Computational Logic
320441 CompLog Lecture Notes

Michael Kohlhase

School of Engineering & Science
Jacobs University, Bremen Germany
m.kohlhase@jacobs-university.de
office: Room 168@Research 1, phone: x3140

November 18, 2015

Preface

Introduction

The ability to *represent knowledge about the world* and to *draw logical inferences* is one of the central components of intelligent behavior. As a consequence, reasoning components of some form are at the heart of many artificial intelligence systems.

Logic: The field of *logic* studies representation and inference systems. It dates back and has its roots in Greek philosophy as presented in the works of Aristotle and others. Since then logic has grown in richness and diversity over the centuries to finally reach the modern methodological approach first expressed in the work of Frege. Logical calculi, which capture an important aspect of human thought, were now amenable to investigation with mathematical rigour and the beginning of this century saw the influence of these developments in the foundations of mathematics, in the work of Hilbert, Russell and Whitehead, in the foundations of syntax and semantics of language, and in philosophical foundations expressed most vividly by the logicians in the Vienna Circle.

Computational Logic: The field of *Computational Logic* looks at computational aspects of logic. It is essentially the computer-science perspective of logic. The idea is that logical statements can be executed on a machine. This has far-reaching consequences that ultimately lead to logic programming, deduction systems for mathematics and engineering, logical design and verification of computer software and hardware, deductive databases and software synthesis as well as logical techniques for analysis in the field of mechanical engineering.

Logic Engineering: As all of these applications require efficient implementations of the underlying inference systems, computational logic focuses on *proof theory* much more than on *model theory* (which is the focus of mathematical logic, a neighboring field). As the respective applications have different requirements on the expressivity and structure of the representation language and on the statements derived or the terms simplified, computational logic focuses on "logic engineering", i.e. the development of representation languages, inference systems, and module systems with specific properties.

Course Concept

Aims: The course 320441 "Computational Logic" (CompLog) is a specialization course offered to third-year undergraduate students and to first-year graduate students at Jacobs University Bremen. The course aims to give these students a solid (and somewhat theoretically oriented) foundation of computational logic and logic engineering techniques.

Prerequisites: The course makes very little assumptions about prior knowledge, but the learning curve is very steep for students who have no prior exposure to logic. As a consequence, the course has a prerequisite to the course 320211 *Formal Languages and Logic* which is a mandatory course in the Computer Science program at Jacobs University. This prerequisite can be waived by the instructor for other students.

Course Contents: We carefully recap the foundations of first-order logic and present the tableau calculus as a computationally inspired inference procedure. Free variable tableaux also introduce unification, and important computational tool in logics. Finally, we introduce the model existence method for proving completeness of calculi.

The next part of the course is about enhancing the expressivity of first-order logic to include functions, predicates, and sets. The intended application is a more adequate representation of mathematical concepts, where these objects are common. Here we introduce the simply typed λ calculus as the main representational vehicle, since it casts function comprehension into an equational theory which we show to be terminating, confluent, and complete. Thus we can build higher-order inference by extending unification and tableaux.

Finally, we explore the realm of decidable logics used for knowledge representation these days. These description logics specialize on representing concepts, and their relations and reasoning about them. Here the game is to add new operators to the language and extend the reasoning algorithms for them without losing decidability and tractability. We present the foundations of knowledge representation starting from semantic networks, over propositional logic with a set-description semantics to ALC, which achieves feature-parity with semantic networks, but has a strong formal basis and well-understood, decision procedures. We conclude the course with a quick walk through the ALC extensions and relate this to the current Semantic Web standards.

This Document

This document contains the course notes for the course Computational Logic held at Jacobs University Bremen in the fall semesters 2004/07/09/11/13/14.

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.

Knowledge Representation Experiment:

This document is also an experiment in knowledge representation. Under the hood, it uses the sTeX package [Koh08, Koh15], a TeX/LaTeX extension for semantic markup, which allows to export the contents into the eLearning platform PantaRhei.

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

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

Acknowledgments

CompLog Students: The following students have submitted corrections and suggestions to this and earlier versions of the notes: Rares Ambrus, Florian Rabe, Deyan Ginev, Fulya Horozal.

Recorded Syllabus for Fall 2015

In this document, we record the progress of the course in Fall 2015 in the form of a "recorded syllabus", i.e. a syllabus that is created after the fact rather than before.

Recorded Syllabus Fall Semester 2014:

#	date	until	slide	page

Here the syllabus of last year's course for reference, the one should be similar.

Recorded Syllabus Fall Semester 2013:

#	date	until	
1	Sep 3.	admin/intro/history	
2	Sep 8.	more overview	
3	Sep 15.	First-Order Semantics	
4	Sep 17.	Capture-Avoiding substitution	
5	Sep 22.	First-Order ND Calculus	
7	Sep 29.	Model Existence	
8	Oct 1.	efficient unification	
9	Oct 6.	tableau soundness/completeness	
10	Oct 8.	Higher-Order predicate Logic	
11	Oct 13.	λ-calculus & Head Reduction	
12	Oct 15.	Termination of β-reduction	
13	Oct 22.	$\alpha\beta\eta$-completeness, Simple Type Theory	
14	Oct 27	Logical Constants for Math	
15	Oct. 29	HOU & General Bindings	
16	Nov 3.	HOU completeness	
17	Nov 5.	Andrews' equality-based HOL	
18	Nov 10.	Higher-Order Tableaux	
19	Nov 12.	Recap FO Tableaux and Course Planning	
20	Nov 17.	Semantic Networks/Semantic Web	
20	Nov 19.	PL0 set description	
21	Nov 24.	ALC intro	
22	Nov 26.	ALC Inference	
23	Dec 1.	Semantic Web & number restrictions	
24	Dec 3.	More ALC extensions: Role Operations	

Contents

Chapter 1

Outline of the Course

In this course, we want to achieve three things: we want to

1) expose you to various logics from a computational perspective, in particular
2) teach you how to build up logics and and express domain theories modularly, and
3) apply that to the foundations of mathematics and of the Semantic Web.

Outline: From Classical Logic to Specialized Inference Procedures

- Recap: First-order Logic (cosolidation)
  - special attention to substitutions, α-renaming (usually glossed over)
  - soundness/completeness (interesting proofs)
  - tableau calculi, unification (basis for later)
- Higher-Order Logic (more expressivity for math)
  - simply typed λ calculus
  - soundness, confluence, termination, completeness
  - higher-order unification?
  - higher-order tableaux
- Axiomatic Set Theory
- Description Logics (expressivity below)
  - propositional logic for concept descriptions
  - ALC+ extensions
  - tableau calculi

 1 JACOBS UNIVERSITY

Chapter 2

320411/CompLog 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 learning as efficient and painless as possible.

Even though the lecture itself will be the main source of information in the course, there are various resources from which to study the material.

Textbooks, Handouts and Information, Forum

- No required textbook, but course notes, posted slides
- Course notes in PDF will be posted at `http://old.kwarc.info/teaching/CompLog.html`
- Everything will be posted on PantaRhei (notes+assignments+course forum)
  - announcements, contact information, course schedule and calendar
  - discussion among your fellow students (careful, we check for academic integrity!)
  - `http://panta.kwarc.info` (use your Jacobs login)
  - if there are problems send e-mail to me.

No Textbook: There is no single textbook that covers the course. Instead we have a comprehensive set of course notes (this document). They are provided in two forms: as a large PDF that is posted at the course web page and on the PantaRhei system. The latter is actually the preferred method of interaction with the course materials, since it allows to discuss the material in place, to play with notations, to give feedback, etc. The PDF file is for printing and as a fallback, if the PantaRhei system, which is still under development, develops problems.

But of course, there is a wealth of literature on the subject of computational logic, and the references at the end of the lecture notes can serve as a starting point for further reading. We will try to point out the relevant literature throughout the notes.

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

Prerequisites, Requirements, Grades

▷ Prerequisites: Motivation, Interest, Curiosity, hard work (mainly,...)

- ▹ exposure to discrete Math, possibly category theory
- ▹ experience in (some) logics

You can do this course if you want! (even without those, but they help)

▷ Grades: (plan your work involvement carefully)

Attendance and Wakefulness	10%
Homework Assignments	60%
Quizzes	30%
No Midterm Exam	–
No Final Exam	–

In particular, no midterm, and no final in the Lab, but attendance is mandatory! (excuses possible)

▷ Note that for the grades, the percentages of achieved points are added with the weights above, and only then the resulting percentage is converted to a grade.

 3 JACOBS UNIVERSITY

Our main motivation in this grading scheme is to entice you to study continuously. This means that you will have to stay involved, do all your homework assignments, and keep abreast with the course. This also means that your continued involvement may be higher than other (graduate) courses, but you are free to concentrate on these during exam week.

Homework assignments

▷ Goal: Reinforce and apply what is taught in class.

▷ Homeworks: will be small individual problem/programming/proof assignments (but take time to solve) group submission if and only if explicitly permitted

▷ Admin: To keep things running smoothly

- ▹ Homeworks will be posted on PantaRhei
- ▹ Homeworks are handed in electronically in `JGrader`(plain text, Postscript, PDF,...)
- ▹ go to the tutorials, discuss with your TA (they are there for you!)
- ▹ materials: sometimes posted ahead of time; then read before class, prepare questions, bring printout to class to take notes

▷ 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.

Homework assignments are a central part of the course, they allow you to review the concepts covered in class, and practice using them. They are usually directly based on concepts covered in the lecture, so reviewing the course notes often helps getting started.

Homework Submissions, Grading, Tutorials

- ▷ Submissions: We use Heinrich Stamerjohanns' `JGrader` system
  - ▷ submit all homework assignments electronically to `https://jgrader.de`.
  - ▷ you can login with your Jacobs account and password. (should have one!)
  - ▷ feedback/grades to your submissions
  - ▷ get an overview over how you are doing! (do not leave to midterm)
- ▷ Tutorials: select a tutorial group and actually go to it regularly
  - ▷ to discuss the course topics after class (lectures need pre/postparation)
  - ▷ to discuss your homework after submission (to see what was the problem)
  - ▷ to find a study group (probably the most determining factor of success)

The next topic is very important, you should take this very seriously, even if you think that this is just a self-serving regulation made by the faculty.

All societies have their rules, written and unwritten ones, which serve as a social contract among its members, protect their interestes, and optimize the functioning of the society as a whole. This is also true for the community of scientists worldwide. This society is special, since it balances intense cooperation on joint issues with fierce competition. Most of the rules are largely unwritten; you are expected to follow them anyway. The code of academic integrity at Jacobs is an attempt to put some of the aspects into writing.

It is an essential part of your academic education that you learn to behave like academics, i.e. to function as a member of the academic community. Even if you do not want to become a scientist in the end, you should be aware that many of the people you are dealing with have gone through an academic education and expect that you (as a graduate of Jacobs) will behave by these rules.

The Code of Academic Integrity

- ▷ Jacobs has a "Code of Academic Integrity"
  - ▷ this is a document passed by the Jacobs community (our law of the university)
  - ▷ you have signed it during enrollment (we take this seriously)
- ▷ It mandates good behaviors from both faculty and students and penalizes bad ones:
  - ▷ honest academic behavior (we don't cheat/falsify)
  - ▷ respect and protect the intellectual property of others (no plagiarism)
  - ▷ treat all Jacobs members equally (no favoritism)

▷ this is to protect you and build an atmosphere of mutual respect

- ▹ academic societies thrive on reputation and respect as primary currency

▷ The Reasonable Person Principle (one lubricant of academia)

- ▹ we treat each other as reasonable persons
- ▹ the other's requests and needs are reasonable until proven otherwise
- ▹ but if the other violates our trust, we are deeply disappointed (severe uncompromising consequences)

To understand the rules of academic societies it is central to realize that these communities are driven by economic considerations of their members. However, in academic societies, the primary good that is produced and consumed consists in ideas and knowledge, and the primary currency involved is academic reputation[1]. Even though academic societies may seem as altruistic — scientists share their knowledge freely, even investing time to help their peers understand the concepts more deeply — it is useful to realize that this behavior is just one half of an economic transaction. By publishing their ideas and results, scientists sell their goods for reputation. Of course, this can only work if ideas and facts are attributed to their original creators (who gain reputation by being cited). You will see that scientists can become quite fierce and downright nasty when confronted with behavior that does not respect other's intellectual property.

Next we come to a special project that is going on in parallel to teaching the course. I am using the coures materials as a research object as well. This gives you an additional resource, but may affect the shape of the coures materials (which now server 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: E-Learning with *OMDoc*/PantaRhei

▷ My research area: deep representation formats for (mathematical) knowledge

▷ Application: E-learning systems (represent knowledge to transport it)

▷ Experiment: Start with this course (Drink my own medicine)

- ▹ Re-Represent the slide materials in *OMDoc* (Open Math Documents)
- ▹ Feed it into the PantaRhei system (`http://panta.kwarc.info`)
- ▹ Try it on you all (to get feedback from you)

▷ Tasks (Unfortunately, I cannot pay you for this; maybe later)

- ▹ 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)

[1] Of course, this is a very simplistic attempt to explain academic societies, and there are many other factors at work there. For instance, it is possible to convert reputation into money: if you are a famous scientist, you may get a well-paying job at a good university,. . .

▷ you will help build better course materials (think of next-year's students)

Chapter 3

What is (Computational) Logic

What is (Computational) Logic?

- ▷ The field of logic studies representation languages, inference systems, and their relation to the world.
- ▷ It dates back and has its roots in Greek philosophy (Aristotle et al.)
- ▷ Logical calculi capture an important aspect of human thought, and make it amenable to investigation with mathematical rigour, e.g. in
  - ▷ foundation of mathematics (Hilbert, Russell and Whitehead)
  - ▷ foundations of syntax and semantics of language(Creswell, Montague, ...)
- ▷ Logics have many practical applications
  - ▷ logic/declarative programming (the third programming paradigm)
  - ▷ program verification: specify conditions in logic, prove program correctness
  - ▷ program synthesis: prove existence of answers constructively, extract program from proof
  - ▷ proof-carrying code: compiler proves safety conditions, user verifies before running.
  - ▷ deductive databases: facts + rules (get more out than you put in)
  - ▷ semantic web: the Web as a deductive database
- ▷ Computational Logic is the study of logic from a computational, proof-theoretic perspective. (model theory is mostly comprised under "mathematical logic".)

 8

What is Logic?

- ▷ 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 ($x, y, f, g, p, 1, \pi, \in, \neg, \wedge \forall, \exists$)

- ▷ Models: things we understand (e.g. number theory)
- ▷ Interpretation: maps formulae into models ($[\![\text{three plus five}]\!] = 8$)
- ▷ Validity: $\mathcal{M} \models \mathbf{A}$, iff $[\![\mathbf{A}]\!]^{\mathcal{M}} = \mathsf{T}$ (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 Wilhelm Schickard (1623), Blaise Pascal (1642), and Gottfried Wilhelm Leibniz (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 logical connective) and a small calculus that gives us a set of rules how to manipulate formulae.

3.1 A History of Ideas in Logic

Before starting with the discussion on particular logics and inference systems, we put things into perspective by previewing ideas in logic from a historical perspective. Even though the presentation (in particular syntax and semantics) may have changed over time, the underlying ideas are still pertinent in today's formal systems.

Many of the source texts of the ideas summarized in this Section can be found in [vH67].

History of Ideas (abbreviated): Propositional Logic

- ▷ General Logic ([ancient Greece, e.g. Aristotle])
  - + conceptual separation of syntax and semantics
  - + system of inference rules ("Syllogisms")
  - – no formal language, no formal semantics
- ▷ Propositional Logic [Boole $\sim$ 1850]
  - + functional structure of formal language (propositions + connectives)
  - + mathematical semantics ($\leadsto$ Boolean Algebra)
  - – abstraction from internal structure of propositions

History of Ideas (continued): Predicate Logic

- ▷ Frege's "Begriffsschrift" [Fre79]
  - + functional structure of formal language (terms, atomic formulae, connectives, quantifiers)
  - – weird graphical syntax, no mathematical semantics
  - – paradoxes e.g. Russell's Paradox [R. 1901] (the set of sets that do not contain themselves)
- ▷ modern form of predicate logic [Peano $\sim$ 1889]
  - + modern notation for predicate logic ($\vee, \wedge, \Rightarrow, \forall, \exists$)

History of Ideas (continued): First-Order Predicate Logic

- ▷ Types ([Russell 1908])
  - – restriction to well-types expression
  - + paradoxes cannot be written in the system
  - + Principia Mathematica ([Whitehead, Russell 1910])
- ▷ Identification of first-order Logic ([Skolem, Herbrand, Gödel $\sim$ 1920 – '30])
  - – quantification only over individual variables (cannot write down induction principle)
  - + correct, complete calculi, semi-decidable

+ set-theoretic semantics ([Tarski 1936])

History of Ideas (continued): Foundations of Mathematics

▷ Hilbert's Program: find logical system and calculus, ([Hilbert ~ 1930])

- ▷ that formalizes all of mathematics
- ▷ that admits sound and complete calculi
- ▷ whose consistence is provable in the system itself

▷ Hilbert's Program is impossible! ([Gödel 1931])

Let $\mathcal{L}$ be a logical system that formalizes arithmetics ($\langle NaturalNumbers, +, *\rangle$),

- ▷ then $\mathcal{L}$ is incomplete
- ▷ then the consistence of $\mathcal{L}$ cannot be proven in $\mathcal{L}$.

History of Ideas (continued): λ-calculus, set theory

▷ Simply typed λ-calculus ([Church 1940])

+ simplifies Russel's types, λ-operator for functions
+ comprehension as β-equality (can be mechanized)
+ simple type-driven semantics (standard semantics $\leadsto$ incompleteness)

▷ Axiomatic set theory

+– type-less representation (all objects are sets)
+ first-order logic with axioms
+ restricted set comprehension (no set of sets)
– functions and relations are derived objects

Part I

Formal Systems

To prepare the ground for the particular developments coming up, let us spend some time on recapitulating the basic concerns of formal systems.

Chapter 4

Logical Systems

The notion of a logical system is at the basis of the field of logic. In its most abstract form, a logical system consists of a formal language, a class of models, and a satisfaction relation between models and expressions of the formal lanugage. The satisfaction relation tells us when an expression is deemed true in this model.

Logical Systems

▷ **Definition 4.0.1** A logical system is a triple $\mathcal{S} := \langle\mathcal{L}, \mathcal{K}, \models\rangle$, where $\mathcal{L}$ is a formal language, $\mathcal{K}$ is a set and $\models \subseteq \mathcal{K} \times \mathcal{L}$. Members of $\mathcal{L}$ are called formulae of $\mathcal{S}$, members of $\mathcal{K}$ models for $\mathcal{S}$, and $\models$ the satisfaction relation.

▷ **Definition 4.0.2** Let $\mathcal{S} := \langle\mathcal{L}, \mathcal{K}, \models\rangle$ be a logical system, $\mathcal{M} \in \mathcal{K}$ be a model and $\mathbf{A} \in \mathcal{L}$ a formula, then we call $\mathbf{A}$

- ▷ satisfied by $\mathcal{M}$, iff $\mathcal{M} \models \mathbf{A}$
- ▷ falsified by $\mathcal{M}$, iff $\mathcal{M} \not\models \mathbf{A}$
- ▷ satisfiable in $\mathcal{K}$, iff $\mathcal{M} \models \mathbf{A}$ for some model $\mathcal{M} \in \mathcal{K}$.
- ▷ valid in $\mathcal{K}$ (write $\models\mathcal{M}$), iff $\mathcal{M} \models \mathbf{A}$ for all models $\mathcal{M} \in \mathcal{K}$
- ▷ falsifiable in $\mathcal{K}$, iff $\mathcal{M} \not\models \mathbf{A}$ for some $\mathcal{M} \in \mathcal{K}$.
- ▷ unsatisfiable in $\mathcal{K}$, iff $\mathcal{M} \not\models \mathbf{A}$ for all $\mathcal{M} \in \mathcal{K}$.

▷ **Definition 4.0.3** Let $\mathcal{S} := \langle\mathcal{L}, \mathcal{K}, \models\rangle$ be a logical system, then we define the entailment relation $\models \subseteq \mathcal{L} \times \mathcal{L}$. We say that $\mathbf{A}$ entails $\mathbf{B}$ (written $\mathbf{A} \models \mathbf{B}$), iff we have $\mathcal{M} \models \mathbf{B}$ for all models $\mathcal{M} \in \mathcal{K}$ with $\mathcal{M} \models \mathbf{A}$.

▷ **Observation 4.0.4** $\mathbf{A} \models \mathbf{B}$ *and* $\mathcal{M} \models \mathbf{A}$ *imply* $\mathcal{M} \models \mathbf{B}$.

 15 JACOBS UNIVERSITY

Example 4.0.5 (First-Order Logic as a Logical System) Let $\mathcal{L} := \mathit{wff}_o(\Sigma)$, $\mathcal{K}$ be the class of first-order models, and $\mathcal{M} \models \mathbf{A} :\Leftrightarrow \mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, then $\langle\mathcal{L}, \mathcal{K}, \models\rangle$ is a logical system in the sense of Definition 4.0.1.

Note that central notions like the entailment relation (which is central for understanding reasoning processes) can be defined independently of the concrete compositional setup we have used for first-order logic, and only need the general assumptions about logical systems.

Let us now turn to the syntactical counterpart of the entailment relation: derivability in a calculus. Again, we take care to define the concepts at the general level of logical systems.

Chapter 5

Calculi, Derivations, and Proofs

The intuition of a calculus is that it provides a set of syntactic rules that allow to reason by considering the form of propositions alone. Such rules are called inference rules, and they can be strung together to derivations — which can alternatively be viewed either as sequences of formulae where all formulae are justified by prior formulae or as trees of inference rule applications. But we can also define a calculus in the more general setting of logical systems as an arbitrary relation on formulae with some general properties. That allows us to abstract away from the homomorphic setup of logics and calculi and concentrate on the basics.

Derivation Systems and Inference Rules

- **Definition 5.0.1** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a relation $\vdash \subseteq \mathcal{P}(\mathcal{L}) \times \mathcal{L}$ a derivation relation for $\mathcal{S}$, if it
  - is proof-reflexive, i.e. $\mathcal{H} \vdash \mathbf{A}$, if $\mathbf{A} \in \mathcal{H}$;
  - is proof-transitive, i.e. if $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H}' \cup \{\mathbf{A}\} \vdash \mathbf{B}$, then $\mathcal{H} \cup \mathcal{H}' \vdash \mathbf{B}$;
  - admits weakening, i.e. $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash \mathbf{A}$.
- **Definition 5.0.2** We call $\langle \mathcal{L}, \mathcal{K}, \models, \vdash \rangle$ a formal system, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and $\vdash$ a derivation relation for $\mathcal{S}$.
- **Definition 5.0.3** Let $\mathcal{L}$ be a formal language, then an inference rule over $\mathcal{L}$

$$\frac{\mathbf{A}_1 \quad \cdots \quad \mathbf{A}_n}{\mathbf{C}} \mathcal{N}$$

  where $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{C}$ are formula schemata for $\mathcal{L}$ and $\mathcal{N}$ is a name.
  The $\mathbf{A}_i$ are called assumptions, and $\mathbf{C}$ is called conclusion.
- **Definition 5.0.4** An inference rule without assumptions is called an axiom (schema).
- **Definition 5.0.5** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a set $\mathcal{C}$ of inference rules over $\mathcal{L}$ a calculus for $\mathcal{S}$.

With formula schemata we mean representations of sets of formulae, we use boldface uppercase letters as (meta)-variables for formulae, for instance the formula schema $\mathbf{A} \Rightarrow \mathbf{B}$ represents the set of formulae whose head is $\Rightarrow$.

Derivations and Proofs

▷ **Definition 5.0.6** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and $\mathcal{C}$ a calculus for $\mathcal{S}$, then a $\mathcal{C}$-derivation of a formula $\mathbf{C} \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses (write $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{C}$) is a sequence $\mathbf{A}_1, \ldots, \mathbf{A}_m$ of $\mathcal{L}$-formulae, such that

- ▷ $\mathbf{A}_m = \mathbf{C}$, (derivation culminates in $\mathbf{C}$)
- ▷ for all $1 \leq i \leq m$, either $\mathbf{A}_i \in \mathcal{H}$, or (hypothesis)
- ▷ there is an inference rule $\frac{\mathbf{A}_{l_1} \quad \cdots \quad \mathbf{A}_{l_k}}{\mathbf{A}_i}$ in $\mathcal{C}$ with $l_j < i$ for all $j \leq k$. (rule application)

Observation: We can also see a derivation as a tree, where the $\mathbf{A}_{l_j}$ are the children of the node $\mathbf{A}_k$.

⊳ **Example 5.0.7** In the propositional Hilbert calculus $\mathcal{H}^0$ we have the derivation $P \vdash_{\mathcal{H}^0} Q \Rightarrow P$: the sequence is $P \Rightarrow Q \Rightarrow P, P, Q \Rightarrow P$ and the corresponding tree on the right.

$$\dfrac{\dfrac{}{P \Rightarrow Q \Rightarrow P}K \qquad P}{Q \Rightarrow P}MP$$

▷ **Observation 5.0.8** *Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and $\mathcal{C}$ a calculus for $\mathcal{S}$, then the $\mathcal{C}$-derivation relation $\vdash_{\mathcal{D}}$ defined in Definition 5.0.6 is a derivation relation in the sense of Definition 5.0.1.*[a]

▷ **Definition 5.0.9** We call $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$ a formal system, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{S}$.

▷ **Definition 5.0.10** A derivation $\emptyset \vdash_{\mathcal{C}} \mathbf{A}$ is called a proof of $\mathbf{A}$ and if one exists (write $\vdash_{\mathcal{C}} \mathbf{A}$) then $\mathbf{A}$ is called a $\mathcal{C}$-theorem.

▷ **Definition 5.0.11** an inference rule $\mathcal{I}$ is called admissible in $\mathcal{C}$, if the extension of $\mathcal{C}$ by $\mathcal{I}$ does not yield new theorems.

   JACOBS UNIVERSITY

[a] EdNote: MK: this should become a view!

Inference rules are relations on formulae represented by formula schemata (where boldface, uppercase letters are used as meta-variables for formulae). For instance, in Example 5.0.7 the inference rule $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$ was applied in a situation, where the meta-variables $\mathbf{A}$ and $\mathbf{B}$ were instantiated by the formulae P and $Q \Rightarrow P$.

As axioms do not have assumptions, they can be added to a derivation at any time. This is just what we did with the axioms in Example 5.0.7.

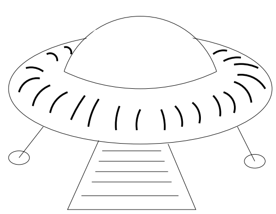
Chapter 6

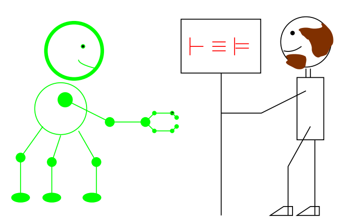
Properties of Calculi

In general formulae can be used to represent facts about the world as propositions; they have a semantics that is a mapping of formulae into the real world (propositions are mapped to truth values.) We have seen two relations on formulae: the entailment relation and the deduction relation. The first one is defined purely in terms of the semantics, the second one is given by a calculus, i.e. purely syntactically. Is there any relation between these relations?

Soundness and Completeness

- **Definition 6.0.1** Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a calculus $\mathcal{C}$ for $\mathcal{S}$
  - sound (or correct), iff $\mathcal{H} \models \mathbf{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, and
  - complete, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, whenever $\mathcal{H} \models \mathbf{A}$.
- Goal: $\vdash \mathbf{A}$ iff $\models \mathbf{A}$ (provability and validity coincide)
  - To TRUTH through PROOF (CALCULEMUS [Leibniz ~1680])





Ideally, both relations would be the same, then the calculus would allow us to infer all facts that can be represented in the given formal language and that are true in the real world, and only those. In other words, our representation and inference is faithful to the world.

A consequence of this is that we can rely on purely syntactical means to make predictions about the world. Computers rely on formal representations of the world; if we want to solve a problem on our computer, we first represent it in the computer (as data structures, which can be seen as a formal language) and do syntactic manipulations on these structures (a form of calculus). Now, if the provability relation induced by the calculus and the validity relation coincide (this will be quite difficult to establish in general), then the solutions of the program will be correct, and we will find all possible ones.

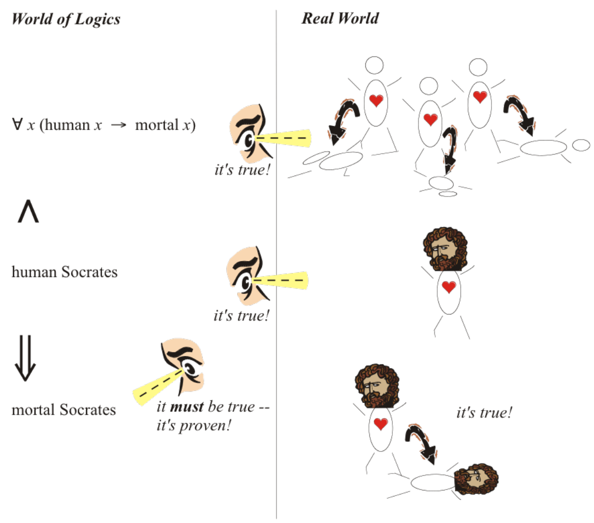
Of course, the logics we have studied so far are very simple, and not able to express interesting facts about the world, but we will study them as a simple example of the fundamental problem of Computer Science: How do the formal representations correlate with the real world.

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

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

The miracle of logics

▷ Purely formal derivations are true in the real world!



 19 JACOBS UNIVERSITY

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

Part II

First-Order Logic and Inference

Chapter 7

First-Order Logic

First-order logic is the most widely used formal system 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, at least three, some,...*)

 20 JACOBS UNIVERSITY

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 Chapter 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.

7.1 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 7.1.1** First-order logic (PL^1), is a formal logical 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; sometimes annotated by type o (like in PL^0)
  - individuals; sometimes annotated by type ι(numbers, foxes, Pokémon,...)
- **Definition 7.1.2** A first-order signature consists of (all disjoint; $k \in \mathbb{N}$)
  - connectives: $\Sigma^o = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$ (functions on truth values)
  - function constants: $\Sigma^f_k = \{f, g, h, \ldots\}$ (functions on individuals)
  - predicate constants: $\Sigma^p_k = \{p, q, r, \ldots\}$ (relations among inds.)
  - (Skolem constants: $\Sigma^{sk}_k = \{f^k_1, f^k_2, \ldots\}$) (witness constructors; countably ∞)
  - We take the signature Σ to be all of these together: $\Sigma := \Sigma^o \cup \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$, where $\Sigma^* := \bigcup_{k \in \mathbb{N}} \Sigma^*_k$.
- We assume a set of individual variables: $\mathcal{V}_\iota = \{X_\iota, Y_\iota, Z, X^1{}_\iota, X^2\}$ (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 is built up from the signature and variables as terms (to represent individuals) and propositions (to represent propositions). The latter include the propositional connectives, but also quantifiers.

PL^1 Syntax (Formulae)

- **Definition 7.1.3** terms: $\mathbf{A} \in wff_\iota(\Sigma_\iota)$ (denote individuals: type ι)
  - $\mathcal{V}_\iota \subseteq wff_\iota(\Sigma_\iota)$,
  - if $f \in \Sigma_k^f$ and $\mathbf{A}^i \in wff_\iota(\Sigma_\iota)$ for $i \leq k$, then $f(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in wff_\iota(\Sigma_\iota)$.
- **Definition 7.1.4** propositions: $\mathbf{A} \in wff_o(\Sigma)$ (denote truth values: type o)
  - if $p \in \Sigma_k^p$ and $\mathbf{A}^i \in wff_\iota(\Sigma_\iota)$ for $i \leq k$, then $p(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in wff_o(\Sigma)$,
  - if $\mathbf{A}, \mathbf{B} \in wff_o(\Sigma)$, then $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X.\mathbf{A} \in wff_o(\Sigma)$.
- **Definition 7.1.5** 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 7.1.6** We use $\exists X.\mathbf{A}$ as an abbreviation for $\neg(\forall X.\neg \mathbf{A})$. (existential quantifier)
- **Definition 7.1.7** Call formulae without connectives or quantifiers atomic else complex.

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

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 7.1.8** 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 7.1.9** We define the set $\mathrm{free}(\mathbf{A})$ of free variables of a formula $\mathbf{A}$ inductively:

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

- **Definition 7.1.10** 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 $cwff_\iota(\Sigma_\iota)$ and the set of sentences with $cwff_o(\Sigma_\iota)$.

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)

- We fix the Universe $\mathcal{D}_o = \{\mathsf{T}, \mathsf{F}\}$ of truth values.
- We assume an arbitrary universe $\mathcal{D}_\iota \neq \emptyset$ of individuals (this choice is a parameter to the semantics)
- **Definition 7.1.11** An interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_o \to \mathcal{D}_o$ with $\mathsf{T} \mapsto \mathsf{F}$, $\mathsf{F} \mapsto \mathsf{T}$, and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}\colon \Sigma_k^f \to \mathcal{F}({\mathcal{D}_\iota}^k; \mathcal{D}_\iota)$ (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 7.1.12** A variable assignment $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}_\iota$ maps variables into the universe.
- A first-order Model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ consists of a universe $\mathcal{D}_\iota$ and an interpretation$\mathcal{I}$.

We do not have to make the universe of truth values part of the model, since it is always the same; we determine the model by choosing a universe and an interpretation function.

Given a first-order model, we can define the evaluation function as a homomorphism over the construction of formulae.

Semantics of PL^1 (Evaluation)

- Given a model $\langle \mathcal{D}, \mathcal{I} \rangle$, the value function $\mathcal{I}_\varphi$ is recursively defined:(two parts: terms & propositions)
  - $\mathcal{I}_\varphi\colon \mathit{wff}_\iota(\Sigma_\iota) \to \mathcal{D}_\iota$ assigns values to terms.
    - $\mathcal{I}_\varphi(X) := \varphi(X)$ and
    - $\mathcal{I}_\varphi(f(\mathbf{A}_1, \ldots, \mathbf{A}_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_k))$
  - $\mathcal{I}_\varphi\colon \mathit{wff}_o(\Sigma) \to \mathcal{D}_o$ 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 $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$.

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 extended 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 $\psi := \varphi,[a/X]$. 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 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 $\psi := \varphi,[a/X]$. 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.

7.2 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 7.2.1** We call $\sigma\colon wff_\iota(\Sigma_\iota) \to wff_\iota(\Sigma_\iota)$ a substitution, iff $\sigma(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)) = f(\sigma(\mathbf{A}_1),\ldots,\sigma(\mathbf{A}_n))$ and the support $\mathbf{supp}(\sigma) := \{X \mid \sigma(X) \neq X\}$ of σ is finite.
- **Observation 7.2.2** *Note that a substitution σ is determined by its values on variables alone, thus we can write σ as $\sigma|_{\mathcal{V}_\iota} = \{[\sigma(X)/X] \mid X \in \mathbf{supp}(\sigma)\}$.*
- **Notation 7.2.3** We denote the substitution σ with $\mathbf{supp}(\sigma) = \{x^i \mid 1 \leq i \leq n\}$ and $\sigma(x^i) = \mathbf{A}_i$ by $[\mathbf{A}_1/x^1],\ldots,[\mathbf{A}_n/x^n]$.
- **Example 7.2.4** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.
- **Definition 7.2.5** We call $\mathbf{intro}(\sigma) := \bigcup_{X\in\mathbf{supp}(\sigma)} \mathrm{free}(\sigma(X))$ the set of variables introduced by σ.

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

- ▷ **Notation 7.2.6 (Substitution Extension)** Let σ be a substitution, then we denote with $\sigma,[\mathbf{A}/X]$ the function $\{(Y,\mathbf{A}) \in \sigma \,|\, Y \neq X\} \cup \{(X,\mathbf{A})\}$. ($\sigma,[\mathbf{A}/X]$ coincides with σ of X, and gives the result $\mathbf{A}$ there.)
- ▷ Note: If σ is a substitution, then $\sigma,[\mathbf{A}/X]$ is also a substitution.
- ▷ **Definition 7.2.7** If σ is a substitution, then we call $\sigma,[\mathbf{A}/X]$ the extension of σ by $[\mathbf{A}/X]$.
- ▷ We also need the dual operation: removing a variable from the support
- ▷ **Definition 7.2.8** We can discharge a variable X from a substitution σ by $\sigma_{-X} := \sigma,[X/X]$.

 27 JACOBS UNIVERSITY

Note that the use of the comma notation for substitutions defined in Notation 7.2.3 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 first-order 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 7.2.9** $\sigma(\forall X.\mathbf{A}) := (\forall X.\sigma_{-X}(\mathbf{A}))$.
- ▷ 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 7.2.10** Let $\mathbf{B} \in \mathit{wff}_\iota(\Sigma_\iota)$ and $\mathbf{A} \in \mathit{wff}_o(\Sigma)$, 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 substitutable 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.)

 28 JACOBS UNIVERSITY

Here we come to a conceptual problem of most introductions to first-order logic: they directly define substitutions to be capture-avoiding 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 7.2.11** *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 $\mathbf{A}$:

P.1.1 depth=0:

P.1.1.1 Then $\mathbf{A}$ is a variable (say Y), or constant, so we have three cases

P.1.1.1.1 $\mathbf{A} = Y = X$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.1.1.2 $\mathbf{A} = Y \neq X$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.1.1.3 $\mathbf{A}$ is a constant: analogous to the preceding case ($Y \neq X$)

P.1.1.2 This completes the base case (depth = 0). □

P.1.2 depth>0: then $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ and we have

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

by inductive hypothesis

P.1.2.2 This completes the inductive case, 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 7.2.12** *Let $\mathbf{B} \in wff_\iota(\Sigma_\iota)$ be substitutable for X in $\mathbf{A} \in wff_o(\Sigma)$, 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 $\mathbf{A}$

P.1.1 $n = 0$: then $\mathbf{A}$ is an atomic proposition, and we can argue like in the inductive case of the substitution value lemma for terms.

P.1.2 $n>0$ and $\mathbf{A} = \neg\mathbf{B}$ or $\mathbf{A} = \mathbf{C} \circ \mathbf{D}$: Here we argue like in the inductive case of the term lemma as well.

P.1.3 $n>0$ and $\mathbf{A} = \forall X.\mathbf{C}$: 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 $\mathcal{I}_\varphi(\forall X.\mathbf{C}) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathsf{T}$.

P.1.4 $n>0$ and $\mathbf{A} = \forall Y.\mathbf{C}$ where $X \neq Y$: 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 inductive hypothesis. So $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\varphi(\forall Y.[\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\forall Y.\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))$ □

 30 JACOBS UNIVERSITY

To understand the proof full, 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.

7.3 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 7.3.1** *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:

P.1 $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{T}$, iff

P.2 $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$, iff

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

P.4 $\mathcal{I}_\varphi(\forall Y.[Y/X](\mathbf{A})) = \mathsf{T}$. □

▷ **Definition 7.3.2** We call two formulae $\mathbf{A}$ and $\mathbf{B}$ alphabetical 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.

 31 JACOBS UNIVERSITY

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, we will consider alphabetical variants as identical

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

▷ Formally: Take $cwff_o(\Sigma_\iota)$ (new) to be the quotient set of $cwff_o(\Sigma_\iota)$ (old) modulo $=_\alpha$. (formulae as syntactic representatives of equivalence classes)

▷ **Definition 7.3.3 (Capture-Avoiding Substitution Application)** Let σ be a substitution, $\mathbf{A}$ a formula, and $\mathbf{A}'$ an alphabetical variant of $\mathbf{A}$, such that $\mathbf{intro}(\sigma) \cap \mathrm{BVar}(\mathbf{A}) = \emptyset$. Then $[\mathbf{A}]_{=_\alpha} = [\mathbf{A}']_{=_\alpha}$ and we can define $\sigma([\mathbf{A}]_{=_\alpha}) := [\sigma(\mathbf{A}')]_{=_\alpha}$.

▷ **Notation 7.3.4** 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.

 32 JACOBS UNIVERSITY

Chapter 8

Inference in First-Order Logic

In this Chapter we will introduce inference systems (calculi) for first-order logic and study their properties, in particular soundness and completeness.

8.1 First-Order Calculi

In this section we will introduce two reasoning calculi for first-order logic, both were invented by Gerhard Gentzen in the 1930's and are very much related. The "natural deduction" calculus was created in order to model the natural mode of reasoning e.g. in everyday mathematical practice. This calculus was intended as a counter-approach to the well-known Hilbert-style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles.

The "sequent calculus" was a rationalized version and extension of the natural deduction calculus that makes certain meta-proofs simpler to push through[2]. EdN:2

Both calculi have a similar structure, which is motivated by the human-orientation: rather than using a minimal set of inference rules, they provide two inference rules for every connective and quantifier, one "introduction rule" (an inference rule that derives a formula with that symbol at the head) and one "elimination rule" (an inference rule that acts on a formula with this head and derives a set of subformulae).

This allows us to introduce the calculi in two stages, first for the propositional connectives and then extend this to a calculus for first-order logic by adding rules for the quantifiers.

8.1.1 Propositional Natural Deduction Calculus

We will now introduce the "natural deduction" calculus for propositional logic. The calculus was created in order to model the natural mode of reasoning e.g. in everyday mathematical practice. This calculus was intended as a counter-approach to the well-known Hilbert style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles.

Rather than using a minimal set of inference rules, the natural deduction calculus provides two/three inference rules for every connective and quantifier, one "introduction rule" (an inference rule that derives a formula with that symbol at the head) and one "elimination rule" (an inference rule that acts on a formula with this head and derives a set of subformulae).

Calculi: Natural Deduction ($\mathcal{ND}^0$; Gentzen [Gen35])

[2]EdNote: say something about cut elimination/analytical calculi somewhere

▷ Idea: $\mathcal{ND}^0$ tries to mimic human theorem proving behavior (non-minimal)

▷ **Definition 8.1.1** The propositional natural deduction calculus $\mathcal{ND}^0$ has rules for the introduction and elimination of connectives

Introduction	Elimination	Axiom
$\dfrac{\mathbf{A}\quad\mathbf{B}}{\mathbf{A}\wedge\mathbf{B}}\wedge I$	$\dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{A}}\wedge E_l \quad \dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{B}}\wedge E_r$	$\dfrac{}{\mathbf{A}\vee\neg\mathbf{A}}\,\mathrm{TND}$
$\dfrac{\begin{array}{c}[\mathbf{A}]^1\\ \vdots \\ \mathbf{B}\end{array}}{\mathbf{A}\Rightarrow\mathbf{B}}\Rightarrow I^1$	$\dfrac{\mathbf{A}\Rightarrow\mathbf{B}\quad\mathbf{A}}{\mathbf{B}}\Rightarrow E$	

▷ TND is used only in classical logic (otherwise constructive/intuitionistic)

The most characteristic rule in the natural deduction calculus is the $\Rightarrow I$ rule. It corresponds to the mathematical way of proving an implication $\mathbf{A}\Rightarrow\mathbf{B}$: We assume that $\mathbf{A}$ is true and show $\mathbf{B}$ from this assumption. When we can do this we discharge (get rid of) the assumption and conclude $\mathbf{A}\Rightarrow\mathbf{B}$. This mode of reasoning is called hypothetical reasoning. Note that the local hypothesis is discharged by the rule $\Rightarrow I$, i.e. it cannot be used in any other part of the proof. As the $\Rightarrow I$ rules may be nested, we decorate both the rule and the corresponding assumption with a marker (here the number 1).

Let us now consider an example of hypothetical reasoning in action.

Natural Deduction: Examples

▷ Inference with local hypotheses

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

$$\dfrac{\dfrac{\begin{array}{c}[A]^1\\ [B]^2\\ A\end{array}}{B\Rightarrow A}\Rightarrow I^2}{A\Rightarrow B\Rightarrow A}\Rightarrow I^1$$

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for $\mathcal{ND}^0$

▷ **Theorem 8.1.2** $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}^0}\mathbf{B}$, *iff* $\mathcal{H}\vdash_{\mathcal{ND}^0}\mathbf{A}\Rightarrow\mathbf{B}$.

▷ Proof: We show the two directions separately

P.1 If $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, then $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by $\Rightarrow I$, and

P.2 If $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$, then $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by weakening and $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ by $\Rightarrow E$. □

Another characteristic of the natural deduction calculus is that it has inference rules (introduction and elimination rules) for all connectives. So we extend the set of rules from Definition 8.1.1 for disjunction, negation and falsity.

More Rules for Natural Deduction

▷ **Definition 8.1.3** $\mathcal{ND}^0$ has the following additional rules for the remaining connectives.

$$\frac{\mathbf{A}}{\mathbf{A} \vee \mathbf{B}}\vee I_l \qquad \frac{\mathbf{B}}{\mathbf{A} \vee \mathbf{B}}\vee I_r \qquad \frac{\mathbf{A} \vee \mathbf{B} \quad \begin{matrix}[\mathbf{A}]^1 \\ \vdots \\ \mathbf{C}\end{matrix} \quad \begin{matrix}[\mathbf{B}]^1 \\ \vdots \\ \mathbf{C}\end{matrix}}{\mathbf{C}}\vee E^1$$

$$\frac{\begin{matrix}[\mathbf{A}]^1 \\ \vdots \\ F\end{matrix}}{\neg \mathbf{A}}\neg I^1 \qquad \frac{\neg\neg \mathbf{A}}{\mathbf{A}}\neg E$$

$$\frac{\neg \mathbf{A} \quad \mathbf{A}}{F}FI \qquad \frac{F}{\mathbf{A}}FE$$

To obtain a first-order calculus, we have to extend $\mathcal{ND}^0$ with (introduction and elimination) rules for the quantifiers.

First-Order Natural Deduction ($\mathcal{ND}^1$; Gentzen [Gen35])

▷ Rules for propositional connectives just as always

▷ **Definition 8.1.4 (New Quantifier Rules)** The first-order natural deduction calculus $\mathcal{ND}^1$ extends $\mathcal{ND}^0$ by the following four rules

$$\frac{\mathbf{A}}{\forall X.\mathbf{A}}\forall I^* \qquad \frac{\forall X.\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\forall E$$

$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X.\mathbf{A}}\exists I \qquad \frac{\exists X.\mathbf{A} \quad \begin{matrix}[[c/X](\mathbf{A})]^1 \\ \vdots \\ \mathbf{C}\end{matrix}}{\mathbf{C}}\exists E^1$$

* means that $\mathbf{A}$ does not depend on any hypothesis in which X is free.

The intuition behind the rule $\forall I$ is that a formula $\mathbf{A}$ with a (free) variable X can be generalized to $\forall X.\mathbf{A}$, if X stands for an arbitrary object, i.e. there are no restricting assumptions about X. The $\forall E$ rule is just a substitution rule that allows to instantiate arbitrary terms $\mathbf{B}$ for X in $\mathbf{A}$. The $\exists I$ rule says if we have a witness $\mathbf{B}$ for X in $\mathbf{A}$ (i.e. a concrete term $\mathbf{B}$ that makes $\mathbf{A}$ true), then we can existentially close $\mathbf{A}$. The $\exists E$ rule corresponds to the common mathematical practice, where we give objects we know exist a new name c and continue the proof by reasoning about this concrete object c. Anything we can prove from the assumption $[c/X](\mathbf{A})$ we can prove outright if $\exists X.\mathbf{A}$ is known.

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for $\mathcal{ND}^0$

▷ **Theorem 8.1.5** $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, *iff* $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$.

▷ Proof: We show the two directions separately

P.1 If $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, then $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by $\Rightarrow I$, and

P.2 If $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$, then $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{A} \Rightarrow \mathbf{B}$ by weakening and $\mathcal{H}, \mathcal{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ by $\Rightarrow E$. □

This is the classical formulation of the calculus of natural deduction. To prepare the things we want to do later (and to get around the somewhat un-licensed extension by hypothetical reasoning in the calculus), we will reformulate the calculus by lifting it to the "judgements level". Instead of postulating rules that make statements about the validity of propositions, we postulate rules that make state about derivability. This move allows us to make the respective local hypotheses in ND derivations into syntactic parts of the objects (we call them "sequents") manipulated by the inference rules.

Natural Deduction in Sequent Calculus Formulation

▷ Idea: Explicit representation of hypotheses (lift calculus to judgments)

▷ **Definition 8.1.6** A judgment is a meta-statement about the provability of propositions

▷ **Definition 8.1.7** A sequent is a judgment of the form $\mathcal{H} \vdash \mathbf{A}$ about the provability of the formula $\mathbf{A}$ from the set $\mathcal{H}$ of hypotheses.

▷ Idea: Reformulate ND rules so that they act on sequents

▷ **Example 8.1.8**

$$\dfrac{\dfrac{\dfrac{}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}}\wedge E_r \quad \dfrac{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}}\wedge E_l}{\dfrac{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}\wedge\mathbf{A}}{\emptyset\vdash\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\Rightarrow I}\wedge I$$

▷ Note: Even though the antecedent of a sequent is written like a sequence, it is actually a set. In particular, we can permute and duplicate members at will.

Sequent-Style Rules for Natural Deduction

▷ **Definition 8.1.9** The following inference rules make up the sequent calculus

$$\frac{}{\Gamma,\mathbf{A}\vdash\mathbf{A}}\mathrm{Ax} \qquad \frac{\Gamma\vdash\mathbf{B}}{\Gamma,\mathbf{A}\vdash\mathbf{B}}\mathrm{weaken} \qquad \frac{}{\Gamma\vdash\mathbf{A}\vee\neg\mathbf{A}}\mathrm{TND}$$

$$\frac{\Gamma\vdash\mathbf{A}\quad\Gamma\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}\wedge I \qquad \frac{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}{\Gamma\vdash\mathbf{A}}\wedge E_l \qquad \frac{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}{\Gamma\vdash\mathbf{B}}\wedge E_r$$

$$\frac{\Gamma\vdash\mathbf{A}}{\Gamma\vdash\mathbf{A}\vee\mathbf{B}}\vee I_l \qquad \frac{\Gamma\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\vee\mathbf{B}}\vee I_r \qquad \frac{\Gamma\vdash\mathbf{A}\vee\mathbf{B}\quad\Gamma,\mathbf{A}\vdash\mathbf{C}\quad\Gamma,\mathbf{B}\vdash\mathbf{C}}{\Gamma\vdash\mathbf{C}}\vee E$$

$$\frac{\Gamma,\mathbf{A}\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\Rightarrow\mathbf{B}}\Rightarrow I \qquad \frac{\Gamma\vdash\mathbf{A}\Rightarrow\mathbf{B}\quad\Gamma\vdash\mathbf{A}}{\Gamma\vdash\mathbf{B}}\Rightarrow E$$

$$\frac{\Gamma,\mathbf{A}\vdash F}{\Gamma\vdash\neg\mathbf{A}}\neg I \qquad \frac{\Gamma\vdash\neg\neg\mathbf{A}}{\mathbf{A}}\neg E$$

$$\frac{\Gamma\vdash\neg\mathbf{A}\quad\Gamma\vdash\mathbf{A}}{\Gamma\vdash F}FI \qquad \frac{\Gamma\vdash F}{\Gamma\vdash\mathbf{A}}FE$$

First-Order Natural Deduction in Sequent Formulation

▷ Rules for propositional connectives just as always

▷ **Definition 8.1.10 (New Quantifier Rules)**

$$\frac{\Gamma\vdash\mathbf{A}\quad X\notin\mathrm{free}(\Gamma)}{\Gamma\vdash\forall X.\mathbf{A}}\forall I \qquad \frac{\Gamma\vdash\forall X.\mathbf{A}}{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}\forall E$$

$$\frac{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}{\Gamma\vdash\exists X.\mathbf{A}}\exists I \qquad \frac{\Gamma\vdash\exists X.\mathbf{A}\quad\Gamma,[c/X](\mathbf{A})\vdash\mathbf{C}\quad c\in\Sigma_0^{sk}\ \mathrm{new}}{\Gamma\vdash\mathbf{C}}\exists E$$

We leave the soundness result for the first-order natural deduction calculus to the reader and turn to the complenesss result, which is much more involved and interesting.

8.2 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 Smullyann, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before.[3] EdN:3

The basic intuition for this method is the following: typically, a logical system $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ 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 situation 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.

 42

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 set $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus-dependent constructions.

In this situation, Raymond Smullyann 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.

[3] EdNote: cite the original papers!

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 8.2.1** Φ is called $\mathcal{C}$-refutable, if there is a formula $\mathbf{B}$, such that $\Phi \vdash_{\mathcal{C}} \mathbf{B}$ and $\Phi \vdash_{\mathcal{C}} \neg \mathbf{B}$.
- ▷ **Definition 8.2.2** We call a pair $\mathbf{A}$ and $\neg \mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ can derive a contradiction from it.
- ▷ **Definition 8.2.3** Φ is called $\mathcal{C}$-consistent, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition 8.2.4** 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 8.2.5** *$\mathcal{C}$-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 specify the calculus (a syntactic device) while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\mathcal{S} = \langle \mathcal{L}, \mathcal{K}, \models \rangle$ 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.

The notion of an "abstract consistency class" provides the a calculus-independent notion of "consistency": A set Φ of sentences is considered "consistent in an abstract sense", iff it is a member of an abstract consistency class ∇.

Abstract Consistency

- ▷ **Definition 8.2.6** Let ∇ be a family of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇.
- ▷ **Notation 8.2.7** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.
- ▷ **Definition 8.2.8** A family $\nabla \subseteq wff_o(\Sigma)$ of sets of formulae is called a (first-order) abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$

  ∇_c) $\mathbf{A} \notin \Phi$ or $\neg \mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in wff_o(\Sigma)$.

  $\nabla_\neg$) $\neg \neg \mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$

  $\nabla_\wedge$) $(\mathbf{A} \wedge \mathbf{B}) \in \Phi$ implies $(\Phi \cup \{\mathbf{A}, \mathbf{B}\}) \in \nabla$

  $\nabla_\vee$) $\neg (\mathbf{A} \wedge \mathbf{B}) \in \Phi$ implies $\Phi * \neg \mathbf{A} \in \nabla$ or $\Phi * \neg \mathbf{B} \in \nabla$

$\nabla_\forall$) If $(\forall X . \mathbf{A}) \in \Phi$, then $\Phi * [\mathbf{B}/X](\mathbf{A}) \in \nabla$ for each closed term $\mathbf{B}$.

$\nabla_\exists$) If $\neg (\forall X . \mathbf{A}) \in \Phi$ and c is an individual constant that does not occur in Φ, then $\Phi * \neg [c/X](\mathbf{A}) \in \nabla$

The conditions are very natural: Take for instance ∇_c, it would be foolish to call a set Φ of sentences "consistent under a complete calculus", if it contains an elementary contradiction. The next condition $\nabla_\neg$ says that if a set Φ that contains a sentence $\neg\neg\mathbf{A}$ is "consistent", then we should be able to extend it by $\mathbf{A}$ without losing this property; in other words, a complete calculus should be able to recognize $\mathbf{A}$ and $\neg\neg\mathbf{A}$ to be equivalent.

We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

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 8.2.9** 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 8.2.10** *If ∇ is compact, then ∇ is closed under subsets.*

▷ Proof:

P.1 Suppose $S \subseteq T$ and $T \in \nabla$.

P.2 Every finite subset A of S is a finite subset of T.

P.3 As ∇ is compact, we know that $A \in \nabla$.

P.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 family ∇ by testing all their finite subsets (which is much simpler).

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).

Compact Abstract Consistency Classes

▷ **Lemma 8.2.11** *Any first-order abstract consistency class can be extended to a compact one.*

▷ Proof:

P.1 We choose $\nabla' := \{\Phi \subseteq \mathit{cwff}_o(\Sigma_\iota) \mid \text{every finite subset of } \Phi \text{is in } \nabla\}$.

P.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 $\nabla \subseteq \nabla'$.

P.3 Let us now show that each ∇' is compact.

P.3.1 Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ.

P.3.2 By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla'$.

P.3.3 Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.

P.3.4 On the other hand, suppose all finite subsets of Φ are in ∇'.

P.3.5 Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi \in \nabla'$. Thus ∇' is compact.

P.4 Note that ∇' is closed under subsets by the Lemma above.

P.5 Next we show that if ∇ satisfies ∇_*, then ∇' satisfies ∇_*.

P.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.

P.5.2 To show $\nabla_\neg$, let $\Phi \in \nabla'$ and $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \nabla'$.

P.5.2.1 Let Ψ be any finite subset of $\Phi * \mathbf{A}$, and $\Theta := (\Psi \backslash \{\mathbf{A}\}) * \neg\neg\mathbf{A}$.

P.5.2.2 Θ is a finite subset of Φ, so $\Theta \in \nabla$.

P.5.2.3 Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A} \in \Theta$, we get $\Theta * \mathbf{A} \in \nabla$ by $\nabla_\neg$.

P.5.2.4 We know that $\Psi \subseteq \Theta * \mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.

P.5.2.5 Thus every finite subset Ψ of $\Phi * \mathbf{A}$ is in ∇ and therefore by definition $\Phi * \mathbf{A} \in \nabla'$.

P.5.3 the other cases are analogous to $\nabla_\neg$. $\square$

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 8.2.12** 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 8.2.13 (Hintikka Properties)** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

$\mathcal{H}_c$) *For all $\mathbf{A} \in wff_o(\Sigma)$ 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 $\mathbf{A} \in \mathcal{H}$.*

$\mathcal{H}_\wedge$) *If $(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ then $\mathbf{A}, \mathbf{B} \in \mathcal{H}$.*

$\mathcal{H}_\vee$) *If $\neg(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ then $\neg\mathbf{A} \in \mathcal{H}$ or $\neg\mathbf{B} \in \mathcal{H}$.*

$\mathcal{H}_\forall$) *If $(\forall X . \mathbf{A}) \in \mathcal{H}$, then $[\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ for each closed term $\mathbf{B}$.*

$\mathcal{H}_\exists$) *If $\neg(\forall X . \mathbf{A}) \in \mathcal{H}$ then $\neg[\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ for some term closed term $\mathbf{B}$.*

Proof:

▷ **P.1** We prove the properties in turn

$\mathcal{H}_c$ goes by induction on the structure of $\mathbf{A}$

P.2.1 $\mathbf{A}$ atomic: Then $\mathbf{A} \notin \mathcal{H}$ or $\neg \mathbf{A} \notin \mathcal{H}$ by ∇_c.

P.2.2 $\mathbf{A} = \neg \mathbf{B}$:

P.2.2.1 Let us assume that $\neg \mathbf{B} \in \mathcal{H}$ and $\neg\neg \mathbf{B} \in \mathcal{H}$,

P.2.2.2 then $\mathcal{H} * \mathbf{B} \in \nabla$ by $\nabla_\neg$, and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.

P.2.2.3 So $\{\mathbf{B}, \neg \mathbf{B}\} \subseteq \mathcal{H}$, which contradicts the inductive hypothesis. □

P.2.3 $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$: similar to the previous case

We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.

P.3.1 If $\neg\neg \mathbf{A} \in \mathcal{H}$, then $\mathcal{H} * \mathbf{A} \in \nabla$ by $\nabla_\neg$.

P.3.2 The maximality of $\mathcal{H}$ now gives us that $\mathbf{A} \in \mathcal{H}$.

The 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 $\Phi \in \mathcal{H}$.

P.4 Extension Theorem

▷ **Theorem 8.2.14** *If ∇ is an abstract consistency class and $\Phi \in \nabla$ finite, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$.*

▷ Proof: Wlog. assume that ∇ compact (else use compact extension)

P.1 Choose an enumeration $\mathbf{A}^1, \mathbf{A}^2, \ldots$ of $\mathit{cwff}_o(\Sigma_\iota)$ and $c^1, c^2, \ldots$ of Σ_0^{sk}.

P.2 and construct a sequence of sets H^i with $H^0 := \Phi$ and

$$H^{n+1} := \begin{cases} H^n & \text{if } H^n * \mathbf{A}^n \notin \nabla \\ H^n \cup \{\mathbf{A}^n, \neg [c^n/X](\mathbf{B})\} & \text{if } H^n * \mathbf{A}^n \in \nabla \text{ and } \mathbf{A}^n = \neg (\forall X . \mathbf{B}) \\ H^n * \mathbf{A}^n & \text{else} \end{cases}$$

P.3 Note that all $H^i \in \nabla$, choose $\mathcal{H} := \bigcup_{i \in \mathbb{N}} H^i$

P.4 $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq H^j$,

P.5 so $\Psi \in \nabla$ as ∇ closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.

P.6 Let $\mathcal{H} * \mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}^j$, so that $\mathbf{B} \in H^{j+1}$ and $H^{j+1} \subseteq \mathcal{H}$

P.7 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 $\mathit{cwff}_o(\Sigma_\iota)$. 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[4] 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. EdN:4

Valuation

- **Definition 8.2.15** A function $\nu\colon cwff_o(\Sigma_\iota) \to \mathcal{D}_o$ is called a (first-order) 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 $\nu(\mathbf{B}) = \mathsf{T}$
  - $\nu(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\nu([\mathbf{B}/X](\mathbf{A})) = \mathsf{T}$ for all closed terms $\mathbf{B}$.
- **Lemma 8.2.16** *If $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}$ is a variable assignment, then $\mathcal{I}_\varphi\colon cwff_o(\Sigma_\iota) \to \mathcal{D}_o$ is a valuation.*
- Proof Sketch: Immediate from the definitions □

Thus a valuation is a weaker notion of evaluation in first-order logic; the other direction is also true, even though the proof of this result is much more involved: The existence of a first-order valuation that makes a set of sentences true entails the existence of a model that satisfies it.[5] EdN:5

Valuation and Satisfiability

- **Lemma 8.2.17** *If $\nu\colon cwff_o(\Sigma_\iota) \to \mathcal{D}_o$ is a valuation and $\Phi \subseteq cwff_o(\Sigma_\iota)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*
- Proof: We construct a model for Φ.

P.1 Let $\mathcal{D}_\iota := cwff_\iota(\Sigma_\iota)$, and

- $\mathcal{I}(f)\colon {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto f(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ for $f \in \Sigma^f$
- $\mathcal{I}(p)\colon {\mathcal{D}_\iota}^k \to \mathcal{D}_o; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto \nu(p(\mathbf{A}_1, \ldots, \mathbf{A}_n))$ for $p \in \Sigma^p$.

P.2 Then variable assignments into $\mathcal{D}_\iota$ are ground substitutions.

P.3 We show $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(\mathbf{A})$ for $\mathbf{A} \in wff_\iota(\Sigma_\iota)$ by induction on $\mathbf{A}$

P.3.1 $\mathbf{A} = X$: then $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(X)$ by definition.

P.3.2 $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_n)$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(f)(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = f(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = \varphi(f(\mathbf{A}_1, \ldots, \mathbf{A}_n)) = \varphi(\mathbf{A})$

P.4 We show $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A}))$ for $\mathbf{A} \in wff_o(\Sigma)$ by induction on $\mathbf{A}$

P.4.1 $\mathbf{A} = p(\mathbf{A}_1, \ldots, \mathbf{A}_n)$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(p)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(p)(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = \nu(p(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n))) = \nu(\varphi(p(\mathbf{A}_1, \ldots, \mathbf{A}_n))) = \nu(\varphi(\mathbf{A}))$

P.4.2 $\mathbf{A} = \neg \mathbf{B}$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{B}) = \nu(\varphi(\mathbf{B})) = \mathsf{F}$, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.

P.4.3 $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$: similar

[4] EdNote: introduce this above
[5] EdNote: I think that we only get a semivaluation, look it up in Andrews.

P.4.4 $\mathbf{A} = \forall X.\mathbf{B}$: then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{B}) = \nu(\psi(\mathbf{B})) = \mathsf{T}$, for all $\mathbf{C} \in \mathcal{D}_\iota$, where $\psi = \varphi, [\mathbf{C}/X]$. This is the case, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.

P.5 Thus $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A})) = \nu(\mathbf{A}) = \mathsf{T}$ for all $\mathbf{A} \in \Phi$.

P.6 Hence $\mathcal{M} \models \mathbf{A}$ for $\mathcal{M} := \langle \mathcal{D}_\iota, \mathcal{I} \rangle$. □

 50 JACOBS UNIVERSITY

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Theorem 8.2.18 (Hintikka-Lemma)** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ Proof:

P.1 we define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$,

P.2 then ν is a valuation by the Hintikka set properties.

P.3 We have $\nu(\mathcal{H}) = \{\mathsf{T}\}$, so $\mathcal{H}$ is satisfiable. □

▷ **Theorem 8.2.19 (Model Existence)** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ **P.1** There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)

We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)

In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable. □

 51 JACOBS UNIVERSITY

8.3 A Completeness Proof for First-Order ND

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 ND-consistency is an abstract consistency property.

P.2 P.3

Consistency, Refutability and Abstract Consistency

▷ **Theorem 8.3.1 (Non-Refutability is an Abstract Consistency Property)** $\Gamma := \{\Phi \subseteq cwff_o(\Sigma_\iota) \mid \Phi \text{ not } \mathcal{ND}^1\text{–refutable}\}$ *is an abstract consistency class.*

▷ Proof: We check the properties of an ACC

P.1 If Φ is non-refutable, then any subset is as well, so Γ is closed under subsets.

P.2 We show the abstract consistency conditions ∇_* for $\Phi \in \Gamma$.

P.2.1 ∇_c: We have to show that $\mathbf{A} \notin \Phi$ or $\neg \mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in wff_o(\Sigma)$.

P.2.1.2 Equivalently, we show the contrapositive: If $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then $\Phi \notin \Gamma$.

P.2.1.3 So let $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then Φ is $\mathcal{ND}^1$-refutable by construction.

P.2.1.4 So $\Phi \notin \Gamma$. □

P.2.2 $\nabla_\neg$: We show the contrapositive again

P.2.2.2 Let $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A} \notin \Gamma$

P.2.2.3 Then we have a refutation $\mathcal{D}\colon \Phi * \mathbf{A} \vdash_{\mathcal{ND}^1} F$

P.2.2.4 By prepending an application of $\neg E$ for $\neg\neg\mathbf{A}$ to $\mathcal{D}$, we obtain a refutation $\mathcal{D}'\colon \Phi \vdash_{\mathcal{ND}^1} F$.

P.2.2.5 Thus $\Phi \notin \Gamma$. □

P.2.3 other ∇_* similar:

 52 JACOBS UNIVERSITY

This directly yields two important results that we will use for the completeness analysis.

Henkin's Theorem

▷ **Corollary 8.3.2 (Henkin's Theorem)** *Every $\mathcal{ND}^1$-consistent set of sentences has a model.*

▷ Proof:

P.1 Let Φ be a $\mathcal{ND}^1$-consistent set of sentences.

P.2 The class of sets of $\mathcal{ND}^1$-consistent propositions constitute an abstract consistency class

P.3 Thus the model existence theorem guarantees a model for Φ. □

▷ **Corollary 8.3.3 (Löwenheim&Skolem Theorem)** *Satisfiable set Φ of first-order sentences has a countable model.*

▷ Proof Sketch: The model we constructed is countable, since the set of ground terms is. □

 53 JACOBS UNIVERSITY

Now, the completeness result for first-order natural deduction is just a simple argument away. We also get a compactness theorem (almost) for free: logical systems with a complete calculus are always compact.

Completeness and Compactness

▷ **Theorem 8.3.4 (Completeness Theorem for $\mathcal{ND}^1$)** *If $\Phi \models \mathbf{A}$, then $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$.*

▷ Proof: We prove the result by playing with negations.

P.1 If $\mathbf{A}$ is valid in all models of Φ, then $\Phi * \neg\mathbf{A}$ has no model

P.2 Thus $\Phi * \neg\mathbf{A}$ is inconsistent by (the contrapositive of) Henkins Theorem.

P.3 So $\Phi \vdash_{\mathcal{ND}^1} \neg\neg\mathbf{A}$ by $\neg I$ and thus $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$ by $\neg E$. □

▷ **Theorem 8.3.5 (Compactness Theorem for first-order logic)** *If $\Phi \models \mathbf{A}$, then there is already a finite set $\Psi \subseteq \Phi$ with $\Psi \models \mathbf{A}$.*

Proof: This is a direct consequence of the completeness theorem

▷ **P.1** We have $\Phi \models \mathbf{A}$, iff $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$.

As a proof is a finite object, only a finite subset $\Psi \subseteq \Phi$ can appear as leaves in the proof. □

8.4 Limits of First-Order Logic

We will now come to the limits of first-order Logic.[6] EdN:6

P.2 Gödel's Incompleteness Theorem

▷ **Theorem 8.4.1** *No logical system that can represent Peano-Arithmetic $(\mathbb{N}, s, 0, +, *)$ admits complete calculi.*

▷ Proof: (Sketch)

P.1 Let $\mathcal{L} := \langle \mathcal{S}, \mathcal{C} \rangle$ be such a system. We show that there is a valid $\mathcal{S}$-sentence $\mathbf{A}_\mathcal{C}$, that is no $\mathcal{C}$-theorem.

P.2 Encode the syntax of $\mathcal{S}$ and the $\mathcal{C}$ in Peano-arithmetic

P.3 We can now talk about $\mathcal{S}$ and $\mathcal{C}$ in $\mathcal{S}$ itself.

P.4 E.g. there is a $\mathcal{S}$-sentence $\mathbf{B}$ with the meaning: $\mathbf{A}$ is a $\mathcal{C}$-theorem.

P.5 Choose $\mathbf{A}_\mathcal{C}$ as "$\mathbf{A}_\mathcal{C}$ is no $\mathcal{C}$-theorem" (cf. Russell's set)

P.6 Obviously: $\mathbf{A}_\mathcal{C}$ ist valid in all standard models.

P.7 So $\mathcal{C}$ is either not correct or cannot derive $\mathbf{A}_\mathcal{C}$. □

[6] EdNote: MK: also present the theorem (whose name I forgot) that show that FOL is the "strongest logic" for first-order models. Maybe also the interpolation theorem.

Chapter 9

First-Order Inference with Tableaux

9.1 First-Order Tableaux

Test Calculi: Tableaux and Model Generation

- ▷ Idea: instead of showing $\emptyset \vdash Th$, show $\neg Th \vdash trouble$ (use $\bot$ for trouble)
- ▷ **Example 9.1.1** Tableau Calculi try to construct models.

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

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 $\mathcal{T}_0$)

- formula is analyzed in a tree to determine satisfiability
- branches correspond to valuations (models)
- one per connective

$$\frac{\mathbf{A}\wedge\mathbf{B}^{\mathsf{t}}}{\begin{array}{c}\mathbf{A}^{\mathsf{t}}\\ \mathbf{B}^{\mathsf{t}}\end{array}}\mathcal{T}_0\wedge \qquad \frac{\mathbf{A}\wedge\mathbf{B}^{\mathsf{f}}}{\mathbf{A}^{\mathsf{f}}\;\big|\;\mathbf{B}^{\mathsf{f}}}\mathcal{T}_0\vee \qquad \frac{\neg\,\mathbf{A}^{\mathsf{t}}}{\mathbf{A}^{\mathsf{f}}}\mathcal{T}_0\overset{\mathsf{T}}{\neg} \qquad \frac{\neg\,\mathbf{A}^{\mathsf{f}}}{\mathbf{A}^{\mathsf{t}}}\mathcal{T}_0\overset{\mathsf{F}}{\neg} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\quad \alpha\neq\beta\end{array}}{\bot}\mathcal{T}_0\mathrm{cut}$$

- Use rules exhaustively as long as they contribute new material
- **Definition 9.1.2** Call a tableau saturated, iff no rule applies, and a branch closed, iff it ends in $\bot$, else open. (open branches in saturated tableaux yield models)
- **Definition 9.1.3 ($\mathcal{T}_0$-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.

  $\Phi \subseteq wff_o(\mathcal{V}_o)$ 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}.

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 $\bot$ (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 9.1.4 We will call a closed tableau with the signed 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.

Definition 9.1.5 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 calculus in section ?sec.hilbert? 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 $\neg\,\mathbf{A}\vee\mathbf{B}$,. . . .)

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

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

▷ Refutation calculus based on trees of labeled formulae

▷ Tableau-Rules: $\mathcal{T}_0$ (propositional tableau rules) plus

$$\frac{\forall X.\mathbf{A}^{\mathsf{t}} \quad \mathbf{C} \in \mathit{cwff}_\iota(\Sigma_\iota)}{[\mathbf{C}/X](\mathbf{A})^{\mathsf{t}}}\mathcal{T}_1{:}\forall \qquad \frac{\forall X.\mathbf{A}^{\mathsf{f}} \quad c \in (\Sigma_0^{sk} \backslash \mathcal{H})}{[c/X](\mathbf{A})^{\mathsf{f}}}\mathcal{T}_1{:}\exists$$

 58 JACOBS UNIVERSITY

The rule $\mathcal{T}_1{:}\forall$ rule 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 $\mathbf{C} \in \mathit{wff}_\iota(\Sigma_\iota)$ for X. This makes the rule infinitely branching.

9.2 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 ($\mathcal{T}_1^f$)

▷ Refutation calculus based on trees of labeled formulae

▷ $\mathcal{T}_0$ (propositional tableau rules) plus

▷ Quantifier rules:

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

▷ Generalized cut rule $\mathcal{T}_1^f{:}\bot$ instantiates the whole tableau by σ.

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

Advantage: no guessing necessary in $\mathcal{T}_1^f{:}\forall$-rule

⊳ New: find suitable substitution (most general unifier)

 59 JACOBS UNIVERSITY

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.

Multiplicity in Tableaux

▷ **Observation 9.2.1** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f{:}\forall$ only need to be applied once.*

▷ **Example 9.2.2** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists x. p(x))$.

Start, close branch	use $\mathcal{T}_1^f{:}\forall$ again
$(p(a) \vee p(b)) \Rightarrow (\exists x. p(x))^{\mathsf{f}}$ $p(a) \vee p(b)^{\mathsf{t}}$ $\exists x. p(x)^{\mathsf{f}}$ $\forall x. \neg p(x)^{\mathsf{t}}$ $\neg p(y)^{\mathsf{t}}$ $p(y)^{\mathsf{f}}$ $p(a)^{\mathsf{t}}$ | $p(b)^{\mathsf{t}}$ $\bot : [a/x]$ |	$(p(a) \vee p(b)) \Rightarrow (\exists x. p(x))^{\mathsf{f}}$ $p(a) \vee p(b)^{\mathsf{t}}$ $\exists x. p(x)^{\mathsf{f}}$ $\forall x. \neg p(x)^{\mathsf{t}}$ $\neg p(a)^{\mathsf{t}}$ $p(a)^{\mathsf{f}}$ $p(a)^{\mathsf{t}}$ | $p(b)^{\mathsf{t}}$ $\bot$ | $\neg p(z)^{\mathsf{t}}$ | $p(z)^{\mathsf{f}}$ | $\bot : [b/z]$

▷ **Definition 9.2.3** 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 9.2.4** *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 9.2.5** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*

▷ Proof Sketch: Otherwise validity in PL^1 would be decidable. □

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

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

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

▷ **Example 9.2.6** Choosing which matters – this tableau does not close!

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

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

▷ 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 (later)

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

▷ **Observation 9.2.7** *$\mathcal{T}_1^f$ 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.

▷ **Definition 9.2.8** Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := (\mathbf{A}_1 =^? \mathbf{A}_1 \wedge \ldots \wedge \mathbf{A}_n =^? \mathbf{B}_n)$ a mating for $\mathcal{T}$, iff $\mathbf{A}_i^{\mathsf{t}}$ and $\mathbf{B}_i^{\mathsf{f}}$ occur in $\mathcal{T}$.

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 9.2.9** *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.

9.3 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[7] EdN:7

computation = 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)

- Problem: For given terms $\mathbf{A}$ and $\mathbf{B}$ find a substitution σ, such that $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$.
- **Notation 9.3.1** We write term pairs as $\mathbf{A} =^? \mathbf{B}$ e.g. $f(X) =^? f(g(Y))$
- Solutions (e.g. $[g(a)/X], [a/Y]$, $[g(g(a))/X], [g(a)/Y]$, or $[g(Z)/X], [Z/Y]$) are called unifiers, $\mathbf{U}(\mathbf{A} =^? \mathbf{B}) := \{\sigma \mid \sigma(\mathbf{A}) = \sigma(\mathbf{B})\}$
- Idea: find representatives in $\mathbf{U}(\mathbf{A} =^? \mathbf{B})$, that generate the set of solutions
- **Definition 9.3.2** 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 9.3.3** σ is called a most general unifier 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 a 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).

[7] EdNote: find the reference, and see what he really said

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 (Equational Systems)

- ▷ Idea: Unification is equation solving.
- ▷ **Definition 9.3.4** We call a formula $\mathbf{A}^1 =^? \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^? \mathbf{B}^n$ an equational system iff $\mathbf{A}^i, \mathbf{B}^i \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.
- ▷ We consider equational systems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).

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 9.3.5** 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 9.3.6** *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 9.3.7** 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 9.3.8** *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 $\theta \in \mathbf{U}(\mathcal{E})$

  P.1 then $\theta(X^i) = \theta(\mathbf{B}^i) = \theta \circ \sigma_\mathcal{E}(X^i)$

  P.2 and thus $\theta = \theta \circ \sigma_\mathcal{E}[\mathbf{supp}(\sigma)]$. □

  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 equational problems whose unifiers we can read off easily. Solved forms serve that need perfectly as Lemma 9.3.8 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 9.3.9** Inference system $\mathcal{U}$

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1, \ldots, \mathbf{A}^n) =^? f(\mathbf{B}^1, \ldots, \mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 =^? \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^? \mathbf{B}^n} \mathcal{U}\,\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathbf{A} =^? \mathbf{A}}{\mathcal{E}} \mathcal{U}\,\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^? \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad X \in \mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =^? \mathbf{A}} \mathcal{U}\,\mathrm{elim}$$

▷ **Lemma 9.3.10** *$\mathcal{U}$ is correct: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$*

▷ **Lemma 9.3.11** *$\mathcal{U}$ is complete: $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ implies $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$*

▷ **Lemma 9.3.12** *$\mathcal{U}$ is confluent: the order of derivations does not matter*

▷ **Corollary 9.3.13** *First-Order Unification is unitary: i.e. most general unifiers are unique up to renaming of introduced variables.*

▷ Proof Sketch: the inference system $\mathcal{U}$ 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 9.3.14 Two similar unification problems:

$\begin{array}{ll} f(g(x,x),h(a)) =^? f(g(a,z),h(z)) & \\ \hline g(x,x) =^? g(a,z) \wedge h(a) =^? h(z) & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? a \wedge x =^? z \wedge h(a) =^? h(z) & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? a \wedge x =^? z \wedge a =^? z & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? a \wedge a =^? z \wedge a =^? z & \mathcal{U}\,\mathrm{elim} \\ \hline x =^? a \wedge z =^? a \wedge a =^? a & \mathcal{U}\,\mathrm{elim} \\ \hline x =^? a \wedge z =^? a & \mathcal{U}\,\mathrm{triv} \end{array}$	$\begin{array}{ll} f(g(x,x),h(a)) =^? f(g(b,z),h(z)) & \\ \hline g(x,x) =^? g(b,z) \wedge h(a) =^? h(z) & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? b \wedge x =^? z \wedge h(a) =^? h(z) & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? b \wedge x =^? z \wedge a =^? z & \mathcal{U}\,\mathrm{dec} \\ \hline x =^? b \wedge b =^? z \wedge a =^? z & \mathcal{U}\,\mathrm{elim} \\ \hline x =^? a \wedge z =^? a \wedge a =^? b & \mathcal{U}\,\mathrm{elim} \end{array}$
MGU: $[a/x], [a/z]$	$a =^? b$ not unifiable

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 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 9.3.15** Let S and T be multisets and $\prec$ a partial ordering on $S \cup T$. Then we define $(S \prec^m T)$, iff $S = C \uplus T'$ and $T = C \uplus \{t\}$, where $s \prec t$ for all $s \in S'$. We call $\prec^m$ the multiset ordering induced by $\prec$.

▷ **Lemma 9.3.16** *If $\prec$ is total/terminating on S, then $\prec^m$ is total/terminating on $\mathcal{P}(S)$.*

▷ **Lemma 9.3.17** *$\mathcal{U}$ is terminating* (*any $\mathcal{U}$-derivation is finite*)

▷ Proof: We prove termination by mapping $\mathcal{U}$ transformation into a Noetherian space.

P.1 Let $\mu(\mathcal{E}) := \langle n, \mathcal{N} \rangle$, where

- ▷ m is the number of unsolved variables in $\mathcal{E}$
- ▷ $\mathcal{N}$ is the multiset of term depths in $\mathcal{E}$

P.2 The lexicographic order $\prec$ on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.

P.2.1 $\mathcal{U}\,\mathrm{dec}$ and $\mathcal{U}\,\mathrm{triv}$ decrease the multiset of term depths without increasing the unsolved variables

P.2.2 $\mathcal{U}\,\mathrm{elim}$ 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.

Unification (decidable)

▷ **Definition 9.3.18** 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 9.3.19** *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.

P.1 There is an unsolved pair $\mathbf{A} =^? \mathbf{B}$ in $\mathcal{E} = \mathcal{E}' \wedge \mathbf{A} =^? \mathbf{B}$.

P.2 we have two cases

P.2.1 $\mathbf{A}, \mathbf{B} \notin \mathcal{V}_\iota$: then $\mathbf{A} = f(\mathbf{A}^1 \ldots \mathbf{A}^n)$ and $\mathbf{B} = f(\mathbf{B}^1 \ldots \mathbf{B}^n)$, and thus $\mathcal{U}\,\mathrm{dec}$ is applicable

P.2.2 $\mathbf{A} = X \in \mathrm{free}(\mathcal{E})$: then $\mathcal{U}\,\mathrm{elim}$ (if $\mathbf{B} \neq X$) or $\mathcal{U}\,\mathrm{triv}$ (if $\mathbf{B} = X$) is applicable. □

▷ **Corollary 9.3.20** *Unification is decidable in* PL^1.

▷ Proof Idea: $\mathcal{U}$-irreducible sets of equations can be obtained in finite time by Lemma 9.3.17 and are either solved or unsolvable by Lemma 9.3.19, so they provide the answer. □

9.4 Efficient Unification

Complexity of Unification

▷ Observation: Naive unification is exponential in time and space.

▷ consider the terms

$$\begin{aligned} s_n &= f(f(x_0,x_0), f(f(x_1,x_1), f(\ldots, f(x_{n-1},x_{n-1}))\ldots)) \\ t_n &= f(x_1, f(x_2, f(x_3, f(\ldots, x_n)\ldots))) \end{aligned}$$

▷ The most general unifier of s_n and t_n is

$[f(x_0,x_0)/x_1], [f(f(x_0,x_0), f(x_0,x_0))/x_2], [f(f(f(x_0,x_0), f(x_0,x_0)), f(f(x_0,x_0), f(x_0,x_0)))/x_3], \ldots$

▷ it contains $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2$ occurrences of the variable x_0. (exponential)

▷ Problem: the variable x_0 has been copied too often

▷ Idea: Find a term representation that re-uses subterms

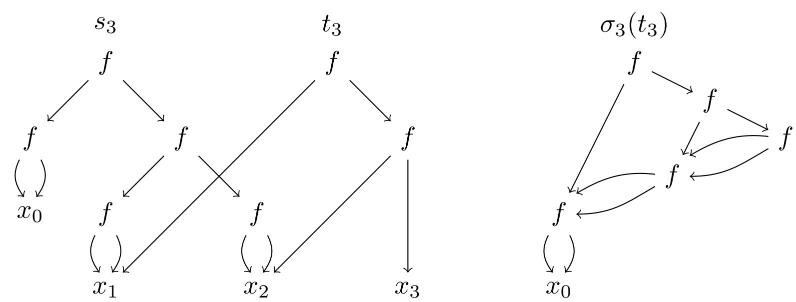
Directed Acyclic Graphs (DAGs)

▷ use directed acyclic graphs for the term representation

- ▷ variables my only occur once in the DAG
- ▷ subterms can be referenced multiply

▷ **Observation 9.4.1** *Terms can be transformed into DAGs in linear time*

▷ **Example 9.4.2** $s_3, t_3, \sigma_3(s_3)$



DAG Unification Algorithm

▷ **Definition 9.4.3** We say that $X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathbf{B}^j)$ for $i \leq j$

▷ **Definition 9.4.4** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}\,\mathrm{dec}$ and $\mathcal{U}\,\mathrm{triv}$ from $\mathcal{U}$ plus the following:

$$\frac{\mathcal{E} \wedge X =^? \mathbf{A} \wedge X =^? \mathbf{B} \quad \mathbf{A}, \mathbf{B} \notin \mathcal{V}_\iota \quad |\mathbf{A}| \leq |\mathbf{B}|}{\mathcal{E} \wedge X =^? \mathbf{A} \wedge \mathbf{A} =^? \mathbf{B}} \mathcal{DU}\,\mathrm{merge}$$

$$\frac{\mathcal{E} \wedge X =^? Y \quad X \neq Y \quad X, Y \in \mathrm{free}(\mathcal{E})}{[Y/X](\mathcal{E}) \wedge X =^? Y} \mathcal{DU}\,\mathrm{evar}$$

where $|\mathbf{A}|$ is the number of symbols in $\mathbf{A}$.

Unification by DAG-chase

▷ Idea: Extend the Input-DAGs by edges that represent unifiers.

▷ write $n.a$, if a is the symbol of node n.

▷ auxiliary procedures: (all linear or constant time)

▷ $\mathtt{find}(n)$ follows the path from n and returns the end node

▷ $\mathtt{union}(n,m)$ adds an edge between n and m.

▷ $\mathtt{occur}(n,m)$ determines whether $n.x$ occurs in the DAG with root m.

Algorithm unify

▷ Input: symmetric pairs of nodes in DAGs

```
fun unify(n,n) = true
    | unify(n.x,m) = if occur(n,m) then true else union(n,m)
  | unify(n.f,m.g) = if g!=f then false
                     else forall (i,j) => unify(find(i),find(j)) (chld m,chld n)
                     end
```

▷ linear in space, since no new nodes are created, and at most one link per variable.

▷ consider terms $f(s_n, f(t'_n, x_n)), f(t_n, f(s'_n, y_n)))$, where $s'_n = [y_i/x_i](s_n)$ und $t'_n = [y_i/x_i](t_n)$.

▷ unify needs exponentially many recursive calls to unify the nodes x_n and y_n. (they are unified after n calls, but checking needs the time)

▷ Idea: Also bind the function nodes, if the arguments are unified.

```
unify(n.f,m.g) = if g!=f then false
        else union(n,m);
              forall (i,j) => unify(find(i),find(j)) (chld m,chld n)
        end
```

▷ this only needs linearly many recursive calls as it directly returns with true or makes a node inaccessible for find.

▷ linearly many calls to linear procedures give quadratic runtime.

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

▷ **Observation 9.4.5** *$\mathcal{T}_1^f$ 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.

▷ **Definition 9.4.6** Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := (\mathbf{A}_1 =^? \mathbf{A}_1 \wedge \ldots \wedge \mathbf{A}_n =^? \mathbf{B}_n)$ a mating for $\mathcal{T}$, iff $\mathbf{A}_i^{\mathsf{t}}$ and $\mathbf{B}_i^{\mathsf{f}}$ occur in $\mathcal{T}$.

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 9.4.7** *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.

Now that we understand basic unification theory, we can come to the meta-theoretical properties of the tableau calculus, which we now discuss to make the understanding of first-order inference complete.

9.5 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 sound, iff it preserves satisfiability and the goal formulae are unsatisfiable.

▷ **Definition 9.5.1** A labeled formula $\mathbf{A}^\alpha$ is valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$.

▷ **Definition 9.5.2** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae in $\mathcal{P}$ is satisfiable.

▷ **Lemma 9.5.3** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ **Theorem 9.5.4 (Soundness)** *A set Φ of propositional formulae is valid, if there is a closed tableau $\mathcal{T}$ for Φ^{f}.*

▷ Proof: by contradiction: Suppose Φ is not valid.

P.1 then the initial tableau is satisfiable (Φ^{f} satisfiable)

P.2 so $\mathcal{T}$ is satisfiable, by Lemma 9.5.3.

P.3 there is a satisfiable branch (by definition)

P.4 but all branches are closed ($\mathcal{T}$ closed)

□

Thus we only have to prove Lemma 9.5.3, 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 assue 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 9.5.5** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ Proof:

P.1 we examine the tableau rules in turn

P.1.1 propositional rules: as in propositional tableaux

P.1.2 $\mathcal{T}_1^f{:}\exists$: by Lemma 9.5.7

P.1.3 $\mathcal{T}_1^f{:}\bot$: by Lemma 7.2.12 (substitution value lemma)

P.1.4 $\mathcal{T}_1^f{:}\forall$:

P.1.4.1 $\mathcal{I}_\varphi(\forall X . \mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$

P.1.4.2 so in particular for some $a \in \mathcal{D}_\iota \neq \emptyset$. □

□

▷ **Corollary 9.5.6** $\mathcal{T}_1^f$ *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 9.5.7** $\mathcal{T}_1^f{:}\exists$ *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 $[f(X^1, \ldots, X^n)/X](\mathbf{A})^{\mathsf{f}}$, where $W := \mathrm{free}(\forall X . \mathbf{A}) = \{X^1, \ldots, X^k\}$

P.1 Let $\mathcal{T}$ be satisfiable in $\mathcal{M} := \langle \mathcal{D}, \mathcal{I} \rangle$, then $\mathcal{I}_\varphi(\forall X . \mathbf{A}) = \mathsf{F}$.

P.2 We need to find a model $\mathcal{M}'$ that satisfies $\mathcal{T}'$ (find interpretation for f)

P.3 By definition $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}$ for some $a \in \mathcal{D}$ (depends on $\varphi|_W$)

P.4 Let $g \colon \mathcal{D}^k \to \mathcal{D}$ be defined by $g(a_1, \ldots, a_k) := a$, if $\varphi(X^i) = a_i$

P.5 choose $\mathcal{M}' = \langle \mathcal{D}, \mathcal{I}' \rangle$ with $\mathcal{I}' := \mathcal{I}, [g/f]$, then by subst. value lemma

$$\begin{aligned}
\mathcal{I}'_\varphi([f(X^1, \ldots, X^k)/X](\mathbf{A})) &= \mathcal{I}'_{\varphi,[\mathcal{I}'_\varphi(f(X^1,\ldots,X^k))/X]}(\mathbf{A}) \\
&= \mathcal{I}'_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}
\end{aligned}$$

P.6 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 (Theorem 8.2.19), 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.

Completeness of ($\mathcal{T}_1^f$)

▷ **Theorem 9.5.8** $\mathcal{T}_1^f$ *is refutation complete.*

▷ Proof: We show that $\nabla := \{\Phi \,|\, \Phi^{\mathsf{T}} \text{ has no closed Tableau}\}$ is an abstract consistency class

P.1 (∇_c, $\nabla_\neg$, $\nabla_\vee$, and $\nabla_\wedge$)as for propositional case.

P.2 ($\nabla_\forall$)by the lifting lemma below

P.3 ($\nabla_\exists$)Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A}) \in \Phi$ and $\Phi^{\mathsf{T}} * [c/X](\mathbf{A})^{\mathsf{F}} \in \nabla$.

$$\begin{array}{c} \Psi^{\mathsf{T}} \\ \forall X.\mathbf{A}^{\mathsf{f}} \\ [c/X](\mathbf{A})^{\mathsf{f}} \\ Rest \end{array} \qquad \begin{array}{c} \Psi^{\mathsf{T}} \\ \forall X.\mathbf{A}^{\mathsf{f}} \\ [f(X^1,\ldots,X^k)/X](\mathbf{A})^{\mathsf{f}} \\ [f(X^1,\ldots,X^k)/c](Rest) \end{array}$$

□

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 9.5.9** *If $\mathcal{T}_\theta$ is a closed tableau for a st $\theta(\Phi)$ 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 $\omega(\mathbf{A}) = \theta(\mathbf{A})$.

P.1 only the tableau-substitution rule is interesting.

P.2 Let $\theta(\mathbf{A}^i)^{\mathsf{t}}$ and $\theta(\mathbf{B}^i)^{\mathsf{f}}$ cut formulae in the branch Θ_θ^i of $\mathcal{T}_\theta$

P.3 there is a joint unifier σ of $\theta(\mathbf{A}^1) =^? \theta(\mathbf{B}^1) \wedge \ldots \wedge \theta(\mathbf{A}^n) =^? \theta(\mathbf{B}^n)$

P.4 thus $\sigma \circ \theta$ is a unifier of $\mathbf{A}$ and $\mathbf{B}$

P.5 hence there is a most general unifier ρ of $\mathbf{A}^1 =^? \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^? \mathbf{B}^n$

P.6 so Θ is closed □

Again, the "lifting lemma for tableaux" is paradigmatic for lifting lemmata for other refutation calculi.

Part III

Higher-Order Logic and λ-Calculus

In this Part we set the stage for a deeper discussions of the logical foundations of mathematics by introducing a particular higher-order logic, which gets around the limitations of first-order logic — the restriction of quantification to individuals. This raises a couple of questions (paradoxes, comprehension, completeness) that have been very influential in the development of the logical systems we know today.

Therefore we use the discussion of higher-order logic as an introduction and motivation for the λ-calculus, which answers most of these questions in a term-level, computation-friendly system.

The formal development of the simply typed λ-calculus and the establishment of its (meta-logical) properties will be the body of work in this Part. Once we have that we can reconstruct a clean version of higher-order logic by adding special provisions for propositions.

Chapter 10

Higher-Order Predicate Logic

The main motivation for higher-order logic is to allow quantification over classes of objects that are not individuals — because we want to use them as functions or predicates, i.e. apply them to arguments in other parts of the formula.

Higher-Order Predicate Logic (PLΩ)

- Quantification over functions and Predicates: $\forall P.\exists F.P(a) \vee \neg P(F(a))$
- Comprehension: (Existence of Functions)
  $\exists F.\forall X.FX = \mathbf{A}$ e.g. $f(x) = 3x^2 + 5x - 7$
- Extensionality: (Equality of functions and truth values)
  $\forall F.\forall G.(\forall X.FX = GX) \Rightarrow F = G$
  $\forall P.\forall Q.(P \Leftrightarrow Q) \Leftrightarrow P = Q$
- Leibniz Equality: (Indiscernability)
  $\mathbf{A} = \mathbf{B}$ for $\forall P.P\mathbf{A} \Rightarrow P\mathbf{B}$

 81 JACOBS UNIVERSITY

Indeed, if we just remove the restriction on quantification we can write down many things that are essential on everyday mathematics, but cannot be written down in first-order logic. But the naive logic we have created (BTW, this is essentially the logic of Frege [Fre79]) is much too expressive, it allows us to write down completely meaningless things as witnessed by Russell's paradox.

Problems with PLΩ

- Problem: Russell's Antinomy: $\forall Q.\mathcal{M}(Q) \Leftrightarrow \neg Q(Q)$
  - the set $\mathcal{M}$ of all sets that do not contain themselves
  - Question: Is $\mathcal{M} \in \mathcal{M}$? Answer: $\mathcal{M} \in \mathcal{M}$ iff $\mathcal{M} \notin \mathcal{M}$.
- What has happened? the predicate Q has been applied to itself
- Solution for this course: Forbid self-applications by types!!
  - ι, o (type of individuals, truth values), $\alpha \to \beta$ (function type)
  - right associative bracketing: $\alpha \to \beta \to \gamma$ abbreviates $\alpha \to (\beta \to \gamma)$

▷ vector notation: $\overline{\alpha_n} \to \beta$ abbreviates $\alpha_1 \to \ldots \to \alpha_n \to \beta$

▷ Well-typed formulae (prohibits paradoxes like $\forall Q.\mathcal{M}(Q) \Leftrightarrow \neg Q(Q)$)

▷ Other solution: Give it a non-standard semantics (Domain-Theory [Scott])

The solution to this problem turns out to be relatively simple with the benefit of hindsight: we just introduce a syntactic device that prevents us from writing down paradoxical formulae. This idea was first introduced by Russell and Whitehead in their Principia Mathematica [WR10].

Their system of "ramified types" was later radically simplified by Alonzo Church to the form we use here in [Chu40]. One of the simplifications is the restriction to unary functions that is made possible by the fact that we can re-interpret binary functions as unary ones using a technique called "Currying" after the Logician Haskell Brooks Curry (∗1900, †1982). Of course we can extend this to higher arities as well. So in theory we can consider n-ary functions as syntactic sugar for suitable higher-order functions. The vector notation for types defined above supports this intuition.

Types

▷ Types are semantic annotations for terms that prevent antinomies

▷ **Definition 10.0.1** 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 10.0.2** We will use ι for the type of individuals and o for the type of truth values.

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

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

Armed with a system of types, we can now define a typed higher-order logic, by insisting that all formulae of this logic be well-typed. One advantage of typed logics is that the natural classes of objects that have otherwise to be syntactically kept apart in the definition of the logic (e.g. the term and proposition levels in first-order logic), can now be distinguished by their type, leading to a much simpler exposition of the logic. Another advantage is that concepts like connectives that were at the language level e.g. in PL^0, can be formalized as constants in the signature, which again makes the exposition of the logic more flexible and regular. We only have to treat the quantifiers at the language level (for the moment).

Well-Typed Formulae ($\mathrm{PL}\Omega$)

▷ signature $\Sigma = \bigcup_{\alpha \in \mathcal{T}} \Sigma_\alpha$ with

▷ connectives: $\neg \in \Sigma_{o \to o}$ $\{\vee, \wedge, \Rightarrow, \Leftrightarrow \ldots\} \subseteq \Sigma_{o \to o \to o}$

▷ variables $\mathcal{V}_{\mathcal{T}} = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$, such that every $\mathcal{V}_\alpha$ countably infinite.

▷ well-typed formula e $\mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ of type α

- ▹ $\mathcal{V}_\alpha \cup \Sigma_\alpha \subseteq \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$
- ▹ If $\mathbf{C} \in \mathit{wff}_{\alpha\to\beta}(\Sigma, \mathcal{V}_\mathcal{T})$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$, then $(\mathbf{CA}) \in \mathit{wff}_\beta(\Sigma, \mathcal{V}_\mathcal{T})$
- ▹ If $\mathbf{A} \in \mathit{wff}_o(\Sigma, \mathcal{V}_\mathcal{T})$, then $(\forall X_\alpha.\mathbf{A}) \in \mathit{wff}_o(\Sigma, \mathcal{V}_\mathcal{T})$

▷ first-order terms have type ι, propositions the type o.

▷ there is no type annotation such that $\forall Q.\mathcal{M}(Q) \Leftrightarrow \neg Q(Q)$ is well-typed. Q needs type α as well as $\alpha \to o$.

 84 JACOBS UNIVERSITY

The semantics is similarly regular: We have universes for every type, and all functions are "typed functions", i.e. they respect the types of objects. Other than that, the setup is very similar to what we already know.

Standard Semantics for PLΩ

▷ **Definition 10.0.3** The universe of discourse (also carrier)

- ▹ arbitrary, non-empty set of individuals $\mathcal{D}_\iota$
- ▹ fixed set of truth values $\mathcal{D}_o = \{\mathsf{T}, \mathsf{F}\}$
- ▹ function universes $\mathcal{D}_{\alpha\to\beta} = \mathcal{F}(\mathcal{D}_\alpha; \mathcal{D}_\beta)$

▷ interpretation of constants: typed mapping $\mathcal{I}\colon \Sigma \to \mathcal{D}$ (i.e. $\mathcal{I}(\Sigma_\alpha) \subseteq \mathcal{D}_\alpha$)

▷ **Definition 10.0.4** We call a structure $\langle \mathcal{D}, \mathcal{I} \rangle$, where $\mathcal{D}$ is a universe and $\mathcal{I}$ an interpretation of constants a standard model of PLΩ.

▷ variable assignment: typed mapping $\varphi\colon \mathcal{V}_\mathcal{T} \to \mathcal{D}$

▷ **Definition 10.0.5** value function: typed mapping $\mathcal{I}_\varphi\colon \mathit{wff}_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T}) \to \mathcal{D}$

- ▹ $\mathcal{I}_\varphi|_{\mathcal{V}_\mathcal{T}} = \varphi \qquad \mathcal{I}_\varphi|_{\Sigma_\mathcal{T}} = \mathcal{I}$
- ▹ $\mathcal{I}_\varphi(\mathbf{AB}) = \mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
- ▹ $\mathcal{I}_\varphi(\forall X_\alpha.\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathbf{A}) = \mathsf{T}$ for all $\mathsf{a} \in \mathcal{D}_\alpha$.

▷ $\mathbf{A}_o$ valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$.

 85 JACOBS UNIVERSITY

We now go through a couple of examples of what we can express in PLΩ, and that works out very straightforwardly. For instance, we can express equality in PLΩ by Leibniz equality, and it has the right meaning.

Equality

▷ "Leibniz equality" (Indiscernability) $\mathbf{Q}^\alpha \mathbf{A}_\alpha \mathbf{B}_\alpha = \forall P_{\alpha\to o}.P\mathbf{A} \Leftrightarrow P\mathbf{B}$

▷ not that $\forall P_{\alpha\to o}.P\mathbf{A} \Rightarrow P\mathbf{B}$ (get the other direction by instantiating P with Q, where $QX \Leftrightarrow \neg PX$)

▷ **Theorem 10.0.6** *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 $\mathcal{D}_\alpha$.*

▷ **Notation 10.0.7** We write $\mathbf{A} = \mathbf{B}$ for $\mathbf{QAB}$ ($\mathbf{A}$ and $\mathbf{B}$ are equal, iff there is no property P that can tell them apart.)

▷ Proof:

P.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 $r \in \mathcal{D}_{\alpha \to o}$.

P.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 $\mathcal{I}_{\varphi,[r/P]}(P\mathbf{B}) = r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{T}$.

P.3 Thus $\mathcal{I}_\varphi(\mathbf{QAB}) = \mathsf{T}$.

P.4 Let $\mathcal{I}_\varphi(\mathbf{A}) \neq \mathcal{I}_\varphi(\mathbf{B})$ and $r = \{\mathcal{I}_\varphi(\mathbf{A})\}$

P.5 so $r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{T}$ and $r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{F}$

P.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 $\mathcal{I}_{\varphi,[r/P]}(P\mathbf{B}) = r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{F}$. □

Another example are the Peano Axioms for the natural numbers, though we omit the proofs of adequacy of the axiomatization here.

Example: Peano Axioms for the Natural Numbers

▷ $\Sigma = \{[\mathbb{N} : \iota \to o], [0 : \iota], [s : \iota \to \iota]\}$

▷ $\mathbb{N}0$ (0 is a natural number)

▷ $\forall X_\iota.\mathbb{N}X \Rightarrow \mathbb{N}(sX)$ (the successor of a natural number is natural)

▷ $\neg(\exists X_\iota.\mathbb{N}X \wedge sX = 0)$ (0 has no predecessor)

▷ $\forall X_\iota.\forall Y_\iota.(sX = sY) \Rightarrow X = Y$ (the successor function is injective)

▷ $\forall P_{\iota \to o}.P0 \Rightarrow (\forall X_\iota.\mathbb{N}X \Rightarrow PX \Rightarrow P(sX)) \Rightarrow (\forall Y_\iota.\mathbb{N}Y \Rightarrow P(Y))$
induction axiom: all properties P, that hold of 0, and with every n for its successor $s(n)$, hold on all $\mathbb{N}$

Finally, we show the expressivity of PLΩ by formalizing a version of Cantor's theorem.

Expressive Formalism for Mathematics

▷ **Example 10.0.8 (Cantor's Theorem)** The cardinality of a set is smaller than that of its power set.

- ▷ smaller-card$(M, N) := \neg(\exists F.\text{surjective}(F, M, N))$
- ▷ surjective$(F, M, N) := (\forall X \in M.\exists Y \in N.FY = X)$

▷ **Example 10.0.9 (Simplified Formalization)** $\neg(\exists F_{\iota \to \iota \to \iota}.\forall G_{\iota \to \iota}.\exists J_\iota.FJ = G)$

▷ Standard-Benchmark for higher-order theorem provers

▷ can be proven by TPS and LEO (see below)

The simplified formulation of Cantor's theorem in Example 10.0.9 uses the universe of type ι for the set S and universe of type $\iota \to \iota$ for the power set rather than quantifying over S explicitly.

The next concern is to find a calculus for PLΩ.

We start out with the simplest one we can imagine, a Hilbert-style calculus that has been adapted to higher-order logic by letting the inference rules range over PLΩ formulae and insisting that substitutions are well-typed.

Hilbert-Calculus

▷ **Definition 10.0.10 ($\mathcal{H}_\Omega$ Axioms)** ▷ $\forall P_o, Q_o . P \Rightarrow Q \Rightarrow P$

- ▷ $\forall P_o, Q_o, R_o . (P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R$
- ▷ $\forall P_o, Q_o . (\neg P \Rightarrow \neg Q) \Rightarrow P \Rightarrow Q$

▷ **Definition 10.0.11 ($\mathcal{H}_\Omega$ Inference rules)**

$$\frac{\mathbf{A}_o \Rightarrow \mathbf{B}_o \quad \mathbf{A}}{\mathbf{B}} \qquad \frac{\forall X_\alpha . \mathbf{A}}{[\mathbf{B}/X_\alpha](\mathbf{A})} \qquad \frac{\mathbf{A}}{\forall X_\alpha . \mathbf{A}} \qquad \frac{X \notin \mathrm{free}(\mathbf{A}) \quad \forall X_\alpha . \mathbf{A} \wedge \mathbf{B}}{\mathbf{A} \wedge (\forall X_\alpha . \mathbf{B})}$$

▷ **Theorem 10.0.12** *Sound, wrt. standard semantics*

▷ Also Complete?

Not surprisingly, $\mathcal{H}_\Omega$ is sound, but it shows big problems with completeness. For instance, if we turn to a proof of Cantor's theorem via the well-known diagonal sequence argument, we will have to construct the diagonal sequence as a function of type $\iota \to \iota$, but up to now, we cannot in $\mathcal{H}_\Omega$. Unlike mathematical practice, which silently assumes that all functions we can write down in closed form exists, in logic, we have to have an axiom that guarantees (the existence of) such a function: the comprehension axioms.

Hilbert-Calculus $\mathcal{H}_\Omega$ (continued)

▷ valid sentences that are not $\mathcal{H}_\Omega$-theorems:

- ▷ Cantor's Theorem:
  $\neg (\exists F_{\iota \to \iota \to \iota} . \forall G_{\iota \to \iota} . (\forall K_\iota . (\mathbb{N} K) \Rightarrow \mathbb{N}(GK)) \Rightarrow (\exists J_\iota . (\mathbb{N} J) \wedge FJ = G))$
  (There is no surjective mapping from $\mathbb{N}$ into the set $\mathcal{F}(\mathbb{N};,)\mathbb{N}$ of natural number sequences)
- ▷ proof attempt fails at the subgoal $\exists G_{\iota \to \iota} . \forall X_\iota . GX = s(fXX)$

▷ Comprehension $\exists F_{\alpha \to \beta} . \forall X_\alpha . FX = \mathbf{A}_\beta$ (for every variable X_α and every term $\mathbf{A} \in \mathit{wff}_\beta(\Sigma, \mathcal{V}_\mathcal{T})$)

▷ extensionality
$\mathbf{Ext}^{\alpha\,\beta}$ $\forall F_{\alpha \to \beta} . \forall G_{\alpha \to \beta} . (\forall X_\alpha . FX = GX) \Rightarrow F = G$
$\mathbf{Ext}^o$ $\forall F_o . \forall G_o . (F \Leftrightarrow G) \Leftrightarrow F = G$

▷ correct! complete? cannot be!! [Göd31]

Actually it turns out that we need more axioms to prove elementary facts about mathematics: the extensionality axioms. But even with those, the calculus cannot be complete, even though empirically it proves all mathematical facts we are interested in.

Way Out: Henkin-Semantics

- Gödel's incompleteness theorem only holds for standard semantics
- find generalization that admits complete calculi:
- Idea: generalize so that the carrier only contains those functions that are requested by the comprehension axioms.
- **Theorem 10.0.13 (Henkin 1950)** $\mathcal{H}_\Omega$ *is complete wrt. this semantics.*
- Proof Sketch: more models $\leadsto$ less valid sentences (these are $\mathcal{H}_\Omega$-theorems) $\square$
- Henkin-models induce sensible measure of completeness for higher-order logic.

Actually, there is another problem with PLΩ: 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 PLΩ 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 difficul 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 $\mathbf{A} \in \mathit{wff}_\beta(\Sigma, \mathcal{V}_\mathcal{T})$ (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 $\mathbf{B} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$.
- **Definition 10.0.14** Axiom of β-equality: $(\lambda X_\alpha.\mathbf{A})\mathbf{B} = [\mathbf{B}/X](\mathbf{A}_\beta)$
- new formulae (λ-calculus [Church 1940])

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

From Extensionality to η-Conversion

- **Definition 10.0.15** 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 10.0.16** We say that $\mathbf{A}$ and $\lambda X_\alpha.\mathbf{A}X$ are η-equal, (write $\mathbf{A}_{\alpha\to\beta} =_\eta (\lambda X_\alpha.\mathbf{A}X)$, if), iff $X \notin \mathrm{free}(\mathbf{A})$.
- **Theorem 10.0.17** *η-equality and Extensionality are equivalent*
- Proof: We show that η-equality is special case of extensionality; the converse entailment is trivial

  P.1 Let $\forall X_\alpha.\mathbf{A}X = \mathbf{B}X$, thus $\mathbf{A}X = \mathbf{B}X$ with $\forall E$

  P.2 $\lambda X_\alpha.\mathbf{A}X = \lambda X_\alpha.\mathbf{B}X$, therefore $\mathbf{A} = \mathbf{B}$ with η

  P.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: $\forall F_o.\forall G_o.(F \Leftrightarrow G) \Leftrightarrow F = G$ 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.

Chapter 11

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)

- Signature $\Sigma = \bigcup_{\alpha \in \mathcal{T}} \Sigma_\alpha$ (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 11.0.1** We call the set $\mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ defined by the rules
  - $\mathcal{V}_\alpha \cup \Sigma_\alpha \subseteq \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$
  - If $\mathbf{C} \in \mathit{wff}_{\alpha \to \beta}(\Sigma, \mathcal{V}_\mathcal{T})$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$, then $(\mathbf{CA}) \in \mathit{wff}_\beta(\Sigma, \mathcal{V}_\mathcal{T})$
  - If $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$, then $(\lambda X_\beta . \mathbf{A}) \in \mathit{wff}_{\beta \to \alpha}(\Sigma, \mathcal{V}_\mathcal{T})$

  the set of well-typed formula e of type α over the signature Σ and use $\mathit{wff}_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T}) := \bigcup_{\alpha \in \mathcal{T}} \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ for the set of all well-typed formulae.
- **Definition 11.0.2** 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. $\sigma(X_\alpha) \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ and capture-avoiding.
- **Definition 11.0.3 (Simply Typed λ-Calculus)** The simply typed λ-calculus $\Lambda^\to$ over a signature Σ has the formulae $\mathit{wff}_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T})$ (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: $(\lambda X . \mathbf{A}X) =_\eta \mathbf{A}$

 94 JACOBS UNIVERSITY

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 $(\lambda X_\alpha.\mathbf{A}X)\mathbf{B} =_\beta \mathbf{AB}$.

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)

- ▷ **Notation 11.0.4 (Application is left-associative)** We abbreviate $(((\mathbf{FA}^1)\mathbf{A}^2)\ldots)\mathbf{A}^n$ with $\mathbf{FA}^1\ldots\mathbf{A}^n$ eliding the brackets and further with $\mathbf{F}\overline{\mathbf{A}^n}$ in a kind of vector notation.
- ▷ A **.** stands for a left bracket whose partner is as far right as is consistent with existing brackets; i.e. $\mathbf{A.BC}$ abbreviates $\mathbf{A}(\mathbf{BC})$.
- ▷ **Notation 11.0.5 (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.
- ▷ **Notation 11.0.6 (Outer brackets)** Finally, we allow ourselves to elide outer brackets where they can be inferred.

 95 JACOBS UNIVERSITY

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 $\mathbf{B}$.[8] EdN:8

In this presentation of the simply typed λ-calculus we build-in α-equality and use capture-avoiding substitutions directly. A clean introduction would followed the steps in Chapter 6 by introducing substitutions with a substitutability condition like the one in Definition 7.2.10, then establishing the soundness of α conversion, and only then postulating defining capture-avoiding substitution application as in Definition 7.3.3. 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)

- ▷ reduction with $\left\{\begin{array}{l}\beta:\ (\lambda X.\mathbf{A})\mathbf{B}\to_\beta[\mathbf{B}/X](\mathbf{A})\\ \eta:\ (\lambda X.\mathbf{A}X)\to_\eta\mathbf{A}\end{array}\right.$ under $=_\alpha$: $\begin{array}{c}\lambda X.\mathbf{A}\\ =_\alpha\\ \lambda Y.[Y/X](\mathbf{A})\end{array}$
- ▷ **Theorem 11.0.7** *$\beta\eta$-reduction is well-typed, terminating and confluent in the presence of $=_\alpha$-conversion.*
- ▷ **Definition 11.0.8 (Normal Form)** We call a λ-term $\mathbf{A}$ a normal form (in

[8] EdNote: rationalize the semantic macros for syntax!

a reduction system $\mathcal{E}$), iff no rule (from $\mathcal{E}$) can be applied to $\mathbf{A}$.

▷ **Corollary 11.0.9** *$\beta\eta$-reduction yields unique normal forms (up to α-equivalence).*

We will now introduce some terminology to be able to talk about λ-terms and their parts.

Syntactic Parts of λ-Terms

▷ **Definition 11.0.10 (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 11.0.11** Head Reduction always has a unique β redex

$$(\lambda \overline{X^n}.(\lambda Y.\mathbf{A})\mathbf{B}^1 \ldots \mathbf{B}^n) \to_\beta^h (\lambda \overline{X^n}.[\mathbf{B}^1/Y](\mathbf{A})\mathbf{B}^2 \ldots \mathbf{B}^n)$$

▷ **Theorem 11.0.12** *The syntactic heads of β-normal forms are constant or variables.*

▷ **Definition 11.0.13** 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 j^{th} bound variable.

▷ **Definition 11.0.14** We call a λ-term a η-long form, iff its matrix has base type.

▷ **Definition 11.0.15** 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 11.0.16** 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 $\overline{\alpha_n} \to \beta$ in η-long form, where $\beta \in \mathcal{B}\mathcal{T}$, then $\mathbf{A}$ must be of the form $\lambda \overline{X_\alpha}^n.\mathbf{B}$, 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 $\Lambda^\to$ separately and thus reduce argumentational complexity.

Chapter 12

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 Chapter we prove the respective properties. We start out with termination, since we will need it later in the proof of confluence.

12.1 Termination of β-reduction

We will use the termination of β 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 12.1.1 (Typed β-Reduction terminates)** *For all $\mathbf{A} \in wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$, 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 12.1.2** $\mathbf{A}$ is called strongly reducing at type α (write $\mathcal{SR}(\mathbf{A},\alpha)$), iff each chain β-reductions from $\mathbf{A}$ terminates.
- 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 $\mathcal{LR}(\mathbf{CA},\beta)$ for all $\mathbf{A}\in wff_\alpha(\Sigma,\mathcal{V}_\mathcal{T})$ with $\mathcal{LR}(\mathbf{A},\alpha)$.

Proof: Termination Proof

- **P.1** $\mathcal{LR}\subseteq\mathcal{SR}$ (Lemma 12.1.4 b))

  $\mathbf{A}\in wff_\alpha(\Sigma,\mathcal{V}_\mathcal{T})$ implies $\mathcal{LR}(\mathbf{A},\alpha)$ (Theorem 12.1.8 with $\sigma=\emptyset$)

  thus $\mathcal{SR}(\mathbf{A},\alpha)$. □

P.2 P.3 ▷ **Lemma 12.1.3 ($\mathcal{SR}$ 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 Idea: Every infinite β-reduction from $\mathbf{B}$ would be one from $\mathbf{A}$. □

 99 JACOBS UNIVERSITY

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 12.1.4 (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

- **P.1.1** α base type:

  P.1.1.1.1 a): $h\overline{\mathbf{A}^n}$ is strongly reducing, since the $\mathbf{A}^i$ are (brackets!)

  P.1.1.1.2 so $\mathcal{LR}(h\overline{\mathbf{A}^n},\alpha)$ as α is a base type ($\mathcal{SR}=\mathcal{LR}$) □

  P.1.1.2 b): by definition □

  $\alpha=\beta\to\gamma$:

  P.1.2.1.1 a): Let $\mathcal{LR}(\mathbf{B},\beta)$.

  P.1.2.1.2 by IH b) we have $\mathcal{SR}(\mathbf{B},\beta)$, and $\mathcal{LR}((h\overline{\mathbf{A}^n})\mathbf{B},\gamma)$ by IH a)

P.1.2.1.1.3 so $\mathcal{LR}(h\overline{\mathbf{A}^n}, \alpha)$ by definition. □

P.1.2.1.2 b): Let $\mathcal{LR}(\mathbf{A}, \alpha)$ and $X_\beta \notin \mathrm{free}(\mathbf{A})$.

P.1.2.1.2.2 $\mathcal{LR}(X, \beta)$ by IH a) with $n = 0$, thus $\mathcal{LR}(\mathbf{A}X, \gamma)$ by definition.

P.1.2.1.2.3 By IH b) we have $\mathcal{SR}(\mathbf{A}X, \gamma)$ and by Lemma 12.1.3 $\mathcal{SR}(\mathbf{A}, \alpha)$. □

□

□

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 12.1.5** *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:

P.1 Let $\alpha = \overline{\gamma_i} \to \delta$ where δ base type and $\mathcal{LR}(\mathbf{C}^i, \gamma^i)$

P.2 It is sufficient to show that $\mathcal{SR}(((\lambda X.\mathbf{A})\mathbf{B})\overline{\mathbf{C}}, \delta)$, as δ base type

P.3 We have $\mathcal{LR}([\mathbf{B}/X](\mathbf{A})\overline{\mathbf{C}}, \delta)$ by hypothesis and definition of $\mathcal{LR}$.

P.4 thus $\mathcal{SR}([\mathbf{B}/X](\mathbf{A})\overline{\mathbf{C}}, \delta)$, as δ base type.

P.5 in particular $\mathcal{SR}([\mathbf{B}/X](\mathbf{A}), \alpha)$ and $\mathcal{SR}(\mathbf{C}^i, \gamma^i)$ (subterms)

P.6 $\mathcal{SR}(\mathbf{B}, \beta)$ by hypothesis and Lemma 12.1.4

P.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 $\mathbf{C}^i$.

P.8 so an infinite reduction from $((\lambda X.\mathbf{A})\mathbf{B})\overline{\mathbf{C}}$ must have the form

$$\begin{array}{rcl} ((\lambda X.\mathbf{A})\mathbf{B})\overline{\mathbf{C}} & \to^*_\beta & ((\lambda X.\mathbf{A}')\mathbf{B}')\overline{\mathbf{C}'} \\ & \to^1_\beta & [\mathbf{B}'/X](\mathbf{A}')\overline{\mathbf{C}'} \\ & \to^*_\beta & \ldots \end{array}$$

where $\mathbf{A} \to^*_\beta \mathbf{A}'$, $\mathbf{B} \to^*_\beta \mathbf{B}'$ and $\mathbf{C}^i \to^*_\beta \mathbf{C}^{i'}$

P.9 so we have $[\mathbf{B}/X](\mathbf{A}) \to^*_\beta [\mathbf{B}'/X](\mathbf{A}')$

P.10 so we have the infinite reduction

$$\begin{array}{rcl} [\mathbf{B}/X](\mathbf{A})\overline{\mathbf{C}} & \to^*_\beta & [\mathbf{B}'/X](\mathbf{A}')\overline{\mathbf{C}'} \\ & \to^*_\beta & \ldots \end{array}$$

which contradicts our assumption □

▷ **Lemma 12.1.6 ($\mathcal{LR}$ 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 $=_\beta$ 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 inductive 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.

$\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ implies $\mathcal{LR}(\mathbf{A}, \alpha)$

▷ **Definition 12.1.7** We write $\mathcal{LR}(\sigma)$ if $\mathcal{LR}(\sigma(X_\alpha), \alpha)$ for all $X \in \mathbf{supp}(\sigma)$.

▷ **Theorem 12.1.8** *If* $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$*, then* $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$ *for any substitution* σ *with* $\mathcal{LR}(\sigma)$*.*

▷ Proof: by induction on the structure of $\mathbf{A}$

P.1.1 $\mathbf{A} = X_\alpha \in \mathbf{supp}(\sigma)$: then $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$ by assumption

P.1.2 $\mathbf{A} = X \notin \mathbf{supp}(\sigma)$: then $\sigma(\mathbf{A}) = \mathbf{A}$ and $\mathcal{LR}(\mathbf{A}, \alpha)$ by Lemma 12.1.4 with $n = 0$.

P.1.3 $\mathbf{A} \in \Sigma$: then $\sigma(\mathbf{A}) = \mathbf{A}$ as above

P.1.4 $\mathbf{A} = \mathbf{BC}$: by IH $\mathcal{LR}(\sigma(\mathbf{B}), \gamma \to \alpha)$ and $\mathcal{LR}(\sigma(\mathbf{C}), \gamma)$

P.1.4.2 so $\mathcal{LR}(\sigma(\mathbf{B})\sigma(\mathbf{C}), \alpha)$ by definition of $\mathcal{LR}$. □

P.1.5 $\mathbf{A} = \lambda X_\beta.\mathbf{C}_\gamma$: Let $\mathcal{LR}(\mathbf{B}, \beta)$ and $\theta := \sigma, [\mathbf{B}/X]$, then θ meets the conditions of the IH.

P.1.5.2 Moreover $\sigma(\lambda X_\beta.\mathbf{C}_\gamma)\mathbf{B} \to_\beta \sigma, [\mathbf{B}/X](\mathbf{C}) = \theta(\mathbf{C})$.

P.1.5.3 Now, $\mathcal{LR}(\theta(\mathbf{C}), \gamma)$ by IH and thus $\mathcal{LR}(\sigma(\mathbf{A})\mathbf{B}, \gamma)$ by Lemma 12.1.6.

P.1.5.4 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 Lemma 12.1.4, 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 Lemma 12.1.4– becomes easy.

The last case on λ-abstraction reveals why we had to strengthen the inductive hypothesis: $=_\beta$ reduction introduces a substitution which may increase the size of the subterm, which in turn keeps us from applying the inductive 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 β-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 β reduction.

Closure under Head β-Expansion (weakly reducing)

▷ **Lemma 12.1.9 ($\mathcal{LR}$ 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

P.1.1 α base type:

P.1.1.1 we have $\mathcal{SR}(\mathbf{D}, \alpha)$ by definition

P.1.1.2 so $\mathcal{SR}(\mathbf{C}, \alpha)$, since head reduction is unique

P.1.1.3 and thus $\mathcal{LR}(\mathbf{C}, \alpha)$. □

P.1.2 $\alpha = \beta \to \gamma$:

P.1.2.1 Let $\mathcal{LR}(\mathbf{B}, \beta)$, by definition we have $\mathcal{LR}(\mathbf{DB}, \gamma)$.

P.1.2.2 but $\mathbf{CB} \to^h_\beta \mathbf{DB}$, so $\mathcal{LR}(\mathbf{CB}, \gamma)$ by IH

P.1.2.3 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 β-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 Lemma 12.1.3 (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.

12.2 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 12.2.1 (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 $a \to^1_R b$ $a \to^1_R c$ there is a $d \in A$ with $b \to^1_R d$ and $c \to^1_R d$.
- ▷ 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 $c \to_R^* d$.

diamond property	confluent	weakly confluent
$\begin{array}{ccc} & a & \\ \swarrow & & \searrow \\ b & & c \\ \searrow & & \swarrow \\ & d & \end{array}$	$\begin{array}{ccc} & a & \\ {*}\swarrow & & \searrow{*} \\ b & & c \\ {*}\searrow & & \swarrow{*} \\ & d & \end{array}$	$\begin{array}{ccc} & a & \\ \swarrow & & \searrow \\ b & & c \\ {*}\searrow & & \swarrow{*} \\ & d & \end{array}$

The diamond property is very simple, but not many reduction relations enjoy it. Confluence is the notion that 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 12.2.2** *If a rewrite relation has a diamond property, then it is weakly confluent.*

▷ **Theorem 12.2.3** *If a rewrite relation has a diamond property, then it is confluent.*

▷ Proof Idea: by a tiling argument, composing 1×1 diamonds to an $n \times m$ diamond. □

▷ **Theorem 12.2.4 (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.

η is very well-behaved – i.e. confluent and terminating

η-Reduction ist terminating and confluent

▷ **Lemma 12.2.5** *η-Reduction ist terminating*

▷ Proof: by a simple counting argument □

▷ **Lemma 12.2.6** *η-reduction is confluent.*

▷ Proof Idea: We show that η-reduction has the diamond property by diagram chase over

$$
\begin{tikzcd}[column sep=small]
 & \lambda X.\mathbf{A}X \arrow[dl] \arrow[dr] & \\
\mathbf{A} \arrow[dr] & & \lambda X.\mathbf{A}'X \arrow[dl] \\
 & \mathbf{A}' &
\end{tikzcd}
$$

where $\mathbf{A} \rightarrow_\eta \mathbf{A}'$. Then the assertion follows by Theorem 12.2.3. □

For β-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 Newman's Lemma

β is confluent

▷ **Lemma 12.2.7** *β-Reduction is weakly confluent.*

▷ Proof Idea: by diagram chase over

$$
\begin{tikzcd}[column sep=small]
 & & (\lambda X.\mathbf{A})\mathbf{B} \arrow[dll] \arrow[d] \arrow[drr] & & \\
(\lambda X.\mathbf{A}')\mathbf{B} \arrow[dr] & & (\lambda X.\mathbf{A})\mathbf{B}' \arrow[dl] \arrow[dr] & & [\mathbf{B}/X](\mathbf{A}) \arrow[dl, "*"] \\
 & (\lambda X.\mathbf{A}')\mathbf{B}' \arrow[dr] & & [\mathbf{B}'/X](\mathbf{A}) \arrow[dl] & \\
 & & [\mathbf{B}'/X](\mathbf{A}') & &
\end{tikzcd}
$$

□

▷ **Corollary 12.2.8** *β-Reduction is confluent.*

▷ Proof Idea: by Newman's Lemma. □

There is one reduction in the diagram in the proof of Lemma 12.2.7 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 β and η 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 12.2.9** 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 $Z \rightarrow_\mathcal{R} W$.

$$
\begin{tikzcd}[column sep=small]
 & X \arrow[dl, "\mathcal{R}"'] \arrow[dr, "\mathcal{S}"] & \\
Y \arrow[dr, "\mathcal{S}"'] & & Z \arrow[dl, "\mathcal{R}"] \\
 & W &
\end{tikzcd}
$$

▷ **Observation 12.2.10** *If $\mathcal{R}$ and $\mathcal{S}$ commute, then $\rightarrow_\mathcal{R}$ and $\rightarrow_\mathcal{S}$ do as well.*

▷ **Observation 12.2.11** *$\mathcal{R}$ is confluent, if $\mathcal{R}$ commutes with itself.*

▷ **Lemma 12.2.12** *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. □

 108 JACOBS UNIVERSITY

This directly gives us our goal.

$\beta\eta$ is confluent

▷ **Lemma 12.2.13** *$\rightarrow^*{}_{\beta}$ and $\rightarrow^*{}_{\eta}$ commute.*

▷ Proof Sketch: diagram chase □

 109 JACOBS UNIVERSITY

Chapter 13

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 13.0.1** We call a collection $\mathcal{D}_{\mathcal{T}} := \{\mathcal{D}_\alpha \,|\, \alpha \in \mathcal{T}\}$ a typed collection (of sets) and a collection $f_{\mathcal{T}} \colon \mathcal{D}_{\mathcal{T}} \to \mathcal{E}_{\mathcal{T}}$, a typed function, iff $f_\alpha \colon \mathcal{D}_\alpha \to \mathcal{E}_\alpha$.
- **Definition 13.0.2** A typed collection $\mathcal{D}_{\mathcal{T}}$ is called a frame, iff $\mathcal{D}_{\alpha\to\beta} \subseteq \mathcal{D}_\alpha \to \mathcal{D}_\beta$
- **Definition 13.0.3** Given a frame $\mathcal{D}_{\mathcal{T}}$, and a typed function $\mathcal{I} \colon \Sigma \to \mathcal{D}$, then we call $\mathcal{I}_\varphi \colon \mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}}) \to \mathcal{D}$ the value function induced by $\mathcal{I}$, iff
  - $\mathcal{I}_\varphi|_{\mathcal{V}_{\mathcal{T}}} = \varphi$, $\quad \mathcal{I}_\varphi|_\Sigma = \mathcal{I}$
  - $\mathcal{I}_\varphi(\mathbf{A}\mathbf{B}) = \mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
  - $\mathcal{I}_\varphi(\lambda X_\alpha.\mathbf{A})$ is that function $f \in \mathcal{D}_{\alpha\to\beta}$, such that $f(a) = \mathcal{I}_{\varphi,[a/X]}(\mathbf{A})$ for all $a \in \mathcal{D}_\alpha$
- **Definition 13.0.4** We call a frame $\langle\mathcal{D},\mathcal{I}\rangle$ comprehension-closed or a Σ-algebra, iff $\mathcal{I}_\varphi \colon \mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}}) \to \mathcal{D}$ is total. (every λ-term has a value)

 110

13.1 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 13.1.1 (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 $\mathbf{A}$

P.1 we have five cases

P.1.1 $\mathbf{A} = X$: Then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.2 $\mathbf{A} = Y \neq X$ and $Y \in \mathcal{V}_\mathcal{T}$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.

P.1.3 $\mathbf{A} \in \Sigma$: This is analogous to the last case.

P.1.4 $\mathbf{A} = \mathbf{C}\mathbf{D}$: then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{C}\mathbf{D})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{C})[\mathbf{B}/X](\mathbf{D})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{C}))(\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{D}))) = \mathcal{I}_\psi(\mathbf{C})(\mathcal{I}_\psi(\mathbf{D})) = \mathcal{I}_\psi(\mathbf{C}\mathbf{D}) = \mathcal{I}_\psi(\mathbf{A})$

P.1.5 $\mathbf{A} = \lambda Y_\alpha.\mathbf{C}$:

P.1.5.1 We can assume that $X \neq Y$ and $Y \notin \mathrm{free}(\mathbf{B})$

P.1.5.2 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 13.1.2** *Let* $\mathcal{A} := \langle \mathcal{D}, \mathcal{I} \rangle$ *be a* Σ*-algebra and* $Y \notin \mathrm{free}(\mathbf{A})$*, 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

$$\begin{aligned} \mathcal{I}_\varphi(\lambda Y.[Y/X]\mathbf{A}) @ \mathsf{a} &= \mathcal{I}_{\varphi,[a/Y]}([Y/X](\mathbf{A})) \\ &= \mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) \\ &= \mathcal{I}_\varphi(\lambda X.\mathbf{A}) @ \mathsf{a} \end{aligned}$$

▷ **Theorem 13.1.3** *If* $\mathcal{A} := \langle \mathcal{D}, \mathcal{I} \rangle$ *is a* Σ*-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 13.1.4** *If $X \notin \mathrm{free}(\mathbf{A})$, then $\mathcal{I}_\varphi(\lambda X.\mathbf{A}X) = \mathcal{I}_\varphi(\mathbf{A})$ for all φ.*

▷ Proof: by calculation

$$\begin{array}{rcll} \mathcal{I}_\varphi(\lambda X.\mathbf{A}X) @ \mathsf{a} & = & \mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathbf{A}X) & \\ & = & \mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathbf{A}) @ \mathcal{I}_{\varphi,[\mathsf{a}/X]}(X) & \\ & = & \mathcal{I}_\varphi(\mathbf{A}) @ \mathcal{I}_{\varphi,[\mathsf{a}/X]}(X) & \text{as } X \notin \mathrm{free}(\mathbf{A}). \\ & = & \mathcal{I}_\varphi(\mathbf{A}) @ \mathsf{a} & \end{array}$$

▷ **Theorem 13.1.5** *$\alpha\beta\eta$-equality is sound wrt. Σ-algebras. (if $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$, then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$ for all assignments φ)*

13.2 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 13.2.1** We call a term $\mathbf{A} \in \mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$ a β normal form iff there is no $\mathbf{B} \in \mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$ 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 $\mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})\downarrow_{\beta\eta}$.

▷ We have just proved that $\beta\eta$-reduction is terminating and confluent, so we have

▷ **Corollary 13.2.2 (Normal Forms)** *Every $\mathbf{A} \in \mathit{wff}_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$ has a unique β normal form ($\beta\eta$, long $\beta\eta$ normal form), which we denote by $\mathbf{A}\downarrow_\beta$ ($\mathbf{A}\downarrow_{\beta\eta}$ $\mathbf{A}\downarrow^l_{\beta\eta}$)*

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 13.2.3** 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 13.2.4** 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 13.2.5** $=_\beta$ ($=_{\beta\eta}$) is a (functional) congruence on $c\,wff_{\mathcal{T}}(\Sigma)$ by definition.

▷ **Theorem 13.2.6** *Let $\mathcal{D}$ be a Σ-frame and $\sim$ a functional congruence on $\mathcal{D}$, then the quotient space $\mathcal{D}/_\sim$ is a Σ-frame.*

▷ Proof:

P.1 $\mathcal{D}/_\sim = \{[f]_\sim \mid f \in \mathcal{D}\}$, define $[f]_\sim([a]_\sim) := [f(a)]_\sim$.

P.2 This only depends on equivalence classes: Let $f' \in [f]_\sim$ and $a' \in [a]_\sim$.

P.3 Then $[f(a)]_\sim = [f'(a)]_\sim = [f'(a')]_\sim = [f(a')]_\sim$

P.4 To see that we have $[f]_\sim = [g]_\sim$, iff $f \sim g$, iff $f(a) = g(a)$ since $\sim$ is functional.

P.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 $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$ and then specialize this result to show that is is also functional on $c\,wff_{\mathcal{T}}(\Sigma)$ by a grounding argument.

$\beta\eta$-Equivalence as a Functional Congruence

▷ **Lemma 13.2.7** *$\beta\eta$-equality is a functional congruence on $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$.*

▷ Proof: Let $\mathbf{AC} =_{\beta\eta} \mathbf{BC}$ for all $\mathbf{C}$ and $X \in (\mathcal{V}_\gamma \backslash (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{B})))$.

P.1 then (in particular) $\mathbf{A}X =_{\beta\eta} \mathbf{B}X$, and

P.2 $(\lambda X.\mathbf{A}X) =_{\beta\eta} (\lambda X.\mathbf{B}X)$, since $\beta\eta$-equality acts on subterms.

P.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 13.2.8** We call an injective substitution $\sigma\colon \mathrm{free}(\mathbf{C}) \to \Sigma$ a grounding substitution for $\mathbf{C} \in wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$, 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 13.2.9** *$\beta\eta$-equality is a functional congruence on $c\,wff_{\mathcal{T}}(\Sigma)$.*

▷ Proof: We use Lemma 13.2.7

P.1 Let $\mathbf{A}, \mathbf{B} \in c\,wff_{(\alpha\to\beta)}(\Sigma)$, such that $\mathbf{A} \neq_{\beta\eta} \mathbf{B}$.

P.2 As $\beta\eta$ is functional on $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$, there must be a $\mathbf{C}$ with $\mathbf{AC} \neq_{\beta\eta} \mathbf{BC}$.

P.3 Now let $\mathbf{C}' := \sigma(\mathbf{C})$, for a grounding substitution σ.

P.4 Any $\beta\eta$ conversion sequence for $\mathbf{AC}' \neq_{\beta\eta} \mathbf{BC}'$ induces one for $\mathbf{AC} \neq_{\beta\eta} \mathbf{BC}$.

P.5 Thus we have shown that $\mathbf{A} \neq_{\beta\eta} \mathbf{B}$ entails $\mathbf{AC}' \neq_{\beta\eta} \mathbf{BC}'$. □

Note that: the result for $c\,wff_{\mathcal{T}}(\Sigma)$ is sharp. For instance, if $\Sigma = \{c_\iota\}$, 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 $\{c\} = c\,wff_\iota(\Sigma)$ (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 $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$ we could always have used a variable.

This completes the preparation and we can define the notion of a term algebra, i.e. a Σ-algebra whose frame is made of $\beta\eta$-normal λ-terms.

A Herbrand Model for $\Lambda^{\to}$

- **Definition 13.2.10** We call $\mathcal{T}_{\beta\eta} := \langle c\,wff_{\mathcal{T}}(\Sigma){\downarrow_{\beta\eta}}, \mathcal{I}^{\beta\eta}\rangle$ the Σ term algebra, if $\mathcal{I}^{\beta\eta} = \mathrm{Id}_\Sigma$.
- The name "term algebra" in the previous definition is justified by the following
- **Theorem 13.2.11** *$\mathcal{T}_{\beta\eta}$ is a Σ-algebra*
- Proof: We use the work we did above

P.1 Note that $c\,wff_{\mathcal{T}}(\Sigma){\downarrow_{\beta\eta}} = c\,wff_{\mathcal{T}}(\Sigma)/_{=_{\beta\eta}}$ and thus a Σ-frame by Theorem 13.2.6 and Lemma 13.2.7.

P.2 So we only have to show that the value function $\mathcal{I}^{\beta\eta} = \mathrm{Id}_\Sigma$ is total.

P.3 Let φ be an assignment into $c\,wff_{\mathcal{T}}(\Sigma){\downarrow_{\beta\eta}}$.

P.4 Note that $\sigma := (\varphi|_{\mathrm{free}(\mathbf{A})})$ is a substitution, since $\mathrm{free}(\mathbf{A})$ is finite.

P.5 A simple induction on the structure of $\mathbf{A}$ shows that $\mathcal{I}^{\beta\eta}_\varphi(\mathbf{A}) = \sigma(\mathbf{A}){\downarrow_{\beta\eta}}$.

P.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 Σ-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 Σ-algebras.

Completetness of $\alpha\beta\eta$-Equality

- **Theorem 13.2.12** *$\mathbf{A} = \mathbf{B}$ is valid in the class of Σ-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 Σ-algebra.

P.1 If $\mathbf{A} = \mathbf{B}$ is valid in all Σ-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 $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$.

P.2 If the equation has free variables, then the argument is more subtle.

P.3 Let σ be a grounding substitution for $\mathbf{A}$ and $\mathbf{B}$ and φ the induced variable assignment.

P.4 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 $\sigma(\mathbf{B})$.

P.5 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 $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$. □

Theorem 13.2.12 and Theorem 13.1.5 complete our study of the sematnics of the simply-typed λ-calculus by showing that it is an adequate logic for modeling (the equality) of functions and their applications.

Chapter 14

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 14.0.1** A grammar for the raw terms of the simply typed λ-calculus:

$$\begin{array}{lcl} \alpha & :== & c \mid \alpha \to \alpha \\ \Sigma & :== & \cdot \mid \Sigma, [c : \mathrm{type}] \mid \Sigma, [c : \alpha] \\ \Gamma & :== & \cdot \mid \Gamma, [x : \alpha] \\ \mathbf{A} & :== & c \mid X \mid \mathbf{A}^1\mathbf{A}^2 \mid \lambda X_\alpha . \mathbf{A} \end{array}$$

▷ 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 14.0.2** Judgments make statements about complex properties of the syntactic entities defined by the grammar.

▷ **Definition 14.0.3** Judgments for the simply typed λ-calculus

$\vdash \Sigma : \mathrm{sig}$	Σ is a well-formed signature
$\Sigma \vdash \alpha : \mathrm{type}$	α is a well-formed type given the type assumptions in Σ
$\Sigma \vdash \Gamma : \mathrm{ctx}$	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

▷ $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$, iff $\Gamma \vdash_\Sigma \mathbf{A} \colon \alpha$ derivable in

$$\frac{\Sigma \vdash \Gamma : \text{ctx} \quad \Gamma(X) = \alpha}{\Gamma \vdash_\Sigma X \colon \alpha}\text{wff:var} \qquad \frac{\Sigma \vdash \Gamma : \text{ctx} \quad \Sigma(c) = \alpha}{\Gamma \vdash_\Sigma c \colon \alpha}\text{wff:const}$$

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} \colon \beta \to \alpha \quad \Gamma \vdash_\Sigma \mathbf{B} \colon \beta}{\Gamma \vdash_\Sigma \mathbf{AB} \colon \alpha}\text{wff:app} \qquad \frac{\Gamma, [X : \beta] \vdash_\Sigma \mathbf{A} \colon \alpha}{\Gamma \vdash_\Sigma \lambda X_\beta.\mathbf{A} \colon \beta \to \alpha}\text{wff:abs}$$

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

$$\frac{}{\vdash \cdot : \text{sig}}\text{sig:empty} \qquad \frac{\vdash \Sigma : \text{sig}}{\vdash \Sigma, [\alpha : \text{type}] : \text{sig}}\text{sig:type}$$

$$\frac{\vdash \Sigma : \text{sig} \quad \Sigma \vdash \alpha : \text{type}}{\vdash \Sigma, [c : \alpha] : \text{sig}}\text{sig:const}$$

$$\frac{\Sigma \vdash \alpha : \text{type} \quad \Sigma \vdash \beta : \text{type}}{\Sigma \vdash \alpha \to \beta : \text{type}}\text{typ:fn} \qquad \frac{\vdash \Sigma : \text{sig} \quad \Sigma(\alpha) = \text{type}}{\Sigma \vdash \alpha : \text{type}}\text{typ:start}$$

$$\frac{\vdash \Sigma : \text{sig}}{\Sigma \vdash \cdot : \text{ctx}}\text{ctx:empty} \qquad \frac{\Sigma \vdash \Gamma : \text{ctx} \quad \Sigma \vdash \alpha : \text{type}}{\Sigma \vdash \Gamma, [X : \alpha] : \text{ctx}}\text{ctx:var}$$

Example: A Well-Formed Signature

▷ Let $\Sigma := [\alpha : \text{type}], [f : \alpha \to \alpha \to \alpha]$, then Σ is a well-formed signature, since we have derivations $\mathcal{A}$ and $\mathcal{B}$

$$\frac{\vdash \cdot : \text{sig}}{\vdash [\alpha : \text{type}] : \text{sig}}\text{sig:type} \qquad \frac{\mathcal{A} \quad [\alpha : \text{type}](\alpha) = \text{type}}{[\alpha : \text{type}] \vdash \alpha : \text{type}}\text{typ:start}$$

and with these we can construct the derivation $\mathcal{C}$

$$\cfrac{\mathcal{A} \quad \cfrac{\mathcal{B} \quad \cfrac{\mathcal{B} \quad \mathcal{B}}{[\alpha : \text{type}] \vdash \alpha \to \alpha : \text{type}}\text{typ:fn}}{[\alpha : \text{type}] \vdash \alpha \to \alpha \to \alpha : \text{type}}\text{typ:fn}}{\vdash \Sigma : \text{sig}}\text{sig:const}$$

Example: A Well-Formed λ-Term

▷ using Σ from above, we can show that $\Gamma := [X : \alpha]$ is a well-formed context:

$$\dfrac{\dfrac{\mathcal{C}}{\Sigma \vdash \cdot : \text{ctx}}\,\text{ctx:empty} \qquad \dfrac{\mathcal{C} \quad \Sigma(\alpha) = \text{type}}{\Sigma \vdash \alpha : \text{type}}\,\text{typ:start}}{\Sigma \vdash \Gamma : \text{ctx}}\,\text{ctx:var}$$

We call this derivation $\mathcal{G}$ and use it to show that

▷ $\lambda X_\alpha . fXX$ is well-typed and has type $\alpha \to \alpha$ in Σ. This is witnessed by the type derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{C} \quad \Sigma(f) = \alpha \to \alpha \to \alpha}{\Gamma \vdash_\Sigma f \colon \alpha \to \alpha \to \alpha}\,\text{wff:const} \qquad \dfrac{\mathcal{G}}{\Gamma \vdash_\Sigma X \colon \alpha}\,\text{wff:var}}{\Gamma \vdash_\Sigma fX \colon \alpha \to \alpha}\,\text{wff:app} \qquad \dfrac{\mathcal{G}}{\Gamma \vdash_\Sigma X \colon \alpha}\,\text{wff:var}}{\Gamma \vdash_\Sigma fXX \colon \alpha}\,\text{wff:app}}{\cdot \vdash_\Sigma \lambda X_\alpha . fXX \colon \alpha \to \alpha}\,\text{wff:abs}$$

$\beta\eta$-Equality by Inference Rules: One-Step Reduction

▷ One-step Reduction ($+ \in \{\alpha, \beta, \eta\}$)

$$\dfrac{\Gamma \vdash_\Sigma \mathbf{A} \colon \alpha \quad \Gamma \vdash_\Sigma \mathbf{B} \colon \beta}{\Gamma \vdash_\Sigma (\lambda X . \mathbf{A})\mathbf{B} \to_\beta^1 [\mathbf{B}/X](\mathbf{A})}\,\text{wff}\beta\text{:top}$$

$$\dfrac{\Gamma \vdash_\Sigma \mathbf{A} \colon \beta \to \alpha \quad X \notin \mathbf{dom}(\Gamma)}{\Gamma \vdash_\Sigma \lambda X . \mathbf{A}X \to_\eta^1 \mathbf{A}}\,\text{wff}\eta\text{:top}$$

$$\dfrac{\Gamma \vdash_\Sigma \mathbf{A} \to_+^1 \mathbf{B} \quad \Gamma \vdash_\Sigma \mathbf{AC} \colon \alpha}{\Gamma \vdash_\Sigma \mathbf{AC} \to_+^1 \mathbf{BC}}\,\text{tr:app}fn$$

$$\dfrac{\Gamma \vdash_\Sigma \mathbf{A} \to_+^1 \mathbf{B} \quad \Gamma \vdash_\Sigma \mathbf{CA} \colon \alpha}{\Gamma \vdash_\Sigma \mathbf{CA} \to_+^1 \mathbf{CB}}\,\text{tr:app}arg$$

$$\dfrac{\Gamma, [X : \alpha] \vdash_\Sigma \mathbf{A} \to_+^1 \mathbf{B}}{\Gamma \vdash_\Sigma \lambda X . \mathbf{A} \to_+^1 \lambda X . \mathbf{B}}\,\text{tr:abs}$$

$\beta\eta$-Equality by Inference Rules: Multi-Step Reduction

▷ Multi-Step-Reduction ($+ \in \{\alpha, \beta, \eta\}$)

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^1_+ \mathbf{B}}{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^*_+ \mathbf{B}}\text{ms:start} \qquad \frac{\Gamma \vdash_\Sigma \mathbf{A} \colon \alpha}{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^*_+ \mathbf{A}}\text{ms:ref}$$

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^*_+ \mathbf{B} \quad \Gamma \vdash_\Sigma \mathbf{B} \rightarrow^*_+ \mathbf{C}}{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^*_+ \mathbf{C}}\text{ms:trans}$$

▷ Congruence Relation

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} \rightarrow^*_+ \mathbf{B}}{\Gamma \vdash_\Sigma \mathbf{A} =_+ \mathbf{B}}\text{eq:start}$$

$$\frac{\Gamma \vdash_\Sigma \mathbf{A} =_+ \mathbf{B}}{\Gamma \vdash_\Sigma \mathbf{B} =_+ \mathbf{A}}\text{eq:sym} \qquad \frac{\Gamma \vdash_\Sigma \mathbf{A} =_+ \mathbf{B} \quad \Gamma \vdash_\Sigma \mathbf{B} =_+ \mathbf{C}}{\Gamma \vdash_\Sigma \mathbf{A} =_+ \mathbf{C}}\text{eq:trans}$$

Chapter 15

Higher-Order Unification

We now come to a very important (if somewhat non-trivial and under-appreciated) algorithm: higher-order unification, i.e. unification in the simply typed λ-calculus, i.e. unification modulo $\alpha\beta\eta$ equality.

15.1 Higher-Order Unifiers

Before we can start solving the problem of higher-order unification, we have to become clear about the terms we want to use. It turns out that "most general $\alpha\beta\eta$ unifiers may not exist – as Theorem 15.1.5 shows, there may be infinitely descending chains of unifiers that become more an more general. Thus we will have to generalize our concepts a bit here.

HOU: Complete Sets of Unifiers

- ▷ Question: Are there most general higher-order Unifiers?
- ▷ Answer: What does that mean anyway?
- ▷ **Definition 15.1.1** $\sigma =_{\beta\eta} \rho[W]$, iff $\sigma(X) =_{\alpha\beta\eta} \rho(X)$ for all $X \in W$. $\sigma =_{\beta\eta} \rho[\mathcal{E}]$ iff $\sigma =_{\beta\eta} \rho[\mathrm{free}(\mathcal{E})]$
- ▷ **Definition 15.1.2** σ is more general than θ on W ($\sigma \leq_{\beta\eta} \theta[W]$), iff there is a substitution ρ with $\theta =_{\beta\eta} \rho \circ \sigma[W]$.
- ▷ **Definition 15.1.3** $\Psi \subseteq \mathbf{U}(\mathcal{E})$ is a complete set of unifiers, iff for all unifiers $\theta \in \mathbf{U}(\mathcal{E})$ there is a $\sigma \in \Psi$, such that $\sigma \leq_{\beta\eta} \theta[\mathcal{E}]$.
- ▷ **Definition 15.1.4** If $\Psi \subseteq \mathbf{U}(\mathcal{E})$ is complete, then $\leq_{\beta\eta}$-minimal elements $\sigma \in \Psi$ are most general unifiers of $\mathcal{E}$.
- ▷ **Theorem 15.1.5** *The set* $\{[\lambda uv.hu/F]\} \cup \{\sigma_i \mid i \in \mathbb{N}\}$ *where*

$$\sigma_i := [\lambda uv.g_n u(u(h_1^n uv))\ldots(u(h_n^n uv))/F], [\lambda v.z/X]$$

is a complete set of unifiers for the equation $FXa_\iota =^? FXb_\iota$, *where* F *and* X *are variables of types* $(\iota \to \iota) \to \iota \to \iota$ *and* $\iota \to \iota$

Furthermore, σ_{i+1} *is more general than* σ_i.

▷ Proof Sketch: [Hue76, Theorem 5] □

The definition of a solved form in $\Lambda^{\rightarrow}$ is just as always; even the argument that solved forms are most general unifiers works as always, we only need to take $\alpha\beta\eta$ equality into account at every level.

Unification

▷ **Definition 15.1.6** $X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ is in solved form, if the X^i are distinct free variables $X^i \notin \mathrm{free}(\mathbf{B}^j)$ and $\mathbf{B}^j$ does not contain Skolem constants for all j.

▷ **Lemma 15.1.7** *If $\mathcal{E} = X^1 =^? \mathbf{B}^1 \wedge \ldots \wedge X^n =^? \mathbf{B}^n$ is in solved form, then $\sigma_{\mathcal{E}} := [\mathbf{B}^1/X^1], \ldots, [\mathbf{B}^n/X^n]$ is the unique most general unifier of $\mathcal{E}$*

▷ Proof:

P.1 $\sigma(X^i) =_{\alpha\beta\eta} \sigma(\mathbf{B}^i)$, so $\sigma \in \mathbf{U}(\mathcal{E})$

P.2 Let $\theta \in \mathbf{U}(\mathcal{E})$, then $\theta(X^i) =_{\alpha\beta\eta} \theta(\mathbf{B}^i) = \theta \circ \sigma(X^i)$

P.3 so $\theta \leq_{\beta\eta} \theta \circ \sigma[\mathcal{E}]$. □

15.2 Higher-Order Unification Transformations

We are now in a position to introduce the higher-order unifiation transformations. We proceed just like we did for first-order unification by casting the unification algorithm as a set of unification inference rules, leaving the control to a second layer of development.

We first look at a group of transformations that are (relatively) well-behaved and group them under the concept of "simplification", since (like the first-order transformation rules they resemble) have good properties. These are usually implemented in a group and applied eagerly.

Simplification $\mathcal{SIM}$

▷ **Definition 15.2.1** The higher-order simplification transformations $\mathcal{SIM}$ consist of the rules below.

$$\frac{(\lambda X_\alpha.\mathbf{A}) =^? (\lambda Y_\alpha.\mathbf{B}) \wedge \mathcal{E} \quad s \in \Sigma_\alpha^{Sk}\text{new}}{[s/X](\mathbf{A}) =^? [s/Y](\mathbf{B}) \wedge \mathcal{E}} \mathcal{SIM}{:}\alpha$$

$$\frac{(\lambda X_\alpha.\mathbf{A}) =^? \mathbf{B} \wedge \mathcal{E} \quad s \in \Sigma_\alpha^{Sk}\text{new}}{[s/X](\mathbf{A}) =^? \mathbf{B}s \wedge \mathcal{E}} \mathcal{SIM}{:}\eta$$

$$\frac{h\overline{\mathbf{U}^n} =^? h\overline{\mathbf{V}^n} \wedge \mathcal{E} \quad h \in (\Sigma \cup \Sigma^{Sk})}{\mathbf{U}^1 =^? \mathbf{V}^1 \wedge \ldots \wedge \mathbf{U}^n =^? \mathbf{V}^n \wedge \mathcal{E}} \mathcal{SIM}{:}\mathrm{dec}$$

$$\frac{\mathcal{E} \wedge X =^? \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad \mathbf{A} \cap \Sigma^{Sk} = \emptyset \quad X \in \mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =^? \mathbf{A}} \mathcal{SIM}{:}\mathrm{elim}$$

After rule applications all λ-terms are reduced to head normal form.

The main new feature of these rules (with respect to their first-order counterparts) is the handling of λ-binders. We eliminate them by replacing the bound variables by Skolem constants in the bodies: The $\mathcal{SIM}:\alpha$ standardizes them to a single one using α-equality, and $\mathcal{SIM}:\eta$ first η-expands the right-hand side (which must be of functional type) so that $\mathcal{SIM}:\alpha$ applies. Given that we are setting bound variables free in this process, we need to be careful that we do not use them in the $\mathcal{SIM}$:elim rule, as these would be variable-capturing.

Consider for instance the higher-order unification problem $(\lambda X.X) =^? (\lambda Y.W)$, which is unsolvable (the left hand side is the identity function and the right hand side some constant function – whose value is given by W). So after an application of $\mathcal{SIM}:\alpha$, we have $c =^? W$, which looks like it could be a solved pair, but the elimination rule prevents that by insisting that instances may not contain Skolem Variables.

Conceptually, $\mathcal{SIM}$ is a direct generalization of first-order unification transformations, and shares it properties; even the proofs go correspondingly.

Properties of Simplification

▷ **Lemma 15.2.2 (Properties of $\mathcal{SIM}$)** *$\mathcal{SIM}$ generalizes first-order unification.*

- ▷ *$\mathcal{SIM}$ is terminating and confluent up to α-conversion*
- ▷ *Unique $\mathcal{SIM}$ normal forms exist (all pairs have the form $h\overline{\mathbf{U}^n} =^? k\overline{\mathbf{V}^m}$)*

▷ **Lemma 15.2.3** *$\mathbf{U}(\mathcal{E} \wedge \mathcal{E}_\sigma) = \mathbf{U}(\sigma(\mathcal{E}) \wedge \mathcal{E}_\sigma)$.*

Proof: by the definitions

▷ **P.1** If $\theta \in \mathbf{U}(\mathcal{E} \wedge \mathcal{E}_\sigma)$, then $\theta \in (\mathbf{U}(\mathcal{E}) \cap \mathbf{U}(\mathcal{E}_\sigma))$.

So $\theta =_{\beta\eta} \theta \circ \sigma[\mathbf{supp}(\sigma)]$,

and thus $(\theta \circ \sigma) \in \mathbf{U}(\mathcal{E})$, iff $\theta \in \mathbf{U}(\sigma(\mathcal{E}))$. □

P.2 P.3 ▷ **Theorem 15.2.4** *If $\mathcal{E} \vdash_{\mathcal{SIM}} \mathcal{F}$, then $\mathbf{U}(\mathcal{E}) \leq_{\beta\eta} \mathbf{U}(\mathcal{F})[\mathcal{E}]$.* *(correct, complete)*

Proof: By an induction over the length of the derivation

P.1 We the $\mathcal{SIM}$ rules individually for the base case

P.1.1 $\mathcal{SIM}{:}\alpha$: by α-conversion □

P.1.2 $\mathcal{SIM}{:}\eta$: By η-conversion in the presence of $\mathcal{SIM}{:}\alpha$ □

P.1.3 $\mathcal{SIM}{:}\mathrm{dec}$: The head $h \in (\Sigma \cup \Sigma^{Sk})$ cannot be instantiated. □

P.1.4 $\mathcal{SIM}{:}\mathrm{elim}$: By Lemma 15.2.3. □

P.2 The step case goes directly by inductive hypothesis and transitivity of derivation. □

Now that we have simplifiation out of the way, we have to deal with unification pairs of the form $h\overline{\mathbf{U}^n} =^? k\overline{\mathbf{V}^m}$. Note that the case where both h and k are contstants is unsolvable, so we can assume that one of them is a variable. The unification problem $F_{\alpha\to\alpha}a =^? a$ is a particularly simple example; it has solutions $[\lambda X_\alpha . a/F]$ and $[\lambda X_\alpha . X/F]$. In the first, the solution comes by instantiating F with a λ-term of type $\alpha \to \alpha$ with head a, and in the second with a 1-projection term of type $\alpha \to \alpha$, which projects the head of the argument into the right position. In both cases, the solution came from a term with a given type and an appropriate head. We will look at the problem of finding such terms in more detail now.

General Bindings

- ▷ Problem: Find all formulae of given type α and head h.
- ▷ sufficient: long $\beta\eta$ head normal form, most general
- ▷ General Bindings: $\mathbf{G}^h_\alpha(\Sigma) := (\lambda \overline{X^k_\alpha} . h(H^1\overline{X}) \ldots (H^n\overline{X}))$
  - ▷ where $\alpha = \overline{\alpha_k} \to \beta$, $h : \overline{\gamma_n} \to \beta$ and $\beta \in \mathcal{BT}$
  - ▷ and $H^i : \overline{\alpha_k} \to \gamma_i$ new variables.
- ▷ **Observation 15.2.5** *General bindings are unique up to choice of names for H^i.*
- ▷ **Definition 15.2.6** If the head h is j^{th} bound variable in $\mathbf{G}^h_\alpha(\Sigma)$, call $\mathbf{G}^h_\alpha(\Sigma)$ j-projection binding (and write $\mathbf{G}^j_\alpha(\Sigma)$) else imitation binding
- ▷ clearly $\mathbf{G}^h_\alpha(\Sigma) \in \mathit{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ and $\mathrm{head}(\mathbf{G}^h_\alpha(\Sigma)) = h$

For the construction of general bindings, note that their construction is completely driven by the intended type α and the (type of) the head h. Let us consider some examples.

Example 15.2.7 The following general bindings may be helpful: $\mathbf{G}^{a_\iota}_{\iota\to\iota}(\Sigma) = \lambda X_\iota . a$, $\mathbf{G}^{a_\iota}_{\iota\to\iota\to\iota}(\Sigma) = \lambda X_\iota Y_\iota . a$, and $\mathbf{G}^{a_{\iota\to\iota}}_{\iota\to\iota\to\iota}(\Sigma) = \lambda X_\iota Y_\iota . a(HXY)$, where H is of type $\iota \to \iota \to \iota$

We will now show that the general bindings defined in Definition 15.2.6 are indeed the most general λ-terms given their type and head.

Approximation Theorem

▷ **Theorem 15.2.8** *If* $\mathbf{A} \in wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ *with* $\mathrm{head}(\mathbf{A}) = h$, *then there is a general binding* $\mathbf{G} = \mathbf{G}^h_\alpha(\Sigma)$ *and a substitution* ρ *with* $\rho(\mathbf{G}) =_{\alpha\beta\eta} \mathbf{A}$ *and* $\mathrm{dp}(\rho){<}\mathrm{dp}(\mathbf{A})$.

▷ Proof: We analyze the term structure of $\mathbf{A}$

P.1 If $\alpha = \overline{\alpha_k} \to \beta$ and $h : \overline{\gamma_n} \to \beta$ where $\beta \in \mathcal{B}\mathcal{T}$, then the long head normal form of $\mathbf{A}$ must be $\lambda \overline{X^k_\alpha}.h\overline{\mathbf{U}^n}$.

P.2 $\mathbf{G} = \mathbf{G}^h_\alpha(\Sigma) = \lambda \overline{X^k_\alpha}.h(H^1\overline{X}) \ldots (H^n\overline{X})$ for some variables $H^i : \overline{\alpha_k} \to \gamma_i$.

P.3 Choose $\rho := [\lambda \overline{X^k_\alpha}.\mathbf{U}^1/H^1], \ldots, [\lambda \overline{X^k_\alpha}.\mathbf{U}^n/H^n]$.

P.4 Then we have
$$\begin{aligned} \rho(\mathbf{G}) &= \lambda \overline{X^k_\alpha}.h(\lambda \overline{X^k_\alpha}.\mathbf{U}^1\overline{X}) \ldots (\lambda \overline{X^k_\alpha}.\mathbf{U}^n\overline{X}) \\ &=_{\beta\eta} \lambda \overline{X^k_\alpha}.h\overline{\mathbf{U}^n} \\ &=_{\beta\eta} \mathbf{A} \end{aligned}$$

P.5 The depth condition can be read off as $\mathrm{dp}(\lambda \overline{X^k_\alpha}.\mathbf{U}^1) \leq \mathrm{dp}(\mathbf{A}) - 1$. □

With this result we can state the higher-order unification transformations.

Higher-Order Unification ($\mathcal{HOU}$)

▷ Recap: After simplification, we have to deal with pairs where one (flex/rigid) or both heads (flex/flex) are variables

▷ **Definition 15.2.9** Let $\mathbf{G} = \mathbf{G}^h_\alpha(\Sigma)$ (imitation) or $\mathbf{G} \in \{\mathbf{G}^j_\alpha(\Sigma) \,|\, 1 \leq j \leq n\}$, then $\mathcal{HOU}$ consists of the transformations (always reduce to $\mathcal{SIM}$ normal form)

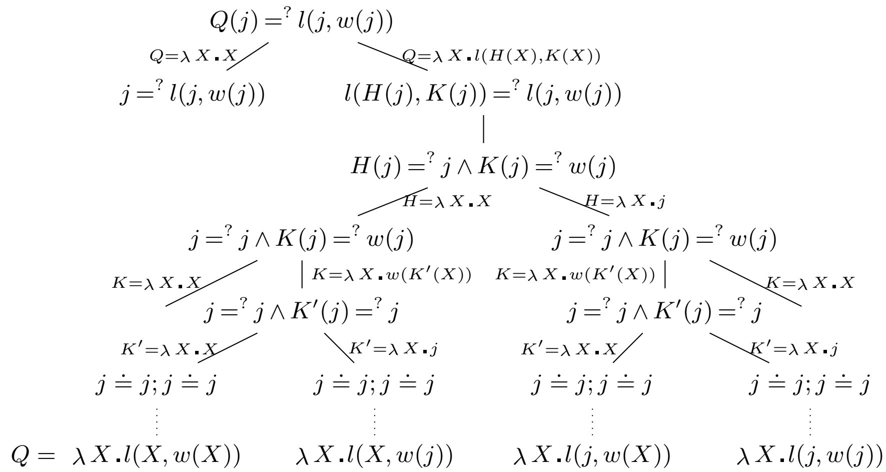
- Rule for flex/rigid pairs:
$$\frac{F_\alpha\overline{\mathbf{U}} =^? h\overline{\mathbf{V}} \wedge \mathcal{E}}{F =^? \mathbf{G} \wedge F\overline{\mathbf{U}} =^? h\overline{\mathbf{V}} \wedge \mathcal{E}} \mathcal{HOU}{:}\mathrm{fr}$$

- Rules for flex/flex pairs:
$$\frac{F_\alpha\overline{\mathbf{U}} =^? H\overline{\mathbf{V}} \wedge \mathcal{E}}{F =^? \mathbf{G} \wedge F\overline{\mathbf{U}} =^? H\overline{\mathbf{V}} \wedge \mathcal{E}} \mathcal{HOU}{:}\mathrm{ff}$$

Let us now fortify our intuition with a simple example.

$\mathcal{HOU}$ Example

Example 15.2.10 Let $Q, w : \iota \to \iota$, $l : \iota \to \iota \to \iota$, and $j : \iota$, then we have the following derivation tree in $\mathcal{HOU}$.



The first thing that meets the eye is that higher-order unification is branching. Indeed, for flex/-rigid pairs, we have to systematically explore the possibilities of binding the head variable the imitation binding and all projection bindings. On the initial node, we have two bindings, the projection binding leads to an unsolvable unification problem, whereas the imitation binding leads to a unification problem that can be decomposed into two flex/rigid pairs. For the first one of them, we have a projection and an imitation binding, which we systematically explore recursively. Eventually, we arrive at four solutions of the initial problem.

The following encoding of natural number arithmetics into $\Lambda^{\to}$ is useful for testing our unification algorithm

A Test Generator for Higher-Order Unification

▷ **Definition 15.2.11 (Church Numerals)** We define closed λ-terms of type $\nu := (\alpha \to \alpha) \to \alpha \to \alpha$

- ▷ Numbers: Church numerals: (n-fold iteration of arg1 starting from arg2)

$$n := (\lambda S_{\alpha\to\alpha}.\lambda O_\alpha.\underbrace{S(S\ldots S}_{n}(O)\ldots))$$

- ▷ Addition (N-fold iteration of S from N)

$$+ := (\lambda N_\nu M_\nu.\lambda S_{\alpha\to\alpha}.\lambda O_\alpha.NS(MSO))$$

- ▷ Multiplication: (N-fold iteration of MS (=$+m$) from O)

$$\cdot := (\lambda N_\nu M_\nu.\lambda S_{\alpha\to\alpha}.\lambda O_\alpha.N(MS)O)$$

▷ **Observation 15.2.12** *Subtraction and (integer) division on Church numberals can be automted via higher-order unification.*

▷ **Example 15.2.13** $5 - 2$ by solving the unification problem $2 + x_\nu =^? 5$

Equation solving for Church numerals yields a very nice generator for test cases for higher-order unification, as we know which solutions to expect.

15.3 Properties of Higher-Order Unification

We will now establish the properties of the higher-order unification problem and the algorithms we have introduced above. We first establish the unidecidability, since it will influence how we go about the rest of the properties.

We establish that higher-order unification is undecidable. The proof idea is a typical for undecidable proofs: we reduce the higher-order unification problem to one that is known to be undecidable: here, the solution of Diophantine equations $\mathbb{N}$.

▷ Undecidability of Higher-Order Unification

- ▷ **Theorem 15.3.1** *Second-order unification is undecidable* *(Goldfarb '82 [Gol81])*
- ▷ Proof Sketch: Reduction to Hilbert's tenth problem (solving Diophantine equations) (known to be undecidable) □
- ▷ **Definition 15.3.2** We call an equation a Diophantine equation, if it is of the form
  - ▷ $x_i\ x_j = x_k$
  - ▷ $x_i + x_j = x_k$
  - ▷ $x_i = c_j$ where $c_j \in \mathbb{N}$

  where the variables x_i range over $\mathbb{N}$.
- ▷ These can be solved by higher-order unification on Church numerals. (cf. Observation 15.2.12)

  .
- ▷ **Theorem 15.3.3** *The general solution for sets of Diophantine equations is undecidable.* *(Matijasevič 1970 [Mat70])*

The argument undecidability proofs is always the same: If higher-order unification were decidable, then via the encoding we could use it to solve Diophantine equations, which we know we cannot by Matijasevič's Theorem.

The next step will be to analyze our transformations for higher-order unification for correctness and completeness, just like we did for first-order unification.

$\mathcal{HOU}$ is Correct

- ▷ **Lemma 15.3.4** *If $\mathcal{E} \vdash_{\mathcal{HOU}\mathrm{fr}} \mathcal{E}'$ or $\mathcal{E} \vdash_{\mathcal{HOU}\mathrm{ff}} \mathcal{E}'$, then $\mathbf{U}(\mathcal{E}') \subseteq \mathbf{U}(\mathcal{E})$.*
- ▷ Proof Sketch: $\mathcal{HOU}$:fr and $\mathcal{HOU}$:ff only add new pair. □
- ▷ **Corollary 15.3.5** *$\mathcal{HOU}$ is correct: If $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$, then $\mathbf{U}(\mathcal{E}') \subseteq \mathbf{U}(\mathcal{E})$.*

Given that higher-order unification is not unitary and undecidable, we cannot just employ the notion of completeness that helped us in the analysis of first-order unification. So the first thing is to establish the condition we want to establish to see that $\mathcal{HOU}$ gives a higher-order unification algorithm.

Completeness of $\mathcal{HOU}$

- We cannot expect completeness in the same sense as for first-order unification: "If $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$, then $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$" (see Lemma 9.3.11) as the rules fix a binding and thus partially commit to a unifier (which excludes others).
- We cannot expect termination either, since HOU is undecidable.
- For a semi-decision procedure we only need termination on unifiable problems.
- **Theorem 15.3.6 ($\mathcal{HOU}$ derives Complete Set of Unifiers)** *If $\theta \in \mathbf{U}(\mathcal{E})$, then there is a $\mathcal{HOU}$-derivation $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{F}$, such that $\mathcal{F}$ is in solved form, $\sigma_{\mathcal{F}} \in \mathbf{U}(\mathcal{E})$, and $\sigma_{\mathcal{F}}$ is more general than θ.*
- Proof Sketch: Given a unifier θ of $\mathcal{E}$, we guide the derivation with a measure μ_θ towards $\mathcal{F}$. □

So we will embark on the details of the completeness proof. The first step is to define a measure that will guide the $\mathcal{HOU}$ transformation out of a unification problem $\mathcal{E}$ given a unifier θ of cE.

Completeness of $\mathcal{HOU}$ (Measure)

- **Definition 15.3.7** We call $\mu(\mathcal{E},\theta) := \langle\mu_1(\mathcal{E},\theta), \mu_2(\theta)\rangle$ the unification measure for $\mathcal{E}$ and θ, if
  - $\mu_1(\mathcal{E},\theta)$ is the multiset of term depths of $\theta(X)$ for the unsolved $X \in \mathbf{supp}(\theta)$.
  - $\mu_2(\mathcal{E})$ the multiset of term depths in $\mathcal{E}$.
  - $\prec$ is the strict lexicographic order on pairs: $(\langle a,b\rangle \prec \langle c,d\rangle$, if $a < c$ or $a = c$ and $b < d)$
  - Component orderings are multiset orderings: $(M \cup \{m\} < M \cup N$ iff $n < m$ for all $n \in N)$
- **Lemma 15.3.8** $\prec$ *is well-founded.* *(by construction)*

This measure will now guide the $\mathcal{HOU}$ transformation in the sense that in any step it chooses whether to use $\mathcal{HOU}$:fr or $\mathcal{HOU}$:ff, and which general binding (by looking at what θ would do). We formulate the details in Theorem 15.3.9 and look at their consequences before we proove it.

Completeness of $\mathcal{HOU}$ (μ-Prescription)

▷ **Theorem 15.3.9** *If $\mathcal{E}$ is unsolved and $\theta \in \mathbf{U}(\mathcal{E})$, then there is a unification problem $\mathcal{E}'$ with $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$ and a substitution $\theta' \in \mathbf{U}(\mathcal{E}')$, such that*

- $\theta =_{\beta\eta} \theta'[\mathcal{E}]$
- $\mu(\mathcal{E}', \theta') \prec \mu(\mathcal{E}, \theta)$.

we call such a $\mathcal{HOU}$-step a μ-prescribed

▷ **Corollary 15.3.10** *If $\mathcal{E}$ is unifiable without μ-prescribed $\mathcal{HOU}$-steps, then $\mathcal{E}$ is solved.*

▷ In other words: μ guides the $\mathcal{HOU}$-transformations to a solved form

We now come to the proof of Theorem 15.3.9, which is a relatively simple consequence of Theorem 15.2.8.

Proof of Theorem 15.3.9

▷ Proof:

P.1 Let $\mathbf{A} =^? \mathbf{B}$ be an unsolved pair of the form $\mathbf{F}\overline{\mathbf{U}} =^? \mathbf{G}\overline{\mathbf{V}}$ in $\mathcal{F}$.

P.2 $\mathcal{E}$ is a $\mathcal{SIM}$ normal form, so $\mathbf{F}$ and $\mathbf{G}$ must be constants or variables,

P.3 but not the same constant, since otherwise $\mathcal{SIM}{:}\mathrm{dec}$ would be applicable.

P.4 We can also exclude $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$, as $\mathcal{SIM}{:}\mathrm{triv}$ would be be appliccable.

P.5 If $\mathbf{F} = \mathbf{G}$ is a variable not in $\mathbf{supp}(\theta)$, then $\mathcal{SIM}{:}\mathrm{dec}$ appliccable. By correctness we have $\theta \in \mathbf{U}(\mathcal{E}')$ and $\mu(\mathcal{E}', \theta) \prec \mu(\mathcal{E}, \theta)$, as $\mu_1(\mathcal{E}', \theta) \preceq \mu_1(\mathcal{E}, \theta)$ and $\mu_2(\mathcal{E}') \prec \mu_2(\mathcal{E})$.

P.6 Otherwise we either have $\mathbf{F} \neq \mathbf{G}$ or $\mathbf{F} = \mathbf{G} \in \mathbf{supp}(\theta)$.

P.7 In both cases $\mathbf{F}$ or $\mathbf{G}$ is an unsolved variable $F \in \mathbf{supp}(\theta)$ of type α, since $\mathcal{E}$ is unsolved.

P.8 Without loss of generality we choose $F = \mathbf{F}$.

P.9 By Theorem 15.2.8 there is a general binding $\mathbf{G} = \mathbf{G}_\alpha^f(\Sigma)$ and a substitution ρ with $\rho(\mathbf{G}) =_{\alpha\beta\eta} \theta(F)$. So,

- if $\mathrm{head}(\mathbf{G}) \notin \mathbf{supp}(\theta)$, then $\mathcal{HOU}{:}\mathrm{fr}$ is appliccable,
- if $\mathrm{head}(\mathbf{G}) \in \mathbf{supp}(\theta)$, then $\mathcal{HOU}{:}\mathrm{ff}$ is appliccable.

P.10 Choose $\theta' := \theta \cup \rho$. Then $\theta =_{\beta\eta} \theta'[\mathcal{E}]$ and $\theta' \in \mathbf{U}(\mathcal{E}')$ by correctness.

P.11 $\mathcal{HOU}{:}\mathrm{ff}$ and $\mathcal{HOU}{:}\mathrm{fr}$ solve $F \in \mathbf{supp}(\theta)$ and replace F by $\mathbf{supp}(\rho)$ in the set of unsolved variable of $\mathcal{E}$.

P.12 so $\mu_1(\mathcal{E}', \theta') \prec \mu_1(\mathcal{E}, \theta)$ and thus $\mu(\mathcal{E}', \theta') \prec \mu(\mathcal{E}, \theta)$. □

We now convince ourselves that if $\mathcal{HOU}$ terminates with a unification problem, then it is either solved – in which case we can read off the solution – or unsolvable.

Terminal $\mathcal{HOU}$-problems are Solved or Unsolvable

▷ **Theorem 15.3.11** *If $\mathcal{E}$ is a unsolved UP and $\theta \in \mathbf{U}(\mathcal{E})$, then there is a $\mathcal{HOU}$-derivation $\mathcal{E} \vdash_{\mathcal{HOU}} \sigma_\sigma$, with $\sigma \leq_{\beta\eta} \theta[\mathcal{E}]$.*

▷ Proof: Let $\mathcal{D}\colon \mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{F}$ a maximal μ-prescribed $\mathcal{HOU}$-derivation from $\mathcal{E}$.

P.1 This must be finite, since $\prec$ is well-founded (ind. over length n of $\mathcal{D}$)

P.2 If $n = 0$, then $\mathcal{E}$ is solved and $\sigma_\mathcal{E}$ most general unifier

P.3 thus $\sigma_\mathcal{E} \leq_{\beta\eta} \theta[\mathcal{E}]$

P.4 If $n > 0$, then there is a μ-prescribed step $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$ and a substitution θ' as in Theorem 15.3.9.

P.5 by IH there is a $\mathcal{HOU}$-derivation $\mathcal{E}' \vdash_{\mathcal{HOU}} \mathcal{F}$ with $\sigma_\mathcal{F} \leq_{\beta\eta} \theta'[\mathcal{E}']$.

P.6 by correctness $\sigma_\mathcal{F} \in \mathbf{U}(\mathcal{E}') \subseteq \mathbf{U}(\mathcal{E})$.

P.7 rules of $\mathcal{HOU}$ only expand free variables, so $\sigma_\mathcal{F} \leq_{\beta\eta} \theta'[\mathcal{E}']$

P.8 Thus $\sigma_\mathcal{F} \leq_{\beta\eta} \theta'[\mathcal{E}]$,

P.9 This completes the proof, since $\theta' =_{\beta\eta} \theta[\mathcal{E}]$ by Theorem 15.3.9. □

We now recap the properties of higher-order unification (HOU) to gain an overview.

Properties of HO-Unification

▷ HOU is undecidable, HOU need not have most general unifiers

▷ The $\mathcal{HOU}$ transformation induce an algorithm that enumerates a complete set of higher-order unifiers.

▷ $\mathcal{HOU}$:ff gives enormous degree of indeterminism

▷ HOU is intractable in practice consider restricted fragments where it is!

▷ HO Matching (decidable up to order four), HO Patterns (unitary, linear), ...

15.4 Pre-Unification

We will now come to a variant of higher-order unification that is used in higher-order theorem proving, where we are only interested in the exgistence of a unifier – e.g. in mating-style tableaux. In these cases, we can do better than full higher-order unification.

Pre-Unification

▷ $\mathcal{HOU}$:ff has a giant branching factor in the search space for unifiers. (makes HOU impracticable)

▷ In most situations, we are more interested in solvability of unification problems than in the unifiers themselves.

▷ More liberal treatment of flex/flex pairs.

▷ **Observation 15.4.1** *flex/flex-pairs $F\overline{\mathbf{U}^n} =^? G\overline{\mathbf{V}^m}$ are always (trivially) solvable by $[\lambda\overline{X^n}.H/F], [\lambda\overline{Y^m}.H/G]$, where H is a new variable*

▷ Idea: consider flex/flex-pairs as pre-solved.

▷ **Definition 15.4.2 (Pre-Unification)** For given terms $\mathbf{A}, \mathbf{B} \in wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ find a substitution σ, such that $\sigma(\mathbf{A}) =^p_{\beta\eta} \sigma(\mathbf{B})$, where $=^p_{\beta\eta}$ is the equality theory that is induced by $\beta\eta$ and $F\overline{\mathbf{U}} = G\overline{\mathbf{V}}$.

▷ **Lemma 15.4.3** *A higher-order unification problem is unifiable, iff it is pre-unifiable.*

The higher-order pre-unification algorithm can be obtained from $\mathcal{HOU}$ by simply omitting the offending $\mathcal{HOU}$:ff rule.

Pre-Unification Algorithm $\mathcal{HOPU}$

▷ **Definition 15.4.4** A unification problem is a pre-solved form, iff all of its pairs are solved or flex/flex

▷ **Lemma 15.4.5** *If $\mathcal{E}$ is solved and $\mathcal{P}$ flex/flex, then σ_σ is a most general unifier of a pre-solved form $\mathcal{E} \wedge \mathcal{P}$.*

▷ Restrict all $\mathcal{HOU}$ rule so that they cannot be applied to pre-solved pairs.

▷ In particular, remove $\mathcal{HOU}$:ff!

▷ $\mathcal{HOPU}$ only consists of $\mathcal{SIM}$ and $\mathcal{HOU}$:fr.

▷ **Theorem 15.4.6** *$\mathcal{HOPU}$ is a correct and complete pre-unification algorithm*

▷ Proof Sketch: with exactly the same methods as higher-order unification □

▷ **Theorem 15.4.7** *Higher-order pre-unification is infinitary, i.e. a unification problem can have infinitely many unifiers.* (Huet 76' [Hue76])

▷ **Example 15.4.8** $Y(\lambda X_\iota.X)a =^? a$, where a is a constant of type ι and Y a variable of type $(\iota \to \iota) \to \iota \to \iota$ has the most general unifiers $\lambda sz.s^n z$ and $\lambda sz.s^n a$, which are mutually incomparable and thus most general.

15.5 Applications of Higher-Order Unification

Application of HOL in NL Semantics: Ellipsis

▷ **Example 15.5.1** *John loves his wife. George does too*

- ▷ $\mathrm{love}(\mathrm{john}, \mathrm{wife\_of}(\mathrm{john})) \wedge Q(\mathrm{george})$
- ▷ "*George* has property some Q, which we still have to determine"

Idea: If *John* has property Q, then it is that he *loves his wife*.

⧐ Equation: $Q(\mathrm{john}) =_{\alpha\beta\eta} \mathrm{love}(\mathrm{john}, \mathrm{wife\_of}(\mathrm{john}))$

▷ Solutions (computed by HOU):

- ▹ $Q = \lambda z.\mathrm{love}(z, \mathrm{wife\_of}(z))$ and $Q = \lambda z.\mathrm{love}(z, \mathrm{wife\_of}(\mathrm{john}))$
- * $Q = \lambda z.\mathrm{love}(\mathrm{john}, \mathrm{wife\_of}(z))$ and $Q = \lambda z.\mathrm{love}(\mathrm{john}, \mathrm{wife\_of}(\mathrm{john}))$

▷ Readings: *George loves his own wife*. and *George loves Johns wife*.

 145 JACOBS UNIVERSITY

Chapter 16

Simple Type Theory

In this Chapter we will revisit the higher-order predicate logic introduced in Chapter 9 with the base given by the simply typed λ-calculus. It turns out that we can define a higher-order logic by just introducing a type of propositions in the λ-calculus and extending the signatures by logical constants (connectives and quantifiers).

Higher-Order Logic Revisited

- ▷ Idea: introduce special base type o for truth values
- ▷ **Definition 16.0.1** We call a Σ-algebra $\langle\mathcal{D},\mathcal{I}\rangle$ a Henkin model, iff $\mathcal{D}_o = \{\mathsf{T},\mathsf{F}\}$.
- ▷ $\mathbf{A}_o$ valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$
- ▷ connectives in Σ: $\neg \in \Sigma_{o\to o}$ and $\{\vee,\wedge,\Rightarrow,\Leftrightarrow,\ldots\} \subseteq \Sigma_{o\to o\to o}$ (with the intuitive $\mathcal{I}$-values)
- ▷ quantifiers: $\Pi^\alpha \in \Sigma_{(\alpha\to o)\to o}$ with $\mathcal{I}(\Pi^\alpha)(p) = \mathsf{T}$, iff $p(a) = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$.
- ▷ quantified formula e: $\forall X_\alpha.\mathbf{A}$ stands for $\Pi^\alpha(\lambda X_\alpha.\mathbf{A})$
- ▷ $\mathcal{I}_\varphi(\forall X_\alpha.\mathbf{A}) = \mathcal{I}(\Pi^\alpha)(\mathcal{I}_\varphi(\lambda X_\alpha.\mathbf{A})) = \mathsf{T}$, iff $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$
- ▷ looks like $\mathrm{PL}\Omega$ (Call any such system $\mathrm{HOL}^{\to}$)

There is a more elegant way to treat quantifiers in $\mathrm{HOL}^{\to}$. 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 $\mathrm{HOL}^{\to}$ 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 $\mathrm{HOL}^{\to}$, we already have variable binder: λ, use that to treat quantification.
- ▷ **Definition 16.0.2** We assume logical constants Π^α and Σ^α of type $(\alpha \to o) \to$

o.

Regain quantifiers as abbreviations:

$$(\forall X_\alpha.\mathbf{A}) := \overset{\alpha}{\Pi}(\lambda X_\alpha.\mathbf{A}) \qquad (\exists X_\alpha.\mathbf{A}) := \overset{\alpha}{\Sigma}(\lambda X_\alpha.\mathbf{A})$$

▷ **Definition 16.0.3** 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:

$$\mathcal{I}_\varphi(\forall X_\iota.\mathbf{A}) = \mathcal{I}_\varphi(\overset{\iota}{\Pi}(\lambda X_\iota.\mathbf{A})) = \mathcal{I}(\overset{\iota}{\Pi})(\mathcal{I}_\varphi(\lambda X_\iota.\mathbf{A})) = \mathsf{T}$$

iff $\mathcal{I}_\varphi(\lambda X_\iota.\mathbf{A})(a) = \mathcal{I}_{[a/X],\varphi}(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\alpha$

But there is another alternative of introducing higher-order logic due to Peter Andrews. Instead of using connectives and quantifiers as primitives and defining equality from them via the Leibniz indiscernability principle, we use equality as a primitive logical constant and define everything else from it.

Alternative: $\mathrm{HOL}^=$

▷ only one logical constant $q^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ with $\mathcal{I}(q^\alpha)(a,b) = \mathsf{T}$, iff $a = b$.

▷ Definitions (D) and Notations (N)

N	$\mathbf{A}_\alpha = \mathbf{B}_\alpha$	for	$q^\alpha \mathbf{A}_\alpha \mathbf{B}_\alpha$
D	T	for	$q^o = q^o$
D	F	for	$(\lambda X_o.T) = (\lambda X_o.X_o)$
D	Π^α	for	$q^{(\alpha\to o)}(\lambda X_\alpha.T)$
N	$\forall X_\alpha.\mathbf{A}$	for	$\Pi^\alpha(\lambda X_\alpha.\mathbf{A})$
D	$\wedge$	for	$\lambda X_o.\lambda Y_o.(\lambda G_{o\to o\to o}.G T T) = (\lambda G_{o\to o\to o}.G X Y)$
N	$\mathbf{A} \wedge \mathbf{B}$	for	$\wedge \mathbf{A}_o \mathbf{B}_o$
D	$\Rightarrow$	for	$\lambda X_o.\lambda Y_o.X = X \wedge Y$
N	$\mathbf{A} \Rightarrow \mathbf{B}$	for	$\Rightarrow \mathbf{A}_o \mathbf{B}_o$
D	$\neg$	for	$q^o F$
D	$\vee$	for	$\lambda X_o.\lambda Y_o.\neg(\neg X \wedge \neg Y)$
N	$\mathbf{A} \vee \mathbf{B}$	for	$\vee \mathbf{A}_o \mathbf{B}_o$
D	$\exists X_\alpha.\mathbf{A}_o$	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, Theorem 10.0.6 does not hold (see [And72] for details). Indeed the proof of Theorem 10.0.6 needs the existence of "singleton sets", 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 will conclude this section with a discussion on two additional "logical constants" (constants with a fixed meaning) that are needed to make any progress in mathematics. Just like above, adding them to the logic guarantees the existence of certain functions in Henkin models. The most important one is the description operator that allows us to make definite descriptions like "the largest prime number" or "the solution to the differential equation $f' = f$.

More Axioms for $\mathrm{HOL}^{\rightarrow}$

- **Definition 16.0.4** unary conditional $\mathbf{w} \in \Sigma_{o\rightarrow\alpha\rightarrow\alpha}$
  $\mathbf{w}\mathbf{A}_o\mathbf{B}_\alpha$ means: "If $\mathbf{A}$, then $\mathbf{B}$"
- **Definition 16.0.5** binary conditional $\mathbf{if} \in \Sigma_{o\rightarrow\alpha\rightarrow\alpha\rightarrow\alpha}$
  $\mathbf{if}\mathbf{A}_o\mathbf{B}_\alpha\mathbf{C}_\alpha$ means: "if $\mathbf{A}$, then $\mathbf{B}$ else $\mathbf{C}$".
- **Definition 16.0.6** description operator $\iota \in \Sigma_{(\alpha\rightarrow o)\rightarrow\alpha}$
  if $\mathbf{P}$ is a singleton set, then $\iota\mathbf{P}_{\alpha\rightarrow o}$ is the element in $\mathbf{P}$,
- **Definition 16.0.7** choice operator $\gamma \in \Sigma_{(\alpha\rightarrow o)\rightarrow\alpha}$
  if $\mathbf{P}$ is non-empty, then $\gamma\mathbf{P}_{\alpha\rightarrow o}$ is an arbitrary element from $\mathbf{P}$
- **Definition 16.0.8 (Axioms for these Operators)**
  - unary conditional: $\forall\varphi_o.\forall X_\alpha.\varphi\Rightarrow\mathbf{w}\varphi X = X$
  - conditional: $\forall\varphi_o.\forall X_\alpha, Y_\alpha, Z_\alpha.(\varphi\Rightarrow\mathbf{if}\varphi XY = X)\wedge(\neg\varphi\Rightarrow\mathbf{if}\varphi ZX = X)$
  - description $\forall P_{\alpha\rightarrow o}.(\exists^1 X_\alpha.PX)\Rightarrow(\forall Y_\alpha.PY\Rightarrow\iota P = Y)$
  - choice $\forall P_{\alpha\rightarrow o}.(\exists X_\alpha.PX)\Rightarrow(\forall Y_\alpha.PY\Rightarrow\gamma P = Y)$

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 singleton sets)
- 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 $\neq o$
  - Define $\iota^o := =(\lambda X_o.X)$ or $\iota^o := (\lambda G_{o\rightarrow o}.GT)$ or $\iota^o := =(=T)$
  - $\iota^{\alpha\rightarrow\beta} := (\lambda H_{(\alpha\rightarrow\beta)\rightarrow o}X_\alpha.\iota^\beta(\lambda Z_\beta.(\exists F_{\alpha\rightarrow\beta}.(HF)\wedge(FX) = Z)))$

Chapter 17

Higher-Order Tableaux

In this Chapter we will extend the ideas from first-order tableaux to higher-order logic.

The rules fo standard tableaux are just like the ones for first-order logic, only that we can take advantage oft higher-order abstract syntax for the quantifiers

Tableau-Rules ($\mathcal{T}_\omega^s$)

▷ **Definition 17.0.1** The rules of $\mathcal{T}_\omega^s$ consist of the propositional tableau rules of $\mathcal{T}_0$ together with

$$\frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{t}}}{\mathbf{A}\mathbf{C}^{\mathsf{t}}}\,\mathcal{T}_\omega^s{:}\forall \qquad \frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{f}} \quad c \in (\Sigma_0^{sk} \backslash \mathcal{H})}{\mathbf{A}c^{\mathsf{f}}}\,\mathcal{T}_\omega^s{:}\exists$$

Higher-order, free-variable tableaux work exactly like first-order tableaux, except that the cut rule uses higher-order unification.

Higher-Order Free-Variable Tableaux ($\mathcal{T}_\omega$ first try)

▷ **Definition 17.0.2** The $\mathcal{T}_\omega$ calculus consists of the propositional tableau rules plus

$$\frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{t}}}{\mathbf{A}X_\alpha{}^{\mathsf{t}}}\,\mathcal{T}_\omega{:}\forall \qquad \frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{f}} \quad \mathrm{free}(\mathbf{A}) = \{Y_{\alpha_1}^1, \ldots, Y_{\alpha_n}^n\} \quad f \in \Sigma_{\alpha_n \to \alpha}^{Sk} \text{ new}}{\mathbf{A}(f\overline{Y^n})^{\mathsf{f}}}\,\mathcal{T}_\omega{:}\exists$$

▷ Problem: Unification in $\Lambda^{\to}$ is undecidable, so we need more

▷ Idea: explicit rule that residuates the unification problem

$$\frac{\begin{array}{c}\mathbf{A}^\alpha\\ \mathbf{B}^\beta\end{array}}{\mathbf{A} \neq^? \mathbf{B}}\ \mathcal{T}_\omega\text{:cut}$$

and adapt the $\mathcal{HOU}$ rules to tableaux (DNF instead of CNF)

Note that we cannot directly use the higher-order unification algorithm, since that is undecidable – this would not result in a fair proof search procedure. Therefore we reinterpret $\mathcal{HOPU}$ rules as tableau rules and mix them into the proof search procedure.

For the reinterpretation of $\mathcal{HOU}$ rules into tableau rules we change notation of the unification pairs, using $\mathbf{A} \neq^? \mathbf{B}$ instead of $\mathbf{A} =^? \mathbf{B}$, since in the tableau setting we want to refute that $\mathbf{A}$ and $\mathbf{B}$ cannot be made equal instead of finding conditions that make them equal (as we did for unification). Correspondingly, we we do not use a "conjunction" of equations, but a disjunction (using tableau branches) of "disequations". But up to this "double negation" the unification algorithm stays the same.

$\mathcal{T}_\omega$ (Pre-Unification)

▷ we can use $\mathcal{SIM}:\alpha$, $\mathcal{SIM}:\eta$, and $\mathcal{SIM}:\mathrm{triv}$ directly, for $\mathcal{SIM}:\mathrm{dec}$ and $\mathcal{SIM}:\mathrm{elim}$ we integrate into tableau setting more closely, obtaining

$$\frac{h\overline{\mathbf{U}^n} \neq^? h\overline{\mathbf{V}^n} \quad h \in (\Sigma \cup \Sigma^{Sk} \cup \mathcal{V}_\mathcal{T})}{\mathbf{U}^1 \neq^? \mathbf{V}^1 \;\Big|\; \ldots \;\Big|\; \mathbf{U}^n \neq^? \mathbf{V}^n}\ \mathcal{T}_\omega\mathrm{:dec}$$

$$\frac{F_\alpha\overline{\mathbf{U}} \neq^? h\overline{\mathbf{V}}}{F \neq^? \mathbf{G} \;\Big|\; F\overline{\mathbf{U}} \neq^? h\overline{\mathbf{V}}}\ \mathcal{T}_\omega\mathrm{:fr}$$

$$\frac{X \neq^? \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad \mathbf{A} \cap \Sigma^{Sk} = \emptyset}{\bot}\ \mathcal{T}_\omega\mathrm{:elim}$$

where $\mathbf{G} = \mathbf{G}^h_\alpha(\Sigma)$ (imitation) or $\mathbf{G} \in \{\mathbf{G}^j_\alpha(\Sigma) \,|\, 1{\le}j{\le}n\}$

▷ **Definition 17.0.3** We call a $\mathcal{T}_\omega$ tableau closed, if all branches end in a $\bot$ or a flex/flex pair.

Note that the elimination rule is particularly elegant in the tableau setting – it comes in the form of a closure rule: If we have a solved pair, then we can just make the branch unsatisfiable by applying its most general unifier to the whole tableau.

Note furthermore, that with the mixed propositional and pre-unification calculus in $\mathcal{T}_\omega$, the decision whether to do regular or matings-style tableaux boils down to a a decision of the strategy used to expand the $\mathcal{T}_\omega$ tableaux.

We will now fortify our intuition with an extended example: a $\mathcal{T}_\omega$ proof of (a version of) Cantor's theorem. The particular formulation we use below uses the whole universe of type ι for the set S and universe of type $\iota \to \iota$ for the power set.

$\mathcal{T}_\omega$ Example: Cantor's Theorem

- **Theorem 17.0.4** *There is no surjective function from the natural numbers into the sequences of natural numbers.*
- Formally: $\neg(\exists F_{\iota\to\iota\to\iota}.\forall G_{\iota\to\iota}.\exists J_\iota.FJ = G)$
- For the proof we use
  - $\forall X_\iota.\neg X = sX$ (the successor function has no fixed points)
  - an extensionality axiom

 154 JACOBS UNIVERSITY

We initialize the tableau with the three formulae discussed above, and then employ the $\mathcal{T}_\omega$ rules.

$\mathcal{T}_\omega$-Proof (Cantor's Theorem)

- First the propositional part (analyzing formula structure)

$$
\begin{array}{c}
\neg(\exists F_{\iota\to\iota\to\iota}.\forall G_{\iota\to\iota}.\exists J_\iota.FJ = G)^{\mathsf{f}}\\
\exists F_{\iota\to\iota\to\iota}.\forall G_{\iota\to\iota}.\exists J_\iota.FJ = G^{\mathsf{t}}\\
\forall G_{\iota\to\iota}.\exists J_\iota.f_{\iota\to\iota\to\iota}J = G^{\mathsf{t}}\\
\exists J_\iota.fJ = G^{\mathsf{t}}\\
f(jG) = G^{\mathsf{t}}\\
H = K \Rightarrow (\forall N_\iota.HN = KN)^{\mathsf{t}}\\
\begin{array}{c|c}
\begin{array}{c}
H = K^{\mathsf{f}}\\
H = K \neq^? f(jG) = G\\
\begin{array}{c|c}
H \neq^? f(jG) & K \neq^? G\\
\bot & \bot
\end{array}
\end{array}
&
\begin{array}{c}
HN = KN^{\mathsf{t}}\\
f(jG)N = GN^{\mathsf{t}}\\
X = sX^{\mathsf{f}}\\
X = sX \neq^? f(jG)N = GN
\end{array}
\end{array}
\end{array}
$$

- then we continue with unification tableau

$$X = sX \neq^? f(jG)N = GN$$

$$X \neq^? f(jG)N \qquad GN \neq^? s(f(jG)N)$$

$$\bot \qquad s(H^1N) \neq^? s(f(jG)N)$$

$$G \neq^? (\lambda Y_\iota . s(H^1Y)) \qquad H^1N \neq^? f(jG)N$$

$$\bot$$

$$H^1 \neq^? (\lambda Y_\iota . f(H^2Y)(H^3Y)) \qquad f(H^2N)(H^3N) \neq^? f(jG)N$$

$$\bot \qquad H^2N \neq^? jG \qquad H^3N \neq^? N$$

$$H^2 \neq^? (\lambda Y_\iota . Y) \qquad N \neq^? jG \qquad H^3 \neq^? (\lambda Y_\iota . Y) \qquad jG \neq^? jG$$

$$\bot \qquad \bot \qquad \bot \qquad \bot$$

▷ We found a closed tableau and completed the $\mathcal{T}_\omega$ proof.

In the higher-order unification tableau above we face the same problem we always face when we try to display the dynamics of free-variable tableaux: in the closure rules we have to instantiate the whole tableau. But this turns the tableau into a standard tableau. So we close the leftmost branch and apply the substitution to the branches to the right of the current branch only.

Note that at first sight $N \neq^? jG$ is not solved (and indeed unsolvable), since j is a Skolem constant. But we only need to forbid the Skolem constants that were introduced by the $\mathcal{SIM}{:}\alpha$ and $\mathcal{SIM}{:}\eta$ rules. So there is no problem here; since they were introduced by $\mathcal{T}_\omega{:}\exists$.

Even though we were successful in proving Cantor's theorem, $\mathcal{T}_\omega$ is not complete as we will see.

Problem for $\mathcal{T}_\omega$

▷ **Theorem 17.0.5** *There is a valid formula* $(\exists X_o . X)$

▷ This is clearly valid, (eg. $\mathbf{A} \vee \neg \mathbf{A}$)

▷ $\mathcal{T}_\omega$ attempt

$$\begin{array}{c} \neg (\forall X_o . \neg X)^{\mathsf{f}} \\ \forall X_o . \neg X^{\mathsf{t}} \\ \neg X^{\mathsf{t}} \\ X^{\mathsf{f}} \end{array}$$

▷ we are stuck!

▷ Observation: We have to instantiate X further, e.g. by $[\neg Q_o/X]$.

▷ then we can continue

$$\begin{array}{c} X^{\mathsf{f}} \\ \neg Q^{\mathsf{f}} \\ Q^{\mathsf{t}} \\ X \neq^? Q \end{array}$$

close with $[Q/X]$.

We see that unlike in first-order unification we cannot obtain all necessary instantiations by unification. Indeed in the presence of predicate variables – in our example above we can view X_o as a nullary predicate – we have to allow instantiations with (all) logical connectives and quantifiers. Fortunately, we can do this in a minimally committing fashion via general bindings, unfortunately, we have to systematically try out all possible ones – which is costly, since there are infinitely many quantifiers.

Primitive Substitutions

- ▷ Unification is not sufficient for $\mathcal{T}_\omega$
- ▷ We need a rule that instantiates head variables with terms that introduce logical constants.
- ▷ **Definition 17.0.6** We extend $\mathcal{T}_\omega$ with the rule $\mathcal{T}_\omega$:prim.

$$\frac{X_\alpha \overline{\mathbf{U}^n}^\alpha \quad \mathbf{G} \in \mathbf{G}_\alpha^k(\Sigma) \quad k \in (\{\wedge, \neg\} \cup \{\overset{\beta}{\Pi} \,|\, \beta \in \mathcal{T}\})}{X \neq^? \mathbf{G} \;\Big|\; X_\alpha \overline{\mathbf{U}^n}^\alpha} \mathcal{T}_\omega\text{:prim}$$

  We call $[\mathbf{G}/X]$ a primitive substitution.

- ▷ In our example $\neg Q = \mathbf{G}_o^\neg(\Sigma)$.

There is another source of incompleteness as another example shows: we can have propositions embedded in formulae. Note that this is different from the situation in first-order logic, but quite natural in mathematics, e.g. for conditional statements of the form "if φ_o then $\mathbf{A}_\alpha$ else $\mathbf{B}_\alpha$.", where φ is a proposition embedded in a term of type α.

Another Example

- ▷ $\mathbf{A} = \neg (c_{o \to o} b_o) \vee (c \neg \neg b)$ is valid
- ▷ $\mathcal{T}_\omega$ proof attempt

$$\begin{array}{c} \neg (cb) \vee (c \neg \neg b)^{\mathsf{f}} \\ \neg (cb)^{\mathsf{f}} \\ c \neg \neg b^{\mathsf{f}} \\ cb^{\mathsf{t}} \\ cb \neq^? c \neg \neg b \\ b \neq^? \neg \neg b \end{array}$$

  and we are stuck (again)

- ▷ Idea: theory unification with $X_o = \neg \neg X_o$
- ▷ But the problem is more general: If $\mathbf{A} \Leftrightarrow \mathbf{B}$ valid, then $\neg (c\mathbf{A}) \wedge (c\mathbf{B})$ must be $\mathcal{T}_\omega$-refutable.
- ▷ Solution: call to the theorem prover recursively.

▷ **Definition 17.0.7** We extend $\mathcal{T}_\omega$ with the rule $\mathcal{T}_\omega$:rec,

$$\frac{\mathbf{A}_o \neq^? \mathbf{B}_o}{\begin{array}{c|c}\mathbf{A}^\mathsf{t} & \mathbf{A}^\mathsf{f} \\ \mathbf{B}^\mathsf{f} & \mathbf{B}^\mathsf{t}\end{array}}\mathcal{T}_\omega\text{:rec}$$

▷ **Observation 17.0.8** *We can prove $\mathbf{A}_o$ by unifying it with T_o.*

The $\mathcal{T}_\omega$:rec rule puts the propositional and unification rules of $\mathcal{T}_\omega$ at an equal footing. $\mathcal{T}_\omega$ can be seen as a calculus for theorem proving or as an unification algorithm that takes the theory of equivalence into account.

Each aspect of $\mathcal{T}_\omega$ can recurse into the the other; this is necessary, since in $\text{HOL}^\to$ the propositional level – which has a fixed interpretation and therefore special $\mathcal{T}_\omega$ rules – and the term level – which is freely interpreted and must thus be handled by unification – can recurse arbitrarily.

To make matters worse, we also have a soundness problem that comes from Skolemization: we can prove a version of the Axiom of Choice that is known to be independent of $\text{HOL}^\to$, and thus should not be provable.

Skolemization is not sound

▷ Axiom of Choice:
$\exists \gamma_{\alpha\to o\to\alpha} . \forall P_{\alpha\to o} . (\exists X_\alpha . PX) \Rightarrow (\forall Y_\alpha . (PY) \Rightarrow \gamma P = Y)$

▷ Weaker Version: (call it $\mathbf{C}$)

$$\forall R_{\alpha\to\alpha\to o} . (\forall X_\alpha . \exists Y_\alpha . RXY) \Rightarrow (\exists F_{(\alpha\to o)\to\alpha} . \forall Z_\alpha . RZ(FZ))$$

▷ Neither $\mathbf{C}$ nor $\neg\mathbf{C}$ are valid in $\text{HOL}^\to$ (independent)

▷ but $\mathbf{C}$ is provable $\mathcal{T}_\omega$. (see next slide)

In this proof, the Skolem constant f introduced for the assumption $\forall X_\alpha . \exists Y_\alpha . RXY$ becomes available as an instance for the variable F in (used to require the existence of a choice operator).

Skolemization is not sound (Choice Proof)

$$\forall R_{\alpha\to\alpha\to o}.(\forall X_\alpha.\exists Y_\alpha.RXY)\Rightarrow(\exists F_{(\alpha\to o)\to\alpha}.\forall Z_\alpha.RZ(FZ))^{\mathsf{f}}$$
$$\forall X_\alpha.\exists Y_\alpha.rXY^{\mathsf{t}}$$
$$\exists F_{(\alpha\to o)\to\alpha}.\forall Z_\alpha.rZ(FZ)^{\mathsf{f}}$$
$$\exists Y_\alpha.rXY^{\mathsf{t}}$$
$$rX(fX)^{\mathsf{t}}$$
$$\forall F_{(\alpha\to o)\to\alpha}.\neg(\forall Z_\alpha.rZ(fZ))^{\mathsf{t}}$$
$$\forall Z_\alpha.rZ(FZ)^{\mathsf{f}}$$
$$r(gF)(F(gF))^{\mathsf{f}}$$
$$rX(fX)\neq^? r(gF)(F(gF))$$

$$\begin{array}{c|c}
\begin{array}{c} X\neq^? gF \\ \bot \end{array} &
\begin{array}{c}
f(gF)\neq^? F(gF) \\
\begin{array}{c|c}
\begin{array}{c} F\neq^? (\lambda Z_\alpha.f(H^1Z)) \\ \bot \end{array} &
\begin{array}{c}
f(g(\lambda Z_\alpha.f(H^1Z)))\neq^? f(H^1(g(\lambda Z_\alpha.f(H^1Z)))) \\
g(\lambda Z_\alpha.f(H^1Z))\neq^? H^1(g(\lambda Z_\alpha.f(H^1Z))) \\
\begin{array}{c|c}
\begin{array}{c} H^1\neq^? (\lambda W_\alpha.W) \\ \bot \end{array} &
\begin{array}{c} g(\lambda Z_\alpha.fZ)\neq^? g(\lambda Z_\alpha.fZ) \\ \bot \end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}$$

 160 JACOBS UNIVERSITY

In first-order logic, Skolemization is sound, since Skolem constants do not "lose their arguments", so they cannot be used to prove the axiom of choice.

The following part is still experimentals; not required for the course

Variable Conditions

- **Definition 17.0.9** Let Γ be an annotated variable context, Then a variable condition $\mathcal{R}$ is a relation on $\mathcal{R}\subseteq\mathbf{dom}(\Gamma)\times\mathbf{dom}(\Gamma^-)$.
- **Definition 17.0.10** We call a substitution σ with $\mathbf{supp}(\sigma)\subseteq\mathbf{dom}(\Gamma)\cup\mathbf{dom}(\Delta)$ a $\mathcal{R}$-substitution, iff $Y\notin\mathrm{free}(\sigma(X))$ for all $(x,y)\in\mathcal{R}$.
- Intuition: If $(X,Y^-)\in\mathcal{R}$, then no formula that contains Y^- freely may be substituted for X.
- We define a judgment $\Delta\vdash\overline{\mathcal{R}}(X,\mathbf{A})$ by
  - $\Delta,\Gamma\vdash_\sigma\mathbf{A}:\Gamma(X)$ und $X\notin\mathrm{free}(\mathbf{A})$,
  - $(\{X\}\times\mathrm{free}(\mathbf{A}))\cap\mathcal{R}=\emptyset$ (no variable $Y\in\mathrm{free}(\mathbf{A})$ is an $\mathcal{R}$-image of X)
- So σ is a $\mathcal{R}$-substitution, iff $\Delta\vdash_\Sigma\overline{\mathcal{R}}(X,\sigma(X))$ for all $X\in\mathbf{supp}(\sigma)$.
- Extension of variable conditions for instantiaton with $[\mathbf{A}/X]$:

$$\mathcal{R}(\mathbf{A}/X):=\{(Z,W)\in\mathcal{R}\,|\,Z\neq X\}\cup\{(Z,W)\,|\,Z\in\mathrm{free}(\mathbf{A}),\mathcal{R}(X,W)\}$$

 161 JACOBS UNIVERSITY

Higher-Order Tableaux (final)

- Higher-order tableaux are triples $\langle\Gamma:\mathcal{R}\rangle.\mathcal{T}$

▷ propositional tableaux as always (do not change Γ or $\mathcal{R}$)

▷ New quantifier rules

$$\frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{t}}}{\mathbf{A}X_\alpha{}^{\mathsf{t}}} \qquad \frac{\overset{\alpha}{\Pi}\mathbf{A}^{\mathsf{f}}}{\mathbf{A}Y^{-\mathsf{f}}}$$

where

- ▹ $X, Y^- \notin \mathbf{dom}(\Gamma)$
- ▹ $\Gamma' := \Gamma, [X : \alpha]$ and $\Gamma' := \Gamma, [Y^- : \alpha]$
- ▹ $\mathcal{R}' := \mathcal{R}$ and $\mathcal{R}' := \mathcal{R} \cup (\mathrm{free}(\mathbf{A}) \times \{Y^-\})$.

▷ substitution rule: If a path in $\langle\Gamma, [X : \alpha] : \mathcal{R}\rangle.\mathcal{T}$ ends in an equation $X =^? \mathbf{A}$ with $\Gamma \vdash_\Sigma \overline{\mathcal{R}}(X, \mathbf{A})$, then generate $\langle\Gamma : \mathcal{R}(\mathbf{A}/X)\rangle.[\mathbf{A}/X]\mathcal{T}$.

▷ Primitive Substitution: If $\langle\Gamma, [X : \alpha] : \mathcal{R}\rangle.\mathcal{T}$ contains a formula $(X\overline{\mathbf{U}^n})^\alpha$, and $\mathbf{A} \in \mathbf{G}^k_\alpha(\Sigma, \Gamma, \mathcal{C})$ with $k \in (\{\wedge, \neg\} \cup \{\Pi^\beta \mid \beta \in \mathcal{T}\})$, then generate $\langle\Gamma \cup \mathcal{C} : \mathcal{R}(\mathbf{A}/X)\rangle.[\mathbf{A}/X]\mathcal{T}$

▷ Closed Tableau: every branch ends in a trivial equation $\mathbf{A} =^? \mathbf{A}$ or a pre-solved equation $F\overline{\mathbf{U}} \neq^? G\overline{\mathbf{V}}$.

▷ tableau-substitution closes the respective branch

▷ Side conditions

$$\frac{(\lambda X_\alpha.\mathbf{A}) \neq^? (\lambda Y_\alpha.\mathbf{B}) \quad Z \notin \mathbf{dom}(\Gamma)}{[Z/X](\mathbf{A}) \neq^? Z(Y)(\mathbf{B})}$$

$$\frac{(\lambda X_\alpha.\mathbf{A}) \neq^? \mathbf{B} \quad Z \notin \mathbf{dom}(\Gamma)}{[Z/X](\mathbf{A}) \neq^? \mathbf{B}Z}$$

where $\Gamma' = \Gamma, [Z^0 : \alpha]$ and $\mathcal{R}' := \mathcal{R}(Z/X)$

$$\frac{h\overline{\mathbf{U}^n} \neq^? h\overline{\mathbf{V}^n} \quad h \in (\Sigma \cup \mathbf{dom}(\Gamma^0) \cup \mathbf{dom}(\Gamma^-))}{\mathbf{U}^1 \neq^? \mathbf{V}^1 \;\Big|\; \ldots \;\Big|\; \mathbf{U}^n \neq^? \mathbf{V}^n}$$

$$\frac{F\overline{\mathbf{U}} =^? h\overline{\mathbf{V}} \quad \Gamma(F) = \alpha \quad \Gamma \vdash_\Sigma \overline{\mathcal{R}}(F, \mathbf{G})}{F \neq^? \mathbf{G} \;\Big|\; F\overline{\mathbf{U}} \neq^? h\overline{\mathbf{V}}}$$

Here we have

- ▹ $\mathbf{G} \in \mathbf{G}^h(\Sigma, \Delta, \mathcal{C})$
- ▹ $\Gamma' = \Gamma \cup \mathcal{C}$ and $\mathcal{R}' := \mathcal{R}$

Part IV

Axiomatic Set Theory (ZFC)

Sets are one of the most useful structures of mathematics. They can be used to form the basis for representing functions, ordering relations, groups, vector spaces, etc. In fact, they can be used as a foundation for all of mathematics as we know it. But sets are also among the most difficult structures to get right: we have already seen that "naive" conceptions of sets lead to inconsistencies that shake the foundations of mathematics.

There have been many attempts to resolve this unfortunate situation and come up a "foundation of mathematics": an inconsistency-free "foundational logic" and "foundational theory" on which all of mathematics can be built.

In this Part we will present the best-known such attempt – and an attempt it must remain as we will see – the axiomatic set theory by Zermelo and Fraenkel (ZFC), a set of axioms for first-order logic that carefully manage set comprehension to avoid introducing the "set of all sets" which leads us into the paradoxes.

Recommended Reading: The – historical and personal – background of the material covered in this Part is delightfully covered in [DPPDD09].

Chapter 18

Naive Set Theory

We will first recap "naive set theory" and try to formalize it in first-order logic to get a feeling for the problems involved and possible solutions.

(Naive) Set Theory [Can95, Can97]

▷ **Definition 18.0.1** A set is "everything that can form a unity in the face of God". (Georg Cantor ($*1845$, $\dagger 1918$))

▷ **Example 18.0.2** (determination by elementhood relation $\in$)

- ▹ "the set that consists of the number 7 and the prime divisors of 510510"
- ▹ $\{7,c\}$, $\{1,2,3,4,5n,\ldots\}$, $\{x\,|\,x$ is an integer$\}$, $\{X\,|\,\mathbf{P}(X)\}$

Questions (extensional/intensional):

▷
- ▹ If $c=7$, is $\{7,c\}=\{7\}$?
- ▹ Is $\{X\,|\,X\in\mathbb{N},X\neq X\}=\{X\,|\,X\in\mathbb{N},X^2<0\}$?
- ▹ yes $\leadsto$ *extensional*; no $\leadsto$ *intensional*;

Georg Cantor was the first to systematically develop a "set theory", introducing the notion of a "power set" and distinguishing finite from infinite sets – and the latter into denumerable and uncountable sets, basing notions of cardinality on bijections.

In doing so, he set a firm foundation for mathematics[1], even if that needed more work as was later discovered.

Now let us see whether we can write down the "theory of sets" as envisioned by Georg Cantor in first-order logic – which at the time Cantor published his seminal articles was just being invented by Gottlob Frege. The main idea here is to consider sets as individuals, and only introduce a single predicate – apart from equality which we consider given by the logic: the binary elementhood predicate.

(Naive) Set Theory: Formalization

[1]David Hilbert famously exclaimed "*No one shall expel us from the Paradise that Cantor has created*" in [Hil26, p. 170]

▷ Idea: Use first-order logic (with equality)

- ▷ Signature: (sets are individuals) $\Sigma := \{\in\}$
- ▷ Extensionality: $\forall M, N . M = N \Leftrightarrow (\forall X . X \in M \Leftrightarrow X \in N)$
- ▷ Comprehension: (all sets that we can write down exist)
  $\exists M . \forall X . X \in M \Leftrightarrow \mathbf{E}$ (schematic in expression $\mathbf{E}$)

▷ Idea: Define set theoretic concepts from $\in$ as signature extensions

Union	$\cup \in \Sigma^f_2$	$\forall M, N, X . X \in (M \cup N) \Leftrightarrow X \in M \vee X \in N$
Intersection	$\cap \in \Sigma^f_2$	$\forall M, N, X . X \in (M \cap N) \Leftrightarrow X \in M \wedge X \in N$
Empty Set	$\emptyset \in \Sigma^f_0$	$\neg (\exists X . X \in \emptyset)$
and so on.	$\vdots$	$\vdots$

 164 JACOBS UNIVERSITY

The central here is the comprehension axiom that states that any set we can describe by writing down a frist-order formula $\mathbf{E}$ – which usually contains the variable X – must exist. This is a direct implementation of Cantor's intuition that sets can be "*. . . everything that forms a unity . . .*". The usual set-theoretic operators ∪, ∩, . . . can be defined by suitable axioms.

This formalization will now allow to understand the problems of set theory: with great power comes great responsibility!

(Naive) Set Theory (Problems)

▷ **Example 18.0.3 (The set of all set and friends)**
$\{M \mid M \text{ set}\}$, $\{M \mid M \text{ set}, M \in M\}$, . . .

▷ **Definition 18.0.4 (Problem)** Russell's Antinomy:

$$\mathcal{M} := \{M \mid M \text{ set}, M \notin M\}$$

the set $\mathcal{M}$ of all sets that do not contain themselves.

▷ Question: Is $\mathcal{M} \in \mathcal{M}$? Answer: $\mathcal{M} \in \mathcal{M}$ iff $\mathcal{M} \notin \mathcal{M}$.

▷ What happened?: We have written something down that makes problems

▷ Solutions: Define away the problems:

weaker comprehension	axiomatic set theory	now
weaker properties	higher-order logic	done
non-standard semantics	domain theory [Scott]	another time

 165 JACOBS UNIVERSITY

The culprit for the paradox is the comprehension axiom that guarantees the existence of the "set of all sets" from which we can then separate out Russell's set. Multiple ways have been proposed to get around the paradoxes induced by the "set of all sets". We have already seen one: (typed) higher-order logic simply does not allow to write down MM which is higher-order (sets-as-predicates) way of representing set theory.

The way we are going to exploren now is to remove the general set comprehension axiom we had introduced above and replace it by more selective ones that only introduce sets that are known to be safe.

Chapter 19

ZFC Axioms

We will now introduce the set theory axioms due to Zermelo and Fraenkel.

We write down a first-order theory of sets by declaring axioms in first-order logic (with equality). The basic idea is that all individuals are sets, and we can therefore get by with a single binary predicate: $\in$ for elementhood.

Axiomatic Set Theory in First-Order Logic

- Idea: Avoid paradoxes by cautious (*axiomatic*) Comprehension. ([Zer08])

Ex	$\exists X.X = X$	There is a set
Ext	$\forall M, N.M = N \Leftrightarrow (\forall X.X \in M \Leftrightarrow X \in N)$	Extensionality
Sep	$\forall N.\exists M.\forall Z.Z \in M \Leftrightarrow Z \in N \wedge \mathbf{E}$ From a given set N we can separate all members described by expression $\mathbf{E}$.	

- **Theorem 19.0.1** $\forall M, N.(M \subseteq N) \wedge (N \subseteq M) \Rightarrow M = N$
- **Theorem 19.0.2** M *is uniquely determined in* **Sep**
- Proof Sketch: With **Ext** □
- **Notation 19.0.3** Write $\{X \in N \mid \mathbf{E}\}$ for the set M guaranteed by **Sep**.

Note that we do not have a general comprehension axiom, which allows the construction of sets from expressions, but the separation axiom **Sep**, which – given a set – allows to "separate out" a subset. As this axiom is insufficient to providing any sets at all, we guarantee that there is one in **Ex** to make the theory less boring.

Before we want to develop the theory further, let us fix the success criteria we have for our foundation.

Quality Control

- Question: Is ZFC good? (make this more precise under various views)

foundational: Is ZFC sufficient for mathematics?

adequate: is the ZFC notion of sets adequate?

formal: is ZFC consistent?

ambitious: Is ZFC complete?

pragmatic: Is the formalization convenient?

computational: does the formalization yield computation-guiding structure?

▷ Questions like these help us determine the quality of a foundational system or theory.

 167

The question about consistency is the most important, so we will address it first. Note that the absence of paradoxes is a big question, which we cannot really answer now. But we *can* convince ourselves that the "set of all sets" cannot exist.

How about Russel's Antinomy?

▷ **Theorem 19.0.4** *There is no universal set*

▷ Proof:

P.1 For each set M, there is a set $M_R := \{X \in M \mid X \notin X\}$ by **Sep**.

P.2 show $\forall M . M_R \notin M$

P.3 If $M_R \in M$, then $M_R \notin M_R$, (also if $M_R \notin M$)

P.4 thus $M_R \notin M$ or $M_R \in M_R$. □

▷ to get the paradox we would have to separate from the universal set $\mathcal{A}$, to get $\mathcal{A}_R$.

▷ Great, then we can continue developing our set theory!

 168

Somewhat surprisingly, we can just use Russell's construction to our advantage here. So back to the other questions.

Are there Interesting Sets at all?

▷ yes, e.g. the empty set

- ▷ let M be a set (there is one by **Ex**; we do not need to know what it is)
- ▷ define $\emptyset := \{X \in M \mid X \neq X\}$
- ▷ $\emptyset$ is empty and uniquely determined by **Ext**.

▷ **Definition 19.0.5** Intersections: $M \cap N := \{X \in M \mid X \in N\}$

Question: How about $M \cup N$? or $\mathbb{N}$?

⊳⊳ Answer: we do not know they exist yet! (need more axioms)
Hint: consider $\mathcal{D}_\iota = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \ldots\}$

So we have identified at least interesting set, the empty set. Unfortunately, the existence of the intersection operator is no big help, if we can only intersect with the empty set. In general, this is a consequence of the fact that **Sep** – in contrast to the comprehension axiom we have abolished – only allows to make sets "smaller". If we want to make sets "larger", we will need more axioms that guarantee these larger sets. The design contribution of axiomatic set theories is to find a balance between "too large" – and therefore paradoxical – and "not large enough" – and therefore inadequate.

Before we have a look at the remaining axioms of ZFC, we digress to a very influential experiment in developing mathematics based on set theory.

"Nicolas Bourbaki" is the collective pseudonym under which a group of (mainly French) 20th-century mathematicians, with the aim of reformulating mathematics on an extremely abstract and formal but self-contained basis, wrote a series of books beginning in 1935. With the goal of grounding all of mathematics on set theory, the group strove for rigour and generality.

Is Set theory enough? $\rightsquigarrow$ Nicolas Bourbaki

▷ Is it possible to develop all of Mathematics from set theory?
$\rightsquigarrow$ N. Bourbaki: Éléments de Mathématique~~s~~ (there is only one mathematics)

▷ Original Goal: A modern textbook on calculus.

▷ Result: 40 volumes in nine books from 1939 to 1968

Set Theory [Bou68]	Functions of one real variable	Commutative Algebra
Algebra [Bou74]	Integration	Lie Theory
Topology [Bou89]	Topological Vector Spaces	Spectral Theory

▷ Contents:

- ▷ starting from set theory all of the fields above are developed.
- ▷ All proofs are carried out, no references to other books.

Even though Bourbaki has dropped in favor in modern mathematics, the universality of axiomatic set theory is generally acknowledged in mathematics and their rigorous style of exposition has influenced modern branches of mathematics.

The first two axioms we add guarantee the unions of sets, either of finitely many – $\cup$**Ax** only guarantees the union of two sets – but can be iterated. And an axiom for unions of arbitrary families of sets, which gives us the infinite case. Note that once we have the ability to make finite sets, $\bigcup$**Ax** makes $\cup$**Ax** redundant, but minimality of the axiom system is not a concern for us currently.

The Axioms for Set Union

▷ **Axiom 19.0.6 (Small Union Axiom ($\cup$Ax))** For any sets M and N there is a set W, that contains all elements of M and N.
$\forall M, N.\exists W.\forall X.(X \in M \vee X \in N) \Rightarrow X \in W$

▷ **Definition 19.0.7** $M \cup N := \{X \in W \mid X \in M \vee X \in N\}$ (exists by **Sep.**)

▷ **Axiom 19.0.8 (large Union Axiom ($\bigcup$Ax))** For each set M there is a set W, that contains the elements of all elements of M.
$\forall M.\exists W.\forall X, Y.Y \in M \Rightarrow X \in Y \Rightarrow X \in W$

▷ **Definition 19.0.9** $\bigcup(M) := \{X \mid \exists Y.Y \in M \wedge X \in Y\}$ (exists by **Sep.**)

▷ This also gives us intersections over families (without another axiom):

▷ **Definition 19.0.10**

$$\bigcap(M) := \{Z \in \bigcup(M) \mid \forall X.X \in M \Rightarrow Z \in X\}$$

  JACOBS UNIVERSITY

In Definition 19.0.10 we note that $\bigcup$**Ax** also guarantees us intersection over families. Note that we could not have defined that in analogy to Definition 19.0.5 since we have no set to separate out of. Intuitively we could just choose one element N from M and define

$$\bigcap(M) := \{Z \in N \mid \forall X.X \in M \Rightarrow Z \in X\}$$

But for choice from an infinite set we need another axiom still.

The power set axiom is one of the most useful axioms in ZFC. It allows to construct finite sets.

The Power Set Axiom

▷ **Axiom 19.0.11 (Power Set Axiom)** For each set M there is a set W that contains all subsets of M: $\wp\mathrm{Ax} := (\forall M.\exists W.\forall X.(X \subseteq M) \Rightarrow X \in W)$

▷ **Definition 19.0.12** Power Set: $\mathcal{P}(M) := \{X \mid X \subseteq M\}$ (Exists by **Sep.**)

▷ **Definition 19.0.13** singleton set: $\{X\} := \{Y \in \mathcal{P}(X) \mid X = Y\}$

▷ **Axiom 19.0.14 (Pair Set (Axiom))** (is often assumed instead of $\cup$**Ax**)
Given sets M and N there is a set W that contains exactly the elements M and N: $\forall M, N.\exists W.\forall X.X \in W \Leftrightarrow (X = N) \vee (X = M)$

▷ Is derivable from $\wp\mathrm{Ax}$: $\{M, N\} := \{M\} \cup \{N\}$.

▷ **Definition 19.0.15 (Finite Sets)** $\{X, Y, Z\} := \{X, Y\} \cup \{Z\}\ldots$

▷ **Theorem 19.0.16** $\forall Z, X_1, \ldots, X_n.Z \in \{X_1, \ldots, X_n\} \Leftrightarrow Z = X_1 \vee \ldots \vee Z = X_n$

  JACOBS UNIVERSITY

The Foundation Axiom

▷ **Axiom 19.0.17 (The foundation Axiom (Fund))** Every non-empty set has a $\in$-minimal element,.
$\forall X.X \neq \emptyset \Rightarrow (\exists Y.Y \in X \wedge \neg(\exists Z.Z \in X \wedge Z \in Y))$

- ▷ **Theorem 19.0.18** *There are no infinite descendig chains* $\ldots, X_2, X_1, X_0$ *and thus no cycles* $\ldots X_1, X_0, \ldots, X_2, X_1, X_0$.
- ▷ **Definition 19.0.19 Fund** guarantees a hierarchical structure (von Neumann Hierarchy) of the universe. 0. order: $\emptyset$, 1. order: $\{\emptyset\}$, 2. order: all subsets of 1. order, $\cdots$
- ▷ Note: In contrast to a Russel-style typing where sets of differernt type are distinct, this categorization is cummulative

The Infinity Axiom

- ▷ We already know a lot of sets
  - ▷ z.B. $\emptyset$, $\{\emptyset\}$, $\{\{\emptyset\}\}$, ... (iterated singleton set)
  - ▷ or $\emptyset$, $\{\emptyset\}$, $\{\emptyset, \{\emptyset\}\}$, ... (iterated pair set)

  But: Does the set $\mathbb{N}$ of all members of these sequences?
- ▷ **Axiom 19.0.20 (Infinity Axiom (∞Ax))** There is a set that contains $\emptyset$ and with each X also $X \cup \{X\}$.
  $\exists M.\emptyset \in M \wedge (\forall Z.Z \in M \Rightarrow (Z \cup \{Z\}) \in M)$.
- ▷ **Definition 19.0.21** M is inductive: $\mathbf{Ind}(M) := \emptyset \in M \wedge (\forall Z.Z \in M \Rightarrow (Z \cup \{Z\}) \in M)$.
- ▷ **Definition 19.0.22** Set of the Inductive Set: $\omega := \{Z \mid \forall W.\mathbf{Ind}(W) \Rightarrow Z \in W\}$
- ▷ **Theorem 19.0.23** ω *is inductive.*

The Replacement Axiom

- ▷ We have ω, $\wp(M)$, but not $\{\omega, \wp(\omega), \wp(\wp(\omega)), \ldots\}$.
- ▷ **Axiom 19.0.24 (The Replacement Axiom (Schema): Rep)** If for each X there is exactly one Y with property $\mathbf{P}(X,Y)$, then for each set U, that contains these X, there is a set V that contains the respective Y.
  $(\forall X.\exists^1 Y.\mathbf{P}(X,Y)) \Rightarrow (\forall U.\exists V.\forall X,Y.X \in U \wedge \mathbf{P}(X,Y) \Rightarrow Y \in V)$
- ▷ Intuitively: A right-unique property $\mathbf{P}$ induces a replacement $\forall U.\exists V.V = \{F(X) \mid X \in U\}$.
- ▷ **Example 19.0.25** Let $U = \{1, \{2,3\}\}$ and $\mathcal{P}(X \Leftrightarrow Y) \Leftrightarrow (\forall Z.Z \in Y \Rightarrow Z = X)$, then the induced function F maps each X to the set V that contains X, i.e. $V = \{\{X\} \mid X \in U = \{\{1\}, \{\{2,3\}\}\}\}$.

Zermelo Fraenkel Set Theory

▷ **Definition 19.0.26 (Zermelo Fraenkel Set Theory)** We call the first-order theory given by the axioms below Zermelo/Fraenkel set theory and denote it by **ZF**.

Ex	$\exists X.X = X$
Ext	$\forall M, N.M = N \Leftrightarrow (\forall X.X \in M \Leftrightarrow X \in N)$
Sep	$\forall N.\exists M.\forall Z.Z \in M \Leftrightarrow Z \in N \wedge \mathbf{E}$
$\cup$**Ax**	$\forall M, N.\exists W.\forall X.(X \in M \vee X \in N) \Rightarrow X \in W$
$\bigcup$**Ax**	$\forall M.\exists W.\forall X, Y.Y \in M \Rightarrow X \in Y \Rightarrow X \in W$
$\wp$Ax	$\forall M.\exists W.\forall X.(X \subseteq M) \Rightarrow X \in W$
∞**Ax**	$\exists M.\emptyset \in M \wedge (\forall Z.Z \in M \Rightarrow (Z \cup \{Z\}) \in M)$
Rep	$(\forall X.\exists^1 Y.\mathbf{P}(X, Y)) \Rightarrow (\forall U.\exists V.\forall X, Y.X \in U \wedge \mathbf{P}(X, Y) \Rightarrow Y \in V)$
Fund	$\forall X.X \neq \emptyset \Rightarrow (\exists Y.Y \in X \wedge \neg(\exists Z.Z \in X \wedge Z \in Y))$

The Axiom of Choice

▷ **Axiom 19.0.27 (The axiom of Choice :AC)** For each set X of non-empty, pairwise disjoint subsets there is a set that contains exactly one element of each element of X.
$\forall X, Y, Z.Y \in X \wedge Z \in X \Rightarrow Y \neq \emptyset \wedge (Y = Z \vee Y \cap Z = \emptyset) \Rightarrow \exists U.\forall V.V \in X \Rightarrow (\exists W.U \cap V = \{W\})$

▷ This axiom assumes the existence of a set of representatives, even if we cannot give a construction for it. $\leadsto$ we can "pick out" an arbitrary element.

▷ Reasons for **AC**:

- ▷ Neither $\mathbf{ZF} \vdash \mathbf{AC}$, nor $\mathbf{ZF} \vdash \neg\mathbf{AC}$
- ▷ So it does not harm?

▷ **Definition 19.0.28 (Zermelo Fraenkel Set Theory with Choice)** The theory **ZF** together with **AC** is called ZFC with choice and denoted as **ZFC**.

Chapter 20

ZFC Applications

Limits of ZFC

▷ **Conjecture 20.0.1 (Cantor's Continuum Hypothesis (CH))** *There is no set whose cardinality is strictly between that of integers and real numbers.*

▷ **Theorem 20.0.2** *If* **ZFC** *is consistent, then neither* **CH** *nor* ¬ **CH** *can be derived.* (**CH** *is independent of* **ZFC**)

▷ The axiomatzation of **ZFC** does not suffice

▷ There are other examples like this.

Ordered Pairs

▷ Empirically: In **ZFC** we can define all mathematical concepts.

▷ For Instance: We would like a set that behaves like an odererd pair

▷ **Definition 20.0.3** Define $\langle X,Y\rangle := \{\{X\},\{X,Y\}\}$

▷ **Lemma 20.0.4** $\langle X,Y\rangle = \langle U,V\rangle \Rightarrow X = U \wedge Y = V$

▷ **Lemma 20.0.5** $U \in X \wedge V \in Y \Rightarrow \langle U,V\rangle \in \mathcal{P}(\mathcal{P}(X\cup Y))$

▷ **Definition 20.0.6** left projection: $\pi_l(X) = \begin{cases} U & \text{if } \exists V\,.\,X = \langle U,V\rangle \\ \emptyset & \text{if } X \text{ is no pair} \end{cases}$

▷ **Definition 20.0.7** right projection π_r analogous.

Relations

▷ All mathematical objects are represented by sets in **ZFC**, in particular relations

▷ **Definition 20.0.8** The Cartesian produkt of X and Y
$X \times Y := \{Z \in \mathcal{P}(\mathcal{P}(X \cup Y)) \mid Z \text{ is ordered pair with } \pi_l(Z) \in X \wedge \pi_r(Z) \in Y\}$
A relation is a subset of a Cartesian product.

▷ **Definition 20.0.9** The domain and codomain of a function are defined as usual

$$\mathrm{Dom}(X) = \begin{cases} \{\pi_l(Z) \,|\, Z \in X\} & \text{if if X is a relation;} \\ \emptyset & \text{else} \end{cases}$$

$$\mathrm{coDom}(X) = \begin{cases} \{\pi_r(Z) \,|\, Z \in X\} & \text{if if } X \text{ is a relation;} \\ \emptyset & \text{else} \end{cases}$$

but they (as first-order functions) must be total, so we (arbitrarily) extend them by the empty set for non-relations

Functions

▷ **Definition 20.0.10** A function f from X to Y is a right-unique relation with $\mathrm{Dom}(f) = X$ and $\mathrm{coDom}(f) = Y$; write $f : X \to Y$.

▷ **Definition 20.0.11** function application: $f(X) = \begin{cases} Y & \text{if } f \text{ function and } \langle X, Y \rangle \in f \\ \emptyset & \text{else} \end{cases}$

Domain Language vs. Representation Language

▷ Note: Relations and functions are objects of set theory, $ZFC \in$ is a predicate of the representation language

▷ predicates and functions of the representation language can be expressed in the object language:

- ▷ $\forall A.\exists R.R = \{\langle U, V \rangle \,|\, U \in A \wedge V \in A \wedge p(U \wedge V)\}$ for all predicates p.
- ▷ $\forall A.\exists F.F = \{\langle X, f(X) \rangle \,|\, X \in A\}$ for all functions f.

▷ As the natural numbers can be epxressed in set theory, the logical calculus can be expressed by Gödelization.

Part V

Knowledge Representation

In the third and final part of the course, we will look into logic-based formalisms for knowledge representation and their application in the "Semantic Web".

The field of "Knowledge Representation" is usually taken to be an area in Artificial Intelligence that studies the representation of knowledge in formal system and how to leverage inferencing techniques to generate new knowledge items from existing ones.

Note that this definition coincides with with what we know as "logical systems" in this course. This is the view taken by the subfield of "description logics", but restricted to the case, where the logical systems have a entailment relation to ensure applicability.

This Part is organized as follows. We will first give a general introduction to the concepts of knowledge representation using semantic networks – an early and very intuitive approach to knowledge representation – as an object-to-think-with. In Chapter 21 we introduce the principles and services of logic-based knowledge-representation using a non-standard interpretation of propositional logic as the basis, this gives us a formal account of the taxonomic part of semantic networks. In Chapter 22 we introduce the logic $\mathcal{ALC}$ that adds relations (called "roles") and restricted quantification and thus gives us the full expressive power of semantic networks. Thus $\mathcal{ALC}$ can be seen as a prototype description logic. In Chapter 23 we show how description logics are applied as the basis of the "semantic web", and finally in Chapter 24 we show various extensions of $\mathcal{ALC}$ an their inference procedures.

Chapter 21

Introduction to Knowledge Representation

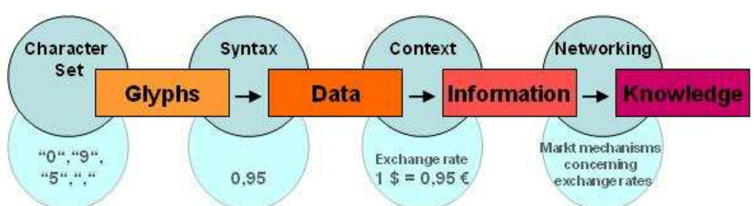
Before we start into the development of description logics, we set the stage by looking into alternatives for knowledge representation.

21.1 Knowledge & Representation

To approach the question of knowledge representation, we first have to ask ourselves, what knowledge might be. This is a difficult question that has kept philosophers occupied for millennia. We will not answer this question in this course, but only allude to and discuss some aspects that are relevant to our cause of knowledge representation.

What is knowledge? Why Representation?

▷ According to Probst/Raub/Romhardt [PRR97]



For the purposes of this course: Knowledge is the information necessary to support intelligent reasoning

▷

representation	can be used to determine
set of words	whether a word is admissible
list of words	the rank of a word
a lexicon	translation or grammatical function
structure	function

According to an influential view of [PRR97], knowledge is appears in layers. Staring with a character set that defines a set of glyphs, we can add syntax that turns mere strings into data.

Adding context information gives information, and finally, by relating the information to other information allows to draw conclusions, turning information into knowledge.

Note that we already have aspects of representation and function in the diagram at the top of the slide. In this, the additional functions added in <the successive layers give the representations more and more function, until we reach the knowledge level, where the function is given by inferencing. In the second example, we can see that representations determine possible functions.

Let us now strengthen our intuition about knowledge by contrasting knowledge representations from "regular" data structures in computation.

Knowledge Representation vs. Data Structures

- ▷ Representation as structure and function.
  - ▷ the representation determines the content theory (what is the data?)
  - ▷ the function determines the process model(what do we do with the data?)
- ▷ Why do we use the term "knowledge representation" rather than
  - ▷ data structures? (sets, lists, ... above)
  - ▷ information representation? (it is information)
- ▷ no good reason other than AI practice, with the intuition that
  - ▷ data is simple and general (supports many algorithms)
  - ▷ knowledge is complex (has distinguished process model)

 184

As knowledge is such a central notion in artificial intelligence, it is not surprising that there are multiple approaches to dealing with it. We will only deal with the first one and leave the others to self-study.

Some Paradigms for Knowledge Representation in AI/NLP

- ▷ GOFAI (good old-fashioned AI)
  - ▷ symbolic knowledge representation, process model based on heuristic search
- ▷ statistical, corpus-based approaches.
  - ▷ symbolic representation, process model based on machine learning
  - ▷ knowledge is divided into symbolic- and statistical (search) knowledge
- ▷ connectionist approach (not in this course)
  - ▷ sub-symbolic representation, process model based on primitive processing elements (nodes) and weighted links
  - ▷ knowledge is only present in activation patters, etc.

 185

When assessing the relative strengths of the respective approaches, we should evaluate them with respect to a pre-determined set of criteria.

KR Approaches/Evaluation Criteria

- Expressive Adequacy: What can be represented, what distinctions are supported.
- Reasoning Efficiency: can the representation support processing that generates results in acceptable speed?
- Primitives: what are the primitive elements of representation, are they intuitive, cognitively adequate?
- Meta-representation: knowledge about knowledge
- Incompleteness: the problems of reasoning with knowledge that is known to be incomplete.

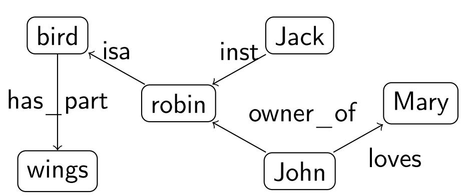
21.2 Semantic Networks

To get a feeling for early knowledge representation approaches from which description logics developed, we take a look at "semantic networks" and contrast them to logical approaches.

Semantic networks are a very simple way of arranging concepts and their relations in a graph.

Semantic Networks [CQ69]

- **Definition 21.2.1** A semantic network is a directed graph for representing knowledge:
  - nodes represent concepts (e.g. bird, John, robin)
  - links represent relations between these (isa, father_of, belongs_to)
- **Example 21.2.2** A semantic net for birds and persons:



  Problem: how do we do inference from such a network?

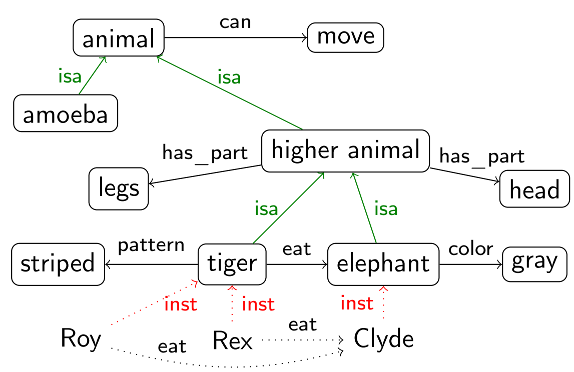
- Idea: encode taxonomic information about concepts and individuals
  - in "isa" links (inclusion of concepts)
  - in "inst" links (concept memberships)
  - use property inheritance along "isa" and "inst" in the process model

Even though the network in Example 21.2.2 is very intuitive (we immediately understand the concepts depicted), it is unclear how we (and more importantly a machine that does not associate meaning with the labels of the nodes and edges) can draw inferences from the "knowledge" represented.

Another problem is that the semantic net in Example 21.2.2 confuses two kinds of concepts: individuals (represented by proper names like *John* and *Jack*) and concepts (nouns like *robin* and *bird*). Even though the "isa" and "inst" links already acknowledge this distinction, the "has_part" and "loves" relations are at different levels entirely, but not distinguished in the networks.

Terminologies and Assertions

▷ **Example 21.2.3** From the network



infer that *elephants* have *legs* and that *Clyde* is *gray*.

▷ **Definition 21.2.4** We call the subgraph of a semantic network N spanned by the "isa" relations the terminology (or TBox, or the famous Isa-Hierarchy) and the subgraph spanned by the "inst" relation the assertions (or ABox) of N.

But there are sever shortcomings of semantic networks: the suggestive shape and node names give (humans) a false sense of meaning, and the inference rules are only given in the process model (the implementation of the semantic network processing system).

This makes it very difficult to assess the strength of the inference system and make assertions e.g. about completeness.

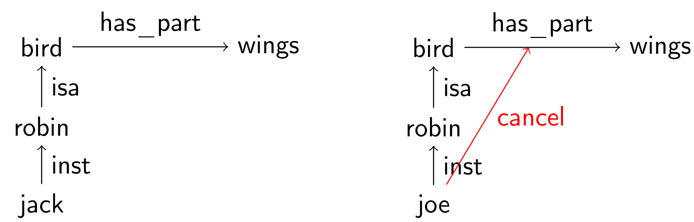
Limitations of Semantic Networks

▷ What is the meaning of a link?

- ▹ link names are very suggestive (misleading for humans)
- ▹ meaning of link types defined in the process model (no denotational semantics)

Problem: No distinction of optional and defining traits

▷ **Example 21.2.5** Consider a robin that has lost its wings in an accident



Cancel-links have been proposed, but their status and process model are debatable.

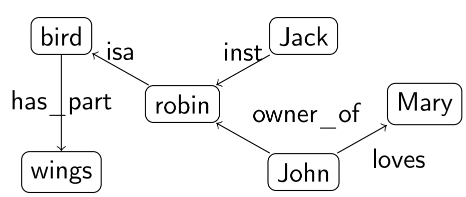
To alleviate the perceived drawbacks of semantic networks, we can contemplate another notation that is more linear and thus more easily implemented: function/argument notation.

Another Notation for Semantic Networks

▷ **Definition 21.2.6 (Idea)** function/argument notation for semantic networks

- ▷ interprets nodes as arguments (reification to individuals)
- ▷ interprets links as functions (logical relations)

▷ **Example 21.2.7**



isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

Evaluation:

▷ + linear notation (equivalent, but better to implement on a computer)
+ easy to give process model by deduction (e.g. in ProLog)
– worse locality properties (networks are associative)

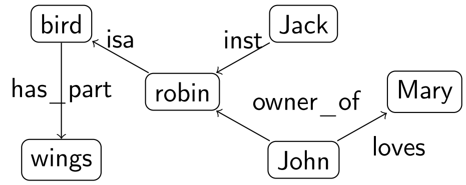
Indeed the function/argument notation is the immediate idea how one would naturally represent semantic networks for implementation.

This notation has been also characterized as subject/predicate/object triples, alluding to simple (English) sentences. This will play a role in the "semantic web" later.

Building on the function/argument notation from above, we can now give a formal semantics for semantic networks: we translate into first-order logic and use the semantics of that.

A Denotational Semantics for Semantic Networks

▷ Extension: take isa/inst concept/individual distinction into account



$robin \subseteq bird$
haspart(bird,wings)
$Jack \in robin$
owner_of(John, Jack)
loves(John,Mary)

▷ Observation: this looks like first-order logic, if we take

- ▷ $A \subseteq B$ to mean $\forall X.A(X) \Rightarrow B(X)$
- ▷ $a \in S$ to mean $S(a)$
- ▷ $haspart(A, B)$ to mean $\forall X.A(X) \Rightarrow (\exists Y.B(Y) \wedge part\_of(X, Y))$

▷ Idea: Take first-order deduction as process model (gives inheritance for free)

Indeed, the semantics induced by the translation to first-order logic, gives the intuitive meaning to the semantic networks. Note that this only holds only for the features of semantic networks that are representable in this way, e.g. the cancel links shown above are not (and that is a feature, not a bug).

But even more importantly, the translation to first-order logic gives a first process model: we can use first-order inference to compute the set of inferences that can be drawn from a semantic network.

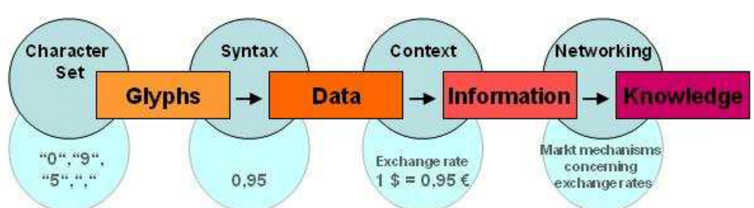
Before we go on, let us have a look at an important application of knowledge representation technologies: the Semantic Web.

21.3 The Semantic Web

The Semantic Web

▷ **Definition 21.3.1** The semantic web is a collaborative movement led by the W3C that promotes the inclusion of semantic content in web pages with the aim of t converting the current web, dominated by unstructured and semi-structured documents into a machine-understandable "web of data".

▷ Idea: Move web content up the ladder, use inference to make connections.



▷ **Example 21.3.2** We want to find information that is not explicitly represented (in one place)

Query: *Who was US president when Barak Obama was born?*

Google: *. . . BIRTH DATE: August 04, 1961. . .*

Query: *Who was US president in 1961?*

Google: *President: Dwight D. Eisenhower [...] John F. Kennedy (starting January 20)*

Humans can read (and understand) the text and combine the information to get the answer.

The term "Semantic Web" was coined by Tim Berners Lee in analogy to semantic networks, only applied to the world wide web. And as for semantic networks, where we have inference processes that allow us the recover information that is not explicitly represented from the network (here the world-wide-web).

To see that problems have to be solved, to arrive at the "Semantic Web", we will now look at a concrete example about the "semantics" in web pages. Here is one that looks typical enough.

What is the Information a User sees?

WWW2002
The eleventh International World Wide Web Conference
Sheraton Waikiki Hotel
Honolulu, Hawaii, USA
7-11 May 2002

Registered participants coming from
Australia, Canada, Chile Denmark, France, Germany, Ghana, Hong Kong, India,
Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, Norway,
Singapore, Switzerland, the United Kingdom, the United States, Vietnam, Zaire

On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.

Speakers confirmed
Tim Berners-Lee: Tim is the well known inventor of the Web,
Ian Foster: Ian is the pioneer of the Grid, the next generation internet.

But as for semantic networks, what you as a human can see ("understand" really) is deceptive, so let us obfuscate the document to confuse your "semantic processor". This gives an impression of what the computer "sees".

What the machine sees

$\mathcal{WWW}\in\prime\prime\in$

$\mathcal{T}\langle]]\updownarrow]\sqsubseteq]\backslash\sqcup\langle\mathcal{I}\backslash\sqcup]\nabla\backslash\dashv\sqcup\rangle\wr\backslash\dashv\updownarrow\mathcal{W}\wr\nabla\updownarrow\lceil\mathcal{W}\rangle\lceil]\mathcal{W}]\lfloor\mathcal{C}\wr\backslash\{]\nabla]\backslash\rfloor]$

S⟨]∇⊣⊔≀\W⊣⟩∥⟩∥⟩H≀⊔]↕
H≀\≀↕⊓↕⊓⇔H⊣⊒⊣⟩⟩⇔USA
↖∞∞M⊣†∈′′∈

R]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫≀↕⟩\}{∇≀↕
A⊓∫⊔∇⊣↕⟩⊣⇔C⊣\⊣⌈⊣⇔C⟨⟩↕]D]\↕⊣∇∥⇔F∇⊣\⌋]⇔G]∇↕⊣\†⇔G⟨⊣\⊣⇔H≀\}K≀\}⇔
I\⌈⟩⊣⇔
I∇]↕⊣\⌈⇔I⊔⊣↕†⇔J⊣√⊣\⇔M⊣↕⊔⊣⇔N]⊒Z]⊣↕⊣\⌈⇔T⟨]N]⊔⟨]∇↕⊣\⌈∫⇔N≀∇⊒⊣†⇔
S⟩\}⊣√≀∇]⇔S⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]U\⟩⊔]⌈K⟩\}⌈≀↕⇔⊔⟨]U\⟩⊔]⌈S⊔⊣⊔]∫⇔V⟩]⊔\⊣↕⇔Z⊣⟩∇]

O\⊔⟨]⊔⟨M⊣†H≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣⌋∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨
I\⊔]∇\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]↙

S√]⊣∥]∇∫⌋≀\{⟩∇↕]⌈
T⟩↕B]∇\]∇∫↖L]]¬T⟩↕⟩∫⊔⟨]⊒]↕‡∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]W]⌊⇔
I⊣\F≀∫⊔]∇¬I⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]G∇⟩⌈⇔⊔⟨]\§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔↙

Obviously, there is not much the computer understands, and as a consequence, there is not a lot the computer can support the reader with. So we have to "help" the computer by providing some meaning. Conventional wisdom is that we add some semantic/functional markup. Here we pick XML without loss of generality, and characterize some fragments of text e.g. as dates.

Solution: XML markup with "meaningful" Tags

<title>WWW∈′′∈
T⟨]]↕]⊑]\⊔⟨I\⊔]∇\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]</title>
<place>S⟨]∇⊣⊔≀\W⊣⟩∥⟩∥⟩H≀⊔]↕H≀\≀↕⊓↕⊓⇔H⊣⊒⊣⟩⟩⇔USA</place>
<date>↖∞∞M⊣†∈′′∈</date>
<participants>R]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫≀↕⟩\}{∇≀↕
A⊓∫⊔∇⊣↕⟩⊣⇔C⊣\⊣⌈⊣⇔C⟨⟩↕]D]\↕⊣∇∥⇔F∇⊣\⌋]⇔G]∇↕⊣\†⇔G⟨⊣\⊣⇔H≀\}K≀\}⇔
I\⌈⟩⊣⇔
I∇]↕⊣\⌈⇔I⊔⊣↕†⇔J⊣√⊣\⇔M⊣↕⊔⊣⇔N]⊒Z]⊣↕⊣\⌈⇔T⟨]N]⊔⟨]∇↕⊣\⌈∫⇔N≀∇⊒⊣†⇔
S⟩\}⊣√≀∇]⇔S⊒⟩⊔‡]∇↕⊣\⌈⇔⊔⟨]U\⟩⊔]⌈K⟩\}⌈≀↕⇔⊔⟨]U\⟩⊔]⌈S⊔⊣⊔]∫⇔V⟩]⊔\⊣↕⇔Z⊣⟩∇]</participants>
</introduction>O\⊔⟨]⊔⟨M⊣†H≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈]⊔⟨]⌊⊣⌋∥⌈∇≀√≀{⊔⟨]]↕]⊑]\⊔⟨
I\⊔]∇\⊣⊔⟩≀\⊣↕W≀∇↕⌈W⟩⌈]W]⌊C≀\{]∇]\⌋]↙</introduction>
<program>S√]⊣∥]∇∫⌋≀\{⟩∇↕]⌈
<speaker>T⟩↕B]∇\]∇∫↖L]]¬T⟩↕⟩∫⊔⟨]⊒]↕‡∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]W]⌊</speaker>
<speaker>I⊣\F≀∫⊔]∇¬I⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]G∇⟩⌈⇔⊔⟨]\§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇↖
\]⊔<speaker>
</program>

What can we do with this?

▷ **Example 21.3.3** Consider the following fragments:

```
<title>𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉⊔⊑⌉\⊓⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉</title>
<place>𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</place>
<date>↖∞∞ℳ⊣†∈′′∈</date>
```

Given the markup above, we can

- ▷ parse ↖∞∞ℳ⊣†∈′′∈ as the date May 7-11 2002 and add this to the user's calendar.
- ▷ parse 𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜 as a destination and find flights.

But: do not be deceived by your ability to understand English

We have to obfuscate the markup as well, since it does not carry any meaning to the machine intrinsically either.

▷ What the machine sees of the XML

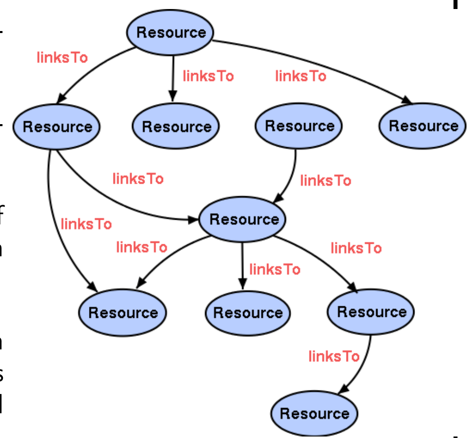
```
<⊔⟩⊔↕⌉>𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉⊔⊑⌉\⊓⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉</⊔⟩⊔↕⌉>
<√↕⊣⌋⌉>𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</√↕⊣⌋⌉>
<⌈⊣⊔⌉>↖∞∞ℳ⊣†∈′′∈</⌈⊣⊔⌉>
<√⊣∇⊔⟩⌋⟩√⊣\⊔∫>ℛ⌉}⟩∫⊔⌉∇⌉⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌊≀⌋⟩\}{∇≀↕
𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕⌉𝒟⌉\↕⊣∇∥⇔ℱ∇⊣\⌋⌉⇔𝒢⌉∇↕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔
ℐ\⌈⟩⊣⇔
ℐ∇⌉↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩⌉⊒𝒵⌉⊣↕⊣\⌈⇔𝒯⟨⌉𝒩⌉⊔⟨⌉∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇⌉⇔𝒮⊒⟩⊔‡⌉∇↕⊣\⌈⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒦⟩\}⌈≀↕⇔⊔⟨⌉𝒰𝒮𝒜⇔𝒱⟩⌉⊔\⊣↕⇔𝒵⊣⟩∇⌉</√⊣∇⊔⟩⌋⟩√⊣\⊔∫>
</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>𝒪\⊔⟨⌉ℳ⊣†ℋ≀\≀↕⊓⊒⟩↕↕√∇≀⊑⟩⌈⌉⊔⟨⌉⌊⊣∥∥⌈∇≀√{⊔⟨⌉⌉⊔⊑⌉\⊔⟨
ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉↙</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>
<√∇≀}∇⊣↕>𝒮√⌉⊣∥⌉∇∫≀\{⟩⌉↕⌈
<∫√⌉⊣∥⌉∇>𝒯⟩↕ℬ⌉∇\⌉∇∫↖ℒ⌉⌉¬𝒯⟩↕⟩∫⊔⟨⌉⊒⌉↕↕∥\≀⊒\⟩\⊑⌉\⊔≀∇≀{⊔⟨⌉𝒲⌉⌊</∫√⌉⊣∥⌉∇>
<∫√⌉⊣∥⌉∇>ℐ⊣\ℱ≀∫⊔⌉∇¬ℐ⊣\⟩∫⊔⟨⌉√≀\⟩⌉⌉∇≀{⊔⟨⌉𝒢∇⟩⌈⇔⊔⟨⌉\⌉§⊔}⌉\⌉∇⊣⊔⟩≀\⟩\⊔⌉∇↖
\⌉⊔<∫√⌉⊣∥⌉∇>
</√∇≀}∇⊣↕>
```


So we have not really gained much either with the markup, we really have to give meaning to the markup as well, this is where techniques from knowledge representation come into play

To understand how we can make the web more semantic, let us first take stock of the current status of (markup on) the web. It is well-known that world-wide-web is a hypertext, where multimedia documents (text, images, videos, etc. and their fragments) are connected by hyperlinks. As we have seen, all of these are largely opaque (non-understandable), so we end up with the following situation (from the viewpoint of a machine).

The Current Web

- ▷ Resources: identified by URI's, untyped
- ▷ Links: `href`, `src`, . . . limited, non-descriptive
- ▷ User: Exciting world - semantics of the resource, however, gleaned from content
- ▷ Machine: Very little information available - significance of the links only evident from the context around the anchor.

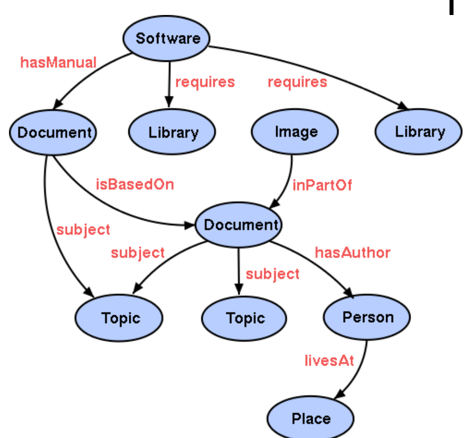


 198

Let us now contrast this with the envisioned semantic web.

The Semantic Web

- ▷ Resources: Globally Identified by URI's or Locally scoped (Blank), Extensible, Relational
- ▷ Links: Identified by URI's, Extensible, Relational
- ▷ User: Even more exciting world, richer user experience
- ▷ Machine: More processable information is available (Data Web)
- ▷ Computers and people: Work, learn and exchange knowledge effectively



 199

Essentially, to make the web more machine-processable, we need to classify the resources by the concepts they represent and give the links a meaning in a way, that we can do inference with that.

The ideas presented here gave rise to a set of technologies jointly called the "semantic web", which we will now summarize before we return to our logical investigations of knowledge representation techniques.

Need to add "Semantics"

- ▷ External agreement on meaning of annotations E.g., Dublin Core
  - ▷ Agree on the meaning of a set of annotation tags
  - ▷ Problems with this approach: Inflexible, Limited number of things can be

expressed

▷ Use Ontologies to specify meaning of annotations

- ▷ Ontologies provide a vocabulary of terms
- ▷ New terms can be formed by combining existing ones
- ▷ Meaning (semantics) of such terms is formally specified
- ▷ Can also specify relationships between terms in multiple ontologies

▷ Inference with annotations and ontologies (get out more than you put in!)

- ▷ Standardize annotations in RDF [KC04] or RDFa [HASB13b] and ontologies on OWL [OWL09]
- ▷ Harvest RDF and RDFa in to a triplestore or OWL reasoner.
- ▷ Query that for implied knowledge (e.g. chaining multiple facts from Wikipedia)

SPARQL: Who was US President when Barack Obama was Born?
DBPedia: John F. Kennedy (was president in August 1961)

21.4 Other Knowledge Representation Approaches

Now that we know what semantic networks mean, let us look at a couple of other approaches that were influential for the development of knowledge representation. We will just mention them for reference here, but not cover them in any depth.

Frame Notation as Logic with Locality

▷ Predicate Logic: (where is the locality?)

$catch\_22 \in catch\_object$	There is an instance of catching
$catcher(catch\_22, jack\_2)$	Jack did the catching
$caught(catch\_22, ball\_5)$	He caught a certain ball

▷ Frame Notation (group everything around the object)

```
(catch_object  catch_22
               (catcher jack_2)
               (caught ball_5))
```

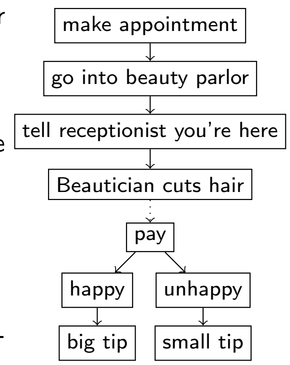
+ Once you have decided on a frame, all the information is local

+ easy to define schemes for concepts (aka. types in feature structures)

– how to determine frame, when to choose frame (log/chair)

KR involving Time (Scripts [Shank '77])

- ▷ Idea: organize typical event sequences, actors and props into representation structure
  - ▷ **Example 21.4.1** getting your hair cut (at a beauty parlor)
    - ▹ props, actors as "script variables"
    - ▹ events in a (generalized) sequence
  - ▷ use script material for
    - ▹ anaphors, bridging references
    - ▹ default common ground
    - ▹ to fill in missing material into situations



Other Representation Formats (not covered)

- ▷ Procedural Representations (production systems)
- ▷ analogical representations (interesting but not here)
- ▷ iconic representations (interesting but very difficult to formalize)
- ▷ If you are interested, come see me off-line

Chapter 22

Logic-Based Knowledge Representation

We now turn to knowledge representation approaches that are based on some kind of logical system. These have the advantage that we know exactly what we are doing: as they are based on symbolic representations and declaratively given inference calculi as process models, we can inspect them thoroughly and even prove facts about them.

Logic-Based Knowledge Representation

- ▷ Logic (and related formalisms) have a well-defined semantics
  - ▷ explicitly (gives more understanding than statistical/neural methods)
  - ▷ transparently (symbolic methods are monotonic)
  - ▷ systematically (we can prove theorems about our systems)
- ▷ Problems with logic-based approaches
  - ▷ Where does the world knowledge come from? (Ontology problem)
  - ▷ How to guide search induced by log. calculi (combinatorial explosion)

One possible answer: Description Logics. (next couple of times)

 204 JACOBS UNIVERSITY

But of course logic-based approaches have big drawbacks as well. The first is that we have to obtain the symbolic representations of knowledge to do anything – a non-trivial challenge, since most knowledge does not exist in this form in the wild, to obtain it, some agent has to experience the word, pass it through its cognitive apparatus, conceptualize the phenomena involved, systematize them sufficiently to form symbols, and then represent those in the respective formalism at hand.

The second drawback is that the process models induced by logic-based approaches (inference with calculi) are quite intractable. We will see that all inferences can be played back to satisfiability tests in the underlying logical system, which are exponential at best, and undecidable or even incomplete at worst.

Therefore a major thrust in logic-based knowledge representation is to investigate logical systems that are expressive enough to be able to represent most knowledge, but still have a decidable – and maybe even tractable in practice – satisfiability problem. Such logics are called "description logics". We will study the basics of such logical systems and their inference procedures in the following.

22.1 Propositional Logic as a Set Description Language

Before we look at "real" description logics in Chapter 22, we will make a "dry run" with a logic we already understand: propositional logic, which we will re-interpret the system as a set description language by giving a new, non-standard semantics. This allows us to already preview most of the inference procedures and knowledge services of knowledge representation systems in the next Section.

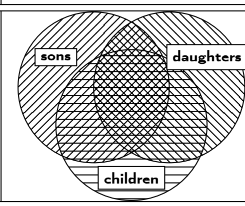
To establish propositional logic as a set description language, we use a different interpretation than usual. We interpret propositional variables as names of sets and the logical connectives as set operations, which is why we give them a different – more suggestive – syntax.

▷ Propositional Logic as Set Description Language

▷ Idea: use propositional logic as a set description language (variant syntax)

- ▷ sets represented as "concepts" (via propositional variables)
- ▷ concept intersection ($\sqcap$) (via conjunction $\wedge$)
- ▷ concept complement ($\bar{\cdot}$) (via negation $\neg$)
- ▷ concept union ($\sqcup$), subsumption ($\sqsubseteq$), and equality ($\equiv$) defined from these. ($\hat{=}\ \wedge$, $\Rightarrow$, and $\Leftrightarrow$)

▷ **Example 22.1.1**

concepts	Set Semantics
child daughter son	

We elaborate the idea of PL^0 as a set description language by giving a variant – set theoretic – semantics as well.

Set-Theoretic Semantics of Axioms

▷ **Definition 22.1.2 (Formal Semantics)**
Let $\mathcal{D}$ be a given set (called the domain) and $\varphi\colon \mathcal{V}_o \to \mathcal{P}(\mathcal{D})$, then

- ▷ $[\![P]\!] := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
- ▷ $[\![\mathbf{A} \sqcup \mathbf{B}]\!] = [\![\mathbf{A}]\!] \cup [\![\mathbf{B}]\!]$ and $[\![\overline{\mathbf{A}}]\!] = \mathcal{D} \backslash [\![\mathbf{A}]\!] \ldots$

Let $\mathcal{L}$ be given by $\mathcal{L} :== C \mid \top \mid \bot \mid \overline{\mathcal{L}} \mid \mathcal{L} \sqcap \mathcal{L} \mid \mathcal{L} \sqcup \mathcal{L} \mid \mathcal{L} \sqsubseteq \mathcal{L} \mid \mathcal{L} \equiv \mathcal{L}$, then we denote the logical system $\mathcal{L}, \mathcal{D}, [\![\cdot]\!]$ with $\mathrm{PL}^0_{\mathrm{DL}}$.

▷ Set-Theoretic Semantics of 'true' and 'false' ($\top = \varphi \sqcup \overline{\varphi} \quad \bot = \varphi \sqcap \overline{\varphi}$)

$$[\![\bot]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \overline{[\![p]\!]} = \mathcal{D} \qquad \text{Analogously: } [\![\bot]\!] = \emptyset$$

Idea: Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)

▷ **Definition 22.1.3 (Set-Theoretic Semantics of Axioms)** $\mathbf{A}$ is true in domain $\mathcal{D}$ iff $[\![\mathbf{A}]\!] = \mathcal{D}$

The main use of the set-theoretic semantics for PL^0 is that we can use it to give meaning to "concept axioms", which we use to describe the "world".

Concept axioms are used to restrict the set of admissible domains to the intended ones. In our situation, we require them to be true – as usual – which here means that they denote the whole domain $\mathcal{D}$.

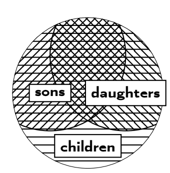
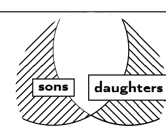
The set-theoretic semantics introduced above is compatible with the regular semantics of propositional logic, therefore we have the same propositional identities. Their validity can be established directly from the settings in Definition 22.1.2.

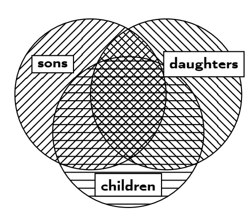
Propositional Identities

Name	for $\sqcap$	for $\sqcup$
Idenpot.	$\varphi \sqcap \varphi = \varphi$	$\varphi \sqcup \varphi = \varphi$
Identity	$\varphi \sqcap \top = \varphi$	$\varphi \sqcup \bot = \varphi$
Absorpt.	$\varphi \sqcup \top = \top$	$\varphi \sqcap \bot = \bot$
Commut.	$\varphi \sqcap \psi = \psi \sqcap \varphi$	$\varphi \sqcup \psi = \psi \sqcup \varphi$
Assoc.	$\varphi \sqcap (\psi \sqcap \theta) = (\varphi \sqcap \psi) \sqcap \theta$	$\varphi \sqcup (\psi \sqcup \theta) = (\varphi \sqcup \psi) \sqcup \theta$
Distrib.	$\varphi \sqcap (\psi \sqcup \theta) = \varphi \sqcap \psi \sqcup \varphi \sqcap \theta$	$\varphi \sqcup \psi \sqcap \theta = (\varphi \sqcup \psi) \sqcap (\varphi \sqcup \theta)$
Absorpt.	$\varphi \sqcap (\varphi \sqcup \theta) = \varphi$	$\varphi \sqcup \varphi \sqcap \theta = \varphi \sqcap \theta$
Morgan	$\overline{\varphi \sqcap \psi} = \overline{\varphi} \sqcup \overline{\psi}$	$\overline{\varphi \sqcup \psi} = \overline{\varphi} \sqcap \overline{\psi}$
dneg	$\overline{\overline{\varphi}} = \varphi$	

Let us fortify our intuition about concept axions with a simple example about the sibling relation. We tive four concept axions and study their effect on the admissible models by looking at the respective Venn diagrams. In the end we see that in all admissible models, the denotations of the concepts son and daughter are disjoint, and child is the union of the two – just as intended.

Effects of Axioms to Siblings

Axioms	Semantics
son $\sqsubseteq$ child iff $[\![\text{son}]\!] \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{son}]\!] \subseteq [\![\text{child}]\!]$ daughter $\sqsubseteq$ child iff $[\![\overline{\text{daughter}}]\!] \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{daughter}]\!] \subseteq [\![\text{child}]\!]$	
$\overline{\text{son} \sqcap \text{daughter}}$ child $\sqsubseteq$ (son $\sqcup$ daughter)	



There is another way we can approach the set description interpretation of propositional logic: by translation into a logic that can express knowledge about sets – first-order logic.

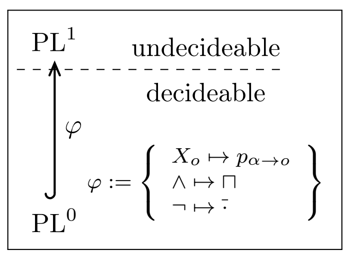
Set-Theoretic Semantics and Predicate Logic

Definition 22.1.4
Translation into PL^1 (borrow semantics from that)

- ▷ recursively add argument variable x
- ▷ change back $\sqcap, \sqcup, \sqsubseteq, \equiv$ to $\wedge, \vee, \Rightarrow, \Leftrightarrow$
- ▷ universal closure for x at formula level.

Definition	Comment
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{\mathbf{A}}}^{fo(x)} := \neg\overline{\mathbf{A}}^{fo(x)}$	
$\overline{\mathbf{A} \sqcap \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \wedge \overline{\mathbf{B}}^{fo(x)}$	$\wedge$ vs. $\sqcap$
$\overline{\mathbf{A} \sqcup \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \vee \overline{\mathbf{B}}^{fo(x)}$	$\vee$ vs. $\sqcup$
$\overline{\mathbf{A} \sqsubseteq \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \Rightarrow \overline{\mathbf{B}}^{fo(x)}$	$\Rightarrow$ vs. $\sqsubseteq$
$\overline{\mathbf{A} = \mathbf{B}}^{fo(x)} := (\overline{\mathbf{A}}^{fo(x)} \Leftrightarrow \overline{\mathbf{B}}^{fo(x)})$	$\Leftrightarrow$ vs. $=$
$\overline{\mathbf{A}}^{fo} := (\forall x.\overline{\mathbf{A}}^{fo(x)})$	for formulae

Normally, we embed PL^0 into PL^1 by mapping propositional variables to atomic predicates and the connectives to themselves. The purpose of this embedding is to "talk about truth/falsity of assertions". For "talking about sets" we use a non-standard embedding: propositional variables in PL^0 are mapped to first-order predicates, and the propositional connectives to corresponding set operations. This uses the convention that a set S is represented by a unary predicate p_S (its characteristic predicate), and set membership $a \in S$ as $p_S(a)$.



Translation Examples

▷ **Example 22.1.5**

$$\begin{aligned}
\overline{\mathsf{son} \sqsubseteq \mathsf{child}}^{fo} &= \forall x.\mathsf{son}(x) \Rightarrow \mathsf{child}(x) \\
\overline{\mathsf{daughter} \sqsubseteq \mathsf{child}}^{fo} &= \forall x.\mathsf{daughter}(x) \Rightarrow \mathsf{child}(x) \\
\overline{\overline{(\mathsf{son} \sqsubseteq \mathsf{daughter})}}^{fo} &= \forall x.\overline{\mathsf{son}(x) \wedge \mathsf{daughter}(x)} \\
\overline{\mathsf{child} \sqsubseteq (\mathsf{son} \sqcup \mathsf{daughter})}^{fo} &= \forall x.\mathsf{child}(x) \Rightarrow \mathsf{son}(x) \vee \mathsf{daughter}(x)
\end{aligned}$$

▷ What are the advantages of translation to PL^1?

- ▷ theoretically: A better understanding of the semantics
- ▷ computationally: Description Logic Framework, but NOTHING fo PL^0
  - ▷ we can follow this pattern for richer description logics
  - ▷ many tests are decidable for PL^0, but not for PL^1 (Description Logics?)

22.2 Ontologies and Description Logics

We have seen how sets of concept axioms can be used to describe the "world" by restricting the set of admissible models. We want to call such concept descriptions "ontologies" – formal descriptions of (classes of) objects and their relations.

Ontologies aka. "World Descriptions"

- **Definition 22.2.1 (Classical)** An ontology is a representation of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse.
- Remark: Definition 22.2.1 is very general, and depends on what we mean by "representation", "entities", "types", and "interrelationships".

  This may be a feature, and not a bug, since we can use the same intuitions across a variety of representations.
- **Definition 22.2.2** An ontology consists of a representation format $\mathcal{L}$ and statements (expressed in $\mathcal{L}$)about
  - Individuals: concrete instances of objects in the domain,
  - concepts: classes of individuals that share properties and aspects, and
  - relations: ways in which classes and individuals can be related to one another
- **Example 22.2.3** Semantic networks are ontologies (relatively informal)
- **Example 22.2.4** $\mathrm{PL}^0_{\mathrm{DL}}$ is an ontology format (formal, but relatively weak)
- **Example 22.2.5** PL^1 is an ontology format as well. (formal, expressive)

 211 JACOBS UNIVERSITY

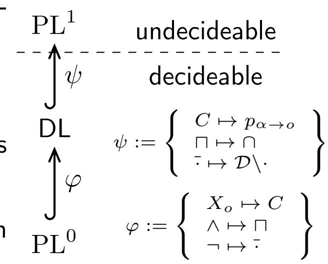
As we will see, the situation for $\mathrm{PL}^0_{\mathrm{DL}}$ is typical for formal ontologies (even though it only offers concepts), so we state the general description logic paradigm for ontologies. The important idea is that having a formal system as an ontology format allows us to capture, study, and implement ontological inference.

The Description Logic Paradigm

- Idea: Build a whole family of logics for describing sets and their relations(tailor their expressivity and computational properties)
- **Definition 22.2.6** A description logic is a formal system for talking about sets and their relations that is at least as expressive as PL^0 with set-theoretic semantics and offers individuals and relations.

  A description logic has the following four components

- a formal language $\mathcal{L}$ with logical constants $\sqcap$, $\overline{\cdot}$, $\sqcup$, $\sqsubseteq$, and $\equiv$.
- a set-theoretic semantics $\mathcal{D}$, $[\![\cdot]\!]$
- a translation into first-order logic that is compatible with $\mathcal{D}$, $[\![\cdot]\!]$.
- a calculus for $\mathcal{L}$ that induces a decision procedure for $\mathcal{L}$-satisfiability.



- **Definition 22.2.7** Given a description logic $\mathcal{D}$, $\mathcal{D}$-ontology consists of
  - a terminology (or TBox): concepts and roles and a set of concept axioms that describe them, and
  - assertions (or ABox): a set of individuals and statements about concept membership and role relationships for them.

For convenience we add concept definitions as a mechanism for defining new concepts from old ones. The so-defined concepts inherit the properties from the concepts they are defined from.

TBoxes in Description Logics

- Let $\mathcal{D}$ be a description logic with concepts $\mathcal{C}$.
- **Definition 22.2.8** A concept definition is a pair $c = \mathbf{C}$, where c is a new concept name and $\mathbf{C} \in \mathcal{C}$ is a $\mathcal{D}$-formula.
- **Definition 22.2.9** A concept definition $c = C$ is called recursive, iff c occurs in C.
- **Example 22.2.10** We can define mother $=$ woman $\sqcap$ has_child.
- **Definition 22.2.11** An TBox is a finite set of concept definitions and concept axioms. It is called acyclic, iff it does not contain recursive definitions.
- **Definition 22.2.12** A formula $\mathbf{A}$ is called normalized wrt. an TBox $\mathcal{T}$, iff it does not contain concept names defined in $\mathcal{T}$. (convenient)
- **Definition 22.2.13 (Algorithm)** (for arbitrary DLs)

  Input: A formula $\mathbf{A}$ and a TBox $\mathcal{T}$.
  - While [$\mathbf{A}$ contains concept name c and $\mathcal{T}$ a concept definition $c = \mathbf{C}$]
    - substitute c by $\mathbf{C}$ in $\mathbf{A}$.
- **Lemma 22.2.14** *This algorithm terminates for acyclic TBoxes, but results can be exponentially large.*

As $\mathrm{PL}^0_{\mathrm{DL}}$ does not offer any guidance on this, we will leave the discussion of ABoxes to Section 23.2 when we have introduced our first proper description logic $\mathcal{ALC}$.

22.3 Description Logics and Inference

Now that we have established the description logic paradigm, we will have a look at the inference services that can be offered on this basis.

Before we go into details of particular description logics, we must ask ourselves what kind of inference support we would want for building systems that support knowledge workers in building, maintaining and using ontologies. An example of such a system is the Protégé system [Pro], which can serve for guiding our intuition.

Kinds of Inference in Description Logics

- ▷ Consistency test (is a concept definition satisfiable?)
- ▷ Subsumption test (does a concept subsume another?)
- ▷ Instance test (is an individual an example of a concept?)
- ▷ . . .
- ▷ Problem: decidability, complexity, algorithm

We will now through these inference-based tests separately.

The consistency test checks for concepts that do not/cannot have instances. We want to avoid such concepts in our ontologies, since they clutter the namespace and do not contribute any meaningful contribution.

Consistency Test

▷ **Example 22.3.1** T-Box

woman	=	person ⊓ $\overline{\text{has\_Y}}$	person without y-chromosome
man	=	person ⊓ has_Y	person with y-chromosome
hermaphrodite	=	man ⊓ woman	man and woman

▷ This specification is inconsistent, i.e. $[\![\text{hermaphrodite}]\!] = \emptyset$ for all $\mathcal{D}$, φ.

▷ Algorithm: propositional satisfiability test (NP-complete) we know how to do this, e.g. tableau, resolution.

Even though consistency in our example seems trivial, large ontologies can make machine support necessary. This is even more true for ontologies that change over time. Say that an ontology initially has the concept definitions woman = person ⊓ long_hair and man = person ⊓ bearded, and then is modernized to a more biologically correct state. In the initial version the concept hermaphrodite is consistent, but becomes inconsistent after the renovation; the authors of the renovation should be made aware of this by the system.

The subsumption test determines whether the sets denoted by two concepts are in a subset relation. The main justification for this is that humans tend to be aware of concept subsumption, and tend to think in taxonomic hierarchies. To cater to this, the subsumption test is useful.

Subsumption Test

▷ **Example 22.3.2** in this case trivial

Axioms	entailed subsumption relation
woman $=$ person $\sqcap \overline{\text{has\_Y}}$	woman $\sqsubseteq$ person
man $=$ person $\sqcap$ has_Y	man $\sqsubseteq$ person

Reduction to consistency test: (need to implement only one)
$Axioms \Rightarrow \mathbf{A} \Rightarrow \mathbf{B}$ is valid iff $Axioms \wedge \mathbf{A} \wedge \neg \mathbf{B}$ is inconsistent.

▷ **Definition 22.3.3** $\mathbf{A}$ subsumes $\mathbf{B}$ (modulo an axiom set $\mathcal{A}$)
iff $[\![\mathbf{B}]\!] \subseteq [\![\mathbf{A}]\!]$ for all interpretations $\mathcal{D}$, that satisfy $\mathcal{A}$
iff $Axioms \Rightarrow \mathbf{B} \Rightarrow \mathbf{A}$ is valid

▷ in our example: person subsumes woman and man

The good news is that we can reduce the subsumption test to the consistency test, so we can re-use our existing implementation.

The main user-visible service of the subsumption test is to compute the actual taxonimy induced by an ontology.

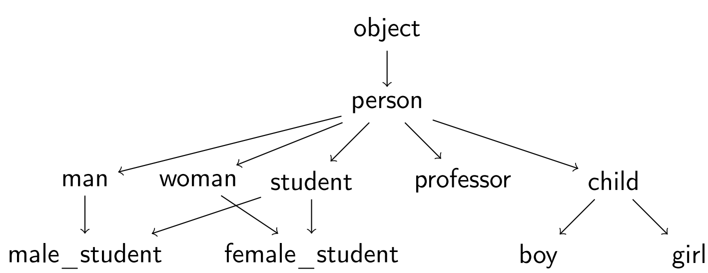
Classification

▷ The subsumption relation among all concepts (subsumption graph)

▷ Visualization of the Subsumption graph for inspection (plausibility)

▷ **Definition 22.3.4** Classification is the computation of the subsumption graph.

▷ **Example 22.3.5** (not always so trivial)



If we take stock of what we have developed so far, then we can see $\mathrm{PL}^0_{\mathrm{DL}}$ as a rational reconstruction of semantic networks restricted to the "isa" relation. We relegate the "instance" relation to Section 23.2.

This reconstruction can now be used as a basis on which we can extend the expressivity and inference procedures without running into problems.

Chapter 23

A simple Description Logic: $\mathcal{ALC}$

In this Chapter, we instantiate the description-logic paradigm further with the prototypical logic $\mathcal{ALC}$, which we will introduce now.

23.1 Basic $\mathcal{ALC}$: Concepts, Roles, and Quantification

In this Section, we instantiate the description-logic paradigm with prototypical logic $\mathcal{ALC}$, which we will devleop now.

Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

- Propositional logic (PL^0) is not expressive enough
- **Example 23.1.1** "mothers are women that have a child"
- Reason: there are no quantifiers in PL^0 (existential ($\exists$) and universal ($\forall$))
- Idea: use first-order predicate logic (PL^1)

$$\forall x.\mathit{mother}(x) \Leftrightarrow \mathit{woman}(x) \wedge (\exists y.\mathit{has\_child}(x, y))$$

- Problem: complex algorithms, non-termination (PL^1 is too expressive)
- Idea: Try to travel the middle ground
  more expressive than PL^0 (quantifiers) but weaker than PL^1 (still tractable)
- Technique: Allow only "restricted quantification", where quantified variables only range over values that can be reached via a binary relation like $\mathit{has\_child}$.

More $\mathcal{ALC}$ Examples

- $\mathsf{car} \sqcap \exists\, \mathsf{has\_part}.(\exists\, \mathsf{made\_in}.\overline{\mathsf{EU}})$ (cars that have at least one part that has not been made in the EU)
- $\mathsf{student} \sqcap \forall\, \mathsf{audits\_course}.\mathsf{graduatelevelcourse}$ (students, that only audit

graduate level courses)

▷ house $\sqcap \forall$ has_parking.off_street (houses with off-street parking)

▷ Note: $p \sqsubseteq q$ can still be used as an abbreviation for $\overline{p} \sqcup q$.

▷ student $\sqcap \forall$ audits_course.($\exists$ hastutorial.$\top \sqsubseteq \forall$ has_TA.woman) (students that only audit courses that either have no tutorial or tutorials that are TAed by women)

As before we allow concept definitions so that we can express new concepts from old ones, and obtain more concise descriptions.

$\mathcal{ALC}$ Concept Definitions

▷ Define new concepts from known ones: $(KD_{\mathcal{ALC}} :== C = F_{\mathcal{ALC}})$

Definition	rec?
man = person $\sqcap \exists$ has_chrom.Y_chrom	-
woman = person $\sqcap \forall$ has_chrom.$\overline{\text{Y\_chrom}}$	-
mother = woman $\sqcap \exists$ has_child.person	-
father = man $\sqcap \exists$ has_child.person	-
grandparent = person $\sqcap \exists$ has_child.(mother $\sqcup$ father)	-
german = person $\sqcap \exists$ has_parents.german	+
number_list = empty_list $\sqcup \exists$ is_first.number $\sqcap \exists$ is_rest.number_list	+

As before, we can be normalize a TBox by definition exapansion – if it is acyclic. With the introduction of roles and quantification, concept definitions in $\mathcal{ALC}$ have a more "interesting" way to be cyclic as Observation 23.1.4 shows.

TBox Normalization in $\mathcal{ALC}$

▷ **Example 23.1.2 (Normalizing grandparent)**

grandparent
$\mapsto$ person $\sqcap \exists$ has_child.(mother $\sqcup$ father)
$\mapsto$ person $\sqcap \exists$ has_child.(woman $\sqcap \exists$ has_child.person), man, ($\exists$ has_child.person)
$\mapsto$ person $\sqcap \exists$ has_child.(person $\sqcap \exists$ has_chrom.Y_chrom $\sqcap \exists$ has_child.person $\sqcap$ person $\sqcap \exists$ has_chrom.Y_chrom $\sqcap \exists$ has_child.person)

▷ **Observation 23.1.3** *Normalization result can be exponential* (*contain redundancies*)

▷ **Observation 23.1.4** *Normalization need not terminate on cyclic TBoxes.*

german $\mapsto$ person $\sqcap \exists$ has_parents.german
$\mapsto$ person $\sqcap \exists$ has_parents.(person $\sqcap \exists$ has_parents.german)
$\mapsto$ ...

Concept Axioms

- **Definition 23.1.5** DL formulae that are not concept definitions are called concept axioms.
- They normally contain additional information about concepts
- **Example 23.1.6**
  - $\overline{\mathsf{person} \sqcap \mathsf{car}}$ (persons and cars are disjoint)
  - $\mathsf{car} \sqsubseteq \mathsf{motor\_vehicle}$ (cars are motor vehicles)
  - $\mathsf{motor\_vehicle} \sqsubseteq (\mathsf{car} \sqcup \mathsf{truck} \sqcup \mathsf{motorcycle})$ (motor vehicles are cars, trucks, or motorcycles)

Now that we have motivated and fixed the syntax of $\mathcal{ALC}$, we will give it a formal semantics. The semantics of $\mathcal{ALC}$ is an extension of the set-theoretic semantics for PL^0, thus the interpretation $[\![\cdot]\!]$ assigns subsets of the domain to concepts and binary relations over the domain to roles.

Semantics of $\mathcal{ALC}$

- $\mathcal{ALC}$ semantics is an extension of the set-semantics of propositional logic.
- **Definition 23.1.7** A model for $\mathcal{ALC}$ is a pair $\langle\mathcal{D}, [\![\cdot]\!]\rangle$, where $\mathcal{D}$ is a non-empty set called the domain and $[\![\cdot]\!]$ a mapping called the interpretation, such that

Op.	formula semantics
	$[\![c]\!] \subseteq \mathcal{D} = [\![\top]\!] \quad [\![\bot]\!] = \emptyset \quad [\![r]\!] \subseteq \mathcal{D} \times \mathcal{D}$
$\overline{\cdot}$	$[\![\overline{\varphi}]\!] = \overline{[\![\varphi]\!]} = \mathcal{D} \backslash [\![\varphi]\!]$
$\sqcap$	$[\![\varphi \sqcap \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$
$\sqcup$	$[\![\varphi \sqcup \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$
$\exists \mathsf{R}.$	$[\![\exists \mathsf{R}.\varphi]\!] = \{x \in \mathcal{D} \mid \exists y.\langle x,y\rangle \in [\![\mathsf{R}]\!] \text{ and } y \in [\![\varphi]\!]\}$
$\forall \mathsf{R}.$	$[\![\forall \mathsf{R}.\varphi]\!] = \{x \in \mathcal{D} \mid \forall y.\text{if } \langle x,y\rangle \in [\![\mathsf{R}]\!] \text{ then } y \in [\![\varphi]\!]\}$

- Alternatively we can define the semantics of $\mathcal{ALC}$ by translation into PL^1.
- **Definition 23.1.8** The translation of $\mathcal{ALC}$ into PL^1 extends the one from Definition 22.1.4 by the following quantifier rules:

$$\overline{\forall \mathsf{R}.\varphi}^{fo(x)} := (\forall y.\mathsf{R}(x,y) \Rightarrow \overline{\varphi}^{fo(y)}) \quad \overline{\exists \mathsf{R}.\varphi}^{fo(x)} := (\exists y.\mathsf{R}(x,y) \wedge \overline{\varphi}^{fo(y)})$$

- **Observation 23.1.9** *The set-theoretic semantics from Definition 23.1.7 and the "sematnics-by-translation" from Definition 23.1.8 induce the same notion of satisfiability.*

The following equivalences will be useful later on. They can be proven directly with the settings from Definition 23.1.7; we carry this out for one of them below.

$\mathcal{ALC}$ Identities

1	$\overline{\exists\mathsf{R}.\varphi} = \forall\mathsf{R}.\overline{\varphi}$	3	$\overline{\forall\mathsf{R}.\varphi} = \exists\mathsf{R}.\overline{\varphi}$
2	$\forall\mathsf{R}.(\varphi\sqcap\psi) = \forall\mathsf{R}.(\varphi\sqcap\forall\mathsf{R}.\psi)$	4	$\exists\mathsf{R}.(\varphi\sqcup\psi) = \exists\mathsf{R}.(\varphi\sqcup\exists\mathsf{R}.\psi)$

▷ Proof of 1

$$\begin{aligned}
[\![\overline{\exists\mathsf{R}.\varphi}]\!] = \mathcal{D}\backslash[\![\exists\mathsf{R}.\varphi]\!] &= \mathcal{D}\backslash\{x\in\mathcal{D}\,|\,\exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ and } (y\in[\![\varphi]\!])\} \\
&= \{x\in\mathcal{D}\,|\,\text{not }\exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ and } (y\in[\![\varphi]\!])\} \\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\notin[\![\varphi]\!])\} \\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\in(\mathcal{D}\backslash[\![\varphi]\!]))\} \\
&= \{x\in\mathcal{D}\,|\,\forall y.\text{if }(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\in[\![\overline{\varphi}]\!])\} \\
&= [\![\forall\mathsf{R}.\overline{\varphi}]\!]
\end{aligned}$$

The form of the identities (interchanging quantification with propositional connectives) is reminiscient of identities in PL^1; this is no coincidence as the "semantics by translation" of Definition 23.1.8 shows.

We can now use the $\mathcal{ALC}$ identities above to establish a useful normal form for $\mathcal{ALC}$. This will play a role in the inference procedures we study next.

Negation Normal Form

▷ **Definition 23.1.10 (NNF)** $\bar{\cdot}$ directly in front of concept names in $\mathcal{ALC}$ formulae

▷ Use the $\mathcal{ALC}$ Identities as rules to compute it. (in linear time)

example	by rule
$\overline{\exists\mathsf{R}.(\forall\mathsf{S}.e\sqcap\overline{\forall\mathsf{S}.d})}$	
$\mapsto \forall\mathsf{R}.\overline{\forall\mathsf{S}.e\sqcap\overline{\forall\mathsf{S}.d}}$	$\overline{\exists\mathsf{R}.\varphi}\mapsto\forall\mathsf{R}.\overline{\varphi}$
$\mapsto \forall\mathsf{R}.(\overline{\forall\mathsf{S}.e}\sqcup\overline{\overline{\forall\mathsf{S}.d}})$	$\overline{\varphi\sqcap\psi}\mapsto\overline{\varphi}\sqcup\overline{\psi}$
$\mapsto \forall\mathsf{R}.(\exists\mathsf{S}.\overline{e}\sqcup\overline{\overline{\forall\mathsf{S}.d}})$	$\overline{\forall\mathsf{R}.\varphi}\mapsto\forall\mathsf{R}.\overline{\varphi}$
$\mapsto \forall\mathsf{R}.(\exists\mathsf{S}.\overline{e}\sqcup\forall\mathsf{S}.d)$	$\overline{\overline{\varphi}}\mapsto\varphi$

Finally, we extend $\mathcal{ALC}$ with an ABox component. This mainly means that we define two new assertions in $\mathcal{ALC}$ and specify their semantics and PL^1 translation.

$\mathcal{ALC}$ with Assertions about Individuals

▷ **Definition 23.1.11 (ABox Formulae)** We define the ABox formulae for $\mathcal{ALC}$

- ▹ $(a\colon\varphi)$ (a is a φ)
- ▹ $a\mathsf{R}b$ (a stands in relation R to b)

▷ **Definition 23.1.12** Let $\langle\mathcal{D},[\![\cdot]\!]\rangle$ be a model for $\mathcal{ALC}$, then we define

- $[\![(a\colon \varphi)]\!] = \mathsf{T}$, iff $[\![a]\!] \in [\![\varphi]\!]$, and
- $[\![a\mathsf{R}b]\!] = \mathsf{T}$, iff $([\![a]\!], [\![b]\!]) \in [\![\mathsf{R}]\!]$.

▷ **Definition 23.1.13** We extend the PL^1 translation of $\mathcal{ALC}$ to ABox formulae by

- $\overline{(a\colon \varphi)}^{fo} := \overline{\varphi}^{fo(a)}$, and
- $\overline{a\mathsf{R}b}^{fo} := \mathsf{R}(a, b)$.

If we take stock of what we have developed so far, then we see that $\mathcal{ALC}$ as a rational reconstruction of semantic networks restricted to the "isa" and "instance" relations – which are the only ones that can really be given a denotational and operational semantics.

23.2 Inference for $\mathcal{ALC}$

In this Section we make good on the motivation from Section 22.1 that description logics enjoy tractable inference procedures: We present a tableau calculus for $\mathcal{ALC}$, show that is is a decision procedures, and study its complexity.

$\mathcal{T}_{\mathcal{ALC}}$: A Tableau-Calculus for $\mathcal{ALC}$

▷ Recap Tableaux: A tableau calculus develops an initial tableau in a tree-formed scheme using tableau extension rules.

A saturated tableau (no rules applicable) constitutes a proof, if all branches are closed (end in $*$).

▷ **Definition 23.2.1** The tableau calculus $\mathcal{T}_{\mathcal{ALC}}$ acts on ABox assertions

- $(x\colon \varphi)$: (x inhabits concept φ)
- $x\mathsf{R}y$: (x and y are in relation R)

with the following rules rules:

$$\frac{\begin{array}{c}(x\colon c)\\(x\colon \overline{c})\end{array}}{*}\mathcal{T}_* \qquad \frac{(x\colon \varphi\sqcap\psi)}{\begin{array}{c}(x\colon \varphi)\\(x\colon \psi)\end{array}}\mathcal{T}_\sqcap \qquad \frac{(x\colon \varphi\sqcup\psi)}{(x\colon \varphi)\mid(x\colon \psi)}\mathcal{T}_\sqcup \qquad \frac{\begin{array}{c}(x\colon \forall\mathsf{R}.\varphi)\\x\mathsf{R}y\end{array}}{(y\colon \varphi)}\mathcal{T}_\forall \qquad \frac{(x\colon \exists\mathsf{R}.\varphi)}{\begin{array}{c}x\mathsf{R}y\\(y\colon \varphi)\end{array}}\mathcal{T}_\exists$$

▷ To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $(x\colon \psi)$, saturate. Open branches $\leadsto$ consistent. (x arbitrary)

In contrast to the tableau calculi for theorem proving we have studied earlier, $\mathcal{T}_{\mathcal{ALC}}$ is run in "model generation mode". Instead of initializing the tableau with the axioms and the negated conjecture and hope that all branches will close, we initialize the $\mathcal{T}_{\mathcal{ALC}}$ tableau with axioms and the "conjecture" that a given concept φ is satisfiable – i.e. φ has a member x, and hope for branches that are open, i.e. that make the conjecture true (and at the same time give a model).

Let us now work through two very simple examples; one unsatisfiable, and a satisfiable one.

$\mathcal{T}_{\mathcal{ALC}}$ Examples

▷ **Example 23.2.2** We have two similar conjectures about children.

▷ $(x\colon \forall \text{has\_child.man} \sqcap \exists \text{has\_child}.\overline{\text{man}})$ (all sons, but a daughter)

▷ $(x\colon \forall \text{has\_child.man} \sqcap \exists \text{has\_child.man})$ (only sons, and at least one)

▷ Tableau Proof

1	$(x\colon \forall \text{has\_child.man} \sqcap \exists \text{has\_child}.\overline{\text{man}})$	initial	$(x\colon \forall \text{has\_child.man} \sqcap \exists \text{has\_child.man})$	initial
2	$(x\colon \forall \text{has\_child.man})$	$\mathcal{T}_{\sqcap}$	$(x\colon \forall \text{has\_child.man})$	$\mathcal{T}_{\sqcap}$
3	$(x\colon \exists \text{has\_child}.\overline{\text{man}})$	$\mathcal{T}_{\sqcap}$	$(x\colon \exists \text{has\_child.man})$	$\mathcal{T}_{\sqcap}$
4	x has_child y	$\mathcal{T}_{\exists}$	x has_child y	$\mathcal{T}_{\exists}$
5	$(y\colon \overline{\text{man}})$	$\mathcal{T}_{\exists}$	$(y\colon \text{man})$	$\mathcal{T}_{\exists}$
6	$(y\colon \text{man})$	$\mathcal{T}_{\forall}$	open	
7	$*$	$\mathcal{T}_{*}$		
	inkonsistent			

The right tableau has a model: there are two persons, x and y. y is the only child of x, y is a man

Another example: this one is more complex, but the concept is satisfiable.

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

▷ **Example 23.2.3**
$\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$ is satisfiable.

▷ Let's try it on the board

▷ Tableau proof for the notes

1	$(x\colon \forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}})$		
2	$(x\colon \forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}))$		$\mathcal{T}_{\sqcap}$
3	$(x\colon \exists \text{has\_child}.\overline{\text{ugrad}})$		$\mathcal{T}_{\sqcap}$
4	x has_child y		$\mathcal{T}_{\exists}$
5	$(y\colon \overline{\text{ugrad}})$		$\mathcal{T}_{\exists}$
6	$(y\colon \text{ugrad} \sqcup \text{grad})$		$\mathcal{T}_{\forall}$
7	$(y\colon \text{ugrad})$	$(y\colon \text{grad})$	$\mathcal{T}_{\sqcup}$
8	$*$	offen	

The left branch is closed, the right one represents a model: y is a child of x, y is a graduate student, x hat exactly one child: y.

After we got an intution about $\mathcal{T}_{\mathcal{ALC}}$, we can now study the properties of the calculus to determine that it is a decision procedure for $\mathcal{ALC}$.

Properties of Tableau Calculi

- We study the following properties of a tableau calculus $\mathcal{C}$:

  Termination there are no infinite sequences of rule applications.

  Correctness If φ is satisfiable, then $\mathcal{C}$ terminates with an open branch.

  Completeness If φ is in unsatisfiable, then $\mathcal{C}$ terminates and all branches are closed.

  Complexity of the algorithm

- Additionally, we are interested in the complexity of the satisfiability itself (as a benchmark)

The correctness result for $\mathcal{T}_{\mathcal{ALC}}$ is as usual: we start with a model of $(x\colon\varphi)$ and show that an $\mathcal{T}_{\mathcal{ALC}}$ tableau must have an open branch.

Correctness

- **Lemma 23.2.4** *If φ satisfiable, then $\mathcal{T}_{\mathcal{ALC}}$ terminates on $(x\colon\varphi)$ with open branch.*

- Proof: Let $\mathcal{M} := \langle\mathcal{D},\mathcal{I}\rangle$ be a model for φ and $w \in [\![\varphi]\!]$.

  P.1 we define $[\![x]\!] := w$ and
  $$\begin{array}{lll} \mathcal{I}\models(x\colon\psi) & \text{iff} & [\![x]\!]\in[\![\psi]\!] \\ \mathcal{I}\models x\mathsf{R}y & \text{iff} & \langle x,y\rangle\in[\![\mathsf{R}]\!] \\ \mathcal{I}\models S & \text{iff} & \mathcal{I}\models c \text{ for all } c\in S \end{array}$$

  P.2 This gives us $\mathcal{I}\models(x\colon\varphi)$ (base case)

  P.3 case analysis: if branch satisfiable, then either
  - no rule applicable to leaf (open branch)
  - or rule applicable and one new branch satisfiable (inductive case)

  P.4 consequence: there must be an open branch (by termination)

  □

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

Case analysis on the rules

$\mathcal{T}_{\sqcap}$ **applies** then $\mathcal{I}\models(x\colon\varphi\sqcap\psi)$, i.e. $[\![x]\!]\in[\![(\varphi\sqcap\psi)]\!]$
so $[\![x]\!]\in[\![\varphi]\!]$ and $[\![x]\!]\in[\![\psi]\!]$, thus $\mathcal{I}\models(x\colon\varphi)$ and $\mathcal{I}\models(x\colon\psi)$.

$\mathcal{T}_{\sqcup}$ **applies** then $\mathcal{I}\models(x\colon\varphi\sqcup\psi)$, i.e $[\![x]\!]\in[\![(\varphi\sqcup\psi)]\!]$
so $[\![x]\!]\in[\![\varphi]\!]$ or $[\![x]\!]\in[\![\psi]\!]$, thus $\mathcal{I}\models(x\colon\varphi)$ or $\mathcal{I}\models(x\colon\psi)$,
wlog. $\mathcal{I}\models(x\colon\varphi)$.

$\mathcal{T}_{\forall}$ **applies** then $\mathcal{I}\models(x\colon\forall\mathsf{R}.\varphi)$ and $\mathcal{I}\models x\mathsf{R}y$, i.e. $[\![x]\!]\in[\![\forall\mathsf{R}.\varphi]\!]$ and $\langle x,y\rangle\in[\![\mathsf{R}]\!]$, so $[\![y]\!]\in[\![\varphi]\!]$

$\mathcal{T}_\exists$ **applies** then $\mathcal{I} \models (x\colon \exists \mathsf{R}.\varphi)$, i.e $[\![x]\!] \in [\![\exists \mathsf{R}.\varphi]\!]$,
so there is a $v \in D$ with $\langle [\![x]\!], v\rangle \in [\![\mathsf{R}]\!]$ and $v \in [\![\varphi]\!]$.
We define $[\![y]\!] := v$, then $\mathcal{I} \models x\mathsf{R}y$ and $\mathcal{I} \models (y\colon \varphi)$

 232 JACOBS UNIVERSITY

For the completeness result for $\mathcal{T}_{\mathcal{ALC}}$ we have to start with an open tableau branch and construct at model that satisfies all judgements in the branch. We proceed by building a Herbrand model, whose domain consists of all the individuals mentioned in the branch and which interprets all concepts and roles as specified in the branch. Not surprisingly, the model thus constructed satisfies the branch.

Completeness of the Tableau Calculus

▷ **Lemma 23.2.5** *Open saturated tableau branches for φ induce models for φ.*

▷ Proof: construct a model for the branch and verify for φ

P.1 (Model Construction)Let $\mathcal{B}$ be an open saturated branch

▷ we define

$$\begin{aligned} \mathcal{D} &:= \{x \mid (x\colon \psi) \in \mathcal{B} \text{ or } z\mathsf{R}x \in \mathcal{B}\} \\ [\![c]\!] &:= \{x \mid (x\colon c) \in \mathcal{B}\} \\ [\![\mathsf{R}]\!] &:= \{\langle x, y\rangle \mid x\mathsf{R}y \in S_n\} \end{aligned}$$

▷ well-defined since never $x\colon c, x\colon \overline{c} \in \mathcal{B}$ (otherwise $\mathcal{T}_*$ applies)

▷ $\mathcal{I}$ satisfies all constraints $(x\colon c)$, $(x\colon \overline{c})$ and $x\mathsf{R}y$, (by construction)

P.2 (Induction)$\mathcal{I} \models (y\colon \psi)$, for all $(y\colon \psi) \in \mathcal{B}$ (on $k = size(\psi)$ next slide)

P.3 (Consequence)$\mathcal{I} \models (x\colon \varphi)$. □

 233 JACOBS UNIVERSITY

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

Case Analysis for Induction

case $(y\colon \psi) = (y\colon \psi_1 \sqcap \psi_2)$ Then $\{y\colon \psi_1, y\colon \psi_2\} \subseteq \mathcal{B}$ ($\mathcal{T}_\sqcap$-rule, saturation)
so $\mathcal{I} \models (y\colon \psi_1)$ and $\mathcal{I} \models (y\colon \psi_2)$ and $\mathcal{I} \models (y\colon \psi_1 \sqcap \psi_2)$ (IH, Definition)

case $(y\colon \psi) = (y\colon \psi_1 \sqcup \psi_2)$ Then $(y\colon \psi_1) \in \mathbf{B}$ or $(y\colon \psi_2) \in \mathbf{B}$ ($\mathcal{T}_\sqcup$, saturation)
so $\mathcal{I} \models (y\colon \psi_1)$ or $\mathcal{I} \models (y\colon \psi_2)$ and $\mathcal{I} \models (y\colon \psi_1 \sqcup \psi_2)$ (IH, Definition)

case $(y\colon \psi) = (y\colon \exists \mathsf{R}.\theta)$ then $\{y\mathsf{R}z, z\colon \theta\} \subseteq \mathbf{B}$ (z new variable) ($\mathcal{T}_\exists$-rules, saturation)
so $\mathcal{I} \models (z\colon \theta)$ and $\mathcal{I} \models y\mathsf{R}z$, thus $\mathcal{I} \models (y\colon \exists \mathsf{R}.\theta)$. (IH, Definition)

case $(y\colon \psi) = (y\colon \forall \mathsf{R}.\theta)$ Let $\langle [\![y]\!], v\rangle \in [\![\mathsf{R}]\!]$ for some $r \in \mathcal{D}$
then $v = z$ for some variable z with $y\mathsf{R}z \in \mathbf{B}$ (construction of $[\![\mathsf{R}]\!]$)
So $(z\colon \theta) \in \mathcal{B}$ and $\mathcal{I} \models (z\colon \theta)$. ($\mathcal{T}_\forall$-rule, saturation, Def)
Since v was arbitrary we have $\mathcal{I} \models (y\colon \forall \mathsf{R}.\theta)$.

Termination

▷ **Theorem 23.2.6** $\mathcal{T}_{\mathcal{ALC}}$ *terminates*

▷ To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\frac{\begin{array}{c}(x\colon c)\\(x\colon \overline{c})\end{array}}{*}\mathcal{T}_{*} \qquad \frac{(x\colon \varphi\sqcap\psi)}{\begin{array}{c}(x\colon \varphi)\\(x\colon \psi)\end{array}}\mathcal{T}_{\sqcap} \qquad \frac{(x\colon \varphi\sqcup\psi)}{(x\colon \varphi)\mid(x\colon \psi)}\mathcal{T}_{\sqcup} \qquad \frac{\begin{array}{c}(x\colon \forall \mathsf{R}.\varphi)\\x\mathsf{R}y\end{array}}{(y\colon \varphi)}\mathcal{T}_{\forall} \qquad \frac{(x\colon \exists \mathsf{R}.\varphi)}{\begin{array}{c}x\mathsf{R}y\\(y\colon \varphi)\end{array}}\mathcal{T}_{\exists}$$

▷ Proof: Sketch (full proof very technical)

P.1 any rule except $\mathcal{T}_{\forall}$ can only be applied once to $(x\colon \psi)$.

P.2 rule $\mathcal{T}_{\forall}$ applicable to $(x\colon \forall \mathsf{R}.\psi)$ at most as the number of R-successors of x. (those y with $x\mathsf{R}y$ above)

P.3 the R-successors are generated by $(x\colon \exists \mathsf{R}.\theta)$ above, (number bounded by size of input formula)

P.4 every rule application to $(x\colon \psi)$ generates constraints $(z\colon \psi')$, where ψ' a proper sub-formula of ψ. □

We can turn the termination result into a worst-case complexity result by examining the sizes of branches.

Complexity

▷ Idea: Work of tableau branches one after the other. (Branch size $\hat{=}$ space complexity)

▷ **Observation 23.2.7** *The size of the branches is* polynomial *in the size of the input formula:*

$$\textit{branch size} = |\textit{input formulae}| + \#(\exists\textit{-formulae}) \cdot \#(\forall\textit{-formulae})$$

▷ Proof Sketch: re-examine the termination proof and count: the first summand comes from P, the second one from P and P □

▷ **Theorem 23.2.8** *The satisfiability problem for $\mathcal{ALC}$ is in* PSPACE.

▷ **Theorem 23.2.9** *The satiability problem for $\mathcal{ALC}$ is* PSPACE-*Complete.*

▷ Proof Sketch: reduce a PSPACE-complete problem to $\mathcal{ALC}$-satisfiability □

▷ **Theorem 23.2.10 (Time Complexity)** *The $\mathcal{ALC}$-satisfiability problem is in* EXPTIME

▷ Proof Sketch: There can be exponentially many branches (already for PL^0)□

In summary, the theoretical complexity of $\mathcal{ALC}$ is the same as that for PL^0, but in practice $\mathcal{ALC}$ is much more expressive. So this is a clear win.

But the description of the tableau algorithm $\mathcal{T}_{\mathcal{ALC}}$ is still quite abstract, so we look at an exemplary implementation in a functional programming language

The functional Algorithm for $\mathcal{ALC}$

- ▷ Observation: leads to treatment for $\exists$
  - ▷ the $\mathcal{T}_\exists$-rule generates the constraints $x\mathsf{R}y$ and $(y\colon\psi)$ from $(x\colon\exists\mathsf{R}.\psi)$
  - ▷ this triggers the $\mathcal{T}_\forall$-rule for $(x\colon\forall\mathsf{R}.\theta_i)$, which generate $(y\colon\theta_1),\ldots,(y\colon\theta_n)$
  - ▷ for y we have $(y\colon\psi)$ and $(y\colon\theta_1),\ldots(y\colon\theta_n)$. (do all of this in a single step)
  - ▷ we are only interested in non-emptiness, not in the particular witnesses (leave them out)

```
consistent(S) =
  if {c,c̄} ⊆ S then false
  elseif '(φ ⊓ ψ)' ∈ S and ('φ' ∉ S or 'ψ' ∉ S)
    then consistent(S ∪ {φ,ψ})
  elseif '(φ ⊔ ψ)' ∈ S and {φ,ψ} ∉ S
     then consistent(S ∪ {φ}) or consistent(S ∪ {ψ})
  elseif forall '∃R.ψ' ∈ S
   consistent({ψ} ∪ {θ ∈ θ | '∀R.θ' ∈ S})
  else true
```

- ▷ relatively simple to implement (good implementations optimized)
- ▷ but: this is restricted to $\mathcal{ALC}$. (extension to other DL difficult)

Note that we have (so far) only considered an empty TBox: we have initialized the tableau with a normalized concept; so we did not need to include the concept definitions. To cover "real" ontologies, we need to consider the case of concept axioms as well.

We now extend $\mathcal{T}_{\mathcal{ALC}}$ with concept axioms. The key idea here is to realize that the concept axioms apply to all individuals. As the individuals are generated by the $\mathcal{T}_\exists$ rule, we can simply extend that rule to apply all the concepts axioms to the newly introduced individual.

Extending the Tableau Algorithm by Concept Axioms

- ▷ Concept axioms, e.g. child $\sqsubseteq$ (son $\sqcup$ daughter) cannot be handled in $\mathcal{T}_{\mathcal{ALC}}$ yet.
- ▷ Idea: Whenever a new variable y is introduced (by $\mathcal{T}_\exists$-rule) add the information that axioms hold for y.
  - ▷ initialize tableau with $\{x\colon\varphi\}\cup\mathcal{CA}$ ($\mathcal{CA}\colon=$ set of concept axioms)

▷ new rule for $\exists$: $\frac{(x\colon \exists \mathsf{R}.\varphi) \;\; \mathcal{CA} = \{a_1, \ldots, a_n\}}{\begin{array}{c}(y\colon \varphi)\\ x\mathsf{R}y\\ (y\colon \alpha_1)\\ \vdots\\ (y\colon \alpha_n)\end{array}} \mathcal{T}_{\mathcal{CA}}^{\exists}$ (instead of $\mathcal{T}_{\exists}$)

▷ Problem: $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $(x\colon d)$ (non-termination)

The problem of this approach is that it spoils termination, since we cannot control the number of rule applications by (fixed) properties of the input formulae. The example shows this very nicely. We only sketch a path towards a solution.

Non-Termination of $\mathcal{T}_{\mathcal{ALC}}$ with Concept Axioms

▷ Problem: $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $(x\colon d)$. (non-termination)

$(x\colon d)$	start
$(x\colon \exists \mathsf{R}.c)$	in $\mathcal{CA}$
$x\mathsf{R}y_1$	$\mathcal{T}_{\exists}$
$(y_1\colon c)$	$\mathcal{T}_{\exists}$
$(y_1\colon \exists \mathsf{R}.c)$	$\mathcal{T}_{\mathcal{CA}}^{\exists}$
$y_1\mathsf{R}y_2$	$\mathcal{T}_{\exists}$
$(y_2\colon c)$	$\mathcal{T}_{\exists}$
$(y_2\colon \exists \mathsf{R}.c)$	$\mathcal{T}_{\mathcal{CA}}^{\exists}$
...	

Solution: Loop-Check:

▷ instead of a new variable y take an old variable z, if we can guarantee that whatever holds for y already holds for z.

▷ we can only do this, iff the $\mathcal{T}_{\forall}$-rule has been exhaustively applied.

23.3 ABoxes, Instance Testing, and $\mathcal{ALC}$

Instance Test

▷ **Example 23.3.1** (will explain TBox and ABox with ALC later)

TBox (terminological Box)		
woman	=	person ⊓ has_Y
man	=	person ⊓ has_Y

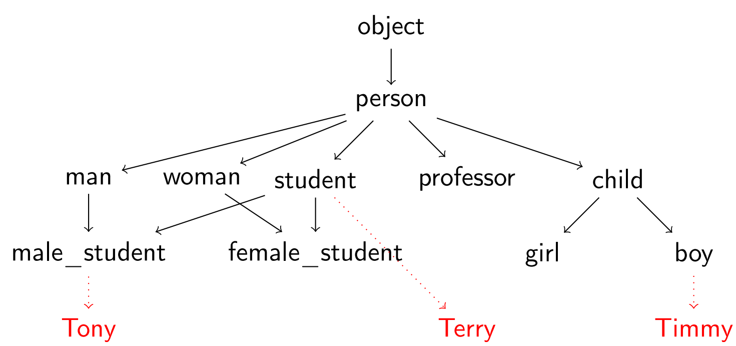
ABox (assertional Box, data base)	
(tony: person)	Tony is a person
(tony: has_Y)	Tony has a y-chromosome

▷ This entails: (tony: man) (Tony is a man).

Realization

▷ **Definition 23.3.2** Realization is the computation of all instance relations between ABox objects and TBox concepts.

▷ sufficient to remember the lowest concepts in the subsumption graph



▷ if (tony: male_student) is known, we do not need (tony: man).

ABox Inference in $\mathcal{ALC}$: Phenomena

▷ There are different kinds of interactions between TBox and ABox in description logics

property	example
internally inconsistent	(tony: student), (tony: $\overline{\text{student}}$)
inconsistent with a TBox	TBox: student $\sqcap$ prof ABox: (tony: student), (tony: prof)
implicit info that is not explicit	Abox: (tony: $\forall$ has_grad.genius) tonyhas_gradmary $\models$ (mary: genius)
info that can be combined with TBox info	TBox: cont_prof $=$ prof $\sqcap \forall$ has_grad.genius ABox: (tony: cont_prof), tonyhas_gradmary $\models$ (mary: genius)

Tableau-based Instance Test and Realization

▷ Query: do the ABox and TBox together entail $(a\colon \varphi)$ $(a \in \varphi?)$

▷ Algorithm: test $(a\colon \overline{\varphi})$ for consistency with ABox and TBox.[9] (use our tableau)

▷ necessary changes: (no big deal)

- ▷ Normalize ABox wrt. TBox (definition expansion)
- ▷ initialize the tableau with ABox in NNF (so it can be used)

Example: add $(\text{mary}\colon \overline{\text{genius}})$ to determine $ABox, TBox \models (\text{mary}\colon \text{genius})$

TBox	cont_prof = prof ⊓ ∀ has_grad.genius	
ABox	(tony: cont_prof) tony has_grad mary	

(tony: prof ⊓ ∀ has_grad.genius)	TBox
tony has_grad mary	ABox
$(\text{mary}\colon \overline{\text{genius}})$	Query
(tony: prof)	$\mathcal{T}_{\sqcap}$
(tony: ∀ has_grad.genius)	$\mathcal{T}_{\sqcap}$
(mary: genius)	$\mathcal{T}_{\forall}$
*	$\mathcal{T}_{*}$

▷ Note: The instance test is the base for the realization (remember?)

▷ extend to more complex ABox queries: (give me all instances of φ)

 ©: Michael Kohlhase 243

[i]EdNote: need to unify abox and tbox judgments.

Chapter 24

Description Logics and the Semantic Web

In this Chapter we discuss how we can apply description logics in the real world, in particular, as a conceptual and algorithmic basis of the "Semantic Web". That tries to transform the "World Wide Web" from a human-understandable web of multimedia documents into a "web of machine-understandable data". In this context, "machine-understandable" means that machines can draw inferences from data they have access to.

Note that the discussion in this digression is not a full-blown introduction to RDF and OWL, we leave that to [SR14, HASB13a, HKP+12] and the respective W3C recommendations. Instead we introduce the ideas behind the mappings from a perspective of the description logics we have discussed above.

The most important component of the "Semantic Web" is a standardized language that can represent "data" about information on the Web in a machine-oriented way.

Resource Description Framework

- ▷ **Definition 24.0.1** The Resource Description Framework (RDF) is a framework for describing resources on the web. It is an XML vocabulary developed by the W3C.
- ▷ Note: RDF is designed to be read and understood by computers, not to be being displayed to people. (it shows)
- ▷ **Example 24.0.2** RDF can be used for describing
  - ▷ properties for shopping items, such as price and availability
  - ▷ time schedules for web events
  - ▷ information about web pages (content, author, created and modified date)
  - ▷ content and rating for web pictures
  - ▷ content for search engines
  - ▷ electronic libraries

 244 JACOBS UNIVERSITY

Note that all these examples have in common that they are about "objects on the Web", which is an aspect we will come to now.

"Objects on the Web" are traditionally called "resources", rather than defining them by their intrinsic properties – which would be ambitious and prone to change – we take an external property to define them: everything that has a URI is a web resource. This has repercussions on the design or RDF.

Resources and URIs

- ▷ RDF describes resources with properties and property values.
- ▷ RDF uses Web identifiers (URIs) to identify resources.
- ▷ **Definition 24.0.3** A resource is anything that can have a URI, such as `http://www.jacobs-university.de`
- ▷ **Definition 24.0.4** A property is a resource that has a name, such as *author* or *homepage*, and a property value is the value of a property, such as *Michael Kohlhase* or `http://kwarc.info/kohlhase` (a property value can be another resource)
- ▷ **Definition 24.0.5** The combination of a resource, a property, and a property value forms a statement (known as the subject, predicate and object of a statement).x
- ▷ **Example 24.0.6** Statement: *[This slide]^{subj} has been [author]^{pred}ed by [Michael Kohlhase]^{obj}*

 245 JACOBS UNIVERSITY

The crucial observation here is that if we map "subjects" and "objects" to "individuals", and "predicates" to "relations", the RDF statements are just relational ABox statements of description logics. As a consequence, the techniques we developed apply.

We now come to the concrete syntax of RDF. This is a relatively conventional XML syntax that combines RDF statements with a common subject into a single "description" of that resource.

XML Syntax for RDF

- ▷ RDF is a concrete XML vocabulary for writing statements
- ▷ **Example 24.0.7** The following RDF document could describe the slides as a resource

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </rdf:Description>
</rdf:RDF>
```

This RDF document makes two statements:

- ▷ The subject of both is given in the about attribute of the `rdf:Description` element
- ▷ The predicates are given by the element names of its children

▷ The objects are given in the elements as URIs or literal content.

Intuitively: RDF is a web-scalable way to write down ABox information.

Note that XML namespaces play a crucial role in using element to encode the predicate URIs. Recall that an element name is a qualified name that consists of a namespace URI and a proper element name (without a colon character). Concatenating them gives a URI in our example the predicate URI induced by the `dc:creator` element is `http://purl.org/dc/elements/1.1/creator`. Note that as URIs go RDF URIs do not have to be URLs, but this one is and it references (is redirected to) the relevant part of the Dublin Core elements specification [DCM12].

RDF was deliberately designed as a standoff markup format, where URIs are used to annotate web resources by pointing to them, so that it can be used to give information about web resources without having to change them. But this also creates maintenance problems, since web resources may change or be deleted without warning.

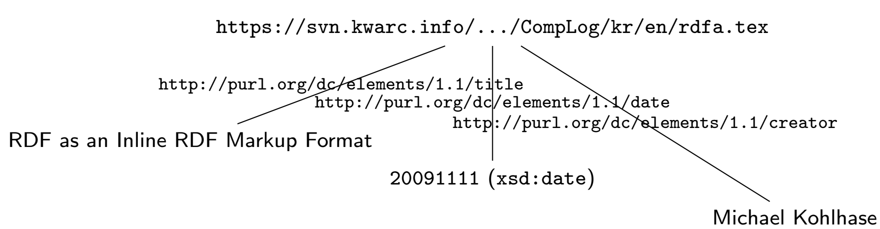
RDFa gives authors a way to embed RDF triples into web resources and make keeping RDF statements about them more in sync.

▷ RDFa as an Inline RDF Markup Format

▷ Problem: RDF is a standoff markup format (annotate by URIs pointing into other files)

▷ **Example 24.0.8**

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/" id="address">
  <h2 about="#address" property="dc:title">RDF as an Inline RDF Markup Format</h2>
  <h3 about="#address" property="dc:creator">Michael Kohlhase</h3>
  <em about="#address" property="dc:date" datatype="xsd:date"
    content="20091111">November 11., 2009</em>
</div>
```



In the example above, the `about` and `property` attribute are reserved by RDFa and specify the subject and predicate of the RDF statement. The object consists of the body of the element, unless otherwise specified e.g. by the `resource` attribute.

Let us now come back to the fact that RDF is just an XML syntax for ABox statements.

RDF as an ABox Language for the Semantic Web

▷ Idea: RDF triples are ABox entries $h\text{R}s$ or $h:\varphi$.

▷ **Example 24.0.9** h is the resource for Ian Horrocks, s is the resource for Ulrike Sattler, R is the the relation "hasColleague", and φ is the class `foaf:Person`

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

Idea: Now, we need an similar language for TBoxes (based on $\mathcal{ALC}$)

In this situation, we want a standardized representation language for TBox information; OWL does just that: it standardizes a set of knowledge representation primitives and specifies a variety of concrete syntaxes for them. OWL is designed to be compatible with RDF, so that the two together can form an ontology language for the web.

▷ OWL as an Ontology Language for the Semantic Web

- ▷ Task: Complement RDF (ABox) with a TBox language
- ▷ Idea: Make use of resources that are values in `rdf:type` (called Classes)
- ▷ **Definition 24.0.10** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- ▷ **Example 24.0.11 (A concept definition for "Mother")** Mother = Woman ⊓ Parent is represented as

XML Syntax	Functional Syntax

`<EquivalentClasses>`
`  <Class IRI="Mother"/>`
`  <ObjectIntersectionOf>`
`    <Class IRI="Woman"/>`
`    <Class IRI="Parent"/>`
`  </ObjectIntersectionOf>`
`</EquivalentClasses>`
	`EquivalentClasses(`
	`  :Mother`
	`  ObjectIntersectionOf(`
	`    :Woman`
	`    :Parent`
	`  )`
	`)`

We have introduced the ideas behind using description logics as the basis of a "machine-oriented web of data". While the first OWL specification (2004) nhad three sublanguages "OWL Lite", "OWL DL" and "OWL Full", of which only the middle was based on description logics, with the OWL2 Recommendation from 2009, the foundation in description logics was nearly universally accepted.

The Semantic Web hype is by now nearly over, the technology has reached the "plateau of productivity" with many applications being pursued in academia and industry. We will not go into these, but briefly instroduce one of the tools that make this work.

SPARQL a RDF Query language

▷ **Definition 24.0.12** A database that stores RDF data is called a triple store

▷ **Definition 24.0.13** SPARQL, the " SPARQL Protocol and RDF Query Language" is an RDF query language, able to retrieve and manipulate data stored in RDF. The SPARQL language was standardize by the World Wide Web Consortium in 2008 [PS08].

▷ SPARQL is pronounced like the word "sparkle".

▷ **Definition 24.0.14** A triple store is called a SPARQL endpoint, iff it answers SPARQL queries.

▷ **Example 24.0.15**
Query for person names and their e-mails from a triple store with FOAF data.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```


SPARQL end-points can be used to build interesting applications, if fed with the appropriate data. An interesting – and by now paradigmatic – example is the DBPedia project.

SPARQL Applications: DBPedia

▷ Typical Application: DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triple store.

▷ **Example 24.0.16 (DBPedia Query)**
People who were born in Berlin before 1900 (`http://dbpedia.org/sparql`)

```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth ?death ?person WHERE {
     ?person dbo:birthPlace :Berlin .
     ?person dbo:birthDate ?birth .
     ?person foaf:name ?name .
     ?person dbo:deathDate ?death .
     FILTER (?birth < "1900-01-01"^^xsd:date) .
}
ORDER BY ?name
```

Chapter 25

ALC Extensions

Language Extensions

- $\mathcal{ALC}$ is much more expressive than propositional logic, (still not enough)
- Idea: study more expressive extensions
- Need to study:
  - which new operators? (are some definable)
  - translation into predicate logic
  - are the inference problems decidable? (consistency, subsumption, instance test,...)
  - what is the complexity of the decision problem?
  - what do the algorithms look like?

Description Logic Naming Scheme

- Idea: Use the name of a description logic to show its expressive power(letters express constructors)
- **Definition 25.0.1** title=DL Naming Conventions Use $\mathcal{S}$ for $\mathcal{ALC}$ with transtive roles (the basic DL) (instead of $\mathcal{ALC}(R^+)$)
  - The letter $\mathcal{H}$ represents subroles (role Hierarchies),
  - $\mathcal{O}$ represents nominals (nOminals),
  - $\mathcal{I}$ represents inverse roles (Iinverse),
  - $\mathcal{N}$ represent number restrictions (Number), and
  - $\mathcal{Q}$ represent qualified number restrictions (Qualified).

The integration of a concrete domain/datatype is indicated by appending its name in parenthesis, but sometimes a âĂIJgenericâĂİ D is used to express that some concrete domain/datatype has been integrated.

▷ **Example 25.0.2** The DL corresponding to the OWL DL ontology language includes all of these constructors and is therefore called $\mathcal{SHOIN}(D)$.

25.1 Functional Roles and Number Restrictions

Functional Roles

▷ **Example 25.1.1** CSR$\widehat{=}$ Car with glass sun roof

- ▹ In $\mathcal{ALC}$: CSR $=$ car $\sqcap \exists$ has_sun_roof.glass
- ▹ potential unwanted interpretation: more than one sun roof.
- ▹ Problem: has_sun_roof is a relation in $\mathcal{ALC}$ (no partial function)

▷ **Example 25.1.2** Humans have exactly one father and mother.

- ▹ in $\mathcal{ALC}$: human $\sqsubseteq \exists$ has_father.human $\sqcap \exists$ has_mother.human
- ▹ Problem: has_father should be a total function (on the set of humans)

Solution: Number Restrictions (see next slide)

▷ **Example 25.1.3** Teenager $=$ human between 13 and 19

- ▹ teenager $=$ human $\sqcap$ $(\text{age} < 20)\text{age} > 12$ (not covered by $\mathcal{ALC}$)
- ▹ Solution: concrete domains (outside the scope of this course)

Number Restrictions

▷ **Example 25.1.4** Car $=$ vehicle with at least four wheels

▷ Trick: In $\mathcal{ALC}$: model car using two new distinguishing concepts p_1 and p_2
vehicle $\sqcap \exists$ has_wheel.$(p_1 \sqcap p_2) \sqcap \exists$ has_wheel.$(\overline{p_1} \sqcap p_2) \sqcap \exists$ has_wheel.$(p_1 \sqcap \overline{p_2}) \sqcap \exists$ has_wheel.$(\overline{p_1} \sqcap \overline{p_2})$

▷ Problem: city $=$ town with at least 1,000,000 inhabitants (oh boy)

▷ Alternative: Operators for number restrictions.

(Unqualified) Number Restrictions

▷ **Definition 25.1.5** $\mathcal{ALCN}$ is $\mathcal{ALC}$ plus operators $\exists^n_\geq$R and $\forall^n_\leq$R (R role, $n \in \mathbb{N}$)

▷ **Example 25.1.6**

$$\mathsf{car} = \mathsf{vehicle} \sqcap \exists^{4}_{\geq} \mathsf{has\_wheel} \quad (25.1)$$

$$\mathsf{city} = \mathsf{town} \sqcap \exists^{1,000,000}_{\geq} \mathsf{has\_inhabitants} \quad (25.2)$$

$$\mathsf{small\_family} = \mathsf{family} \sqcap \forall^{2}_{\leq} \mathsf{has\_child} \quad (25.3)$$

▷

$$[\![\exists^{n}_{\geq} \mathsf{R}]\!] = \{x \in \mathcal{D} \,|\, \#(\{y \,|\, \langle x, y\rangle \in [\![\mathsf{R}]\!]\}) \geq n\} \quad (25.4)$$

$$[\![\forall^{n}_{\leq} \mathsf{R}]\!] = \{x \in \mathcal{D} \,|\, \#(\{y \,|\, \langle x, y\rangle \in [\![\mathsf{R}]\!]\}) \leq n\} \quad (25.5)$$

▷ Intuitively: $\exists^{n}_{\geq}\mathsf{R}$ is the set of objects that have at least n R-successors.

▷ **Example 25.1.7** $\exists^{1,000,000}_{\geq}\mathsf{has\_inhabitants}$ is the set of objects that have at least 1,000,000 inhabitants.

Translation into Predicate Logic

▷ Two extra rules for number restrictions: (very cumbersome)

$\overline{\exists^{n}_{\geq}\mathsf{R}}^{fo(x)}$	$\overline{\forall^{n}_{\leq}\mathsf{R}}^{fo(x)}$
$\exists y_1 . \mathsf{R}(x,y_1) \wedge \ldots \wedge \exists y_n . \mathsf{R}(x,y_n)$ $\wedge y_1 \neq y_2 \wedge \ldots \wedge y_1 \neq y_n$ $\wedge\; y_2 \neq y_3 \wedge \ldots \wedge y_2 \neq y_n \wedge$ $y_{n-1} \neq y_n$	$\neg \exists y . \mathsf{R}(x,y) \vee$ $(\exists y_1 . \mathsf{R}(x,y_1) \wedge \ldots \wedge \exists y_n . \mathsf{R}(x,y_n))$ $\forall y . \mathsf{R}(x,y) \Rightarrow (y = y_1 \vee \ldots \vee y = y_n))$

▷ Definable Operator: $\overset{n}{=}\mathsf{R} := \exists^{n}_{\geq}\mathsf{R} \sqcap \forall^{n}_{\leq}\mathsf{R}$
defines the set of objects that have exactly n R-successors.

▷ **Example 25.1.8** $\mathsf{car} = \mathsf{vehicle} \sqcap \overset{n}{=} \mathsf{has\_wheel}$ (vehicles with exactly 4 wheels)

Functional Roles

▷ **Example 25.1.9** $\mathsf{CSR} = \mathsf{car} \sqcap \overset{1}{=} \mathsf{has\_sun\_roof}$ (CSR = car with sun roof)
has_sun_roof is a relation, but restricted to CSR it is a total function.

▷ Partial functions: $\mathsf{Chd} = \mathsf{computer} \sqcap \forall^{1}_{\leq}\mathsf{has\_hd}$ (computer with at most one hard drive) has_hd is a partial function on the set Chd

▷ Intuition: number restrictions can be used to encode partial and total functions, but not to specify the range type.

 258

Negation Rules

▷ Observation: to compute the negation normal form, need the rules for the new operators $\boxed{\overline{\exists^n_{\geq}\mathsf{R}} \mapsto \forall^{n-1}_{\leq}\mathsf{R}}$ $\boxed{\overline{\forall^n_{\leq}\mathsf{R}} \mapsto \exists^{n+1}_{\geq}\mathsf{R}}$

▷ Proof Sketch: by the semantics of the operators □

▷ **Example 25.1.10**
1: $\overline{\exists^5_{\geq}\mathsf{has\_child}} = \forall^4_{\leq}\mathsf{has\_child}$
2: $\overline{\forall^5_{\leq}\mathsf{has\_child}} = \exists^6_{\geq}\mathsf{has\_child}$

 259

Tableaux Rules (without ABox Information)

$$\frac{\begin{array}{c} x\mathsf{R}a_1 \\ \vdots \\ x\mathsf{R}a_{n-k} \\ (x\colon \exists^n_{\geq}\mathsf{R}) \end{array} \quad y_1,\ldots,y_k \ \text{new}}{\begin{array}{c} x\mathsf{R}y_1 \\ \vdots \\ x\mathsf{R}y_k \end{array}} \qquad \frac{\begin{array}{c} x\mathsf{R}a_1 \\ \vdots \\ x\mathsf{R}a_m \\ (x\colon \forall^n_{\leq}\mathsf{R}) \end{array} \quad \begin{array}{c} m>n \\ 1\leq i,j\leq m \end{array}}{[a_j/a_i] \ \text{everywhere}}$$

▷ Basic Intuition (but when do we fail? Can we always identify)

- ▷ $\exists^n_{\geq}\mathsf{R}$: Introduce as many R-successors as necessary
- ▷ $\forall^n_{\leq}\mathsf{R}$: Identify two R-successors if there are too many (repeat as needed)

 260

25.2 Unique Names

Unique Name Assumption

▷ Problem: assuming UNA for ABox constants (but not always)

▷ **Definition 25.2.1 (Unique Name Assumption)** (UNA)
Different names for objects denote different objects, (cannot be equated)

▷ **Example 25.2.2**

(Bob: gardener)
(Bill: gardener)
(1UNAbomber: gardener)

- ▷ Bill and Bob are different
- ▷ but the UNAbomber can be Bill or Bob or someone else.

▷ Assumption: mark every ABox constant with 'UNA' or '$\overline{\mathrm{UNA}}$'

Tableau Rules (with ABox Information)

▷ **Definition 25.2.3** The rules for $\mathcal{ALC}$ with unique name assumption are

$$\frac{\begin{array}{c} x\mathsf{R}a_1 \\ \vdots \\ x\mathsf{R}a_{n-k} \\ (x\colon \exists^n_{\geq}\mathsf{R}) \end{array} \quad \begin{array}{l} y_1,\ldots,y_k\colon \overline{\mathrm{UNA}} \text{ new} \\ a_1,\ldots,a_{n-k}\colon \mathrm{UNA} \end{array}}{\begin{array}{c} x\mathsf{R}y_1 \\ \vdots \\ x\mathsf{R}y_k \end{array}} \qquad \frac{\begin{array}{c} x\mathsf{R}a_1 \vdots \\ x\mathsf{R}a_m \\ (x\colon \forall^n_{\leq}\mathsf{R}) \end{array} \quad \begin{array}{c} m>n \\ 1\leq i,j\leq m \\ a_i\colon \overline{\mathrm{UNA}} \end{array}}{[a_j/a_i] \text{ everywhere}}$$

$$\frac{\begin{array}{c} x\mathsf{R}a_1 \\ \vdots \\ x\mathsf{R}a_m \\ (x\colon \forall^n_{\leq}\mathsf{R}) \end{array} \quad \begin{array}{c} m>n \\ a_1,\ldots,a_m\colon \mathrm{UNA} \end{array}}{*}$$

Example: Solving a Crime with Number Restrictions

▷ **Example 25.2.4** 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.)

Bill: UNA, Bob: UNA, tony: UNA, muderer: $\overline{\mathrm{UNA}}$

$$\begin{array}{c} (\mathsf{tony}\colon \forall^2_{\leq}\mathsf{observes}) \\ \mathsf{tony}\mathsf{observes}\mathsf{Bill} \\ \mathsf{tony}\mathsf{observes}\mathsf{Bob} \\ \mathsf{tony}\mathsf{observes}\mathsf{muderer} \\ (\mathsf{muderer}\colon \mathsf{black\_hair}) \\ (\mathsf{Bill}\colon \overline{\mathsf{black\_hair}}) \\ (\mathsf{Bob}\colon \mathsf{black\_hair}) \\ \begin{array}{c|c} \mathsf{tony}\mathsf{observes}\mathsf{Bill} & \mathsf{tony}\mathsf{observes}\mathsf{Bob} \\ (\mathsf{Bill}\colon \mathsf{black\_hair}) & (\mathsf{Bob}\colon \mathsf{black\_hair}) \\ * & \end{array} \end{array}$$

25.3 Qualified Number Restrictions

Qualified Number Restrictions

- **Definition 25.3.1** $\mathcal{ALCQ}$ is $\mathcal{ALC}$ plus operators $\exists^n_{\geq}\mathsf{R}.\varphi$ and $\forall^n_{\leq}\mathsf{R}.\varphi$ (R role, $n\in\mathbb{N}$, φ formula)
- **Example 25.3.2** $\mathsf{person}\sqcap(\forall^2_{\leq}\mathsf{has\_child}.\mathsf{blond})$ (persons with ≤ 2 blond kids)
- **Example 25.3.3** $\mathsf{comp}\sqcap(\exists^5_{\geq}\mathsf{has\_client}.\mathsf{car\_comp})$ (company with at least 5 clients in the automobile industry)
- Special case: Unqualified Number restrictions ($\exists^n_{\geq}\mathsf{R}.\top$, $\forall^n_{\leq}\mathsf{R}.\top$.)
- $[[\exists^n_{\geq}\mathsf{R}.\varphi)]] = \{x\in\mathcal{D}\,|\,\#(\{y\,|\,\langle x,y\rangle\in[\![\mathsf{R}]\!]\text{ and }y\in[\![\varphi]\!]\})\geq n\}[[\forall^n_{\leq}\mathsf{R}.\varphi)]] = \{x\in\mathcal{D}\,|\,\#(\{y\,|\,\langle x,y\rangle\in[\![\mathsf{R}]\!]\text{ and }y\in[\![$

Negation and Quantifier Elimination

- $\boxed{\overline{\exists^n_{\geq}\mathsf{R}.\varphi}=\forall^{n-1}_{\leq}\mathsf{R}.\varphi \;\; \overline{\forall^n_{\leq}\mathsf{R}.\varphi}=\exists^{n+1}_{\geq}\mathsf{R}.\varphi}$
- **Example 25.3.4** $\overline{\exists^3_{\geq}\mathsf{has\_child}.\mathsf{teacher}}=\forall^2_{\leq}\mathsf{has\_child}.\mathsf{teacher}$
- **Example 25.3.5** $\overline{\forall^3_{\leq}\mathsf{has\_child}.\mathsf{teacher}}=\exists^4_{\geq}\mathsf{has\_child}.\mathsf{teacher}$
- Quantifier elimination (regular quantifiers no longer necessary)
  - $\exists\mathsf{R}.\varphi=\exists^1_{\geq}\mathsf{R}.\varphi$
  - $\forall\mathsf{R}.\varphi=\overline{\exists\mathsf{R}.\overline{\varphi}}=\overline{\exists^1_{\geq}\mathsf{R}.\overline{\varphi}}=\forall^0_{\leq}\mathsf{R}.\overline{\varphi}$

Optimizied Tableau Rules [Tob00]

- **Definition 25.3.6** $\mathcal{T}_{\mathcal{ALC}}$ rules plus:

$$\frac{\begin{array}{c}\mathcal{B}\\(x\colon\exists^n_{\geq}r.\varphi)\end{array}\quad \#(\{y\,|\,x\mathsf{R}y, y\colon\varphi\in\mathcal{B}\})<n\quad y\text{ new}}{\begin{array}{c}x\mathsf{R}y\\(y\colon\varphi)\\(y\colon\xi_1)\\\vdots\\(y\colon\xi_k)\end{array}}$$

where $\{\psi_1,\ldots,\psi_k\}=\{\psi\,|\,(x\colon\exists^m_{\geq}\mathsf{R}.\psi)\in\mathcal{B}\text{ or }(x\colon\forall^m_{\leq}\mathsf{R}.\psi)\in\mathcal{B}\}$ and $\xi_i=\psi$

or $\xi = \overline{\psi}$.

$$\frac{\begin{array}{c}\mathcal{B}\\ (x\colon \forall^{n}_{\leq} r.\varphi)\end{array} \quad \#(\{y \mid x\mathsf{R}y, y\colon \varphi \in \mathcal{B}\}) > n}{*}$$

Example Tableau

▷ **Example 25.3.7**

$$\begin{array}{c}
(x\colon (\exists^{3}_{\geq}\mathsf{R}.\varphi) \sqcap (\forall^{1}_{\leq}\mathsf{R}.\psi) \sqcap (\forall^{1}_{\leq}\mathsf{R}.\overline{\psi}))\\
(x\colon \exists^{3}_{\geq}\mathsf{R}.\varphi)\\
(x\colon \forall^{1}_{\leq}\mathsf{R}.\psi)\\
(x\colon \forall^{1}_{\leq}\mathsf{R}.\overline{\psi})\\
x\mathsf{R}y_1\\
(y_1\colon \varphi)
\end{array}$$

$$\begin{array}{c|c}
\begin{array}{c}
(y_1\colon \psi)\\ x\mathsf{R}y_2\\ (y_2\colon \varphi)\\
\begin{array}{c|c}
\begin{array}{c}(y_2\colon \psi)\\ *\end{array} &
\begin{array}{c}(y_2\colon \overline{\psi})\\ x\mathsf{R}y_3\\ (y_3\colon \varphi)\\
\begin{array}{c|c}\begin{array}{c}(y_3\colon \psi)\\ *\end{array} & \begin{array}{c}(y_3\colon \overline{\psi})\\ *\end{array}\end{array}
\end{array}
\end{array}
\end{array}
&
\begin{array}{c}
(y_1\colon \overline{\psi})\\ x\mathsf{R}y_2\\ (y_2\colon \varphi)\\
\begin{array}{c|c}
\begin{array}{c}(y_2\colon \psi)\\ x\mathsf{R}y_3\\ (y_2\colon \varphi)\\
\begin{array}{c|c}\begin{array}{c}(y_3\colon \psi)\\ *\end{array} & \begin{array}{c}(y_3\colon \overline{\psi})\\ *\end{array}\end{array}
\end{array} &
\begin{array}{c}(y_2\colon \overline{\psi})\\ *\end{array}
\end{array}
\end{array}
\end{array}$$

▷ Problem: Naive Implementation: exponential path lengths

Implementation by “Traces”

▷ Algorithm SAT(φ) = sat($x_0, \{x_0\colon \varphi\}$)

$sat(x, S)$:

allocate counter $\#r^{S}(x,\psi) := 0$ for all roles R and positive or negative subformulae ψ in S.

apply rules $\mathcal{T}_{\sqcap}$ and $\mathcal{T}_{\sqcup}$ as long as possible

If S contains an inconsistency, RETURN $*$.

`while`($\mapsto_{\geq}$ is applicable to x) `do`:

$S_{neu} := \{\mathcal{T}_{\mathcal{ALC}}\mathsf{R}xy, y\colon \varphi, y\colon \xi_1, \ldots y\colon \xi_k\}$
`where`
y is a new variable,
$(x\colon \exists^{n}_{\geq}\mathsf{R}.\varphi)$ triggers rule $\mapsto_{\geq}$,
$\{\psi_1, \ldots, \psi_k\} = \{\psi \mid (x\colon \exists^{m}_{\geq}\mathsf{R}.\psi) \in \mathcal{B}$ or $(x\colon \forall^{m}_{\leq}\mathsf{R}.\psi) \in \mathcal{B}\}$ and
$\xi_i = \psi$ oder $\xi = \neg\psi$.

For each $(y\colon \psi) \in S_{new}$: $\#r^S(x,\psi)+ = 1$ If $(x\colon \forall^m_{\leq}\mathsf{R}.\psi) \in \mathcal{B}$ and $\#r^S(x,\psi) > m$ RETURN $*$

If $sat(y, S_{neu}) = *$ RETURN $*$ od

RETURN "'consistent"'.

 268

Analysis

- Idea: Each R-successor of x triggers a recursive call of sat.
- There may be exponentially many R-successor, but they are treated one-by-one, so their space can be re-used.
- The chains of R-successors are at most as long as the nesting depth of operators (linear)
- **Lemma 25.3.8** *Space consumption is polynomial.*
- **Lemma 25.3.9** *This algorithm is complete.*
- Proof Sketch: The global counters $\#r^S(x,\psi)$ count the R-successors and trigger rule $\mapsto_{\leq}$. □
- **Theorem 25.3.10** *The algorithm is correct, complete and terminating, and* PSPACE *(no worse than $\mathcal{ALC}$)*

 269

25.4 Role Operators

The DL-Zoo: Operator Types

- Operators on role names (construct roles on the fly)
- role hierarchy and role axioms (knowledge about roles)
- nominals (names for domain elements)
- features (partial functions)
- concrete domains (e.g. $\mathbb{N}, \mathbb{Z}, trees$)
- external data structures (for programming)
- epistemic operators (belief,...)
- ...

 270

Role Hierarchies

- ▷ Idea: specification of subset relations among relations.
- ▷ **Example 25.4.1** role hierarchy as a directed graph $\mathcal{R}$

$$\begin{gathered}\mathsf{has\_daughter} \sqsubseteq \mathsf{has\_child}\\ \mathsf{has\_son} \sqsubseteq \mathsf{has\_child}\\ \mathsf{talks\_to} \sqsubseteq \mathsf{communicates\_with}\\ \mathsf{calls} \sqsubseteq \mathsf{communicates\_with}\\ \mathsf{buys} \sqsubseteq \mathsf{obtains}\\ \mathsf{steals} \sqsubseteq \mathsf{obtains}\end{gathered}$$

$\mathcal{ALC}$ with Role hierarchies (without role operators)

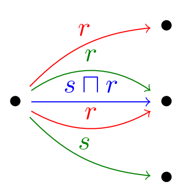
- ▷ **Definition 25.4.2** $\mathcal{T}_{\mathcal{ALC}}$ + complex roles instead of role names

$$\frac{(x\colon \exists \mathsf{R}.\varphi)}{\begin{matrix}x\mathsf{R}y\\ (y\colon \varphi)\end{matrix}}\mathcal{T}_\exists \qquad \frac{\begin{matrix}x\mathsf{S}y\\ (x\colon \forall \mathsf{R}.\varphi)\end{matrix} \quad \mathsf{S}\sqsubseteq\mathsf{R}\in\mathcal{R}}{(y\colon \varphi)}\forall_\sqsubseteq$$

The $\mathcal{T}_\exists$ rule is the same as before

Operators on Roles: Role Conjunction

- ▷ **Example 25.4.3** $\mathsf{person}\sqcap\exists(\mathsf{has\_teacher}\sqcap\mathsf{has\_friend}).\mathsf{swiss}$ (persons that have a Swiss teacher that is also their friend)
- ▷ **Example 25.4.4** $\mathsf{com}\sqcap\exists(\mathsf{has\_employee}\sqcap\mathsf{has\_attorney}).\mathsf{lawyer}$ (companies that have an employed attorney that is a lawyer)
- ▷ $[\![\mathsf{R}\sqcap\mathsf{S}]\!] = [\![\mathsf{R}]\!]\cap[\![\mathsf{S}]\!] = \{\langle x,y\rangle\in\mathcal{D} \mid \langle x,y\rangle\in[\![\mathsf{R}]\!] \text{ and } \langle x,y\rangle\in[\![\mathsf{S}]\!]\}$

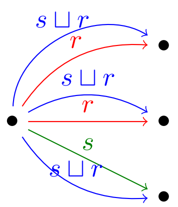


Inference Rules

$$\begin{gathered}\forall \mathsf{R}\sqcap\mathsf{S}.\varphi \sqsubseteq \forall\mathsf{R}.\varphi\sqcap\forall\mathsf{S}.\varphi\\ \exists \mathsf{R}\sqcap\mathsf{S}.\varphi \sqsubseteq \exists\mathsf{R}.\varphi\sqcap\exists\mathsf{S}.\varphi\\ (\exists^n_\geq\mathsf{R}\sqcap\mathsf{S}.\varphi) \sqsubseteq (\exists^n_\geq\mathsf{R}.\varphi)\sqcap(\exists^n_\geq\mathsf{S}.\varphi)\\ (\forall^{n\mp m}_\leq\mathsf{R}\sqcap\mathsf{S}.\varphi) \sqsubseteq (\bar{\exists}^n_\geq\mathsf{R}.\varphi)\sqcap(\bar{\exists}^m_\geq\mathsf{S}.\varphi)\end{gathered}$$

Role Disjunction ⊔

- ▷ **Example 25.4.5** person $\sqcap \forall$ has_child $\sqcup$ has_friend.teacher (persons whose children and friends are all teachers)
- ▷ **Example 25.4.6** com $\sqcap \exists$ has_employee $\sqcup$ has_consultant.member_of_congress (companies with an employee or consultant who is member of congress)
- ▷ $[\![\mathsf{R} \sqcup \mathsf{S}]\!] = [\![\mathsf{R}]\!] \cup [\![\mathsf{S}]\!] = \{\langle x, y\rangle \in \mathcal{D} \mid \langle x, y\rangle \in [\![\mathsf{R}]\!] \quad \text{or} \quad \langle x, y\rangle \in [\![\mathsf{S}]\!]\}$



Inference Rules
$\forall \mathsf{R} \sqcup \mathsf{S}.\varphi = \forall \mathsf{R}.\varphi \sqcup \forall \mathsf{S}.\varphi$
$\exists \mathsf{R} \sqcup \mathsf{S}.\varphi = \exists \mathsf{R}.\varphi \sqcup \exists \mathsf{S}.\varphi$
$\exists^n_\geq \mathsf{R} \sqcup \mathsf{S}.\varphi = ??$
$(\forall^n_\leq \mathsf{R} \sqcup \mathsf{S}.\varphi) \sqsubseteq (\forall^n_\leq \mathsf{R}.\varphi) \sqcap (\forall^n_\leq \mathsf{S}.\varphi)$
$(\forall^{n+m}_\leq \mathsf{R} \sqcup \mathsf{S}.\varphi) \sqsubseteq (\forall^n_\leq \mathsf{R}.\varphi) \sqcap (\forall^m_\leq \mathsf{S}.\varphi)$
$(\exists^{max(n,m)}_\geq \mathsf{R} \sqcup \mathsf{S}.\varphi) \sqsubseteq ((\exists^n_\geq \mathsf{R}.\varphi) \sqcup (\exists^m_\geq \mathsf{S}.\varphi))$

Role Complement $\bar{\cdot}$

- ▷ **Example 25.4.7** univ $\sqcap \forall$ has_employee $\sqcap$ $\overline{\text{has\_prof}}$.unionized (universities whose employees that are not professors are unionized)
- ▷ **Example 25.4.8** house $\sqcap \exists$ resident $\sqcap$ $\overline{\text{owner}}$.swiss (houses whose residents that are not owners are Swiss)
- ▷ $[\![\overline{\mathsf{R}}]\!] = \mathcal{D}^2 \backslash [\![\mathsf{R}]\!] = \{\langle x, y\rangle \in \mathcal{D}^2 \mid \langle x, y\rangle \notin [\![\mathsf{R}]\!]\}$
- ▷ Observation: $\sqcap, \sqcup, \bar{\cdot}$ is a Boolean algebra (propositional logic)
  We can compute with role terms built up from $\sqcap, \sqcup, \bar{\cdot}$ exactly like with propositional formulae built up from $\wedge, \vee, \neg$.
- ▷ **Example 25.4.9** $\forall \overline{\mathsf{R} \sqcap \mathsf{S}}.\varphi = \forall \overline{\mathsf{R}} \sqcup \overline{\mathsf{S}}.\varphi$
- ▷ more rules: if $\mathsf{R} \sqsubseteq \mathsf{S}$ is a tautology, then $\forall \mathsf{S}.\varphi \sqsubseteq \forall \mathsf{R}.\varphi$ and $\exists \mathsf{R}.\varphi \sqsubseteq \exists \mathsf{S}.\varphi$

Special Relations 0 and 1

$\mathsf{R} \sqcap \overline{\mathsf{R}} = 0$	empty relation
$\mathsf{R} \sqcup \overline{\mathsf{R}} = 1$	universal relation

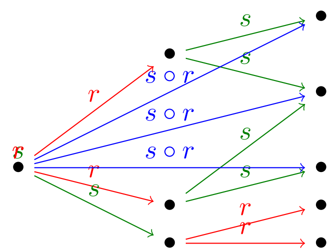
- ▷ Question: what does $\forall 1.\varphi$ mean?

Role composition $\circ$

- ▷ **Example 25.4.10** person $\sqcap \exists$ has_child $\circ$ has_child.prof (persons that have grandchild that is a professor)

▷ **Example 25.4.11** $\mathsf{univ} \sqcap \forall\, \mathsf{has\_student} \circ \mathsf{has\_Partner} \circ \mathsf{lives\_in}).\mathsf{Texas}$(universities whose students all have partners that live in Texas)

▷ $[\![\mathsf{R} \circ \mathsf{S}]\!] = [\![\mathsf{R}]\!][\![\mathsf{S}]\!] = \{\langle x, z\rangle \in \mathcal{D}^2 \mid \exists\, y.(\langle x, y\rangle \in [\![\mathsf{S}]\!]) \text{ and } (\langle y, z\rangle \in [\![\mathsf{R}]\!])\}$



Converse Roles $(\cdot^{-1})$

▷ **Example 25.4.12** (set of objects whose parents are teachers)

$$\begin{aligned} [\![\forall\, \mathsf{has\_child}^{-1}.\mathsf{teacher}]\!] &= \{x \mid \forall y.\langle x, y\rangle \in [\![\mathsf{has\_child}^{-1}]\!] \Rightarrow y \in [\![\mathsf{teacher}]\!]\} \\ &= \{x \mid \forall y.\langle y, x\rangle \in [\![\mathsf{has\_child}]\!] \Rightarrow y \in [\![\mathsf{teacher}]\!]\} \\ &= \{x \mid \forall y.\langle x, y\rangle \in [\![\mathsf{has\_parents}]\!] \Rightarrow y \in [\![\mathsf{teacher}]\!]\} \end{aligned}$$

▷ **Definition 25.4.13** $[\![\mathsf{R}^{-1}]\!] = [\![\mathsf{R}]\!]^{-1} = \{\langle y, x\rangle \in \mathcal{D}^2 \mid \langle x, y\rangle \in [\![\mathsf{R}]\!]\}$

▷ **Example 25.4.14**

$$\begin{aligned} \mathsf{has\_child}^{-1} &= \mathsf{has\_parents} \\ \mathsf{is\_part\_of}^{-1} &= \mathsf{contains\_as\_part} \\ \mathsf{owns}^{-1} &= \mathsf{belongs\_to} \\ &\ldots \end{aligned}$$

Translation of Role Terms

▷ **Definition 25.4.15** Translation Rules:

$\mathsf{tr}(\mathsf{R}) := \mathsf{R}(x, y)$	
$\mathsf{tr}(\mathsf{R} \sqcap \mathsf{S}) := \mathsf{tr}(\mathsf{R}) \wedge \mathsf{tr}(\mathsf{S})$	$\mathsf{tr}(\mathsf{R} \sqcup \mathsf{S}) := \mathsf{tr}(\mathsf{R}) \vee \mathsf{tr}(\mathsf{S})$
$\mathsf{tr}(\mathsf{R} \sqsubseteq \mathsf{S}) := \mathsf{tr}(\mathsf{R}) \Rightarrow \mathsf{tr}(\mathsf{S})$ $\mathsf{tr}(\mathsf{R}^{-1}) := \mathsf{tr}(\mathsf{R})$	$\mathsf{tr}(\mathsf{R} \circ \mathsf{S}) := \exists\, z.\mathsf{tr}(\mathsf{R}), \mathsf{tr}(\mathsf{S})$ $\mathsf{tr}(\overline{\mathsf{R}}) := \neg\, \mathsf{tr}(\mathsf{R})$
$\overline{\forall\, \mathsf{R}.\varphi}^{fo(x)} := \forall\, y.\mathsf{tr}(\mathsf{R}) \Rightarrow \overline{\varphi}^{fo(y)}$	$\overline{\exists\, \mathsf{R}.\varphi}^{fo(x)} := \exists\, y.\mathsf{tr}(\mathsf{R}), \overline{\varphi}^{fo(y)}$

▷ **Example 25.4.16**

$$\overline{\forall \overline{\mathsf{R} \circ \mathsf{S} \sqcap \mathsf{T}^{-1}}.c}^{fo(x)}$$

$$
\begin{aligned}
&= \forall y.\mathsf{tr}(\overline{\mathsf{R} \circ \mathsf{S} \sqcap \mathsf{T}^{-1}}) \Rightarrow \overline{c}^{fo(y)} \\
&= \forall y.\neg\,\mathsf{tr}(\mathsf{R} \circ \mathsf{S} \sqcap \mathsf{T}^{-1}) \Rightarrow c(y) \\
&= \forall y.\neg\,(\exists z.(\mathsf{R}(x \wedge z) \wedge \mathsf{tr}(\mathsf{S} \sqcap \mathsf{T}^{-1}))) \Rightarrow c(y) \\
&= \forall y.\neg\,(\exists z.(\mathsf{R}(x \wedge z) \wedge \mathsf{tr}(\mathsf{S} \sqcap \mathsf{T}))) \Rightarrow c(y) \\
&= \forall y.\neg\,(\exists z.(\mathsf{R}(x \wedge z) \wedge \mathsf{S}(y \wedge z) \wedge \mathsf{T}(y \wedge z))) \Rightarrow c(y)
\end{aligned}
$$

Connection to dynamic Logic

- ▷ Dynamic Logic is used for specification and verification of imperative programs (including non-deterministic, parallel)
- ▷ Similar to $\mathcal{ALC}$ with role terms (role terms as program fragments)
- ▷ Domain of interpretation of a DynL formula is the set of states of the processes ($[\![\forall\,\mathsf{R}.\varphi]\!]$ = "in all states after executing R, φ holds")

$\mathsf{R} \sqcap \mathsf{S}$	parallel execution of R and S
$\mathsf{R} \sqcup \mathsf{S}$	execution of R or S (nondeterministically)
$\mathsf{R} \circ \mathsf{S}$	execution of S after R
$\overline{\mathsf{R}}$	execution of a program that is not R
R^{-1}	execution of an undo operation
$?\psi$	test whether ψ holds (not in $\mathcal{ALC}$)

Tableaux Calculus: $\mathcal{ALC}$ + Role Terms

▷ **Definition 25.4.17** complex roles instead of role names

$$\frac{\begin{array}{c}(x\colon \exists\,\mathsf{R}.\varphi)\\ x\mathsf{R}z\end{array}}{\begin{array}{c}x\mathsf{R}y\\ (y\colon \varphi)\end{array}}\mathcal{T}_{\exists} \qquad \frac{\begin{array}{c}\mathcal{B}\\ (x\colon \forall\,\mathsf{R}.\varphi)\end{array}\quad \mathcal{B} \models x\mathsf{R}y}{(y\colon \varphi)}\forall_{\mathsf{R}}$$

- ▷ Problem: What is $\mathcal{B} \models x\mathsf{R}y$ ($\mathcal{B}$ is the current branch)
- ▷ Simple case: no role composition $\circ$ and no converse roles $\cdot^{-1}$.
  - ▷ then $\mathcal{B} \models x\mathsf{R}y$, iff $\{\mathsf{S} \mid x\mathsf{S}y \in \mathcal{B}\} \cup \{\overline{\mathsf{R}}\}$ inconsistent in PL^0 (decidable)
- ▷ General case: $\mathcal{B} \models x\mathsf{R}y$, iff $\{\mathsf{tr}(\mathsf{S}) \mid u\mathsf{S}v \in \mathcal{B}\} \cup \{\mathsf{tr}(\overline{\mathsf{R}})\}$ inconsistent in PL^1 (undecidable in general)

Special Cases for $\mathcal{B} \models x\mathsf{R}y$

- no role composition $\circ$ (decidable)
  - then $\mathcal{B} \models x\mathsf{R}y$, iff $\{\mathsf{tr}(\mathsf{S}) \mid x\mathsf{S}y \in \mathcal{B}\} \cup \{\mathsf{tr}(\overline{\mathsf{R}})\}$ inconsistent in PL^1 (as set of ground formulae).
- role complement only for role names (decidable)
  - then $\{\mathsf{tr}(\mathsf{S}) \mid u\mathsf{S}v \in \mathcal{B}\}$ is a set of ground formulae and $\mathsf{tr}(\overline{\mathsf{R}})$ only contains constants and variables in the clause normal form.
- The general case is undecidable, therefore the naive tableau approach is unsuitable

25.5 Role Axioms

General Role Axioms

has_daughter $\sqsubseteq$ has_child	daughters are children
has_son $\sqsubseteq$ has_child	sons are children
$\overline{\text{has\_daughter} \sqcap \text{has\_son}}$	sons and daughters are disjoint
has_child $\sqsubseteq$ has_son $\sqcup$ has_daughter	children are either sons or daughters

- Translation of an axiom ρ: $\mathsf{trr}(\rho) = \forall x, y.\mathsf{tr}(\rho)$

$$\begin{aligned} & \mathsf{trr}(\mathsf{has\_child} \sqsubseteq (\mathsf{has\_son} \sqcup \mathsf{has\_daughter})) \\ = \; & \forall x, y.\mathsf{tr}(\mathsf{has\_child} \sqsubseteq \mathsf{has\_son} \sqcup \mathsf{has\_daughter}) \\ = \; & \forall x, y.\mathsf{has\_child}(x \Rightarrow y) \Rightarrow \mathsf{has\_son}(x \vee y) \vee \mathsf{has\_daughter}(x \vee y) \end{aligned}$$

$\mathcal{ALC}$ + Role Terms + Role Axioms ρ

- Idea: Tableau like for $\mathcal{ALC}$ + role terms ($\mathcal{B}, \rho \models x\mathsf{R}y$ instead of $\mathcal{B} \models x\mathsf{R}y$)
- Simple case: no role composition $\circ$ and no converse roles $\cdot^{-1}$. (decidable)
  - then $\mathcal{B}, \rho \models x\mathsf{R}y$ iff $\{\mathsf{S} \mid x\mathsf{S}y \in \mathcal{B}\} \cup \rho \cup \{\overline{\mathsf{R}}\}$ inconsistent in PL^0
- General case: $\mathcal{B}, \rho \models x\mathsf{R}y$, iff $\{\mathsf{tr}(\mathsf{S}) \mid u\mathsf{S}v \in (\mathcal{B} \cup \mathsf{trr}(\rho) \cup \{\mathsf{tr}(\overline{\mathsf{R}})\})\}$ inconsistent in PL^1 (undecidable in

general)

▷ no role composition $\circ$ (decidable)

▹ then $\mathcal{B}, \rho \models x\mathsf{R}y$, iff $\{\mathsf{tr}(\mathsf{S}) \mid x\mathsf{S}y \in (S \cup \mathsf{trr}(\rho) \cup \{\mathsf{tr}(\overline{\mathsf{R}})\})\}$ inconsistent in PL^1 (as set of formulae without functions).

▷ role complement only for role names (decidable)

▹ then $\{\mathsf{tr}(\mathsf{S}) \mid u\mathsf{S}v \in \mathcal{B}\}$ is a set of ground formulae and both $\mathsf{tr}(\rho)$ and $\mathsf{tr}(\overline{\mathsf{R}})$ only contain constants and variables in CNF

25.6 Features

$\mathcal{ALCF}$: Features

▷ Features are partial functions.

▷ Idea: $\mathcal{ALCF}$ is $\mathcal{ALC}$ + features + special constraints on feature paths

▷ **Definition 25.6.1** Let $\mathcal{F} := \{f, g, f_1, \ldots\}$ be a set of features, then we define the $\mathcal{ALCF}$ formulae by
$F_{\mathcal{ALCF}} ::= F_{\mathcal{ALC}} \mid \mathsf{R}.F_{\mathcal{ALCF}} \mid (\pi)\Uparrow \mid \pi = \pi \mid \pi \neq \pi$ where $\pi ::= f \mid f \circ \pi$

▷ **Definition 25.6.2** The semantics of the $\mathcal{ALC}$ part is as always.

1) The meaning of a feature f is a partial function $[\![f]\!]\colon \mathcal{D} \times \mathcal{D} \to \mathcal{D}$.
2) $[\![f \circ \pi]\!](x) := [\![\pi]\!]([\![f]\!](x))$
3) $[\![(\pi)\Uparrow]\!] := \mathcal{D} \backslash \mathbf{dom}([\![\pi]\!])$
4) $[\![f.\varphi]\!] := \{x \in \mathbf{dom}([\![\pi]\!]) \mid [\![f]\!](x) \in [\![\varphi]\!]\}$
5) $[\![\varphi = \omega]\!] := \{x \in \mathbf{dom}([\![\pi]\!]) \cap \mathbf{dom}([\![\omega]\!]) \mid [\![\pi]\!](x) = [\![\omega]\!](x)\}$
6) $[\![\varphi \neq \omega]\!] := \{x \in \mathbf{dom}([\![\pi]\!]) \cap \mathbf{dom}([\![\omega]\!]) \mid [\![\pi]\!](x) \neq [\![\omega]\!](x)\}$

Examples

▷ **Example 25.6.3** *persons, whose father is a teacher*: $\mathsf{person} \sqcap \mathsf{had\_father.teacher}$

▷ **Example 25.6.4** *persons that have no father*: $\mathsf{person} \sqcap (\mathsf{had\_father})\Uparrow$

▷ **Example 25.6.5** *companies, whose bosses have no company car*: $\mathsf{company} \sqcap (\mathsf{has\_boss} \circ \mathsf{has\_comp\_car})\Uparrow$

▷ **Example 25.6.6** *cars whose exterior color is the same as the interior color*: $\mathsf{car} \sqcap \mathsf{color\_exterior} = \mathsf{color\_interior}$

▷ **Example 25.6.7** *cars whose exterior color is different from the interior color*: $\mathsf{car} \sqcap \mathsf{color\_exterior} \neq \mathsf{color\_interior}$

▷ **Example 25.6.8** *companies whose Bosses and Vice Presidents have the same company car*: $\mathsf{company} \sqcap \mathsf{has\_boss} \circ \mathsf{has\_comp\_car} = \mathsf{has\_VP} \circ \mathsf{has\_comp\_car}$

Normalization

▷ Normalization rules

$$\begin{array}{rcl} \overline{f.\varphi} & \to & (f)\Uparrow \sqcup f.\overline{\varphi} \\ \overline{\pi = \omega} & \to & ((\pi)\Uparrow)(\omega)\Uparrow \sqcup \pi \neq \omega \\ \overline{\pi \neq \omega} & \to & ((\pi)\Uparrow)(\omega)\Uparrow \sqcup \pi = \omega \\ (f \circ \pi)\Uparrow & \to & (f)\Uparrow \sqcup f \circ (\pi)\Uparrow \end{array}$$

▷ **Example 25.6.9** (for the last transformation)

$$(\mathsf{has\_boss} \circ \mathsf{has\_comp\_car} \circ \mathsf{has\_sun\_roof})\Uparrow = \ldots$$

i.e. the set of objects that do not have a boss, plus the set of objects whose boss does not have a company car plus the set of objects whose bosses have company cars without sun roofs

Tableau Calculus

▷ **Definition 25.6.10** The calculus is an extension of $\mathcal{T}_{\mathcal{ALC}}$.

$$\frac{(x\colon f.\varphi)}{\begin{array}{c} xfy \\ (y\colon \varphi) \end{array}} \qquad \frac{(x\colon \pi = \omega)}{\begin{array}{c} x\pi y \\ x\omega y \end{array}} \qquad \frac{(x\colon \pi \neq \omega)}{\begin{array}{c} x\pi y \\ x\omega z \\ y \neq z \end{array}} \qquad \frac{xf \circ \pi y}{\begin{array}{c} xfy \\ z\pi y \end{array}} \qquad \frac{\begin{array}{l} \mathcal{B} \\ xfy \quad \neq y, z \\ xfz \end{array}}{[y/z](\mathcal{B})}$$

$$\frac{(x\colon \bot)}{*} \qquad \frac{\begin{array}{c} (x\colon c) \\ (x\colon \overline{c}) \end{array}}{*} \qquad \frac{\begin{array}{c} xfy \\ (x\colon (f)\Uparrow) \end{array}}{*} \qquad \frac{x \neq x}{*}$$

▷ **Theorem 25.6.11** *The calculus is correct, complete and terminating.*

▷ **Theorem 25.6.12** *It can be implemented in* PSPACE

Example

▷ **Example 25.6.13** $(\mathsf{has\_boss} \circ \mathsf{has\_comp\_car})\Uparrow \sqcap \mathsf{has\_boss}.\mathsf{has\_comp\_car}.\mathsf{has\_sun\_roof}.\top$ is inconsistent.

▷ Normalize: ((has_boss)⇑ ⊔ has_boss.(has_comp_car)⇑) ⊓ has_boss.has_comp_car.has_sun_roof.⊤

▷ Tableau

$$
\begin{array}{c}
(x\colon (\text{has\_boss})\Uparrow \sqcup \text{has\_boss}.(\text{has\_comp\_car})\Uparrow)\\
(x\colon \text{has\_boss}.\text{has\_comp\_car}.\text{has\_sun\_roof}.\top)\\
x\,\text{has\_boss}\,y\\
(y\colon \text{has\_comp\_car}.\text{has\_sun\_roof}.\top)\\
y\,\text{has\_comp\_car}\,z\\
(z\colon \text{has\_sun\_roof}.\top)\\
\begin{array}{c|c}
(x\colon (\text{has\_boss})\Uparrow) & (x\colon \text{has\_boss}.(\text{has\_comp\_car})\Uparrow)\\
* & x\,\text{has\_boss}\,v\\
 & (v\colon (\text{has\_comp\_car})\Uparrow)\\
 & (y\colon (\text{has\_comp\_car})\Uparrow)\quad (y=v)\\
 & *
\end{array}
\end{array}
$$

25.7 Concrete Domains

$\mathcal{ALC}$ with "concrete Domains" (Examples)

Formula	Concrete Domain
person ⊓ age < 20	real numbers
persons younger than 20	
company ⊓ has_CEO ∘ has_comp_car ∘ price) > \$100000	natural numbers
companies with CEOs with expensive car	
car ⊓ height > width	natural numbers
cars that are higher than wide	
person ⊓ first_name < last_name	strings
persons whose first name is lexicographically smaller than their last name	
person ⊓ has_father ∘ studiesbefore(has_mother ∘ studies	temporal interval logic
persons whose fathers have studied before their mothers	

Concrete Domain

▷ **Definition 25.7.1** A concrete domain is a pair $\langle\mathcal{C},\mathcal{P}\rangle$, where $\mathcal{C}$ is a set and $\mathcal{P}$ a set of predicates.

▷ **Example 25.7.2** ▷ $\mathcal{C}=\mathbb{N}$ and $\mathcal{P}=\{=,<,\leq,>,\geq\}$ (natural numbers)

- ▷ $\mathcal{C}=\mathbb{R}$ and $\mathcal{P}=\{=,<,\leq,>,\geq\}$ (real numbers)
- ▷ $\mathcal{C}=$ temporal intervals, $\mathcal{P}=\{\text{before},\text{after},\text{overlaps},\ldots\}$ (Allen's interval logic)
- ▷ $\mathcal{C}=$ facts in a relational data base, $\mathcal{P}=$ SQL relations

Admissible Concrete Domains

- Idea: concrete domains are admissible, iff $\mathcal{P}$ is decidable.
- **Definition 25.7.3** Let $\{P_1, \ldots, P_n\} \subseteq \mathcal{P}$, then conjunctions $P_1(x_1, \ldots) \wedge \ldots \wedge P_n(x_n, \ldots)$ are called satisfiable, iff there is a satisfying variable assignment $[a_i/x_i]$ with $a_i \in \mathcal{C}$. (the model is fixed in a concrete domain)
- **Example 25.7.4** $\mathcal{C}$ = real numbers

$P_1(x,y) = \exists z.(x+z^2=y)$	satisfiable ($z=\sqrt{y-x}$, e.g. $x=y=1, z=0$)
$P_2(x,y) = P_1(x,y) \wedge x > y$	unsatisfiable

- **Definition 25.7.5** A concrete domain $\langle \mathcal{C}, \mathcal{P} \rangle$ is called admissible, iff
  1) the satisfiability problem for conjunctions is decidable
  2) $\mathcal{P}$ is closed under negation and contains a name for $\mathcal{C}$.

$\mathcal{ALC}(\mathcal{C})$

- $F_{\mathcal{ALC}(\mathcal{C})}$:== $F_{\mathcal{ALCF}} \mid P(\pi, \ldots, \pi)$
- **Example 25.7.6** a female human under 21 can become a woman by having a child

$$\begin{aligned} \text{mother} &= \text{human} \sqcap ♀ \sqcap \exists \text{has\_child}.\text{human} \\ \text{woman} &= \text{human} \sqcap ♀ \sqcap (\text{mother} \sqcup \text{age} \geq 21) \end{aligned}$$

  here $\text{age} \geq 21 \in F_{\mathcal{ALC}(\mathcal{C})}$, since it is of the form $P(\text{age})$ ($P = \lambda x.x \geq 21$)
- Semantics of $\mathcal{ALC}(\mathcal{D})$
  - $\mathcal{D}$ and $\mathcal{C}$ are disjoint.
  - $P(\pi_1, \ldots, \pi_n) = x\mathcal{D}$there are $y_1 = [\![\pi_1]\!](x), \ldots, y_n = [\![\pi_n]\!](x) \in \mathcal{C}$, with $\langle y_1, \ldots, y_n \rangle \in [\![P]\!]$

  Warning: $[\![\overline{\varphi}]\!] = \mathcal{D} \backslash [\![\varphi]\!]$, but not $[\![\overline{\varphi}]\!] = (\mathcal{D} \cup \mathcal{C}) \backslash [\![\varphi]\!]$

!

Negation Rules and Tableau Calculus

- Let $\top_{\mathcal{C}}$ be the name for the concrete domain (as a set) and $\overline{P}$ the negated predicate for P ($\mathcal{C}$ is admissible)
- New negation rule: $\overline{P(\pi_1, \ldots, \pi_n)} \rightarrow \overline{P}(\pi_1, \ldots, \pi_n) \sqcup \forall \pi_1.\top_{\mathcal{C}} \sqcup \ldots \sqcup \forall \pi_n.\top_{\mathcal{C}}$

▷ New tableau rule

$$\frac{\begin{array}{c} P_1(x_{11}, \ldots, x_{1n_1}) \\ \vdots \\ P_k(x_{k1}, \ldots, x_{kn_k}) \end{array} \quad \bigwedge_{1 \leq i \leq k} P_i(x_{i1}, \ldots, x_{in_i}) \text{ inconsistent}}{*} \perp^p$$

 294

Example: $\mathsf{car} \sqcap \mathsf{height} = 2 \sqcap \mathsf{width} = 1 \sqsubseteq \mathsf{car} \sqcap \mathsf{height} > \mathsf{width}$

$$\begin{array}{c}
(x\colon \mathsf{car} \sqcap \mathsf{height} = 2 \sqcap \mathsf{width} = 1) \\
(x\colon \overline{\mathsf{car}} \sqcap \mathsf{width} \leq \mathsf{height}) \\
(x\colon \mathsf{car}) \\
(x\colon \mathsf{height} = 2) \\
(x\colon \mathsf{width} = 1)
\end{array}$$

$$\begin{array}{c|c}
(x\colon \overline{\mathsf{car}}) & (x\colon \mathsf{width} \leq \mathsf{height}) \\
* & x \mathsf{height}\ y_1 \\
 & y_1 = 2 \\
 & (x\colon \mathsf{width} = y_2) \\
 & y_2 = 1 \\
 & (x\colon \mathsf{width}\ y_3) \\
 & (x\colon \mathsf{height} = y_4) \\
 & y_3 \leq y_4 \\
 & y_1 \leq y_2 \\
 & *
\end{array}$$

 295

25.8 Nominals

Nominals

▷ **Definition 25.8.1 (Idea)** nominal are names for domain elements that can be used in the T-Box.

▷ **Example 25.8.2** *Students that study on Bremen or Hamburg*: $\mathsf{student} \sqcap \exists \mathsf{studies\_in}.\{Bremen, Hamburg\}$

▷ **Example 25.8.3** *Students that have a friend with name Eva*: $\mathsf{student} \sqcap \exists \mathsf{has\_friend} \circ \mathsf{has\_name}.\{Eva\}$

▷ **Example 25.8.4** *persons that have phoned Bill, Bob, or the murderer*: $\mathsf{person} \sqcap \exists \mathsf{has\_phoned}.\{Bill, Bob, murderer\}$

▷ **Example 25.8.5** *friends of Eva*: $\mathsf{person} \sqcap \mathsf{has\_friend}\colon Eva$

▷ **Example 25.8.6** *companies whose employees all bank at Sparda Bank*: $\mathsf{company} \sqcap \forall \mathsf{has\_empl}.\mathsf{has\_bank}\colon Sparda$

▷ **Example 25.8.7** *employees of Jacobs that bank at Sparda*: $\mathsf{employed\_at}\colon Jacobs \sqcap \mathsf{has\_bank}\colon Sparda$

 296

Semantics

▷ **Definition 25.8.8** $[\![\{a_1, \ldots, a_n\}]\!]$ is the set of objects with names $a_1, \ldots a_n$.

▷ **Definition 25.8.9** $[\![\mathsf{R}\colon a]\!]$ is the set of objects that have $[\![a]\!]$ as R-successor

$$
\begin{aligned}
[\![\{a_1, \ldots, a_n\}]\!] &= \{[\![a_1]\!], \ldots, [\![a_n]\!]\} \\
[\![\mathsf{R}\colon a]\!] &= \{x \in \mathcal{D} \,|\, \langle x, [\![a]\!]\rangle \in [\![\mathsf{R}]\!]\}
\end{aligned}
$$

▷ **Definition 25.8.10 (Negation Rules)**

$$
\begin{aligned}
\overline{\{a_1, \ldots, a_n\}} &= \text{invariant} \\
\overline{\mathsf{R}\colon a} &= \forall\,\mathsf{R}.\overline{\{a\}}
\end{aligned}
$$

▷ **Example 25.8.11** $\overline{\mathsf{had\_friend}\colon Eva}$ (the complement of the set of friends of Eva)

$= \forall\,\mathsf{had\_friend}.\overline{\{Eva\}}$ (the set of objects that do not have Eva as a friend)

Example Language with Nominals

▷ We consider the following language: $\mathcal{ALC}$ + unqualified number restrictions ($\exists^n_{\geq}\mathsf{R}$, $\forall^n_{\leq}\mathsf{R}$), some role operators ($\sqcap$, $\circ$, $\cdot^{-1}$), $\{a_1, \ldots, a_n\}$, $\mathsf{R}\colon a$

▷ **Example 25.8.12** *persons that have at most two friends among their neighbors and whose neighbors are Bill, Bob, or the gardener* $\mathsf{person} \sqcap \forall^2_{\leq}(\mathsf{has\_friend} \sqcap \mathsf{has\_neighbor}) \sqcap \forall\,\mathsf{has\_ne}$

▷ **Example 25.8.13** *companies with at least 100 employees that have a car and live in Bremen* $\mathsf{company} \sqcap \exists^{100}_{\geq}\mathsf{has\_empl} \circ \mathsf{has\_comp\_car} \sqcap \mathsf{has\_empl} \circ \mathsf{lives\_in}\colon Bremen$

Tableaux Calculus (only T-Box)

▷ **Definition 25.8.14** The calculus consists of the $\mathcal{T}_{\mathcal{ALC}}$ rules together with:

$$
\frac{(a\colon \overline{\{\ldots, a, \ldots\}})}{*}
\qquad
\frac{\begin{array}{c}\mathcal{B} \\ (x\colon \{a_1, \ldots, a_n\})\end{array}}{[x/a_1](\mathcal{B}) \;\Big|\; \ldots [x/a_n](\mathcal{B})}
\qquad
\frac{(x\colon \mathsf{R}\colon a)}{x\mathsf{R}a}
$$

$$
\frac{x\mathsf{R}^{-1}y}{y\mathsf{R}x}
\qquad
\frac{x\mathsf{R} \sqcap \mathsf{S}y}{\begin{array}{c}x\mathsf{R}y \\ x\mathsf{S}y\end{array}}
\qquad
\frac{x\mathsf{R} \circ \mathsf{S}y}{\begin{array}{c}x\mathsf{R}z \\ z\mathsf{S}y\end{array}}
$$

▷ **Theorem 25.8.15** *The calculus is correct, complete, and terminating*

▷ Proof Sketch: very technical but not terribly difficult using the techniques developed so far. □

 299 JACOBS UNIVERSITY

Bibliography

[And72] Peter B. Andrews. General models and extensionality. *Journal of Symbolic Logic*, 37(2):395–397, 1972.

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Kluwer Academic Publishers, second edition, 2002.

[Bou68] Nicolas Bourbaki. *Theory of Sets.* Elements of Mathematics. Springer Verlag, 1968.

[Bou74] Nicolas Bourbaki. *Algebra I.* Elements of Mathematics. Springer Verlag, 1974.

[Bou89] N. Bourbaki. *General Topology 1-4.* Elements of Mathematics. Springer Verlag, 1989.

[Can95] Georg Cantor. Beiträge zur begründung der transfiniten mengenlehre (1). *Mathematische Annalen*, 46:481–512, 1895.

[Can97] Georg Cantor. Beiträge zur begründung der transfiniten mengenlehre (2). *Mathematische Annalen*, 49:207–246, 1897.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[CQ69] Allan M. Collins and M. Ross Quillian. Retrieval time from semantic memory. *Journal of verbal learning and verbal behavior*, 8(2):240–247, 1969.

[DCM12] DCMI Usage Board. DCMI metadata terms. DCMI recommendation, Dublin Core Metadata Initiative, 2012.

[DPPDD09] A.K. Doxiadēs, C.H. Papadimitriou, A. Papadatos, and A. Di Donna. *Logicomix: An Epic Search for Truth.* Bloomsbury, 2009.

[Fre79] Gottlob Frege. Begriffsschrift: eine der arithmetischen nachgebildete formelsprache des reinen denkens, 1879.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39:176–210, 1935.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte der Mathematischen Physik*, 38:173–198, 1931. English Version in [vH67].

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[HASB13a] Ivan Herman, Ben Adida, Manu Sporny, and Mark Birbeck. RDF 1.1 primer (second edition). W3C working group note, World Wide Web Consortium (W3C), 2013.

[HASB13b] Ivan Herman, Ben Adida, Manu Sporny, and Mark Birbeck. RDFa 1.1 primer – second edition. W3C Working Goup Note, World Wide Web Consortium (W3C), 2013.

[Hil26] David Hilbert. Über das unendliche. *Mathematische Annalen*, 95:161–190, 1926.

[HKP+12] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 web ontology language primer (second edition). W3C recommendation, World Wide Web Consortium (W3C), 2012.

[Hue76] Gérard P. Huet. *Résolution d'Équations dans des Langages d'ordre 1,2,...,w.* Thèse d'état, Université de Paris VII, 1976.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM (JACM)*, 27(4):797–821, 1980.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. W3C recommendation, World Wide Web Consortium (W3C), 2004.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

[Koh15] Michael Kohlhase. sTeX: Semantic markup in TeX/LaTeX. Technical report, Comprehensive TeX Archive Network (CTAN), 2015.

[Mat70] Ju. V. Matijasevič. Enumerable sets are diophantine. *Soviet Math. Doklady*, 11:354–358, 1970.

[OWL09] OWL Working Group. OWL 2 web ontology language: Document overview. W3C recommendation, World Wide Web Consortium (W3C), October 2009.

[Pro] Protégé. Project Home page at `http://protege.stanford.edu`.

[PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen.* Gabler Verlag, 4 (2003) edition, 1997.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium (W3C), January 2008.

[SR14] Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C working group note, World Wide Web Consortium (W3C), 2014.

[Sta85] Rick Statman. Logical relations and the typed lambda calculus. *Information and Computation*, 65, 1985.

[Tob00] Stephan Tobies. PSpace reasoning for graded modal logics. *Journal of Logic and Computation*, 11:85–106, 2000.

[vH67] Jean van Heijenoort. *From Frege to Gödel: a source book in mathematical logic 1879-1931.* Source books in the history of the sciences series. Harvard Univ. Press, Cambridge, MA, 3rd printing, 1997 edition, 1967.

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, UK, 2 edition, 1910.

[Zer08] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre. I. *Mathematische Annalen*, 65:261–281, 1908.