathbb{S}4}w_3$.

▷ **Corollary F.0.13.** *$\mathbb{T}$ ($\mathbb{S}4$) is complete wrt. reflexive (reflexive transitive) Kripke-models*