

# Computational Logic

## 320441 CompLog Lecture Notes

Michael Kohlhase

School of Engineering & Science  
Jacobs University, Bremen Germany  
`m.kohlhase@jacobs-university.de`

November 27, 2013

## 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.

**Corequisites and Extensions:** In its incarnation as an undergraduate course, there are no co-requisites, but in its incarnation as a graduate course, the course 320612 *Computational Logic Lab* is required. It teaches the graduate-level material, in particular encoding logics in meta-logical frameworks and gives hands-on experience with the LF and MMT systems.

Graduate students can optionally take the course 320434 *Computational Logic Project*, which covers research-near techniques in a guided research project.

**Course Contents:** We carefully recap the foundations of first-order logic and present the tableau calculus as a computationally inspired inference procedure.<sup>1</sup>

---

<sup>1</sup>EdNOTE: continue, when the plan is fixed.

## 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.

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

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

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

**Knowledge Representation Experiment:** This document is also an experiment in knowledge representation. Under the hood, it uses the  $\text{\LaTeX}$  package [Koh08, Koh12], a  $\text{\TeX}/\text{\LaTeX}$  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 2013

In this document, we record the progress of the course in Fall 2013 in the form of a “recorded syllabus”, i.e. a syllabus that is created after the fact rather than before.

[Recorded Syllabus Fall Semester 2013:](#)

#	date	until	slide	page
1	Sep 9.	admin/intro/history	11	10
2	Sep 11.	more overview	13	10
3	Sep 16.	substitution value lemma	24	21
4	Sep 18.	inference for FOL	33	28
5	Sep 23.	abstract consistency	45	35
6	Sep 25.	model existence	52	39
7	Sep 30.	unification algorithm	63	48
8	Oct 2.	efficient unification	67	50
9	Oct 7.	tableau soundness/completeness	76	55
10	Oct 14.	Simply typed $\lambda$ calculus & Head Reduction	93	71
11	Oct 16.	Termination of $\beta$ -reduction	99	76
12	Nov 5.	$\alpha\beta\eta$ -completeness	112	83
13	Nov 6.	HOU & General Bindings	117	87
14	Nov 12.	HOU completeness	125	90
15	Nov 13.	Andrews’ equality-based HOL	133	96
16	Nov 19.	Higher-Order Tableaux	143	102
17	Nov 20.	PL0 set description	171	122
18	Nov 26.	Semantic Web/Ontologies	??	??

# Contents

Preface . . . . .	ii
Introduction . . . . .	ii
Course Concept . . . . .	ii
This Document . . . . .	ii
Acknowledgments . . . . .	iii
Recorded Syllabus for Fall 2013 . . . . .	iv
<b>1 Outline of the Course</b>	<b>1</b>
<b>2 320411/CompLog Administrativa</b>	<b>3</b>
<b>3 A History of Ideas in Logic</b>	<b>9</b>
<b>I First-Order Logic and Inference</b>	<b>13</b>
<b>4 First-Order Logic</b>	<b>15</b>
4.1 First-Order Logic: Syntax and Semantics . . . . .	16
4.2 First-Order Substitutions . . . . .	19
4.3 Alpha-Renaming for First-Order Logic . . . . .	22
4.4 Entailment Theorem . . . . .	22
<b>5 Inference in First-Order Logic</b>	<b>25</b>
5.1 Formal Systems . . . . .	25
5.1.1 Logical Systems . . . . .	25
5.1.2 Calculi, Derivations, and Proofs . . . . .	26
5.1.3 Properties of Calculi . . . . .	27
5.2 First-Order Calculi . . . . .	29
5.2.1 Propositional Natural Deduction Calculus . . . . .	29
5.3 Abstract Consistency and Model Existence . . . . .	34
5.4 A Completeness Proof for First-Order ND . . . . .	40
5.5 Limits of First-Order Logic . . . . .	41
<b>6 First-Order Inference with Tableaux</b>	<b>43</b>
6.1 First-Order Tableaux . . . . .	43
6.2 Free Variable Tableaux . . . . .	45
6.3 First-Order Unification . . . . .	46
6.4 Efficient Unification . . . . .	50
6.5 Soundness and Completeness of First-Order Tableaux . . . . .	52
<b>II Higher-Order Logic and <math>\lambda</math>-Calculus</b>	<b>57</b>
<b>7 Higher-Order Predicate Logic</b>	<b>61</b>

<b>8</b>	<b>Simply Typed <math>\lambda</math>-Calculus</b>	<b>69</b>
<b>9</b>	<b>Computational Properties of <math>\lambda</math>-Calculus</b>	<b>73</b>
9.1	Termination of $\beta$ -reduction . . . . .	73
9.2	Confluence of $\beta\eta$ Conversion . . . . .	77
<b>10</b>	<b>The Semantics of the Simply Typed <math>\lambda</math>-Calculus</b>	<b>79</b>
10.1	Soundness of the Simply Typed $\lambda$ -Calculus . . . . .	79
10.2	Completeness of $\alpha\beta\eta$ -Equality . . . . .	81
<b>11</b>	<b>Higher-Order Unification</b>	<b>85</b>
11.1	Higher-Order Unifiers . . . . .	85
11.2	Higher-Order Unification Transformations . . . . .	86
11.3	Properties of Higher-Order Unification . . . . .	89
11.4	Pre-Unification . . . . .	92
11.5	Applications of Higher-Order Unification . . . . .	93
<b>12</b>	<b>Simple Type Theory (Higher-Order Logic based on the Simply Typed <math>\lambda</math>-Calculus)</b>	<b>95</b>
<b>13</b>	<b>Higher-Order Tableaux</b>	<b>99</b>
<b>III</b>	<b>Knowledge Representation</b>	<b>103</b>
<b>14</b>	<b>Introduction to Knowledge Representation</b>	<b>107</b>
14.1	Semantic Networks . . . . .	109
14.2	The Semantic Web . . . . .	112
14.3	Other Knowledge Representation Approaches . . . . .	117
<b>15</b>	<b>Logic-Based Knowledge Representation</b>	<b>119</b>
15.1	Propositional Logic as a Set Description Language . . . . .	120
15.2	Description Logics and Inference . . . . .	123
<b>16</b>	<b>Description Logics and the Semantic Web</b>	<b>127</b>
<b>17</b>	<b>A simple Description Logic: ALC</b>	<b>131</b>
<b>18</b>	<b>ALC Extensions</b>	<b>143</b>
18.1	Functional Roles and Number Restrictions . . . . .	143
18.2	Unique Names . . . . .	146
18.3	Qualified Number Restrictions . . . . .	147
18.4	Role Operators . . . . .	150
18.5	Role Axioms . . . . .	154
18.6	Features . . . . .	155
18.7	Concrete Domains . . . . .	157
18.8	Nominals . . . . .	160

# 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 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 (consolidation)
  - ▷ special attention to substitutions,  $\alpha$ -renaming (usually glossed over)
  - ▷ soundness/completeness (interesting proofs)
  - ▷ tableau calculi, unification (basis for later)
- ▷ Higher-Order Logic (more expressivity for math)
  - ▷ simply typed  $\lambda$  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







## 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.

Let us first look at the setup of the CompLog Course in the Joacbs curriculum.

### The Course/Lab Combo

- ▷ The course is both for undergraduate students in their third year and graduate students in their first year.
- ▷ For graduate students, the course is accompanied by a lab, which
  - ▷ **During the Semester**: teaches graduate-level (more advanced) concepts.
  - ▷ **During the semester**: deepens the understanding of the course topics by practical exercise
  - ▷ **Over the break**: let's you experiment with them in a little research project
- ▷ Course and Lab are 5 credits each (⚠ about one day of work per week each!)
- ▷ They are graded independently



©: Michael Kohlhase

2



For graduate students, the Course/Lab combo is one of two they will take in the first year. They also have to take one graduate project, which is also offered in the CompLog context. The topic of this is individually negotiated with the instructor and serves as a possible preparation (or a test run) for a master's thesis topic.

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://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 [c.prodescu@jacobs-university.de](mailto:c.prodescu@jacobs-university.de)



**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)

Course		Lab	
Homework Assignments	70%	Attendance and Wakefulness	10%
Quizzes	30%	Graded Lab Assignments	50%
No Midterm Exam	–	Intersession Project	20%
No Final Exam	–	Discussion	20%

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.



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 interests, 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)



©: Michael Kohlhasse

7



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<sup>1</sup>. 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 course materials as a research object as well. This gives you an additional resource, but may affect the shape of the course materials (which now serve double purpose). Of course I can use all the help on the research project I can get, so please give me feedback, report errors and shortcomings, and suggest improvements.

## Experiment: E-Learning with OMDoc/PantaRhei

- ▷ My research area: deep representation formats for (mathematical) knowledge
- ▷ Application: E-learning systems (represent knowledge to transport it)

<sup>1</sup>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,...

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





## Chapter 3

# 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 chapter 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 ~ 1850]
  - + functional structure of formal language (propositions + connectives)
  - + mathematical semantics ( $\rightsquigarrow$  Boolean Algebra)
  - abstraction from internal structure of propositions



©: Michael Kohlhase

9



### 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 ~ 1889]
  - + modern notation for predicate logic ( $\forall, \wedge, \Rightarrow, \forall, \exists$ )



©: Michael Kohlhase

10



## 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 ~ 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 ( $\langle \text{NaturalNumbers}, +, * \rangle$ ),

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



## History of Ideas (continued): $\lambda$ -calculus, set theory

- ▷ Simply typed  $\lambda$ -calculus ([Church 1940])
  - + simplifies Russel's types,  $\lambda$ -operator for functions
  - + comprehension as  $\beta$ -equality (can be mechanized)
  - + simple type-driven semantics (standard semantics  $\rightsquigarrow$  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

# First-Order Logic and Inference



## Chapter 4

# 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<sup>1</sup>)

- ▷ **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, ...*)



©: Michael Kohlhase

14



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 ( $\alpha$ -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  $\alpha$ -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  $\alpha$ -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.

## 4.1 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<sup>1</sup> Syntax (Signature and Variables)

- ▷ **Definition 4.1.1** **First-order logic** (PL<sup>1</sup>), 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<sup>1</sup> talks about two kinds of objects: (so we have two kinds of symbols)
  - ▷ **truth values**; sometimes annotated by type  $o$  (like in PL<sup>0</sup>)
  - ▷ **individuals**; sometimes annotated by type  $\iota$  (numbers, foxes, Pokémon, ...)
- ▷ **Definition 4.1.2** A **first-order signature** consists of (all disjoint;  $k \in \mathbb{N}$ )
  - ▷ **connectives**:  $\Sigma^o = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\}$  (functions on truth values)
  - ▷ **function constants**:  $\Sigma_k^f = \{f, g, h, \dots\}$  (functions on individuals)
  - ▷ **predicate constants**:  $\Sigma_k^p = \{p, q, r, \dots\}$  (relations among inds.)
  - ▷ (**Skolem constants**:  $\Sigma_k^{sk} = \{f_1^k, f_2^k, \dots\}$ ) (witness constructors; countably  $\infty$ )
  - ▷ We take the signature  $\Sigma$  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**:  $\mathcal{V}_\iota = \{X_\iota, Y_\iota, Z, X^1_\iota, X^2_\iota\}$  (countably  $\infty$ )



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<sup>1</sup> Syntax (Formulae)

- ▷ **Definition 4.1.3** **terms**:  $A \in \text{wff}_\iota(\Sigma_\iota)$  (denote individuals: type  $\iota$ )

▷  $\mathcal{V}_t \subseteq \text{wff}_t(\Sigma_t)$ ,

▷ if  $f \in \Sigma_k^f$  and  $\mathbf{A}^i \in \text{wff}_t(\Sigma_t)$  for  $i \leq k$ , then  $f(\mathbf{A}^1, \dots, \mathbf{A}^k) \in \text{wff}_t(\Sigma_t)$ .

▷ **Definition 4.1.4** **propositions**:  $\mathbf{A} \in \text{wff}_o(\Sigma)$  (denote truth values: type  $o$ )

▷ if  $p \in \Sigma_k^p$  and  $\mathbf{A}^i \in \text{wff}_t(\Sigma_t)$  for  $i \leq k$ , then  $p(\mathbf{A}^1, \dots, \mathbf{A}^k) \in \text{wff}_o(\Sigma)$ ,

▷ if  $\mathbf{A}, \mathbf{B} \in \text{wff}_o(\Sigma)$ , then  $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X. \mathbf{A} \in \text{wff}_o(\Sigma)$ .

▷ **Definition 4.1.5** 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 4.1.6** We use  $\exists X. \mathbf{A}$  as an abbreviation for  $\neg(\forall X. \neg \mathbf{A})$ . (existential quantifier)

▷ **Definition 4.1.7** 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 4.1.8** 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  $\mathbf{BVar}(\mathbf{A})$  (and  $\mathbf{free}(\mathbf{A})$ ) for the set of bound (free) variables of  $\mathbf{A}$ , i.e. variables that have a free/bound occurrence in  $\mathbf{A}$ .

▷ **Definition 4.1.9** We define the set  $\mathbf{free}(\mathbf{A})$  of **free variables** of a formula  $\mathbf{A}$  inductively:

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

▷ **Definition 4.1.10** We call a formula  $\mathbf{A}$  **closed** or **ground**, iff  $\mathbf{free}(\mathbf{A}) = \emptyset$ . We call a closed proposition a **sentence**, and denote the set of all ground terms with  $\text{cwff}_t(\Sigma_t)$  and the set of sentences with  $\text{cwff}_o(\Sigma_t)$ .



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 give 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<sup>1</sup> (Models)

- ▷ We fix the **Universe**  $\mathcal{D}_o = \{\mathsf{T}, \mathsf{F}\}$  of **truth values**.
- ▷ We assume an arbitrary **universe**  $\mathcal{D}_i \neq \emptyset$  of **individuals** (this choice is a parameter to the semantics)
- ▷ **Definition 4.1.11** An **interpretation**  $\mathcal{I}$  assigns values to constants, e.g.
  - ▷  $\mathcal{I}(\neg): \mathcal{D}_o \rightarrow \mathcal{D}_o$  with  $\mathsf{T} \mapsto \mathsf{F}$ ,  $\mathsf{F} \mapsto \mathsf{T}$ , and  $\mathcal{I}(\wedge) = \dots$  (as in PL<sup>0</sup>)
  - ▷  $\mathcal{I}: \Sigma_k^f \rightarrow \mathcal{F}(\mathcal{D}_i^k; \mathcal{D}_i)$  (interpret function symbols as arbitrary functions)
  - ▷  $\mathcal{I}: \Sigma_k^p \rightarrow \mathcal{P}(\mathcal{D}_i^k)$  (interpret predicates as arbitrary relations)
- ▷ **Definition 4.1.12** A **variable assignment**  $\varphi: \mathcal{V}_i \rightarrow \mathcal{D}_i$  maps variables into the universe.
- ▷ A first-order **Model**  $\mathcal{M} = \langle \mathcal{D}_i, \mathcal{I} \rangle$  consists of a universe  $\mathcal{D}_i$  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<sup>1</sup> (Evaluation)

- ▷ Given a model  $\langle \mathcal{D}, \mathcal{I} \rangle$ , the **value function**  $\mathcal{I}_\varphi$  is recursively defined: (two parts: terms & propositions)
- ▷  $\mathcal{I}_\varphi: \text{wff}_i(\Sigma_i) \rightarrow \mathcal{D}_i$  assigns values to terms.
  - ▷  $\mathcal{I}_\varphi(X) := \varphi(X)$  and
  - ▷  $\mathcal{I}_\varphi(f(\mathbf{A}_1, \dots, \mathbf{A}_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \dots, \mathcal{I}_\varphi(\mathbf{A}_k))$
- ▷  $\mathcal{I}_\varphi: \text{wff}_o(\Sigma) \rightarrow \mathcal{D}_o$  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<sup>0</sup>)
  - ▷  $\mathcal{I}_\varphi(p(\mathbf{A}^1, \dots, \mathbf{A}^k)) := \mathsf{T}$ , iff  $\langle \mathcal{I}_\varphi(\mathbf{A}^1), \dots, \mathcal{I}_\varphi(\mathbf{A}^k) \rangle \in \mathcal{I}(p)$
  - ▷  $\mathcal{I}_\varphi(\forall X. \mathbf{A}) := \mathsf{T}$ , iff  $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = \mathsf{T}$  for all  $a \in \mathcal{D}_i$ .



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  $\varphi$ , the assignment  $\psi$  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  $\mathcal{I}_\varphi(\forall x.\neg\mathbf{A}) = \mathcal{I}_\psi(\neg\mathbf{A}) = \mathbf{F}$  for all  $\mathbf{a} \in \mathcal{D}_\iota$  and  $\psi := \varphi, [a/X]$ . This is the case, iff  $\mathcal{I}_\psi(\mathbf{A}) = \mathbf{T}$  for some  $\mathbf{a} \in \mathcal{D}_\iota$ . So our definition of the existential quantifier yields the appropriate semantics.

## 4.2 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 4.2.1** We call  $\sigma: \text{wff}_\iota(\Sigma_\iota) \rightarrow \text{wff}_\iota(\Sigma_\iota)$  a **substitution**, iff  $\sigma(f(\mathbf{A}_1, \dots, \mathbf{A}_n)) = f(\sigma(\mathbf{A}_1), \dots, \sigma(\mathbf{A}_n))$  and the **support**  $\text{supp}(\sigma) := \{X \mid \sigma(X) \neq X\}$  of  $\sigma$  is finite.
- ▷ **Observation 4.2.2** Note that a substitution  $\sigma$  is determined by its values on variables alone, thus we can write  $\sigma$  as  $\sigma|_{\mathcal{V}_\iota} = \{[\sigma(X)/X] \mid X \in \text{supp}(\sigma)\}$ .
- ▷ **Notation 4.2.3** We denote the substitution  $\sigma$  with  $\text{supp}(\sigma) = \{x^i \mid 1 \leq i \leq n\}$  and  $\sigma(x^i) = \mathbf{A}_i$  by  $[\mathbf{A}_1/x^1], \dots, [\mathbf{A}_n/x^n]$ .
- ▷ **Example 4.2.4**  $[a/x], [f(b)/y], [a/z]$  instantiates  $g(x, y, h(z))$  to  $g(a, f(b), h(a))$ .
- ▷ **Definition 4.2.5** We call  $\text{intro}(\sigma) := \bigcup_{X \in \text{supp}(\sigma)} \text{free}(\sigma(X))$  the set of variables **introduced** by  $\sigma$ .



The extension of a substitution is an important operation, which you will run into from time to time. Given a substitution  $\sigma$ , a variable  $x$ , and an expression  $\mathbf{A}$ ,  $\sigma, [\mathbf{A}/x]$  extends  $\sigma$  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  $\sigma$  may not show it.

### Substitution Extension

- ▷ **Notation 4.2.6 (Substitution Extension)** Let  $\sigma$  be a substitution, then we denote with  $\sigma, [\mathbf{A}/X]$  the function  $\{\langle Y, \mathbf{A} \rangle \in \sigma \mid Y \neq X\} \cup \{\langle X, \mathbf{A} \rangle\}$ . ( $\sigma, [\mathbf{A}/X]$  coincides with  $\sigma$  of  $X$ , and gives the result  $\mathbf{A}$  then
- ▷ **Note:** If  $\sigma$  is a substitution, then  $\sigma, [\mathbf{A}/X]$  is also a substitution.
- ▷ **Definition 4.2.7** If  $\sigma$  is a substitution, then we call  $\sigma, [\mathbf{A}/X]$  the **extension** of  $\sigma$  by  $[\mathbf{A}/X]$ .
- ▷ We also need the dual operation: removing a variable from the support
- ▷ **Definition 4.2.8** We can **discharge** a variable  $X$  from a substitution  $\sigma$  by  $\sigma_{-X} := \sigma, [X/X]$ .



Note that the use of the comma notation for substitutions defined in Notation 4.2.3 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  $\sigma(\forall X.A)$ ?

▷ **Idea:**  $\sigma$  should not instantiate bound variables. ( $[A/X](\forall X.B) = \forall A.B'$  ill-formed)

▷ **Definition 4.2.9**  $\sigma(\forall X.A) := \forall X.\sigma_{-X}(A)$ .

▷ **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 4.2.10** Let  $B \in \text{wff}_\iota(\Sigma_\iota)$  and  $A \in \text{wff}_o(\Sigma)$ , then we call  $B$  **substitutable** for  $X$  in  $A$ , iff  $A$  has no occurrence of  $X$  in a subterm  $\forall Y.C$  with  $Y \in \text{free}(B)$ .

▷ **Solution:** Forbid substitution  $[B/X]A$ , when  $B$  is not substitutable for  $X$  in  $A$ .

▷ **Better Solution:** Rename away the bound variable  $X$  in  $\forall X.p(X, Y)$  before applying the substitution. (see alphabetic renaming later.)



Here we come to a conceptual problem of most introductions to first-order logic: they directly define substitutions to be capture-avoiding by stipulating that bound variables are renamed in the to ensure substitutability. 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 4.2.11** Let  $A$  and  $B$  be terms, then  $\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\psi(A)$ , where  $\psi = \varphi, [\mathcal{I}_\varphi(B)/X]$ .

▷ **Proof:** by induction on the depth of  $A$ :

**P.1.1.1** *depth=0*:

**P.1.1.1.1** Then  $\mathbf{A}$  is a variable (say  $Y$ ), or constant, so we have three cases

**P.1.1.1.1.1**  $\mathbf{A} = Y = X$ : 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.1.1.1.1.2**  $\mathbf{A} = Y \neq X$ : 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.1.1.1.1.3**  $\mathbf{A}$  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  $\mathbf{A} = f(\mathbf{A}_1, \dots, \mathbf{A}_n)$  and we have

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

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 4.2.12** Let  $\mathbf{B} \in \text{wff}_\iota(\Sigma_\iota)$  be substitutable for  $X$  in  $\mathbf{A} \in \text{wff}_o(\Sigma)$ , 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  $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall X. \mathbf{C}) = \text{T}$ , iff  $\mathcal{I}_{\psi, [a/X]}(\mathbf{C}) = \mathcal{I}_{\varphi, [a/X]}(\mathbf{C}) = \text{T}$ , for all  $a \in \mathcal{D}_\iota$ , which is the case, iff  $\mathcal{I}_\varphi(\forall X. \mathbf{C}) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \text{T}$ .

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



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.

### 4.3 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 4.3.1** *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**  $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \top$ , iff

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

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

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

▷ **Definition 4.3.2** We call two formulae  $\mathbf{A}$  and  $\mathbf{B}$  **alphabetical variants** (or  **$\alpha$ -equal**); write  **$\mathbf{A} =_\alpha \mathbf{B}$** , iff  $\mathbf{A} = \forall X.\mathbf{C}$  and  $\mathbf{B} = \forall Y.[Y/X](\mathbf{C})$  for some variables  $X$  and  $Y$ .



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

#### Avoiding Variable Capture by Built-in $\alpha$ -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  $\text{cwff}_o(\Sigma_\iota)$  (new) to be the quotient set of  $\text{cwff}_o(\Sigma_\iota)$  (old) modulo  $=_\alpha$ .  
(formulae as syntactic representatives of equivalence classes)

▷ **Definition 4.3.3 (Capture-Avoiding Substitution Application)** Let  $\sigma$  be a substitution,  $\mathbf{A}$  a formula, and  $\mathbf{A}'$  an alphabetical variant of  $\mathbf{A}$ , such that  $\text{intro}(\sigma) \cap \mathbf{BVar}(\mathbf{A}) = \emptyset$ . Then  $[\mathbf{A}]_{=_\alpha} = [\mathbf{A}']_{=_\alpha}$  and we can define  $\sigma([\mathbf{A}]_{=_\alpha}) := [\sigma(\mathbf{A}')]_{=_\alpha}$ .

▷ **Notation 4.3.4** 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.



### 4.4 Entailment Theorem

The next theorem shows that the implication connective and the entailment relation are closely related: we can move a hypothesis of the entailment relation into an implication assumption in the conclusion of the entailment relation. Note that however close the relationship between implication

and entailment, the two should not be confused. The implication connective is a syntactic formula constructor, whereas the entailment relation lives in the semantic realm. It is a relation between formulae that is induced by the evaluation mapping.

### The Entailment Theorem

▷ **Theorem 4.4.1** *If  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$ , then  $\mathcal{H} \models (\mathbf{A} \Rightarrow \mathbf{B})$ .*

▷ **Proof:** We show that  $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathbf{T}$  for all assignments  $\varphi$  with  $\mathcal{I}_\varphi(\mathcal{H}) = \mathbf{T}$  whenever  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$

**P.1** Let us assume there is an assignment  $\varphi$ , such that  $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathbf{F}$ .

**P.2** Then  $\mathcal{I}_\varphi(\mathbf{A}) = \mathbf{T}$  and  $\mathcal{I}_\varphi(\mathbf{B}) = \mathbf{F}$  by definition.

**P.3** But we also know that  $\mathcal{I}_\varphi(\mathcal{H}) = \mathbf{T}$  and thus  $\mathcal{I}_\varphi(\mathbf{B}) = \mathbf{T}$ , since  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$ .

**P.4** This contradicts our assumption  $\mathcal{I}_\varphi(\mathbf{B}) = \mathbf{F}$  from above.

**P.5** So there cannot be an assignment  $\varphi$  that  $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathbf{F}$ ; in other words,  $\mathbf{A} \Rightarrow \mathbf{B}$  is valid.  $\square$



Now, we complete the theorem by proving the converse direction, which is rather simple.

### The Entailment Theorem (continued)

▷ **Corollary 4.4.2**  *$\mathcal{H}, \mathbf{A} \models \mathbf{B}$ , iff  $\mathcal{H} \models (\mathbf{A} \Rightarrow \mathbf{B})$*

▷ **Proof:** In the light of the previous result, we only need to prove that  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$ , whenever  $\mathcal{H} \models (\mathbf{A} \Rightarrow \mathbf{B})$

**P.1** To prove that  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$  we assume that  $\mathcal{I}_\varphi(\mathcal{H}, \mathbf{A}) = \mathbf{T}$ .

**P.2** In particular,  $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathbf{T}$  since  $\mathcal{H} \models (\mathbf{A} \Rightarrow \mathbf{B})$ .

**P.3** Thus we have  $\mathcal{I}_\varphi(\mathbf{A}) = \mathbf{F}$  or  $\mathcal{I}_\varphi(\mathbf{B}) = \mathbf{T}$ .

**P.4** The first cannot hold, so the second does, thus  $\mathcal{H}, \mathbf{A} \models \mathbf{B}$ .  $\square$





## Chapter 5

# 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.

### 5.1 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.

#### 5.1.1 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 language. The satisfaction relation tells us when an expression is deemed true in this model.

#### Logical Systems

▷ **Definition 5.1.1** 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 5.1.2** 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 5.1.3** 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 5.1.4**  $\mathbf{A} \models \mathbf{B}$  and  $\mathcal{M} \models \mathbf{A}$  imply  $\mathcal{M} \models \mathbf{B}$ .



**Example 5.1.5 (First-Order Logic as a Logical System)** Let  $\mathcal{L} := \text{wff}_o(\Sigma)$ ,  $\mathcal{K}$  be the class of first-order models, and  $\mathcal{M} \models \mathbf{A} :\Leftrightarrow \mathcal{I}_\varphi(\mathbf{A}) = \mathbf{T}$ , then  $\langle \mathcal{L}, \mathcal{K}, \models \rangle$  is a logical system in the sense of Definition 5.1.1.

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.

### 5.1.2 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 5.1.6** 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 5.1.7** 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 5.1.8** Let  $\mathcal{L}$  be a formal language, then an **inference rule** over  $\mathcal{L}$

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

where  $\mathbf{A}_1, \dots, \mathbf{A}_n$  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 5.1.9** An inference rule without assumptions is called an **axiom** (schema).

▷ **Definition 5.1.10** 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 5.1.11** 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  $\mathbf{A}_1, \dots, \mathbf{A}_m$  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  $\frac{\mathbf{A}_{l_1} \cdots \mathbf{A}_{l_k}}{\mathbf{A}_i}$  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 5.1.12** 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.

$$\frac{\frac{}{P \Rightarrow Q \Rightarrow P} \mathbf{K} \quad P}{Q \Rightarrow P} \text{MP}$$

▷ **Observation 5.1.13** 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  $\vdash_{\mathcal{C}}$  defined in Definition 5.1.11 is a **derivation relation** in the sense of Definition 5.1.6.<sup>2</sup>

▷ **Definition 5.1.14** Correspondingly, 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 5.1.15** 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 5.1.16** 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.



<sup>2</sup>EdNOTE: MK: this should become a view!

Inference rules are relations on formulae represented by formula schemata (where boldface, upper-case letters are used as meta-variables for formulae). For instance, in Example 5.1.12 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 5.1.12.

### 5.1.3 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 5.1.17** 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

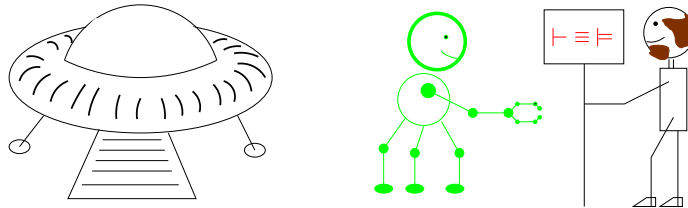
▷ **complete**, iff  $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$ , whenever  $\mathcal{H} \models \mathbf{A}$ .

▷ Goal:  $\vdash \mathbf{A}$  iff  $\models \mathbf{A}$

(provability and validity coincide)

▷ To TRUTH through PROOF

(CALCULEMUS [Leibniz ~1680])



©: Michael Kohlhase

32



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.

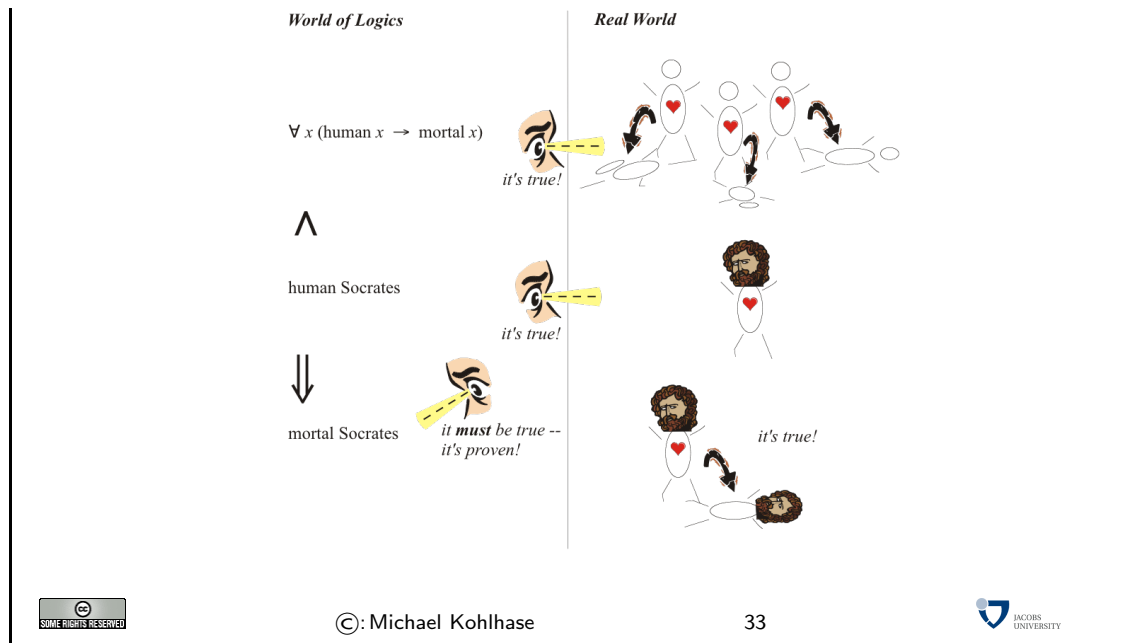
Of course, the logics we have studied so far are very simple, and not able to express interesting facts about the world, but we will study them as a simple example of the fundamental problem of Computer Science: How do the formal representations correlate with the real world.

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 *Socrates 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!



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!)

## 5.2 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<sup>3</sup>.

EdN:3

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.

### 5.2.1 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

<sup>3</sup>EdNOTE: say something about cut elimination/analytical calculi somewhere

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])

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

▷ **Definition 5.2.1** The **propositional natural deduction calculus**  $\mathcal{ND}^0$  has rules for the introduction and elimination of connectives

$$\begin{array}{ccc}
 \text{Introduction} & \text{Elimination} & \text{Axiom} \\
 \frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \wedge I & \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{A}} \wedge E_l \quad \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{B}} \wedge E_r & \frac{}{\mathbf{A} \vee \neg \mathbf{A}} \text{TND} \\
 \frac{\begin{array}{c} [\mathbf{A}]^1 \\ \hline \mathbf{B} \end{array}}{\mathbf{A} \Rightarrow \mathbf{B}} \Rightarrow I^1 & \frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}} \Rightarrow E & 
 \end{array}$$

▷ 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

$$\begin{array}{ccc}
 \frac{[\mathbf{A} \wedge \mathbf{B}]^1}{\mathbf{B}} \wedge E_r & \frac{[\mathbf{A} \wedge \mathbf{B}]^1}{\mathbf{A}} \wedge E_l & \\
 \frac{\mathbf{B} \quad \mathbf{A}}{\mathbf{B} \wedge \mathbf{A}} \wedge I & & \\
 \frac{\mathbf{B} \wedge \mathbf{A}}{\mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A}} \Rightarrow I^1 & & \\
 & \frac{[\mathbf{A}]^1}{[\mathbf{B}]^2} & \\
 & \frac{\mathbf{A}}{\mathbf{B} \Rightarrow \mathbf{A}} \Rightarrow I^2 & \\
 & \frac{\mathbf{B} \Rightarrow \mathbf{A}}{\mathbf{A} \Rightarrow \mathbf{B} \Rightarrow \mathbf{A}} \Rightarrow I^1 & 
 \end{array}$$



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 5.2.2**  $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$ , 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  $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$  by weakening and  $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$  by  $\Rightarrow E$ .  $\square$



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 5.2.1 for disjunction, negation and falsity.

### More Rules for Natural Deduction

▷ **Definition 5.2.3**  $\mathcal{ND}^0$  has the following additional rules for the remaining connectives.

$$\begin{array}{c}
 \frac{\mathbf{A}}{\mathbf{A} \vee \mathbf{B}} \vee I_l \quad \frac{\mathbf{B}}{\mathbf{A} \vee \mathbf{B}} \vee I_r \quad \frac{\begin{array}{c} [\mathbf{A}]^1 \quad [\mathbf{B}]^1 \\ \vdots \quad \vdots \\ \mathbf{A} \vee \mathbf{B} \quad \mathbf{C} \end{array}}{\mathbf{C}} \vee E^1 \\
 \frac{\begin{array}{c} [\mathbf{A}]^1 \\ \vdots \\ \mathbf{F} \end{array}}{\neg \mathbf{A}} \neg I^1 \quad \frac{\neg \neg \mathbf{A}}{\mathbf{A}} \neg E \\
 \frac{\neg \mathbf{A} \quad \mathbf{A}}{\mathbf{F}} FI \quad \frac{\mathbf{F}}{\mathbf{A}} FE
 \end{array}$$



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

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

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



©: Michael Kohlhasse

38



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

▷ **Proof:** We show the two directions separately

**P.1** If  $\mathcal{H}, \mathbf{A} \vdash_{\mathcal{MD}^0} \mathbf{B}$ , then  $\mathcal{H} \vdash_{\mathcal{MD}^0} \mathbf{A} \Rightarrow \mathbf{B}$  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  $\Rightarrow E$ .



©: Michael Kohlhasse

39



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 5.2.6** A **judgment** is a meta-statement about the provability of propositions

▷ **Definition 5.2.7** 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 5.2.8**

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

$$\frac{\mathbf{A} \wedge \mathbf{B} \vdash \mathbf{B} \wedge \mathbf{A}}{\emptyset \vdash \mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A}} \Rightarrow I$$

▷ **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 5.2.9** The following inference rules make up the **sequent calculus**

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

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

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

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

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

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



## First-Order Natural Deduction in Sequent Formulation

▷ Rules for propositional connectives just as always

▷ **Definition 5.2.10 (New Quantifier Rules)**

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

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



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

### 5.3 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.<sup>4</sup>

EdN:4

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  $\Phi$  is abstract consistent, then  $\Phi$  can be extended to a Hintikka set.
- ▷ **Corollary:** If  $\Phi$  is abstract consistent, then  $\Phi$  is satisfiable
- ▷ **Application:** Let  $\mathcal{C}$  be a calculus, if  $\Phi$  is  $\mathcal{C}$ -consistent, then  $\Phi$  is abstract consistent.
- ▷ **Corollary:**  $\mathcal{C}$  is complete.



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  $\Phi$  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.

To carry out the “model-existence”/“abstract consistency” method, we will first have to look at the notion of consistency.

<sup>4</sup>EdNOTE: cite the original papers!



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 5.3.1**  $\Phi$  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 5.3.2** We call a pair  $\mathbf{A}$  and  $\neg \mathbf{A}$  a **contradiction**.
- ▷ So a set  $\Phi$  is  $\mathcal{C}$ -refutable, if  $\mathcal{C}$  can derive a contradiction from it.
- ▷ **Definition 5.3.3**  $\Phi$  is called  **$\mathcal{C}$ -consistent**, iff there is a formula  $\mathbf{B}$ , that is not derivable from  $\Phi$  in  $\mathcal{C}$ .
- ▷ **Definition 5.3.4** 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 5.3.5**  $\mathcal{C}$ -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  $\Phi$  of sentences is considered “consistent in an abstract sense”, iff it is a member of an abstract consistency class  $\nabla$ .

## Abstract Consistency

- ▷ **Definition 5.3.6** Let  $\nabla$  be a family of sets. We call  $\nabla$  **closed under subsets**, iff for each  $\Phi \in \nabla$ , all subsets  $\Psi \subseteq \Phi$  are elements of  $\nabla$ .
- ▷ **Notation 5.3.7** We will use  $\Phi * \mathbf{A}$  for  $\Phi \cup \{\mathbf{A}\}$ .
- ▷ **Definition 5.3.8** A family  $\nabla \subseteq \text{wff}_o(\Sigma)$  of sets of formulae is called a (first-order) **abstract consistency class**, iff it is closed under subsets, and for each  $\Phi \in \nabla$ 
  - $\nabla_c$ )  $\mathbf{A} \notin \Phi$  or  $\neg \mathbf{A} \notin \Phi$  for atomic  $\mathbf{A} \in \text{wff}_o(\Sigma)$ .
  - $\nabla_{\neg}$ )  $\neg \neg \mathbf{A} \in \Phi$  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  $\Phi$ , then  $\Phi * \neg[c/X](\mathbf{A}) \in \nabla$



The conditions are very natural: Take for instance  $\nabla_c$ , it would be foolish to call a set  $\Phi$  of sentences “consistent under a complete calculus”, if it contains an elementary contradiction. The next condition  $\nabla_{\neg}$  says that if a set  $\Phi$  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 5.3.9** We call a collection  $\nabla$  of sets **compact**, iff for any set  $\Phi$  we have  $\Phi \in \nabla$ , iff  $\Psi \in \nabla$  for every finite subset  $\Psi$  of  $\Phi$ .

▷ **Lemma 5.3.10** If  $\nabla$  is compact, then  $\nabla$  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  $\nabla$  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  $\nabla$  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 5.3.11** Any first-order abstract consistency class can be extended to a compact one.

▷ **Proof:**

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

**P.2** Now suppose that  $\Phi \in \nabla$ .  $\nabla$  is closed under subsets, so every finite subset of  $\Phi$  is in  $\nabla$  and thus  $\Phi \in \nabla'$ . Hence  $\nabla \subseteq \nabla'$ .

**P.3** Let us now show that each  $\nabla'$  is compact.

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

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

**P.3.3** Thus all finite subsets of  $\Phi$  are in  $\nabla'$  whenever  $\Phi$  is in  $\nabla'$ .

**P.3.4** On the other hand, suppose all finite subsets of  $\Phi$  are in  $\nabla'$ .

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

**P.4** Note that  $\nabla'$  is closed under subsets by the Lemma above.

**P.5** Next we show that if  $\nabla$  satisfies  $\nabla_*$ , then  $\nabla'$  satisfies  $\nabla_*$ .

**P.5.1** To show  $\nabla_c$ , 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  $\nabla_c$ .

**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  $\Psi$  be any finite subset of  $\Phi * \mathbf{A}$ , and  $\Theta := ((\Psi \setminus \{\mathbf{A}\}) * \neg\neg\mathbf{A})$ .

**P.5.2.2**  $\Theta$  is a finite subset of  $\Phi$ , so  $\Theta \in \nabla$ .

**P.5.2.3** Since  $\nabla$  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  $\nabla$  is closed under subsets, so  $\Psi \in \nabla$ .

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

**P.5.3** the other cases are analogous to  $\nabla_{\neg}$ . □



Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

## $\nabla$ -Hintikka Set

▷ **Definition 5.3.12** Let  $\nabla$  be an abstract consistency class, then we call a set  $\mathcal{H} \in \nabla$  a  **$\nabla$ -Hintikka Set**, iff  $\mathcal{H}$  is maximal in  $\nabla$ , i.e. for all  $\mathbf{A}$  with  $\mathcal{H} * \mathbf{A} \in \nabla$  we already have  $\mathbf{A} \in \mathcal{H}$ .

▷ **Theorem 5.3.13 (Hintikka Properties)** Let  $\nabla$  be an abstract consistency class and  $\mathcal{H}$  be a  $\nabla$ -Hintikka set, then

$\mathcal{H}_c$ ) For all  $\mathbf{A} \in \text{wff}_o(\Sigma)$  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  $\mathbf{A}$

**IP21**  $\mathbf{A}$  atomic: Then  $\mathbf{A} \notin \mathcal{H}$  or  $\neg\mathbf{A} \notin \mathcal{H}$  by  $\nabla_c$ .

**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  $\nabla$ .

**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  $\Phi$  it allows us to construct a Hintikka set  $\mathcal{H}$  with  $\Phi \in \mathcal{H}$ .

#### P.4 Extension Theorem

▷ **Theorem 5.3.14** *If  $\nabla$  is an abstract consistency class and  $\Phi \in \nabla$  finite, then there is a  $\nabla$ -Hintikka set  $\mathcal{H}$  with  $\Phi \subseteq \mathcal{H}$ .*

▷ **Proof:** Wlog. assume that  $\nabla$  compact (else use compact extension)

**P.1** Choose an enumeration  $\mathbf{A}^1, \mathbf{A}^2, \dots$  of  $\text{cwff}_o(\Sigma_\iota)$  and  $c^1, c^2, \dots$  of  $\Sigma_0^{sk}$ .

**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  $\nabla$  closed under subsets and  $\mathcal{H} \in \nabla$  as  $\nabla$  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  $\nabla$ -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  $\nabla$ , 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  $\Phi$ , but depends on the choice of the enumeration of  $\text{cwff}_o(\Sigma_\iota)$ . 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  $\nabla$ -consistent<sup>5</sup> with  $\Phi$ , 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:5

#### Valuation

▷ **Definition 5.3.15** A function  $\nu: \text{cwff}_o(\Sigma_\iota) \rightarrow \mathcal{D}_o$  is called a (first-order) **valuation**, iff

▷  $\nu(\neg\mathbf{A}) = \text{T}$ , iff  $\nu(\mathbf{A}) = \text{F}$

▷  $\nu(\mathbf{A} \wedge \mathbf{B}) = \text{T}$ , iff  $\nu(\mathbf{A}) = \text{T}$  and  $\nu(\mathbf{B}) = \text{T}$

<sup>5</sup>EdNOTE: introduce this above

▷  $\nu(\forall X.A) = \top$ , iff  $\nu([B/X](A)) = \top$  for all closed terms  $B$ .

▷ **Lemma 5.3.16** *If  $\varphi: \mathcal{V}_\iota \rightarrow \mathcal{D}$  is a variable assignment, then  $\mathcal{I}_\varphi: \text{cwff}_o(\Sigma_\iota) \rightarrow \mathcal{D}_o$  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.<sup>6</sup>

EdN:6

## Valuation and Satisfiability

▷ **Lemma 5.3.17** *If  $\nu: \text{cwff}_o(\Sigma_\iota) \rightarrow \mathcal{D}_o$  is a valuation and  $\Phi \subseteq \text{cwff}_o(\Sigma_\iota)$  with  $\nu(\Phi) = \{\top\}$ , then  $\Phi$  is satisfiable.*

▷ **Proof:** We construct a model for  $\Phi$ .

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

- ▷  $\mathcal{I}(f): \mathcal{D}_\iota^k \mapsto \mathcal{D}_\iota \langle \mathbf{A}_1, \dots, \mathbf{A}_k \rangle f(\mathbf{A}_1, \dots, \mathbf{A}_k)$  for  $f \in \Sigma^f$
- ▷  $\mathcal{I}(p): \mathcal{D}_\iota^k \mapsto \mathcal{D}_o \langle \mathbf{A}_1, \dots, \mathbf{A}_k \rangle \nu(p(\mathbf{A}_1, \dots, \mathbf{A}_n))$  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  $\mathbf{A} \in \text{wff}_\iota(\Sigma_\iota)$  by induction on  $\mathbf{A}$

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

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

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

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

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

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

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

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

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



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

## Model Existence

▷ **Theorem 5.3.18 (Hintikka-Lemma)** *If  $\nabla$  is an abstract consistency class and  $\mathcal{H}$  a  $\nabla$ -Hintikka set, then  $\mathcal{H}$  is satisfiable.*

<sup>6</sup>EdNOTE: I think that we only get a semivaluation, look it up in Andrews.

▷ **Proof:**

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

**P.2** then  $\nu$  is a valuation by the Hintikka set properties.

**P.3** We have  $\nu(\mathcal{H}) = \{\top\}$ , so  $\mathcal{H}$  is satisfiable.  $\square$

▷ **Theorem 5.3.19 (Model Existence)** *If  $\nabla$  is an abstract consistency class and  $\Phi \in \nabla$ , then  $\Phi$  is satisfiable.*

**Proof:**

▷ **P.1** There is a  $\nabla$ -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.  $\square$



## 5.4 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 5.4.1 (Non-Refutability is an Abstract Consistency Property)**  $\Gamma := \{\Phi \subseteq \text{wff}_o(\Sigma_\iota) \mid \Phi \text{ not } \mathcal{ND}^1\text{-refutable}\}$  is an abstract consistency class.

▷ **Proof:** We check the properties of an ACC

**P.1** If  $\Phi$  is non-refutable, then any subset is as well, so  $\Gamma$  is closed under subsets.

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

**P.2.1**  $\nabla_c$ : We have to show that  $\mathbf{A} \notin \Phi$  or  $\neg\mathbf{A} \notin \Phi$  for atomic  $\mathbf{A} \in \text{wff}_o(\Sigma)$ .

**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  $\Phi$  is  $\mathcal{ND}^1$ -refutable by construction.

**P.2.1.4** So  $\Phi \notin \Gamma$ .  $\square$

**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  $\mathcal{D}: \Phi * \mathbf{A} \vdash_{\mathcal{ND}^1} F$

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

**P.2.2.5** Thus  $\Phi \notin \Gamma$ .  $\square$

**P.2.3** other  $\nabla_*$  similar:



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

### Henkin's Theorem

▷ **Corollary 5.4.2 (Henkin's Theorem)** *Every  $\mathcal{ND}^1$ -consistent set of sentences has a model.*

▷ **Proof:**

**P.1** Let  $\Phi$  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  $\Phi$ .  $\square$

▷ **Corollary 5.4.3 (Löwenheim&Skolem Theorem)** *Satisfiable set  $\Phi$  of first-order sentences has a countable model.*

▷ **Proof Sketch:** The model we constructed is countable, since the set of ground terms is.  $\square$



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 5.4.4 (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  $\Phi$ , 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  $\neg E$ .  $\square$

▷ **Theorem 5.4.5 (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.  $\square$



## 5.5 Limits of First-Order Logic

We will now come to the limits of first-order Logic.<sup>7</sup>

EdN:7

### P.2 Gödel's Incompleteness Theorem

<sup>7</sup>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.

▷ **Theorem 5.5.1** *No logical system that can 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  $\mathbf{A}_{\mathcal{C}}$ . □





## Chapter 6

# First-Order Inference with Tableaux

### 6.1 First-Order Tableaux

#### Test Calculi: Tableaux and Model Generation

▷ **Idea:** instead of showing  $\emptyset \vdash Th$ , show  $\neg Th \vdash \text{trouble}$  (use  $\perp$  for trouble)

▷ **Example 6.1.1** Tableau Calculi try to construct models.

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

**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  $\perp$ .

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

$$\frac{\mathbf{A} \wedge \mathbf{B}^T}{\mathbf{A}^T \mid \mathbf{B}^T} \mathcal{T}_0 \wedge \quad \frac{\mathbf{A} \wedge \mathbf{B}^F}{\mathbf{A}^F \mid \mathbf{B}^F} \mathcal{T}_0 \vee \quad \frac{\neg \mathbf{A}^T}{\mathbf{A}^F} \mathcal{T}_0 \neg^T \quad \frac{\neg \mathbf{A}^F}{\mathbf{A}^T} \mathcal{T}_0 \neg^F \quad \frac{\mathbf{A}^\alpha \mid \mathbf{A}^\beta \quad \alpha \neq \beta}{\perp} \mathcal{T}_0 \text{cut}$$

- ▷ Use rules exhaustively as long as they contribute new material
- ▷ **Definition 6.1.2** Call a tableau **saturated**, iff no rule applies, and a branch **closed**, iff it ends in  $\perp$ , else **open**. (open branches in saturated tableaux yield models)
- ▷ **Definition 6.1.3 ( $\mathcal{T}_0$ -Theorem/Derivability)**  $\mathbf{A}$  is a  **$\mathcal{T}_0$ -theorem** ( $\vdash_{\mathcal{T}_0} \mathbf{A}$ ), iff there is a closed tableau with  $\mathbf{A}^F$  at the root.
- $\Phi \subseteq \text{wff}_o(\mathcal{V}_o)$  **derives**  $\mathbf{A}$  in  $\mathcal{T}_0$  ( $\Phi \vdash_{\mathcal{T}_0} \mathbf{A}$ ), iff there is a closed tableau starting with  $\mathbf{A}^F$  and  $\Phi^T$ .



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  $\perp$  (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

**Definition 6.1.4** 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  $\alpha$ ; since all branches are closed (contain contradictions) this is impossible.

**Definition 6.1.5** We will call a tableau refutation for  $\mathbf{A}^F$  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 techniques 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: propositional tableau rules plus

$$\frac{\forall X.\mathbf{A}^\top \quad \mathbf{C} \in \text{cwoff}_\iota(\Sigma_\iota)}{[C/X](\mathbf{A})^\top} \mathcal{T}_1:\forall \quad \frac{\forall X.\mathbf{A}^\text{F} \quad c \in (\Sigma_0^{sk} \setminus \mathcal{H})}{[c/X](\mathbf{A})^\text{F}} \mathcal{T}_1:\exists$$



©: Michael Kohlhase

59



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  $\forall X.\mathbf{A}^\text{F}$  existentially — i.e. as  $\exists X.\neg\mathbf{A}^\top$ , 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  $\Sigma_0^{sk}$ , which we have given ourselves expressly for this purpose when we defined first-order syntax. In other words  $[c/X](\neg\mathbf{A})^\top = [c/X](\mathbf{A})^\text{F}$  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  $\mathbf{C} \in \text{woff}_\iota(\Sigma_\iota)$  for  $X$ . This makes the rule infinitely branching.

## 6.2 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

▷ Tableau rules

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

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

$$\frac{\mathbf{A}^\alpha \quad \mathbf{B}^\beta \quad \alpha \neq \beta \quad \sigma(\mathbf{A}) = \sigma(\mathbf{B})}{\perp} \mathcal{T}_1^f:\perp$$

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

▷ **New:** find suitable substitution

(most general unifier)



©: Michael Kohlhase

60



**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  $\mathcal{T}_1^f:\perp$  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.

## 6.3 First-Order Unification

We will now look into the problem of finding a substitution  $\sigma$  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<sup>8</sup>

EdN:8

$$\text{computation} = \text{logic} + \text{control}$$

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  $\sigma$ , such that  $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$ .
- ▷ **Notation 6.3.1** 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 6.3.2** Let  $\sigma$  and  $\theta$  be substitutions and  $W \subseteq \mathcal{V}_t$ , we say that a substitution  $\sigma$  is **more general** than  $\theta$  (on  $W$  write  $\sigma \leq \theta[W]$ ), iff there is a substitution  $\rho$ , such that  $\theta = \rho \circ \sigma[W]$ , where  $\sigma = \rho[W]$ , iff  $\sigma(X) = \rho(X)$  for all  $X \in W$ .
- ▷ **Definition 6.3.3**  $\sigma$  is called a **most general unifier** of  $\mathbf{A}$  and  $\mathbf{B}$ , iff it is minimal in  $\mathbf{U}(\mathbf{A} =^? \mathbf{B})$  wrt.  $\leq [\text{free}(\mathbf{A}) \cup \text{free}(\mathbf{B})]$ .

<sup>8</sup>EDNOTE: find the reference, and see what he really said



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).

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 6.3.4** We call a formula  $\mathbf{A}^1 =? \mathbf{B}^1 \wedge \dots \wedge \mathbf{A}^n =? \mathbf{B}^n$  an **equational system** iff  $\mathbf{A}^i, \mathbf{B}^i \in \text{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$ .
- ▷ We consider equational systems as sets of equations ( $\wedge$  is ACI), and equations as two-element multisets ( $=?$  is C).
- ▷ **Definition 6.3.5** We call a pair  $\mathbf{A} =? \mathbf{B}$  **solved** in a unification problem  $\mathcal{E}$ , iff  $\mathbf{A} = X$ ,  $\mathcal{E} = X =? \mathbf{A} \wedge cE$ , and  $X \notin (\text{free}(\mathbf{A}) \cup \text{free}(\mathcal{E}))$ . We call an unification problem  $\mathcal{E}$  a **solved form**, iff all its pairs are solved.
- ▷ **Lemma 6.3.6** Solved forms are of the form  $X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$  where the  $X^i$  are distinct and  $X^i \notin \text{free}(\mathbf{B}^j)$ .
- ▷ **Lemma 6.3.7** If  $\mathcal{E} = X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$  is a solved form, then  $\mathcal{E}$  has the unique most general unifier  $\sigma_\mathcal{E} := [\mathbf{B}^1/X^1], \dots, [\mathbf{B}^n/X^n]$ .
- ▷ **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  $\theta = \theta \circ \sigma_\mathcal{E}[\text{supp}(\sigma)]$ . □

**Note:** we can rename the introduced variables in most general unifiers!



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.

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 6.3.7 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 6.3.8** Inference system  $\mathcal{U}$

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1, \dots, \mathbf{A}^n) = ? f(\mathbf{B}^1, \dots, \mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 = ? \mathbf{B}^1 \wedge \dots \wedge \mathbf{A}^n = ? \mathbf{B}^n} \mathcal{U}_{\text{dec}} \quad \frac{\mathcal{E} \wedge \mathbf{A} = ? \mathbf{A}}{\mathcal{E}} \mathcal{U}_{\text{triv}}$$

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

▷ **Lemma 6.3.9**  $\mathcal{U}$  is *correct*:  $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$  implies  $\mathbf{U}(\mathcal{F}) \subseteq \mathbf{U}(\mathcal{E})$

▷ **Lemma 6.3.10**  $\mathcal{U}$  is *complete*:  $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$  implies  $\mathbf{U}(\mathcal{E}) \subseteq \mathbf{U}(\mathcal{F})$

▷ **Lemma 6.3.11**  $\mathcal{U}$  is *confluent*: the order of derivations does not matter

▷ **Corollary 6.3.12** 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  $\mathcal{U}_{\text{dec}}$  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  $\mathcal{U}_{\text{triv}}$  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  $\mathcal{U}_{\text{dec}}$  rule in the somewhat degenerate case without arguments.

Finally, we observe that the first of the two variable conditions in  $\mathcal{U}_{\text{elim}}$  (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 6.3.13** Two similar unification problems:

$  \begin{array}{c}  f(g(x, x), h(a)) = ? f(g(a, z), h(z)) \\  \hline  g(x, x) = ? g(a, z) \wedge h(a) = ? h(z) \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? a \wedge x = ? z \wedge h(a) = ? h(z) \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? a \wedge x = ? z \wedge a = ? z \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? a \wedge x = ? z \wedge a = ? z \quad \mathcal{U}_{\text{elim}} \\  \hline  x = ? a \wedge a = ? z \wedge a = ? z \quad \mathcal{U}_{\text{elim}} \\  \hline  x = ? a \wedge z = ? a \wedge a = ? a \quad \mathcal{U}_{\text{triv}} \\  \hline  x = ? a \wedge z = ? a  \end{array}  $	$  \begin{array}{c}  f(g(x, x), h(a)) = ? f(g(b, z), h(z)) \\  \hline  g(x, x) = ? g(b, z) \wedge h(a) = ? h(z) \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? b \wedge x = ? z \wedge h(a) = ? h(z) \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? b \wedge x = ? z \wedge a = ? z \quad \mathcal{U}_{\text{dec}} \\  \hline  x = ? b \wedge x = ? z \wedge a = ? z \quad \mathcal{U}_{\text{elim}} \\  \hline  x = ? b \wedge b = ? z \wedge a = ? z \quad \mathcal{U}_{\text{elim}} \\  \hline  x = ? a \wedge z = ? a \wedge a = ? b  \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 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 6.3.14** 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 6.3.15** If  $\prec$  is total/terminating on  $S$ , then  $\prec^m$  is total/terminating on  $\mathcal{P}(S)$ .

▷ **Lemma 6.3.16**  $\mathcal{U}$  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

- ▷  $n$  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**  $\mathcal{U}_{\text{dec}}$  and  $\mathcal{U}_{\text{triv}}$  decrease the multiset of term depths without increasing the unsolved variables

**P.2.2**  $\mathcal{U}_{\text{elim}}$  decreases the number of unsolved variables (by one), but may increase term depths.  $\square$



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 6.3.17** 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 6.3.18** 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**  $\mathbf{A}, \mathbf{B} \notin \mathcal{V}_i$ : then  $\mathbf{A} = f(\mathbf{A}^1 \dots \mathbf{A}^n)$  and  $\mathbf{B} = f(\mathbf{B}^1 \dots \mathbf{B}^n)$ , and thus  $\mathcal{U}_{\text{dec}}$  is applicable

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

▷ **Corollary 6.3.19** Unification is decidable in  $PL^1$ .

▷ **Proof Idea:**  $\mathcal{U}$ -irreducible sets of equations can be obtained in finite time by Lemma 6.3.16 and are either solved or unsolvable by Lemma 6.3.18, so they provide the answer.  $\square$



## 6.4 Efficient Unification

### Complexity of Unification

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

▷ consider the terms

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

▷ The most general unifier of  $s_n$  and  $t_n$  is

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

▷ 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

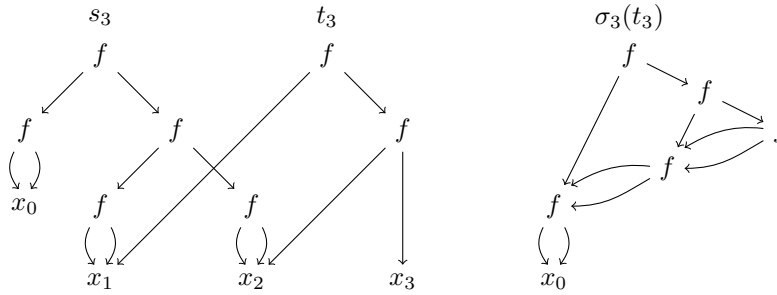


- ▷ **Idea:** Find a term representation that re-uses subterms



## Directed Acyclic Graphs (DAGs)

- ▷ use directed acyclic graphs for the term representation
- ▷ variables may only occur once in the DAG
  - ▷ subterms can be referenced multiply
- ▷ **Observation 6.4.1** Terms can be transformed into DAGs in linear time
- ▷ **Example 6.4.2**  $s_3, t_3, \sigma_3(s_3)$



## DAG Unification Algorithm

- ▷ **Definition 6.4.3** We say that  $X^1 = ? B^1 \wedge \dots \wedge X^n = ? B^n$  is a **DAG solved form**, iff the  $X^i$  are distinct and  $X^i \notin \text{free}(B^j)$  for  $i \leq j$
- ▷ **Definition 6.4.4** The inference system  $\mathcal{DU}$  contains rules  $\mathcal{Udec}$  and  $\mathcal{Utriv}$  from  $\mathcal{U}$  plus the following:

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

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

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)
  - ▷  $\text{find}(n)$  follows the path from  $n$  and returns the end node
  - ▷  $\text{union}(n, m)$  adds an edge between  $n$  and  $m$ .
  - ▷  $\text{occur}(n, m)$  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  $t'_n = [y_i/x_i](t_n)$ .
  - ▷  $\text{unify}$  needs exponentially many recursive calls to unify the nodes  $x_n$  and  $y_n$ . (they are unified after  $n$  calls, but checking
  - ▷ **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  $\text{find}$ .
  - ▷ linearly many calls to linear procedures give quadratic runtime.



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.

## 6.5 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 6.5.1** A labeled formula  $\mathbf{A}^\alpha$  is valid under  $\varphi$ , iff  $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$ .
- ▷ **Definition 6.5.2** 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 6.5.3** *Tableau rules transform satisfiable tableaux into satisfiable ones.*
- ▷ **Theorem 6.5.4 (Soundness)** *A set  $\Phi$  of propositional formulae is valid, if there is a closed tableau  $\mathcal{T}$  for  $\Phi^F$ .*
- ▷ **Proof:** by contradiction: Suppose  $\Phi$  is not valid.
  - P.1 then the initial tableau is satisfiable ( $\Phi^F$  satisfiable)
  - P.2  $\mathcal{T}$  satisfiable, by our Lemma.
  - 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 6.5.3, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains  $\mathbf{A} \wedge \mathbf{B}^T$  and is satisfiable, then it must have a satisfiable branch. If  $\mathbf{A} \wedge \mathbf{B}^T$  is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus  $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = T$  for some variable assignment  $\varphi$ . Thus  $\mathcal{I}_\varphi(\mathbf{A}) = T$  and  $\mathcal{I}_\varphi(\mathbf{B}) = T$ , so after the extension (which adds the formulae  $\mathbf{A}^T$  and  $\mathbf{B}^T$  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 6.5.5** *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 6.5.7
    - P.1.3  $\mathcal{T}_1^f : \perp$ : by Lemma 4.2.12(substitution value lemma)
    - P.1.4  $\mathcal{T}_1^f : \forall$ :
      - P.1.4.1  $\mathcal{I}_\varphi(\forall X. \mathbf{A}) = T$ , iff  $\mathcal{I}_\psi(\mathbf{A}) = T$  for all  $a \in \mathcal{D}_\iota$
      - P.1.4.2 so in particular for some  $a \in \mathcal{D}_\iota \neq \emptyset$ .

□

□

- ▷ **Corollary 6.5.6**  $\mathcal{T}_1^f$  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 6.5.7**  $\mathcal{T}_1^f : \exists$  transforms satisfiable tableaux into satisfiable ones.

▷ **Proof:** Let  $\mathcal{T}'$  be obtained by applying  $\mathcal{T}_1^f : \exists$  to  $\forall X. \mathbf{A}^F$  in  $\mathcal{T}$ , extending it with  $[f(X^1, \dots, X^n)/X](\mathbf{A})^F$ , where  $W := \text{free}(\forall X. \mathbf{A}) = \{X^1, \dots, X^k\}$

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

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

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

**P.4** Let  $g : \mathcal{D}^k \rightarrow \mathcal{D}$  be defined by  $g(a_1, \dots, a_k) := a$ , 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

$$\mathcal{I}'_\varphi([f(X^1, \dots, X^k)/X](\mathbf{A})) = \mathcal{I}'_{\varphi, [g/f]}(\mathbf{A}) = \mathcal{I}'_{\varphi, [a/X]}(\mathbf{A}) = F$$

**P.6** So  $[f(X^1, \dots, X^k)/X](\mathbf{A})^F$  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 5.3.19), 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 6.5.8**  $\mathcal{T}_1^f$  is refutation complete.

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

**P.1** ( $\nabla_c, \nabla_\neg, \nabla_\vee$ , 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  $\Phi^T * [c/X](\mathbf{A})^F \in \nabla$ .

$$\begin{array}{ccc} \Psi^T & & \Psi^T \\ \forall X. \mathbf{A}^F & & \forall X. \mathbf{A}^F \\ [c/X](\mathbf{A})^F & & [f(X^1, \dots, X^k)/X](\mathbf{A})^F \\ \text{Rest} & & [f(X^1, \dots, X^k)/c](\text{Rest}) \end{array}$$

□



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 6.5.9** *If  $\mathcal{T}_\theta$  is a closed tableau for a st  $\theta(\Phi)$  of formulae, then there is a closed tableau  $\mathcal{T}$  for  $\Phi$ .*

▷ **Proof:** by induction over the structure of  $\mathcal{T}_\theta$  we build an isomorphic tableau  $\mathcal{T}$ , and a tableau-isomorphism  $\omega: \mathcal{T} \rightarrow \mathcal{T}_\theta$ , such that  $\omega(\mathbf{A}) = \theta(\mathbf{A})$ .

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

**P.2** Let  $\theta(\mathbf{A}^i)^\top$  and  $\theta(\mathbf{B}^i)^\top$  cut formulae in the branch  $\Theta_\theta^i$  of  $\mathcal{T}_\theta$

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

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

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

**P.6** so  $\Theta$  is closed □



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



Part II

Higher-Order Logic and  
 $\lambda$ -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  $\lambda$ -calculus, which answers most of these questions in a term-level, computation-friendly system.

The formal development of the simply typed  $\lambda$ -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 7

# 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 ( $PL\Omega$ )

- ▷ Quantification over functions and Predicates:  $\forall P.\exists F.P(a) \vee \neg P(F(a))$
- ▷ **Comprehension**: (Existence of Functions)  
 $\exists F.\forall X.FX = A$  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**)  
 $A = B$  for  $\forall P.PA \Rightarrow PB$



©: Michael Kohlhase

77



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 $PL\Omega$

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

- ▷ vector notation:  $\overline{\alpha_n} \rightarrow \beta$  abbreviates  $\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$
- ▷ 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 7.0.10** Given a set  $\mathcal{BT}$  of **base types**, construct **function types**:  $\alpha \rightarrow \beta$  is the type of functions with **domain type**  $\alpha$  and **range type**  $\beta$ . We call the closure  $\mathcal{T}$  of  $\mathcal{BT}$  under function types the set of **types** over  $\mathcal{BT}$ .
- ▷ **Definition 7.0.11** We will use  $\iota$  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  $\alpha \rightarrow \beta \rightarrow \gamma$  as an abbreviation for  $\alpha \rightarrow (\beta \rightarrow \gamma)$
- ▷ We will use a kind of vector notation for function types, abbreviating  $\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$  with  $\overline{\alpha_n} \rightarrow \beta$ .



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 ( $PL\Omega$ )

- ▷ **signature**  $\Sigma = \bigcup_{\alpha \in \mathcal{T}} \Sigma_\alpha$  with
- ▷ **connectives**:  $\neg \in \Sigma_{o \rightarrow o} \quad \{\vee, \wedge, \Rightarrow, \Leftrightarrow \dots\} \subseteq \Sigma_{o \rightarrow o \rightarrow o}$
- ▷ **variables**  $\mathcal{V}_\mathcal{T} = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ , such that every  $\mathcal{V}_\alpha$  countably infinite.

- ▷ **well-typed formulae**  $wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$  of type  $\alpha$ 
  - ▷  $\mathcal{V}_\alpha \cup \Sigma_\alpha \subseteq wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$
  - ▷ If  $\mathbf{C} \in wff_{(\alpha \rightarrow \beta)}(\Sigma, \mathcal{V}_\mathcal{T})$  and  $\mathbf{A} \in wff_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ , then  $(\mathbf{C}\mathbf{A}) \in wff_\beta(\Sigma, \mathcal{V}_\mathcal{T})$
  - ▷ If  $\mathbf{A} \in wff_o(\Sigma, \mathcal{V}_\mathcal{T})$ , then  $(\forall X_\alpha.\mathbf{A}) \in wff_o(\Sigma, \mathcal{V}_\mathcal{T})$
- ▷ first-order terms have type  $\iota$ , formulae (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  $\alpha$  as well as  $\alpha \rightarrow o$ .



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

- ▷ **Definition 7.0.12** The **universe** of discourse (also **carrier**)
  - ▷ arbitrary, non-empty **set of individuals**  $\mathcal{D}_\iota$
  - ▷ fixed **set of truth values**  $\mathcal{D}_o = \{\mathbf{T}, \mathbf{F}\}$
  - ▷ **function universes**  $\mathcal{D}_{\alpha \rightarrow \beta} = \mathcal{F}(\mathcal{D}_\alpha; \mathcal{D}_\beta)$
- ▷ **interpretation of constants**: typed mapping  $\mathcal{I}: \Sigma \rightarrow \mathcal{D}$  (i.e.  $\mathcal{I}(\Sigma_\alpha) \subseteq \mathcal{D}_\alpha$ )
- ▷ **Definition 7.0.13** 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  $\text{PL}\Omega$ .
- ▷ **variable assignment**: typed mapping  $\varphi: \mathcal{V}_\mathcal{T} \rightarrow \mathcal{D}$
- ▷ **Definition 7.0.14 value function**: typed mapping  $\mathcal{I}_\varphi: wff_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T}) \rightarrow \mathcal{D}$ 
  - ▷  $\mathcal{I}_\varphi|_{\mathcal{V}_\mathcal{T}} = \varphi$        $\mathcal{I}_\varphi|_{\Sigma_\mathcal{T}} = \mathcal{I}$
  - ▷  $\mathcal{I}_\varphi(\mathbf{AB}) = \mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
  - ▷  $\mathcal{I}_\varphi(\forall X_\alpha.\mathbf{A}) = \mathbf{T}$ , iff  $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = \mathbf{T}$  for all  $a \in \mathcal{D}_\alpha$ .
- ▷  $\mathbf{A}_o$  **valid** under  $\varphi$ , iff  $\mathcal{I}_\varphi(\mathbf{A}) = \mathbf{T}$ .



We now go through a couple of examples of what we can express in  $\text{PL}\Omega$ , and that works out very straightforwardly. For instance, we can express equality in  $\text{PL}\Omega$  by Leibniz equality, and it has the right meaning.

## Equality

- ▷ “Leibniz equality” (**Indiscernability**)  $\mathbf{Q}^\alpha \mathbf{A}_\alpha \mathbf{B}_\alpha = \forall P_{\alpha \rightarrow o}. P\mathbf{A} \Leftrightarrow P\mathbf{B}$
- ▷ not that  $\forall P_{\alpha \rightarrow o}. P\mathbf{A} \Rightarrow P\mathbf{B}$  (get the other direction by instantiating  $P$  with  $Q$ , where  $QX \Leftrightarrow \neg PX$ )
- ▷ **Theorem 7.0.15** 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 7.0.16** 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**  $\mathcal{I}_\varphi(\mathbf{QAB}) = \mathcal{I}_\varphi(\forall P.PA \Rightarrow PB) = \mathsf{T}$ , iff  
 $\mathcal{I}_{\varphi, [r/P]}(PX \Rightarrow PY) = \mathsf{T}$  for all  $r \in \mathcal{D}_{\alpha \rightarrow o}$ .

**P.2** For  $\mathbf{A} = \mathbf{B}$  we have  $\mathcal{I}_{\varphi, [r/P]}(PA) = r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{F}$  or  $\mathcal{I}_{\varphi, [r/P]}(PA) = r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{T}$ .

**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  $\mathcal{I}_{\varphi, [r/P]}(PA \Rightarrow PB) = \mathsf{F}$ , since  $\mathcal{I}_{\varphi, [r/P]}(PA) = r(\mathcal{I}_\varphi(\mathbf{A})) = \mathsf{T}$  and  
 $\mathcal{I}_{\varphi, [r/P]}(PB) = r(\mathcal{I}_\varphi(\mathbf{B})) = \mathsf{F}$ .  $\square$



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

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

▷  $\mathbb{N}0$  (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)

▷  $\forall P_{\iota \rightarrow o}. P0 \Rightarrow (\forall X_\iota. \mathbb{N}X \Rightarrow PX \Rightarrow P(sX)) \Rightarrow (\forall Y_\iota. \mathbb{N}Y \Rightarrow P(Y))$   
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  $\text{PL}\Omega$  by formalizing a version of Cantor's theorem.

### Expressive Formalism for Mathematics

▷ **Example 7.0.17 (Cantor's Theorem)** The cardinality of a set is smaller than that of its power set.

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

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

**Simplified Formalization:**  $\neg \exists F_{\iota \rightarrow \iota \rightarrow \iota}. \forall G_{\iota \rightarrow \iota}. \exists J_\iota. FJ = G$

▷ Standard-Benchmark for higher-order theorem provers

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



The next concern is to find a calculus for  $\text{PL}\Omega$ .

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  $\text{PL}\Omega$  formulae and insisting that substitutions are well-typed.

## Hilbert-Calculus

▷ **Definition 7.0.18 ( $\mathcal{H}_\Omega$  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 7.0.19 ( $\mathcal{H}_\Omega$  Inference rules)**

$$\frac{A_o \Rightarrow B_o \quad \mathbf{A}}{\mathbf{B}} \quad \frac{\forall X_\alpha. \mathbf{A}}{[\mathbf{B}/X_\alpha](\mathbf{A})} \quad \frac{\mathbf{A}}{\forall X_\alpha. \mathbf{A}} \quad \frac{X \notin \text{free}(\mathbf{A}) \quad \forall X_\alpha. \mathbf{A} \wedge \mathbf{B}}{\mathbf{A} \wedge (\forall X_\alpha. \mathbf{B})}$$

▷ **Theorem 7.0.20** *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  $\iota \rightarrow \iota$ , 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:**

$$\neg(\exists F_{\iota \rightarrow \iota}. \forall G_{\iota \rightarrow \iota}. (\forall K_\iota. (\mathbb{N}K) \Rightarrow \mathbb{N}(GK)) \Rightarrow (\exists J_\iota. (\mathbb{N}J) \wedge FJ = G))$$

(There is no surjective mapping from  $\mathbb{N}$  into the set  $\mathcal{F}(\mathbb{N}; \cdot)\mathbb{N}$  of natural number sequences)

▷ proof attempt fails at the subgoal  $\exists G_{\iota \rightarrow \iota}. \forall X_\iota. GX = s(fXX)$

▷ **Comprehension**  $\exists F_{\alpha \rightarrow \beta}. \forall X_\alpha. FX = \mathbf{A}_\beta$  (for every variable  $X_\alpha$  and every term  $\mathbf{A} \in \text{wff}_\beta(\Sigma, \mathcal{V}_\tau)$ )

▷ **extensionality**

$$\text{Ext}^{\alpha\beta} \quad \forall F_{\alpha \rightarrow \beta}. \forall G_{\alpha \rightarrow \beta}. (\forall X_\alpha. FX = GX) \Rightarrow F = G$$

$$\text{Ext}^o \quad \forall F_o. \forall G_o. (F \Leftrightarrow G) \Leftrightarrow F = G$$

▷ **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 7.0.21 (Henkin 1950)**  $\mathcal{H}_\Omega$  is complete wrt. this semantics.
- ▷ **Proof Sketch:** □
  - more models  $\rightsquigarrow$  less valid sentences (these are  $\mathcal{H}_\Omega$ -theorems)
- ▷ Henkin-models induce sensible measure of completeness for higher-order logic.



Actually, there is another problem with  $\text{PL}\Omega$ : 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  $\text{PL}\Omega$  are equal. Second we observe that there are countably infinitely many of them (they are parametric in the term  $\mathbf{A}$ , the type  $\alpha$  and the variable name), which makes dealing with them difficult in practice. Finally, axioms with both existential and universal quantifiers are always difficult to reason with.

Therefore we would like to have a formulation of higher-order logic without comprehension axioms. In the next slide we take a close look at the comprehension axioms and transform them into a form without quantifiers, which will turn out useful.

## From Comprehension to $\beta$ -Conversion

- ▷  $\exists F_{\alpha \rightarrow \beta} \cdot \forall X_\alpha \cdot FX = \mathbf{A}_\beta$  for arbitrary variable  $X_\alpha$  and term  $\mathbf{A} \in \text{wff}_\beta(\Sigma, \mathcal{V}_\mathcal{T})$   
(for each term  $\mathbf{A}$  and each variable  $X$  there is a function  $f \in \mathcal{D}_{\alpha \rightarrow \beta}$ , with  $f(\varphi(X)) = \mathcal{I}_\varphi(\mathbf{A})$ )
  - ▷ schematic in  $\alpha, \beta, X_\alpha$  and  $\mathbf{A}_\beta$ , very inconvenient for deduction
- ▷ Transformation in  $\mathcal{H}_\Omega$ 
  - ▷  $\exists F_{\alpha \rightarrow \beta} \cdot \forall X_\alpha \cdot FX = \mathbf{A}_\beta$
  - ▷  $\forall X_\alpha \cdot (\lambda X_\alpha \cdot \mathbf{A})X = \mathbf{A}_\beta$  ( $\exists E$ )  
Call the function  $F$  whose existence is guaranteed " $(\lambda X_\alpha \cdot \mathbf{A})$ "
  - ▷  $(\lambda X_\alpha \cdot \mathbf{A})\mathbf{B} = [\mathbf{B}/X]\mathbf{A}_\beta$  ( $\forall E$ ), in particular for  $\mathbf{B} \in \text{wff}_\alpha(\Sigma, \mathcal{V}_\mathcal{T})$ .
- ▷ **Definition 7.0.22 Axiom of  $\beta$ -equality:**  $(\lambda X_\alpha \cdot \mathbf{A})\mathbf{B} = [\mathbf{B}/X](\mathbf{A}_\beta)$
- ▷ new formulae ( $\lambda$ -calculus [Church 1940])



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

## From Extensionality to $\eta$ -Conversion

- ▷ **Definition 7.0.23 Extensionality Axiom:**  $\forall F_{\alpha \rightarrow \beta} \cdot \forall G_{\alpha \rightarrow \beta} \cdot (\forall X_\alpha \cdot FX = GX) \Rightarrow F = G$



- ▷ **Idea:** Maybe we can get by with a simplified equality schema here as well.
- ▷ **Definition 7.0.24** We say that  $\mathbf{A}$  and  $\lambda X_\alpha. \mathbf{A}X$  are  **$\eta$ -equal**, (write  $\mathbf{A}_{\alpha \rightarrow \beta} =_\eta \lambda X_\alpha. \mathbf{A}X$ , if), iff  $X \notin \text{free}(\mathbf{A})$ .
- ▷ **Theorem 7.0.25**  *$\eta$ -equality and Extensionality are equivalent*
- ▷ **Proof:** We show that  $\eta$ -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  $\eta$
- P.3** Hence  $\forall F_o. \forall G_o. (F \Leftrightarrow G) \Leftrightarrow F = G$  □
- ▷ 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  $\lambda$ -generated names we used in the transformation as (generic) witnesses for the existential quantifier. Alonzo Church did just that with his “simply typed  $\lambda$ -calculus” which we will introduce next.



## Chapter 8

# Simply Typed $\lambda$ -Calculus

In this section we will present a logic that can deal with functions – the simply typed  $\lambda$ -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 $\lambda$ -Calculus (Syntax)

▷ **Signature**  $\Sigma = \bigcup_{\alpha \in \mathcal{T}} \Sigma_{\alpha}$  (includes countably infinite Signatures  $\Sigma_{\alpha}^{Sk}$  of **Skolem constants**).

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

▷ **Definition 8.0.26** We call the set  $wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$  defined by the rules

▷  $\mathcal{V}_{\alpha} \cup \Sigma_{\alpha} \subseteq wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$

▷ If  $\mathbf{C} \in wff_{(\alpha \rightarrow \beta)}(\Sigma, \mathcal{V}_{\mathcal{T}})$  and  $\mathbf{A} \in wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$ , then  $(\mathbf{C}\mathbf{A}) \in wff_{\beta}(\Sigma, \mathcal{V}_{\mathcal{T}})$

▷ If  $\mathbf{A} \in wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$ , then  $(\lambda X_{\beta}.\mathbf{A}) \in wff_{(\beta \rightarrow \alpha)}(\Sigma, \mathcal{V}_{\mathcal{T}})$

the set of **well-typed formulae** of type  $\alpha$  over the signature  $\Sigma$  and use  $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}}) := \bigcup_{\alpha \in \mathcal{T}} wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$  for the set of all well-typed formulae.

▷ **Definition 8.0.27** 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.  $\sigma(X_{\alpha}) \in wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$  and capture-avoiding.

▷ **Definition 8.0.28 (Simply Typed  $\lambda$ -Calculus)** The **simply typed  $\lambda$ -calculus**  $\Lambda^{\rightarrow}$  over a signature  $\Sigma$  has the formulae  $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$  (they are called  **$\lambda$ -terms**) and the following equalities:

▷  **$\alpha$  conversion**:  $\lambda X.\mathbf{A} =_{\alpha} \lambda Y.[Y/X](\mathbf{A})$

▷  **$\beta$  conversion**:  $(\lambda X.\mathbf{A})\mathbf{B} =_{\beta} [\mathbf{B}/X](\mathbf{A})$

▷  **$\eta$  conversion**:  $\lambda X.\mathbf{A}X =_{\eta} \mathbf{A}$



The intuitions about functional structure of  $\lambda$ -terms and about free and bound variables are encoded into three transformation rules  $\Lambda^{\rightarrow}$ : The first rule ( $\alpha$ -conversion) just says that we can rename bound variables as we like.  $\beta$ -conversion codifies the intuition behind function application by replacing bound variables with argument. The equality relation induced by the  $\eta$ -reduction is a special case of the extensionality principle for functions ( $f = g$  iff  $f(a) = g(a)$  for all possible

arguments  $a$ ): If we apply both sides of the transformation to the same argument – say  $\mathbf{B}$  and then we arrive at the right hand side, since  $\lambda X_{\alpha}.\mathbf{A}X\mathbf{B} =_{\beta} \mathbf{A}\mathbf{B}$ .

We will use a set of bracket elision rules that make the syntax of  $\Lambda^{\rightarrow}$  more palatable. This makes  $\Lambda^{\rightarrow}$  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 $\lambda$ -Calculus (Notations)

- ▷ **Notation 8.0.29 (Application is left-associative)** We abbreviate  $((\mathbf{F}\mathbf{A}^1)\mathbf{A}^2)\dots\mathbf{A}^n$  with  $\mathbf{F}\mathbf{A}^1\dots\mathbf{A}^n$  eliding the brackets and further with  $\mathbf{F}\overline{\mathbf{A}^n}$  in a kind of vector notation.
- ▷  $\mathbf{A}.$  stands for a left bracket whose partner is as far right as is consistent with existing brackets; i.e.  $\mathbf{A}(\mathbf{B}\mathbf{C})$  abbreviates  $\mathbf{A}(\mathbf{B}\mathbf{C})$ .
- ▷ **Notation 8.0.30 (Abstraction is right-associative)** We abbreviate  $\lambda X^1.\lambda X^2.\dots\lambda X^n.\mathbf{A}\dots$  with  $\lambda X^1\dots X^n.\mathbf{A}$  eliding brackets, and further to  $\lambda\overline{X^n}.\mathbf{A}$  in a kind of vector notation.
- ▷ **Notation 8.0.31 (Outer brackets)** Finally, we allow ourselves to elide outer brackets where they can be inferred.



EdN:9

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  $\mathbf{B}$ .<sup>9</sup>

In this presentation of the simply typed  $\lambda$ -calculus we build-in  $\alpha$ -equality and use capture-avoiding substitutions directly. A clean introduction would followed the steps in Chapter3 by introducing substitutions with a substitutability condition like the one in Definition 4.2.10, then establishing the soundness of  $\alpha$  conversion, and only then postulating defining capture-avoiding substitution application as in Definition 4.3.3. The development for  $\Lambda^{\rightarrow}$  is directly parallel to the one for  $\text{PL}^1$ , so we leave it as an exercise to the reader and turn to the computational properties of the  $\lambda$ -calculus.

Computationally, the  $\lambda$ -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  $\lambda$ -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  $\Lambda^{\rightarrow}$  as we will see below.

### $\alpha\beta\eta$ -Equality (Overview)

- ▷ reduction with  $\left\{ \begin{array}{l} \beta : (\lambda X.\mathbf{A})\mathbf{B} \rightarrow_{\beta} [\mathbf{B}/X](\mathbf{A}) \\ \eta : \lambda X.\mathbf{A}X \rightarrow_{\eta} \mathbf{A} \end{array} \right.$  under  $=_{\alpha} : \begin{array}{l} \lambda X.\mathbf{A} \\ =_{\alpha} \\ \lambda Y.[Y/X](\mathbf{A}) \end{array}$
- ▷ **Theorem 8.0.32**  $\beta\eta$ -reduction is well-typed, terminating and confluent in the presence of  $=_{\alpha}$ -conversion.
- ▷ **Definition 8.0.33 (Normal Form)** We call a  $\lambda$ -term  $\mathbf{A}$  a **normal form** (in a reduction system  $\mathcal{E}$ ), iff no rule (from  $\mathcal{E}$ ) can be applied to  $\mathbf{A}$ .
- ▷ **Corollary 8.0.34**  $\beta\eta$ -reduction yields unique normal forms (up to  $\alpha$ -equivalence).

<sup>9</sup>EDNOTE: rationalize the semantic macros for syntax!

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

## Syntactic Parts of $\lambda$ -Terms

▷ **Definition 8.0.35 (Parts of  $\lambda$ -Terms)** We can always write a  $\lambda$ -term in the form  $\mathbf{T} = \lambda X^1 \dots X^k. \mathbf{H} \mathbf{A}^1 \dots \mathbf{A}^n$ , where  $\mathbf{H}$  is not an application. We call

- ▷  $\mathbf{H}$  the **syntactic head** of  $\mathbf{T}$
- ▷  $h\mathbf{A}^1 \dots \mathbf{A}^n$  the **matrix** of  $\mathbf{T}$ , and
- ▷  $\lambda X^1 \dots X^k$  the **binder** of  $\mathbf{T}$

▷ **Definition 8.0.36 Head Reduction** always has a unique  $\beta$  redex

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

▷ **Theorem 8.0.37** The syntactic heads of  $\beta$ -normal forms are constant or variables.

▷ **Definition 8.0.38** Let  $\mathbf{A}$  be a  $\lambda$ -term, then the syntactic head of the  $\beta$ -normal form of  $\mathbf{A}$  is called the **head symbol** of  $\mathbf{A}$  and written as  $\text{head}(\mathbf{A})$ . We call a  $\lambda$ -term a  **$j$ -projection**, iff its head is the  $j^{\text{th}}$  bound variable.

▷ **Definition 8.0.39** We call a  $\lambda$ -term a  **$\eta$ -long form**, iff its matrix has base type.

▷ **Definition 8.0.40  $\eta$ -Expansion:**  $\eta[\lambda X^1 \dots X^n. \mathbf{A}] := \lambda X^1 \dots X^n Y^1 \dots Y^m. \mathbf{A} Y^1 \dots Y^m$

▷ **Definition 8.0.41 Long  $\beta\eta$ -normal form**, iff it is  $\beta$ -normal and  $\eta$ -long.

$\eta$  long forms are structurally convenient since for them, the structure of the term is isomorphic to the structure of its type (argument types correspond to binders): if we have a term  $\mathbf{A}$  of type  $\overline{\alpha_n} \rightarrow \beta$  in  $\eta$ -long form, where  $\beta \in \mathcal{BT}$ , then  $\mathbf{A}$  must be of the form  $\lambda \overline{X^n}. \mathbf{B}$ , where  $\mathbf{B}$  has type  $\beta$ . Furthermore, the set of  $\eta$ -long forms is closed under  $\beta$ -equality, which allows us to treat the two equality theories of  $\Lambda^\rightarrow$  separately and thus reduce argumentational complexity.



## Chapter 9

# Computational Properties of $\lambda$ -Calculus

As we have seen above, the main contribution of the  $\lambda$ -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.

### 9.1 Termination of $\beta$ -reduction

We will use the termination of  $\beta$  reduction to present a very powerful proof method, called the “logical relations method”, which is one of the basic proof methods in the repertoire of a proof theorist, since it can be extended to many situations, where other proof methods have no chance of succeeding.

Before we start into the termination proof, we convince ourselves that a straightforward induction over the structure of expressions will not work, and we need something more powerful.

#### Termination of $\beta$ -Reduction

- ▷ only holds for the typed case  
 $(\lambda X.XX)(\lambda X.XX) \rightarrow_{\beta} (\lambda X.XX)(\lambda X.XX)$
- ▷ **Theorem 9.1.1 (Typed  $\beta$ -Reduction terminates)** *For all  $A \in wff_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$ , the chain of reductions from  $A$  is finite.*
- ▷ proof attempts:
  - ▷ Induction on the structure  $A$  must fail, since this would also work for the untyped case.
  - ▷ Induction on the type of  $A$  must fail, since  $\beta$ -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  $\lambda$ -terms and their types. The first is the one that we are interested in,  $\mathcal{LR}(A, \alpha)$  essentially states the property that  $\beta\eta$  reduction terminates at  $A$ . 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 9.1.2**  $\mathbf{A}$  is called **strongly reducing** at type  $\alpha$  (write  $\mathcal{SR}(\mathbf{A}, \alpha)$ ), iff each chain  $\beta$ -reductions from  $\mathbf{A}$  terminates.

▷ We define a **logical relation**  $\mathcal{LR}$  inductively on the structure of the type

▷  $\alpha$  base type:  $\mathcal{LR}(\mathbf{A}, \alpha)$ , iff  $\mathcal{SR}(\mathbf{A}, \alpha)$

▷  $\mathcal{LR}(\mathbf{C}, \alpha \rightarrow \beta)$ , iff  $\mathcal{LR}(\mathbf{CA}, \beta)$  for all  $\mathbf{A} \in \text{wff}_\alpha(\Sigma, \mathcal{V}_T)$  with  $\mathcal{LR}(\mathbf{A}, \alpha)$ .

**Proof:** Termination Proof

▷ **P.1**  $\mathcal{LR} \subseteq \mathcal{SR}$

(Lemma 9.1.4b))

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

(Theorem 9.1.6 with  $\sigma = \emptyset$ )

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

**P.2** **P.3** **Lemma 9.1.3** ( $\mathcal{SR}$  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  $\beta$ -reduction from  $\mathbf{B}$  would be one from  $\mathbf{A}$ . □



The termination proof proceeds in two steps, the first one shows that  $\mathcal{LR}$  is a sub-relation of  $\mathcal{SR}$ , and the second that  $\mathcal{LR}$  is total on  $\lambda$ -terms. Together 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 9.1.4** (Rollercoaster Lemma)

a) If  $h$  is a constant or variable of type  $\overline{\alpha_n} \rightarrow \alpha$  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  $\alpha$

▷ **P.1.1**  $\alpha$  base type:

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

**P.1.1.1.1.2** so  $\mathcal{LR}(h\overline{\mathbf{A}^n}, \alpha)$  as  $\alpha$  is a base type ( $\mathcal{SR} = \mathcal{LR}$ ) □

**P.1.1.1.2 b):** by definition □

$\alpha = \beta \rightarrow \gamma$ :

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

**P.1.2.1.1.2** 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  $\mathcal{LR}(h\overline{\mathbf{A}^n}, \beta)$  by definition. □



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

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

□  
□  
□



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  $\mathcal{LR}$ .

### $\mathbf{A} \in \text{wff}_\alpha(\Sigma, \mathcal{V}_\tau)$ implies $\mathcal{LR}(\mathbf{A}, \alpha)$

▷ **Definition 9.1.5** We write  $\mathcal{LR}(\sigma)$  if  $\mathcal{LR}(\sigma(X_\alpha), \alpha)$  for all  $X \in \text{supp}(\sigma)$ .

▷ **Theorem 9.1.6** If  $\mathbf{A} \in \text{wff}_\alpha(\Sigma, \mathcal{V}_\tau)$ , then  $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$  for any substitution  $\sigma$  with  $\mathcal{LR}(\sigma)$ .

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

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

**P.1.2**  $\mathbf{A} = X \notin \text{supp}(\sigma)$ : then  $\sigma(\mathbf{A}) = \mathbf{A}$  and  $\mathcal{LR}(\mathbf{A}, \alpha)$  by Lemma 9.1.4 with  $n = 0$ .

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

**P.1.4**  $\mathbf{A} = \mathbf{B}\mathbf{C}$ : by IH  $\mathcal{LR}(\sigma(\mathbf{B}), \gamma \rightarrow \alpha)$  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  $\theta$  meets the conditions of the IH.

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

**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 9.1.8.

**P.1.5.4** So  $\mathcal{LR}(\sigma(\mathbf{A}), \alpha)$  by definition of  $\mathcal{LR}$ . □

□  
□



### $\beta$ -Expansion Lemma

▷ **Lemma 9.1.7** If  $\mathcal{LR}([\mathbf{B}/X](\mathbf{A}), \alpha)$  and  $\mathcal{LR}(\mathbf{B}, \beta)$  for  $X_\beta \notin \text{free}(\mathbf{B})$ , then  $\mathcal{LR}((\lambda X_\alpha. \mathbf{A})\mathbf{B}, \alpha)$ .

▷ **Proof:**

**P.1** Let  $\alpha = \overline{\gamma}_i \rightarrow \delta$  where  $\delta$  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  $\delta$  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  $\delta$  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 9.1.4

**P.7** So an infinite reduction from  $((\lambda X.A)B)\overline{C}$  cannot solely consist of redexes from  $[B/X](A)$  and the  $C^i$ .

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

$$\begin{aligned} ((\lambda X.A)B)\overline{C} &\rightarrow_{\beta}^* ((\lambda X.A')B')\overline{C'} \\ &\rightarrow_{\beta}^1 [B'/X](A')\overline{C'} \\ &\rightarrow_{\beta}^* \dots \end{aligned}$$

where  $A \rightarrow_{\beta}^* A'$ ,  $B \rightarrow_{\beta}^* B'$  and  $C^i \rightarrow_{\beta}^* C'^i$

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

**P.10** so we have the infinite reduction

$$\begin{aligned} [B/X](A)\overline{C} &\rightarrow_{\beta}^* [B'/X](A')\overline{C'} \\ &\rightarrow_{\beta}^* \dots \end{aligned}$$

which contradicts our assumption □

▷ **Lemma 9.1.8 ( $\mathcal{LR}$  is closed under  $\beta$ -expansion)**

If  $C \rightarrow_{\beta} D$  and  $\mathcal{LR}(D, \alpha)$ , so is  $\mathcal{LR}(C, \alpha)$ .



This was the last result we needed to complete the proof of termination of  $\beta$ -reduction.

**Remark:** If we are only interested in the termination of head reductions, we can get by with a much simpler version of this lemma, that basically relies on the uniqueness of head  $\beta$  reduction.

### Closure under Head $\beta$ -Expansion (weakly reducing)

▷ **Lemma 9.1.9 ( $\mathcal{LR}$  is closed under head  $\beta$ -expansion)** If  $C \rightarrow_{\beta}^h D$  and  $\mathcal{LR}(D, \alpha)$ , so is  $\mathcal{LR}(C, \alpha)$ .

▷ **Proof:** by induction over the structure of  $\alpha$

**P.1.1  $\alpha$  base type:**

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

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

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

**P.1.2  $\alpha = \beta \rightarrow \gamma$ :**

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

**P.1.2.2** but  $CB \rightarrow_{\beta}^h DB$ , so  $\mathcal{LR}(CB, \gamma)$  by IH

**P.1.2.3** and  $\mathcal{LR}(C, \alpha)$  by definition. □

**Note:** This result only holds for weak reduction (any chain of  $\beta$  head reductions terminates) for strong reduction we need a stronger Lemma. □



For the termination proof of head  $\beta$ -reduction we would just use the same proof as above, just

for a variant of  $SR$ , where  $SR\mathbf{A}\alpha$  that only requires that the head reduction sequence out of  $\mathbf{A}$  terminates. Note that almost all of the proof except Lemma 9.1.3 (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.

## 9.2 Confluence of $\beta\eta$ Conversion

### ▷ $\eta$ -Reduction ist terminating and confluent

▷ **Lemma 9.2.1**  $\eta$ -Reduction ist terminating

▷ **Proof:** by a simple counting argument □

▷ **Lemma 9.2.2**  $\eta$ -Reduction ist confluent

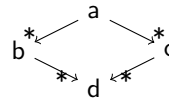
▷ **Proof Idea:** by diagram chase □



### Newman's Lemma

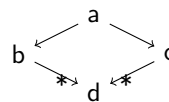
▷ **Definition 9.2.3 (Confluence)**

We call a rewrite relation  $R \subseteq A^2$  **confluent** iff for every  $a, b, c \in A$  with  $a \rightarrow_R b$   $a \rightarrow_R c$  there is a  $d \in A$  with  $b \rightarrow_R d$  and  $c \rightarrow_R d$



▷ **Definition 9.2.4 (Weak Confluence)**

We call a rewrite relation  $R \subseteq A^2$  **weakly confluent** iff for every  $a, b, c \in A$  with  $a \rightarrow_R b$   $a \rightarrow_R c$  there is a  $d \in A$  with  $b \rightarrow_R^* d$  and  $c \rightarrow_R^* d$ .



▷ **Theorem 9.2.5 (Newman's Lemma)** If a relation is terminating and weakly confluent, then it is also confluent.



### $\beta$ is confluent

▷ **Lemma 9.2.6**  $\beta$ -Reduction ist weakly confluent

▷ **Proof Idea:** by diagram chase □

▷ **Corollary 9.2.7**  $\beta$ -Reduction ist confluent

▷ **Proof Idea:** by Newman's Lemma □



$\beta\eta$  is confluent

▷ **Lemma 9.2.8**  $\rightarrow_{\beta}^*$  and  $\rightarrow_{\eta}^*$  commute.

▷ **Proof Sketch:** diagram chase

□



## Chapter 10

# The Semantics of the Simply Typed $\lambda$ -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 10.0.9** We call a collection  $\mathcal{D}_\mathcal{T} := \{\mathcal{D}_\alpha \mid \alpha \in \mathcal{T}\}$  a **typed collection** (of sets) and a collection  $f_\mathcal{T}: \mathcal{D}_\mathcal{T} \rightarrow \mathcal{E}_\mathcal{T}$ , a **typed function**, iff  $f_\alpha: \mathcal{D}_\alpha \rightarrow \mathcal{E}_\alpha$ .
- ▷ **Definition 10.0.10** A typed collection  $\mathcal{D}_\mathcal{T}$  is called a **frame**, iff  $\mathcal{D}_{\alpha \rightarrow \beta} \subseteq \mathcal{D}_\alpha \rightarrow \mathcal{D}_\beta$
- ▷ **Definition 10.0.11** Given a frame  $\mathcal{D}_\mathcal{T}$ , and a typed function  $\mathcal{I}: \Sigma \rightarrow \mathcal{D}$ , then we call  $\mathcal{I}_\varphi: \text{wff}_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T}) \rightarrow \mathcal{D}$  the **value function** induced by  $\mathcal{I}$ , iff
  - ▷  $\mathcal{I}_\varphi|_{\mathcal{V}_\mathcal{T}} = \varphi$ ,  $\mathcal{I}_\varphi|_\Sigma = \mathcal{I}$
  - ▷  $\mathcal{I}_\varphi(\mathbf{AB}) = \mathcal{I}_\varphi(\mathbf{A})(\mathcal{I}_\varphi(\mathbf{B}))$
  - ▷  $\mathcal{I}_\varphi(\lambda X_\alpha. \mathbf{A})$  is that function  $f \in \mathcal{D}_{\alpha \rightarrow \beta}$ , such that  $f(a) = \mathcal{I}_{\varphi, [a/X]}(\mathbf{A})$  for all  $a \in \mathcal{D}_\alpha$
- ▷ **Definition 10.0.12** We call a frame  $\langle \mathcal{D}, \mathcal{I} \rangle$  **comprehension-closed** or a  **$\Sigma$ -algebra**, iff  $\mathcal{I}_\varphi: \text{wff}_\mathcal{T}(\Sigma, \mathcal{V}_\mathcal{T}) \rightarrow \mathcal{D}$  is total.  
(every  $\lambda$ -term has a value)



©: Michael Kohlhase

104



## 10.1 Soundness of the Simply Typed $\lambda$ -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  $\varphi$ . We say that the reductions are sound. As always, the main tool for proving soundness 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 $\lambda$ -Terms

- ▷ **Lemma 10.1.1 (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  $Y \in \mathcal{V}_T$ : 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**  $\mathbf{A} = \mathbf{CD}$ : then  $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{CD})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{C})[\mathbf{B}/X](\mathbf{D})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{C}))(\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{D}))) = \mathcal{I}_\psi(\mathbf{C})(\mathcal{I}_\psi(\mathbf{D})) = \mathcal{I}_\psi(\mathbf{CD}) = \mathcal{I}_\psi(\mathbf{A})$

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

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

**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 10.1.2** Let  $\mathcal{A} := \langle \mathcal{D}, \mathcal{I} \rangle$  be a  $\Sigma$ -algebra and  $Y \notin \text{free}(\mathbf{A})$ , then  $\mathcal{I}_\varphi(\lambda X. \mathbf{A}) = \mathcal{I}_\varphi(\lambda Y. [Y/X]\mathbf{A})$  for all assignments  $\varphi$ .

▷ **Proof:** by substitution value lemma

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

▷ **Theorem 10.1.3** If  $\mathcal{A} := \langle \mathcal{D}, \mathcal{I} \rangle$  is a  $\Sigma$ -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 10.1.4** If  $X \notin \text{free}(\mathbf{A})$ , then  $\mathcal{I}_\varphi(\lambda X. \mathbf{A}X) = \mathcal{I}_\varphi(\mathbf{A})$  for all  $\varphi$ .

▷ **Proof:** by calculation

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

▷ **Theorem 10.1.5**  $\alpha\beta\eta$ -equality is sound wrt.  $\Sigma$ -algebras. (if  $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$ , then  $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$  for all assignments  $\varphi$ )



## 10.2 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  $\varphi$ , 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  $\varphi$ .

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  $\lambda$ -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 $\lambda$ -calculus

▷ **Definition 10.2.1** We call a term  $\mathbf{A} \in wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$  a  **$\beta$  normal form** iff there is no  $\mathbf{B} \in wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$  with  $\mathbf{A} \rightarrow_{\beta} \mathbf{B}$ .

We call  $\mathbf{N}$  a  **$\beta$  normal form of  $\mathbf{A}$** , iff  $\mathbf{N}$  is a  $\beta$ -normal form and  $\mathbf{A} \rightarrow_{\beta} \mathbf{N}$ .

We denote the set of  $\beta$ -normal forms with  $wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}}) \downarrow_{\beta\eta}$ .

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

▷ **Corollary 10.2.2 (Normal Forms)** Every  $\mathbf{A} \in wff_{\mathcal{T}}(\Sigma, \mathcal{V}_{\mathcal{T}})$  has a unique  $\beta$  normal form ( $\beta\eta$ , long  $\beta\eta$  normal form), which we denote by  $\mathbf{A} \downarrow_{\beta}$  ( $\mathbf{A} \downarrow_{\beta\eta}$   $\mathbf{A} \downarrow_{\beta\eta}^l$ )



The term frames will be a quotient spaces over the equality relations of the  $\lambda$ -calculus, so we introduce this construction generally.

### Frames and Quotients

▷ **Definition 10.2.3** 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 10.2.4** We call a congruence  $\sim$  **functional**, iff for all  $f, g \in \mathcal{D}_{\alpha \rightarrow \beta}$  the fact that  $f(a) \sim g(a)$  holds for all  $a \in \mathcal{D}_{\alpha}$  implies that  $f \sim g$ .

▷ **Example 10.2.5**  $=_{\beta}$  ( $=_{\beta\eta}$ ) is a (functional) congruence on  $cwff_{\mathcal{T}}(\Sigma)$  by definition.

▷ **Theorem 10.2.6** Let  $\mathcal{D}$  be a  $\Sigma$ -frame and  $\sim$  a functional congruence on  $\mathcal{D}$ , then the quotient space  $\mathcal{D}/\sim$  is a  $\Sigma$ -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  $[a]_\sim \in \mathcal{D}/\sim$ .  $\square$



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 it is also functional on  $cwff_{\mathcal{T}}(\Sigma)$  by a grounding argument.

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

▷ **Lemma 10.2.7**  $\beta\eta$ -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  $X \in (\mathcal{V}_{\mathcal{T}} \setminus (\text{free}(\mathbf{A}) \cup \text{free}(\mathbf{B})))$ .

**P.1** then (in particular)  $\mathbf{AX} =_{\beta\eta} \mathbf{BX}$ , and

**P.2**  $\lambda X. \mathbf{AX} =_{\beta\eta} \lambda X. \mathbf{BX}$ , since  $\beta\eta$ -equality acts on subterms.

**P.3** By definition we have  $\mathbf{A} =_{\eta} \lambda X_{\alpha}. \mathbf{AX} =_{\beta\eta} \lambda X_{\alpha}. \mathbf{BX} =_{\eta} \mathbf{B}$ .  $\square$

▷ **Definition 10.2.8** We call an injective substitution  $\sigma: \text{free}(\mathbf{C}) \rightarrow \Sigma$  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  $\Sigma_{\alpha}$  are infinite and  $\text{free}(\mathbf{C})$  is finite.

▷ **Theorem 10.2.9**  $\beta\eta$ -equality is a functional congruence on  $cwff_{\mathcal{T}}(\Sigma)$ .

▷ **Proof:** We use Lemma 10.2.7

**P.1** Let  $\mathbf{A}, \mathbf{B} \in cwff_{(\alpha \rightarrow \beta)}(\Sigma)$ , 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  $\sigma$ .

**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  $\mathbf{AC}' \neq_{\beta\eta} \mathbf{BC}'$ .  $\square$



**Note that:** the result for  $cwff_{\mathcal{T}}(\Sigma)$  is sharp. For instance, if  $\Sigma = \{c_i\}$ , then  $\lambda X.X \neq_{\beta\eta} \lambda X.c$ , but  $(\lambda X.X)c =_{\beta\eta} c =_{\beta\eta} (\lambda X.c)c$ , as  $\{c\} = cwff_{\iota}(\Sigma)$  (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  $d_i$  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  $\Sigma$ -algebra whose frame is made of  $\beta\eta$ -normal  $\lambda$ -terms.



### A Herbrand Model for $\Lambda^\rightarrow$

▷ **Definition 10.2.10** We call  $\mathcal{T}_{\beta\eta} := \langle \text{cwf}_{\mathcal{T}}(\Sigma) \downarrow_{\beta\eta}, \mathcal{I}^{\beta\eta} \rangle$  the  $\Sigma$  term algebra, if  $\mathcal{I}^{\beta\eta} = \text{Id}_{\Sigma}$ .

▷ The name “term algebra” in the previous definition is justified by the following

▷ **Theorem 10.2.11**  $\mathcal{T}_{\beta\eta}$  is a  $\Sigma$ -algebra

▷ **Proof:** We use the work we did above

**P.1** Note that  $\text{cwf}_{\mathcal{T}}(\Sigma) \downarrow_{\beta\eta} = \text{cwf}_{\mathcal{T}}(\Sigma) /_{=\beta\eta}$  and thus a  $\Sigma$ -frame by Theorem 10.2.6 and Lemma 10.2.7.

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

**P.3** Let  $\varphi$  be an assignment into  $\text{cwf}_{\mathcal{T}}(\Sigma) \downarrow_{\beta\eta}$ .

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

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

**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  $\Sigma$ -algebras, i.e. if the equation  $\mathbf{A} = \mathbf{B}$  is valid, then  $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$ . Thus  $\alpha\beta\eta$  equivalence fully characterizes equality in the class of all  $\Sigma$ -algebras.

### Completeness of $\alpha\beta\eta$ -Equality

▷ **Theorem 10.2.12**  $\mathbf{A} = \mathbf{B}$  is valid in the class of  $\Sigma$ -algebras, iff  $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$ .

▷ **Proof:** For  $\mathbf{A}, \mathbf{B}$  closed this is a simple consequence of the fact that  $\mathcal{T}_{\beta\eta}$  is a  $\Sigma$ -algebra.

**P.1** If  $\mathbf{A} = \mathbf{B}$  is valid in all  $\Sigma$ -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  $\sigma$  be a grounding substitution for  $\mathbf{A}$  and  $\mathbf{B}$  and  $\varphi$  the induced variable assignment.

**P.4** Thus  $\mathcal{I}_{\varphi}^{\beta\eta}(\mathbf{A}) = \mathcal{I}_{\varphi}^{\beta\eta}(\mathbf{B})$  is the  $\beta\eta$ -normal form of  $\sigma(\mathbf{A})$  and  $\sigma(\mathbf{B})$ .

**P.5** Since  $\varphi$  is a structure preserving homomorphism on well-formed formulae,  $\varphi^{-1}(\mathcal{I}_{\varphi}^{\beta\eta}(\mathbf{A}))$  is the  $\beta\eta$ -normal form of both  $\mathbf{A}$  and  $\mathbf{B}$  and thus  $\mathbf{A} =_{\alpha\beta\eta} \mathbf{B}$ . □



Theorem 10.2.12 and Theorem 10.1.5 complete our study of the semantics of the simply-typed  $\lambda$ -calculus by showing that it is an adequate logic for modeling (the equality) of functions and their applications.



# Chapter 11

## 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  $\lambda$ -calculus, i.e. unification modulo  $\alpha\beta\eta$  equality.

### 11.1 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 (there may be infinitely descending chains of unifiers that become more and more general), so 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 11.1.1**  $\sigma =_{\beta\eta} \rho[W]$ , iff  $\sigma(X) =_{\alpha\beta\eta} \rho(X)$  for all  $X \in W$ .  $\sigma =_{\beta\eta} \rho[\mathcal{E}]$  iff  $\sigma =_{\beta\eta} \rho[\text{free}(\mathcal{E})]$
- ▷ **Definition 11.1.2**  $\sigma$  is **more general** than  $\theta$  on  $W$  ( $\sigma \leq_{\beta\eta} \theta[W]$ ), iff there is a substitution  $\rho$  with  $\theta =_{\beta\eta} \rho \circ \sigma[W]$ .
- ▷ **Definition 11.1.3**  $\Psi \subseteq \mathbf{U}(\mathcal{E})$  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 11.1.4** If  $\Psi \subseteq \mathbf{U}(\mathcal{E})$  is complete, then  $\leq_{\beta\eta}$ -minimal elements  $\sigma \in \Psi$  are **most general unifiers** of  $\mathcal{E}$ .



©: Michael Kohlhase

113



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 11.1.5**  $X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$  is in **solved form**, if the  $X^i$  are distinct free

variables  $X^i \notin \text{free}(\mathbf{B}^j)$  and  $\mathbf{B}^j$  does not contain Skolem constants for all  $j$ .

▷ **Lemma 11.1.6** *If  $\mathcal{E} = X^1 =? \mathbf{B}^1 \wedge \dots \wedge X^n =? \mathbf{B}^n$  is in solved form, then  $\sigma_{\mathcal{E}} := [\mathbf{B}^1/X^1], \dots, [\mathbf{B}^n/X^n]$  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  $\theta \leq_{\beta\eta} \theta \circ \sigma[\mathcal{E}]$ . □



## 11.2 Higher-Order Unification Transformations

We are now in a position to introduce the higher-order unification 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 11.2.1** The **higher-order simplification transformations**  $\mathcal{SIM}$  consist of the rules below.

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

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

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

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

After rule applications all  $\lambda$ -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  $\lambda$ -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  $\alpha$ -equality, and  $\mathcal{SIM}:\eta$  first 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  $STM$ :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  $STM$ : $\alpha$ , 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,  $STM$  is a direct generalization of first-order unification transformations, and shares its properties; even the proofs go correspondingly.

▷ **Lemma 11.2.2 (Properties of  $STM$ )**  $STM$  generalizes first-order unification.

▷  $STM$  is terminating and confluent up to  $\alpha$ -conversion

▷ Unique  $STM$  normal forms exist (all pairs have the form  $h\overline{U}^n =^? k\overline{V}^m$ )

▷ **Lemma 11.2.3** If  $\mathcal{E} \vdash_{STM} \mathcal{E}'$ , then  $U(\mathcal{E}) \leq_{\beta\eta} U(\mathcal{E}')[\mathcal{E}]$ . (correct, complete)



Now that we have simplification out of the way, we have to deal with unification pairs of the form  $h\overline{U}^n =^? k\overline{V}^m$ . Note that the case where both  $h$  and  $k$  are constants is unsolvable, so we can assume that one of them is a variable. The unification problem  $F_{(\alpha \rightarrow \alpha)} a =^? a$  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  $\lambda$ -term of type  $\alpha \rightarrow \alpha$  with head  $a$ , and in the second with a 1-projection term of type  $\alpha \rightarrow \alpha$ , 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  $\alpha$  and head  $h$ .

▷ **sufficient:** long  $\beta\eta$  head normal form, most general

▷ **General Bindings:**  $G_{\alpha}^h(\Sigma) := \lambda \overline{X}^k. h(H^1 \overline{X}) \dots (H^n \overline{X})$

▷ where  $\alpha = \overline{\alpha}_k \rightarrow \beta$ ,  $h : \overline{\gamma}_n \rightarrow \beta$  and  $\beta \in \mathcal{BT}$

▷ and  $H^i : \overline{\alpha}_k \rightarrow \gamma_i$  new variables.

▷ **Observation 11.2.4** General bindings are unique up to choice of names for  $H^i$ .

▷ **Definition 11.2.5** If the head  $h$  is  $j^{\text{th}}$  bound variable in  $G_{\alpha}^h(\Sigma)$ , call  $G_{\alpha}^h(\Sigma)$   **$j$ -projection binding** (and write  $G_{\alpha}^j(\Sigma)$ ) else **imitation binding**

▷ clearly  $G_{\alpha}^h(\Sigma) \in \text{wff}_{\alpha}(\Sigma, \mathcal{V}_{\mathcal{T}})$  and  $\text{head}(G_{\alpha}^h(\Sigma)) = h$



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

**Example 11.2.6** The following general bindings may be helpful:  $G_{\iota \rightarrow \iota}^{a_{\iota}}(\Sigma) = \lambda X_{\iota}. a$ ,  $G_{\iota \rightarrow \iota \rightarrow \iota}^{a_{\iota \rightarrow \iota}}(\Sigma) = \lambda X_{\iota} Y_{\iota}. a$ , and  $G_{\iota \rightarrow \iota \rightarrow \iota}^{a_{\iota \rightarrow \iota \rightarrow \iota}}(\Sigma) = \lambda X_{\iota} Y_{\iota}. a(HXY)$ , where  $H$  is of type  $\iota \rightarrow \iota \rightarrow \iota$

We will now show that the general bindings defined in Definition 11.2.5 are indeed the most general  $\lambda$ -terms given their type and head.

## Approximation Theorem

▷ **Theorem 11.2.7** If  $\mathbf{A} \in \text{wff}_\alpha(\Sigma, \mathcal{V}_T)$  with  $\text{head}(\mathbf{A}) = h$ , then there is a general binding  $\mathbf{G} = \mathbf{G}_\alpha^h(\Sigma)$  and a substitution  $\rho$  with  $\rho(\mathbf{G}) =_{\alpha\beta\eta} \mathbf{A}$  and  $\text{dp}(\rho) < \text{dp}(\mathbf{A})$ .

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

**P.1** If  $\alpha = \overline{\alpha_k} \rightarrow \beta$  and  $h : \overline{\gamma_n} \rightarrow \beta$  where  $\beta \in \mathcal{BT}$ , then the long head normal form of  $\mathbf{A}$  must be  $\lambda \overline{X_\alpha^k}. h \overline{U^n}$ .

**P.2**  $\mathbf{G} = \mathbf{G}_\alpha^h(\Sigma) = \lambda \overline{X_\alpha^k}. h (H^1 \overline{X}) \dots (H^n \overline{X})$  for some variables  $H^i : \overline{\alpha_k} \rightarrow \gamma_i$ .

**P.3** Choose  $\rho := [(\lambda \overline{X_\alpha^k}. U^1)/H^1], \dots, [(\lambda \overline{X_\alpha^k}. U^n)/H^n]$ .

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

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



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 11.2.8** Let  $\mathbf{G} = \mathbf{G}_\alpha^h(\Sigma)$  (imitation) or  $\mathbf{G} \in \{\mathbf{G}_\alpha^j(\Sigma) \mid 1 \leq j \leq n\}$ , then  $\mathcal{HOU}$  consists of the transformations (always reduce to  $\mathcal{SLM}$  normal form)

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

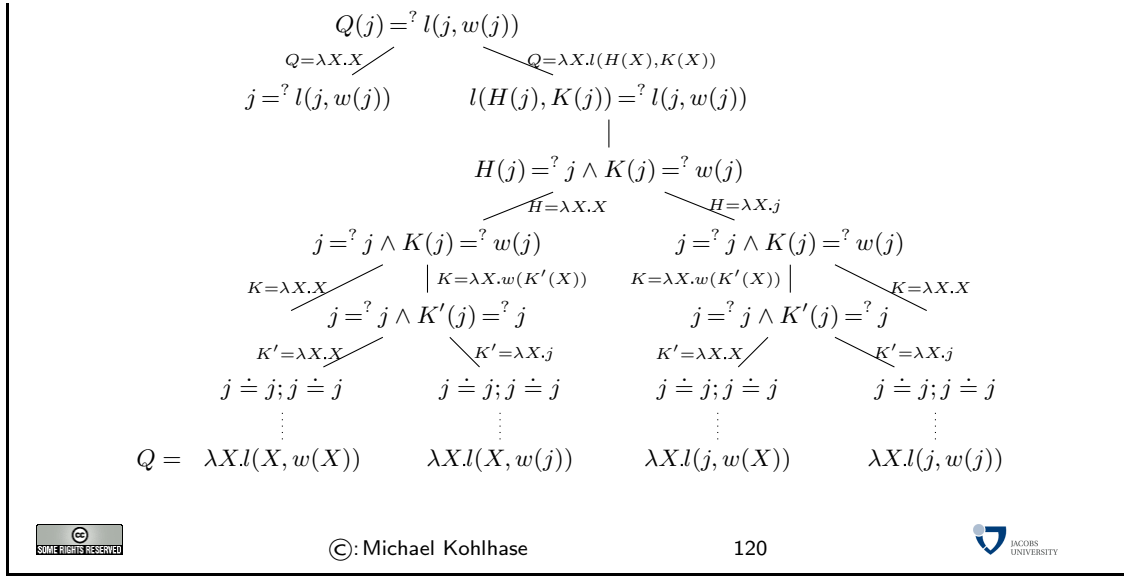
▷ Rules for flex/flex pairs: 
$$\frac{F_\alpha \overline{U} =? H \overline{V} \wedge \mathcal{E}}{F =? \mathbf{G} \wedge F \overline{U} =? H \overline{V} \wedge \mathcal{E}} \mathcal{HOU}:\text{ff}$$



Let us now fortify our intuition with a simple example.

## $\mathcal{HOU}$ Example

**Example 11.2.9** Let  $Q, w : \iota \rightarrow \iota$ ,  $l : \iota \rightarrow \iota \rightarrow \iota$ , 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.

### 11.3 Properties of Higher-Order Unification

We will now establish the properties of the higher-order unification problem and the algorithms we have introduced above.

#### Undecidability of Higher-Order Unification

▷ **Theorem 11.3.1** (Goldfarb '82) *Second-order unification is undecidable*

▷ **Proof Sketch:** Reduction to Hilbert's tenth problem (known to be undecidable)

□

▷ **Definition 11.3.2** Diophantine equations over  $\mathbb{N}$ .

- ▷  $x_i \cdot x_j = x_k$
- ▷  $x_i + x_j = x_k$
- ▷  $x_i = c_j$  where  $c_j \in \mathbb{N}$

▷ **Theorem 11.3.3** *The general solution of Diophantine equations is undecidable.*

#### Undecidability (Reduction by Church numerals)

- ▷ Diophantine equations become unification pairs,
- ▷ **Definition 11.3.4 (Church Numerals)** We define closed  $\lambda$ -terms of type  $\nu := (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$ 
  - ▷ **Numbers**:  $n := \lambda S_{\alpha \rightarrow \alpha} . \lambda O_{\alpha} . \underbrace{S(S \dots S(O) \dots)}_n (n\text{-fold iteration of arg1 starting from arg2})$
  - ▷ **Addition**  $+$   $:= \lambda N_{\nu} M_{\nu} . \lambda S_{\alpha \rightarrow \alpha} . \lambda O_{\alpha} . N S (M S O)$  ( $N$ -fold iteration of  $S$  from  $N$ )
  - ▷ **Multiplication**:  $\cdot := \lambda N_{\nu} M_{\nu} . \lambda S_{\alpha \rightarrow \alpha} . \lambda O_{\alpha} . N (M S) O$  ( $N$ -fold iteration of  $MS (=+m)$  from  $O$ )
- ▷ Subtraction and division by higher-order unification.



## Properties of HO-Unification

- ▷ **Theorem 11.3.5** If  $\sigma \in \mathbf{U}(\mathcal{E})$ , then there is a  $\mathcal{HOU}$ -derivation  $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$ , such that
  - ▷  $\mathcal{E}' = X^1 =^? \mathbf{A}^1 \wedge \dots \wedge X^n =^? \mathbf{A}^n$  is in solved form,
  - ▷  $\theta = [\mathbf{A}^1/X^1], \dots, [\mathbf{A}^n/X^n] \in \mathbf{U}(\mathcal{E})$
  - ▷  $\theta$  is more general than  $\sigma$ .
- ▷ **HOU is undecidable, HOU need not have most general unifiers**
- ▷  **$\mathcal{HOU}:\text{ff}$  gives enormous degree of indeterminism**
- ▷ HOU is intractable  $\implies$  **consider restricted fragments where it is!**
- ▷ HO Matching (up to order 4 decidable), HO Patterns (unitary, linear)...



## $\mathcal{HOU}$ is Correct

- ▷ **Lemma 11.3.6** If  $\mathcal{E} \vdash_{\mathcal{HOU}\text{fr}} \mathcal{E}'$  or  $\mathcal{E} \vdash_{\mathcal{HOU}\text{ff}} \mathcal{E}'$ , then  $\mathbf{U}(\mathcal{E}') \subseteq \mathbf{U}(\mathcal{E})$ .
- ▷ **Proof Sketch**:  $\mathcal{HOU}:\text{fr}$  and  $\mathcal{HOU}:\text{ff}$  only add new pair. □
- ▷ completeness cannot be expected since a binding is fixed (choice of solutions)
- ▷ **Corollary 11.3.7**  $\mathcal{HOU}$  is correct: If  $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$ , then  $\mathbf{U}(\mathcal{E}') \subseteq \mathbf{U}(\mathcal{E})$ .
- ▷ We know that Unification in  $\Lambda^\rightarrow$  is not decidable,
- ▷ therefore,  $\mathcal{HOU}$  cannot be terminating!



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

- ▷ **Definition 11.3.8** We call  $\mu(\mathcal{E}, \theta) := \langle \mu_1(\mathcal{E}, \theta), \mu_2(\theta) \rangle$  the **unification measure** for  $\mathcal{E}$  and  $\theta$ , if



- ▷  $\mu_1(\mathcal{E}, \theta)$  is the multiset of term depths of  $\theta(X)$  for the unsolved  $X \in \text{supp}(\theta)$ .
- ▷  $\mu_2(\mathcal{E})$  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 11.3.9**  $\prec$  is well-founded.



## Completeness of $\mathcal{HOU}$ (Semi-Termination)

▷ **Theorem 11.3.10** If  $\theta \in \mathbf{U}(\mathcal{E})$ , 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  $\mu$ -prescribed

▷ **Corollary 11.3.11** If  $\mathcal{E}$  is unifiable without  $\mu$ -prescribed  $\mathcal{HOU}$ -steps, then  $\mathcal{E}$  is solved.

▷ **Theorem 11.3.12** 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  $\mathcal{D}: \mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{F}$  a maximal  $\mu$ -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  $\mu$ -prescribed step  $\mathcal{E} \vdash_{\mathcal{HOU}} \mathcal{E}'$  and a substitution  $\theta'$  as in Theorem 11.3.10.

**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 ?prescribed.thm?.

□



## Proof of Theorem 11.3.10

▷ **Proof:**

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

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

**P.3** but not the same constant, since otherwise *SIM*:dec would be applicable.

- P.4** We can also exclude  $A =_{\alpha\beta\eta} B$ , as  $STM:triv$  would be applicable.
- P.5** If  $F = G$  is a variable not in  $\text{supp}(\theta)$ , then  $STM:dec$  applicable. By correctness we have  $\theta \in U(\mathcal{E}')$  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  $F \neq G$  or  $F = G \in \text{supp}(\theta)$ .
- P.7** In both cases  $F$  or  $G$  is an unsolved variable  $F \in \text{supp}(\theta)$  of type  $\alpha$ , since  $\mathcal{E}$  is unsolved.
- P.8** Without loss of generality we choose  $F = G$ .
- P.9** By Theorem 11.2.7 there is a general binding  $G = G_\alpha^f(\Sigma)$  and a substitution  $\rho$  with  $\rho(G) =_{\alpha\beta\eta} \theta(F)$ . So,
- ▷ if  $\text{head}(G) \notin \text{supp}(\theta)$ , then  $HOU:fr$  is applicable,
  - ▷ if  $\text{head}(G) \in \text{supp}(\theta)$ , then  $HOU:ff$  is applicable.
- P.10** Choose  $\theta' := \theta \cup \rho$ . Then  $\theta =_{\beta\eta} \theta'[\mathcal{E}]$  and  $\theta' \in U(\mathcal{E}')$  by correctness.
- P.11**  $HOU:ff$  and  $HOU:fr$  solve  $F \in \text{supp}(\theta)$  and replace  $F$  by  $\text{supp}(\rho)$  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  $\mu(\mathcal{E}', \theta') \prec \mu(\mathcal{E}, \theta)$ . □



## 11.4 Pre-Unification

### Pre-Unification

- ▷  $HOU:ff$  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 11.4.1** *flex/flex-pairs  $F\overline{U}^n =^? G\overline{V}^m$  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 11.4.2 (Pre-Unification)** For given terms  $A, B \in wff_\alpha(\Sigma, \mathcal{V}_T)$  find a substitution  $\sigma$ , such that  $(\sigma(A) =_{\beta\eta}^p \sigma(B))$ , where  $=_{\beta\eta}^p$  is the equality theory that is induced by  $\beta\eta$  and  $F\overline{U} = G\overline{V}$ .



### Pre-Unification Algorithm $HOPU$

- ▷ **Definition 11.4.3** A unification problem is a **pre-solved form**, iff all of its pairs are solved or flex/flex
- ▷ **Lemma 11.4.4** *If  $\mathcal{E}$  is solved and  $\mathcal{P}$  flex/flex, then  $\sigma_\sigma$  is a most general unifier of a pre-solved form  $\mathcal{E} \wedge \mathcal{P}$ .*
- ▷ Restrict all  $HOU$  rule so that they cannot be applied to pre-solved pairs.

- ▷ In particular, remove  $HOU:ff$ !
- ▷  $HOPU$  only consists of  $SIM$  and  $HOU:fr$ .
- ▷ **Theorem 11.4.5**  $HOPU$  is a correct and complete pre-unification algorithm
- ▷ **Proof Sketch:** with exactly the same methods as higher-order unification □



## 11.5 Applications of Higher-Order Unification

### Application of HOL in NL Semantics: Ellipsis

- ▷ **Example 11.5.1** *John loves his wife. George does too*
  - ▷  $\text{love}(\text{john}, \text{wife\_of}(\text{john})) \wedge Q(\text{george})$
  - ▷ “*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:**  $Q(\text{john}) =_{\alpha\beta\eta} \text{love}(\text{john}, \text{wife\_of}(\text{john}))$
- ▷ **Solutions (computed by HOU):**
  - ▷  $Q = \lambda z.\text{love}(z, \text{wife\_of}(z))$  and  $Q = \lambda z.\text{love}(z, \text{wife\_of}(\text{john}))$
  - \*  $Q = \lambda z.\text{love}(\text{john}, \text{wife\_of}(z))$  and  $Q = \lambda z.\text{love}(\text{john}, \text{wife\_of}(\text{john}))$
- ▷ **Readings:** *George loves his own wife.* and *George loves Johns wife.*





## Chapter 12

# Simple Type Theory (Higher-Order Logic based on the Simply Typed $\lambda$ -Calculus)

In this chapter we will revisit the higher-order predicate logic introduced in Chapter 6 with the base given by the simply typed  $\lambda$ -calculus. It turns out that we can define a higher-order logic by just introducing a type of propositions in the  $\lambda$ -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 12.0.2** We call a  $\Sigma$ -algebra  $\langle \mathcal{D}, \mathcal{I} \rangle$  a **Henkin model**, iff  $\mathcal{D}_o = \{\mathbf{T}, \mathbf{F}\}$ .
- ▷  **$\mathbf{A}_o$  valid** under  $\varphi$ , iff  $\mathcal{I}_\varphi(\mathbf{A}) = \mathbf{T}$
- ▷ **connectives** in  $\Sigma$ :  $\neg \in \Sigma_{o \rightarrow o}$  and  $\{\vee, \wedge, \Rightarrow, \Leftrightarrow, \dots\} \subseteq \Sigma_{o \rightarrow o \rightarrow o}$  (with the intuitive  $\mathcal{I}$ -values)
- ▷ **quantifiers**:  $\Pi^\alpha \in \Sigma_{(\alpha \rightarrow o) \rightarrow o}$  with  $\mathcal{I}(\Pi^\alpha)(p) = \mathbf{T}$ , iff  $p(a) = \mathbf{T}$  for all  $a \in \mathcal{D}_\alpha$ .
- ▷ **quantified formulae**:  $\forall X_\alpha. \mathbf{A}$  stands for  $\Pi^\alpha(\lambda X_\alpha. \mathbf{A})$
- ▷  $\mathcal{I}_\varphi(\forall X_\alpha. \mathbf{A}) = \mathcal{I}(\Pi^\alpha)(\mathcal{I}_\varphi(\lambda X_\alpha. \mathbf{A})) = \mathbf{T}$ , iff  $\mathcal{I}_{\varphi, [a/X]}(\mathbf{A}) = \mathbf{T}$  for all  $a \in \mathcal{D}_\alpha$
- ▷ **looks like PL $\Omega$**  (Call any such system **HOL $^\rightarrow$** )



©: Michael Kohlhase

131



There is a more elegant way to treat quantifiers in  $\text{HOL}^\rightarrow$ . It builds on the realization that the  $\lambda$ -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  $\text{HOL}^\rightarrow$  to introduce quantifiers; only the “lexicon”, i.e. the set of logical constants. Since  $\Pi^\alpha$  and  $\Sigma^\alpha$  are logical constants, we need to fix their semantics.

### Higher-Order Abstract Syntax

- ▷ **Idea:** In  $\text{HOL}^\rightarrow$ , we already have variable binder:  $\lambda$ , use that to treat quantification.

▷ **Definition 12.0.3** We assume logical constants  $\Pi^\alpha$  and  $\Sigma^\alpha$  of type  $\alpha \rightarrow o \rightarrow o$ .

Regain quantifiers as abbreviations:

$$\forall X_\alpha. \mathbf{A} := \Pi^\alpha(\lambda X_\alpha. \mathbf{A}) \quad \exists X_\alpha. \mathbf{A} := \Sigma^\alpha(\lambda X_\alpha. \mathbf{A})$$

▷ **Definition 12.0.4** We must fix the semantics of logical constants:

1.  $\mathcal{I}(\Pi^\alpha)(p) = \top$ , iff  $p(a) = \top$  for all  $a \in \mathcal{D}_\alpha$  (i.e. if  $p$  is the universal set)
2.  $\mathcal{I}(\Sigma^\alpha)(p) = \top$ , iff  $p(a) = \top$  for some  $a \in \mathcal{D}_\alpha$  (i.e. iff  $p$  is non-empty)

▷ With this, we re-obtain the semantics we have given for quantifiers above:

$$\mathcal{I}_\varphi(\forall X_\iota. \mathbf{A}) = \mathcal{I}_\varphi(\Pi^\iota(\lambda X_\iota. \mathbf{A})) = \mathcal{I}(\Pi^\iota)(\mathcal{I}_\varphi(\lambda X_\iota. \mathbf{A})) = \top$$

$$\text{iff } \mathcal{I}_\varphi(\lambda X_\iota. \mathbf{A})(a) = \mathcal{I}_{[a/X], \varphi}(\mathbf{A}) = \top \text{ for all } a \in \mathcal{D}_\alpha$$



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: HOL<sup>=</sup>

▷ only one logical constant  $q^\alpha \in \Sigma_{\alpha \rightarrow \alpha \rightarrow o}$  with  $\mathcal{I}(q^\alpha)(a, b) = \top$ , 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 | $\Pi^\alpha$                               | for | $q^{(\alpha \rightarrow o)}(\lambda X_\alpha. T)$                                                                           |
| N | $\forall X_\alpha. \mathbf{A}$             | for | $\Pi^\alpha(\lambda X_\alpha. \mathbf{A})$                                                                                  |
| D | $\wedge$                                   | for | $\lambda X_o. \lambda Y_o. \lambda G_{o \rightarrow o \rightarrow o}. GTT = \lambda G_{o \rightarrow o \rightarrow o}. GXY$ |
| 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 7.0.15 does not hold (see [And72] for details). Indeed the proof of Theorem 7.0.15 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  $\lambda$ -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 HOL $\rightarrow$

- ▷ **Definition 12.0.5 unary conditional**  $\mathbf{w} \in \Sigma_{\alpha \rightarrow o \rightarrow \alpha}$   
 $\mathbf{wA}_o\mathbf{B}_\alpha$  means: “If  $\mathbf{A}$ , then  $\mathbf{B}$ ”
- ▷ **Definition 12.0.6 binary conditional**  $\mathbf{if} \in \Sigma_{\alpha \rightarrow \alpha \rightarrow o \rightarrow \alpha}$   
 $\mathbf{ifA}_o\mathbf{B}_\alpha\mathbf{C}_\alpha$  means: “if  $\mathbf{A}$ , then  $\mathbf{B}$  else  $\mathbf{C}$ ”.
- ▷ **Definition 12.0.7 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 12.0.8 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  $\mathbf{P}$
- ▷ **Formalization**
  - $\forall X_\alpha. \mathbf{A} \Rightarrow \mathbf{wA}X = X$
  - $\forall X_\alpha, Y_\alpha, Z_\alpha. (\mathbf{A} \Rightarrow \mathbf{ifAXY} = X) \wedge (\neg \mathbf{A} \Rightarrow \mathbf{ifAZX} = X)$
  - $\forall P_{\alpha \rightarrow o}. (\exists^1 X_\alpha. PX) \Rightarrow (\forall Y_\alpha. PY \Rightarrow \iota P = Y)$
  - $\forall P_{\alpha \rightarrow o}. (\exists X_\alpha. PX) \Rightarrow (\forall Y_\alpha. PY \Rightarrow \gamma P = Y)$
- ▷ Ensure a much larger supply of functions in Henkin models.



## More on the Description Operator

- ▷  $\iota$  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  $\mathcal{I}_{[a/X]}(=X) = \{a\}$
  - ▷ 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 13

# Higher-Order Tableaux

### Tableau-Rules ( $\mathcal{T}_\omega^s$ )

▷ **Definition 13.0.9** The rules of  $\mathcal{T}_\omega^s$  consist of the propositional tableau rules of  $\mathcal{T}_0$  together with

$$\frac{\Pi^\alpha \mathbf{A}^\top}{\mathbf{A} \mathbf{C}^\top} \mathcal{T}_\omega^s : \forall \quad \frac{\Pi^\alpha \mathbf{A}^\top \quad c \in (\Sigma_0^{sk} \setminus \mathcal{H})}{\mathbf{A} c^\top} \mathcal{T}_\omega^s : \exists$$



©: Michael Kohlhase

136



### Higher-Order Free-Variable Tableaux ( $\mathcal{T}_\omega$ first try)

▷ **Definition 13.0.10** The  $\mathcal{T}_\omega$  calculus consists of the propositional tableau rules plus

$$\frac{\Pi^\alpha \mathbf{A}^\top}{\mathbf{A} X_\alpha^\top} \mathcal{T}_\omega : \forall \quad \frac{\Pi^\alpha \mathbf{A}^\top \quad \text{free}(\mathbf{A}) = \{Y_{\alpha_1}^1, \dots, Y_{\alpha_n}^n\} \quad f \in \Sigma_{\alpha_n \rightarrow \alpha}^{sk} \text{ new}}{\mathbf{A} (f \overline{Y^n})^\top} \mathcal{T}_\omega : \exists$$

▷ **Problem:** Unification in  $\Lambda^\rightarrow$  is undecidable, so we need more

▷ **Idea:** explicit rule that residuates the unification problem

$$\frac{\mathbf{A}^\alpha \quad \mathbf{B}^\beta}{\mathbf{A} = ? \mathbf{B}} \mathcal{T}_\omega : \text{cut}$$

and adapt the  $\mathcal{HOU}$  rules to tableaux

(DNF instead of CNF)



©: Michael Kohlhase

137



## $\mathcal{T}_\omega$ (Unification)

▷ we can use  $\mathcal{SIM}:\alpha$ ,  $\mathcal{SIM}:\eta$ , and  $\mathcal{SIM}:\text{triv}$  directly, for  $\mathcal{SIM}:\text{dec}$  and  $\mathcal{SIM}:\text{elim}$  we integrate into tableau setting more closely, obtaining

$$\frac{h\overline{\mathbf{U}}^n = ? h\overline{\mathbf{V}}^n \quad h \in (\Sigma \cup \Sigma^{Sk} \cup \mathcal{V}_T)}{\mathbf{U}^1 = ? \mathbf{V}^1 \mid \dots \mid \mathbf{U}^n = ? \mathbf{V}^n} \mathcal{T}_\omega:\text{dec}$$

$$\frac{F_\alpha \overline{\mathbf{U}} = ? h\overline{\mathbf{V}}}{F = ? \mathbf{G} \mid F\overline{\mathbf{U}} = ? h\overline{\mathbf{V}}} \mathcal{T}_\omega:\text{fr}$$

$$\frac{X = ? \mathbf{A} \quad X \notin \text{free}(\mathbf{A}) \quad \mathbf{A} \cap \Sigma^{Sk} = \emptyset}{\perp} \mathcal{T}_\omega:\text{elim}$$

where  $\mathbf{G} = \mathbf{G}_\alpha^h(\Sigma)$  (imitation) or  $\mathbf{G} \in \{\mathbf{G}_\alpha^j(\Sigma) \mid 1 \leq j \leq n\}$



## Example: Cantor's Theorem

▷ **Theorem 13.0.11** *There is no surjective function from the natural numbers into the sequences of natural numbers.*

▷ **Formally:**  $\neg(\exists F_{\iota \rightarrow \iota} \forall G_{\iota \rightarrow \iota} \exists J_\iota. FJ = G)$

▷ For the proof we use

▷  $\forall X_\iota. \neg X = sX$  (the successor function has no fixed points)

▷ an extensionality axiom



## $\mathcal{T}_\omega$ -Proof (Cantor's Theorem)

▷ First the propositional part (analyzing formula structure)

$$\begin{array}{c}
 \neg(\exists F_{\iota \rightarrow \iota \rightarrow \iota} \forall G_{\iota \rightarrow \iota} \exists J_{\iota} F J = G)^F \\
 \exists F_{\iota \rightarrow \iota \rightarrow \iota} \forall G_{\iota \rightarrow \iota} \exists J_{\iota} F J = G^T \\
 \forall G_{\iota \rightarrow \iota} \exists J_{\iota} f_{\iota \rightarrow \iota \rightarrow \iota} J = G^T \\
 \exists J_{\iota} f J = G^T \\
 f(jG) = G^T \\
 H = K \Rightarrow (\forall N_{\iota} H N = K N)^T \\
 H = K^F \quad H N = K N^T \\
 H = K = ? f(jG) = G \quad f(jG) N = G N^T \\
 H = ? f(jG) \mid K = ? G \quad X = s X^F \\
 \perp \quad \perp \quad (X = s X) = ? (f(jG) N = G N)
 \end{array}$$

▷ then we continue with unification

$$\begin{array}{c}
 (X = s X) = ? (f(jG) N = G N) \\
 \swarrow \quad \downarrow \\
 X = ? f(jG) N \quad G N = ? s(f(jG) N) \\
 \swarrow \quad \downarrow \\
 G = ? (\lambda Y_{\iota} s(H^1 Y)) \quad s(H^1 N) = ? s(f(jG) N) \\
 \downarrow \\
 H^1 N = ? f(jG) N \\
 \swarrow \quad \downarrow \\
 H^1 = ? (\lambda Y_{\iota} f(H^2 Y)(H^3 Y)) \quad f(H^2 N)(H^3 N) = ? f(jG) N \\
 \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \\
 H^2 N = ? jG \quad H^3 N = ? N \\
 \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \\
 H^2 = ? (\lambda Y_{\iota} Y) \quad N = ? jG \quad H^3 = ? (\lambda Y_{\iota} Y) \quad jG = ? jG \\
 \perp \quad \perp \quad \perp \quad \perp
 \end{array}$$



## Problem for $\mathcal{T}_{\omega}$

▷ **Theorem 13.0.12** *There is a valid vormala  $(\exists X_o. X)$*

▷ This is clearly valid,

(eg.  $\forall \mathbf{A}, \neg \mathbf{A}$ )

▷  $\mathcal{T}_{\omega}$  attempt

$$\begin{array}{c}
 \neg(\forall X_o. \neg X)^F \\
 \forall X_o. \neg X^T \\
 \neg X^T \\
 X^F
 \end{array}$$

▷ we are stuck!

▷ **Observation:** We have to instantiate  $X$  further, e.g. by  $[\neg Q_o / X]$ .

▷ then we can continue

$$\begin{array}{c} X^F \\ \neg Q^F \\ Q^T \\ X = ? Q \end{array}$$

close with  $[Q/X]$ .



## 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 13.0.13** If  $\mathcal{T}$  contains a term  $X\overline{U}^n{}^\alpha$ , and  $\mathbf{G} \in \mathbf{G}_\alpha^k(\Sigma)$  for  $k \in (\{\wedge, \neg\} \cup \{\Pi^\beta \mid \beta \in \mathcal{T}\})$ , then instantiate the tableau with  $[\mathbf{G}/X]$ .



## Another Example

▷  $\mathbf{A} = \neg(c_{(o \rightarrow o)}b_o) \vee (c\neg b)$  is valid

▷  $\mathcal{T}_\omega$  proof attempt

$$\begin{array}{c} \neg cb \vee c(\neg\neg b)^F \\ \neg cb^F \\ c(\neg\neg b)^F \\ cb^T \\ cb = ? c(\neg\neg b) \\ b = ? \neg\neg b \end{array}$$

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:** Recursive call to the theorem prover



## Part III

# 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 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 an entailment relation to ensure applicability.

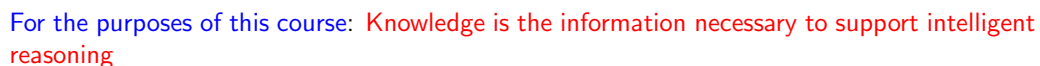




# Introduction to Knowledge Representation

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.

- ▷ According to Probst/Raub/Romhardt [PRR97]



|                |                                     |
|----------------|-------------------------------------|
| 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                            |

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)



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 AI/NLP

- ▷ GOF AI (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 patterns, etc.



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.



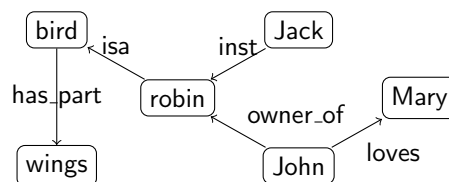
## 14.1 Semantic Networks

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 14.1.1** A **semantic network** is a graph structure for representing knowledge:
  - ▷ nodes represent concepts (e.g. *bird*, *John*, *robin*)
  - ▷ links represent relations between these (*isa*, *father\_of*, *belongs\_to*)
- ▷ **Example 14.1.2** 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



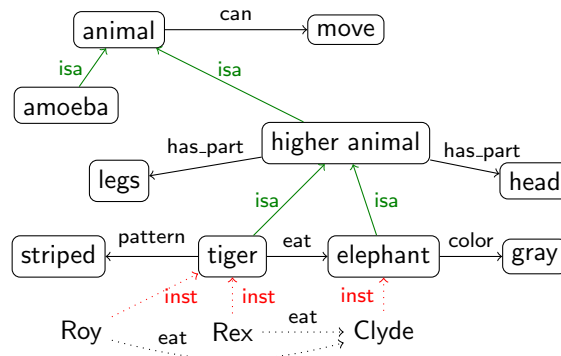
Even though the network in Example 14.1.2 is very intuitive (we immediately understand the concepts depicted), it is unclear how we (and more importantly a machine that does not associate meaning with the labels of the nodes and edges) can draw inferences from the “knowledge” represented.

Another problem is that the semantic net in Example 14.1.2 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 14.1.3** From the network



infer that *elephants* have *legs* and that *Clyde* is *gray*.

▷ **Definition 14.1.4** 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 several 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.

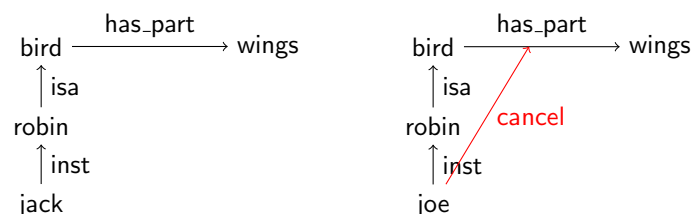
## Limitations of Semantic Networks

▷ 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 14.1.5** 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

▷ **Idea:** use function/argument notation

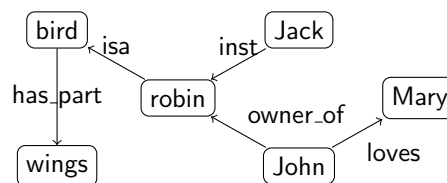
▷ Interpret nodes as arguments

(reification to individuals)

▷ Interpret links as functions

(logical relations)

▷ **Example 14.1.6**



```

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)



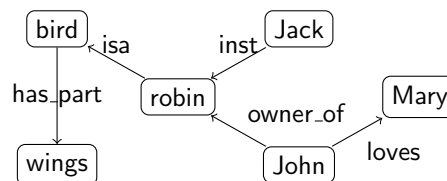
Indeed the function/argument notation is the immediate idea how one would naturally represent semantic networks for implementation.

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 ⊆ bird
haspart(bird,wings)
Jack ∈ 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  $S(a)$

$\triangleright haspart(A, B)$  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)



©: Michael Kohlhasse

152



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.

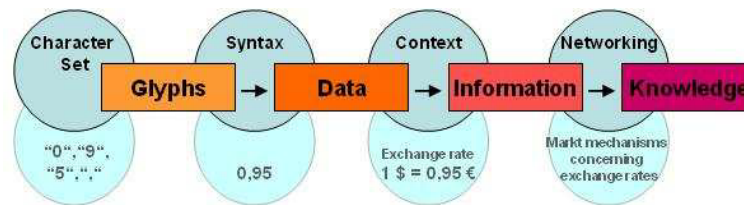
Before we go on, let us have a look at an important application of knowledge representation technologies: the Semantic Web.

## 14.2 The Semantic Web

## The Semantic Web

▷ **Definition 14.2.1** 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 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 14.2.2** 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.



©: Michael Kohlhasse

153



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 to recover information that is not explicitly represented from the network (here the world-wide-web).

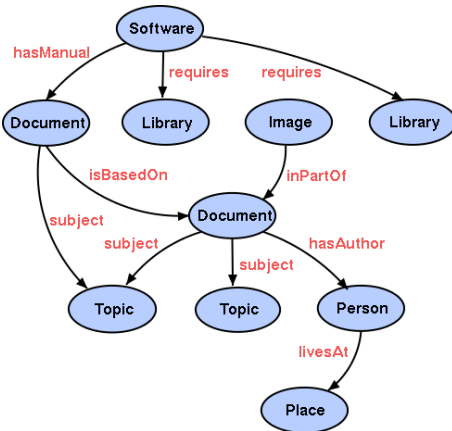








- ▷ **Resources:** Globally Identified by URI's or Locally scoped (Blank), Extensible, Relational
- ▷ **Links:** Identified by URI's, Extensible, Relational
- ▷ **User:** Even more exciting world, richer user experience
- ▷ **Machine:** More processable information is available (Data Web)
- ▷ **Computers and people:** Work, learn and exchange knowledge effectively



©: Michael Kohlhase

160



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 [HASB13] 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)



©: Michael Kohlhase

161



## 14.3 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* ∈ *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])

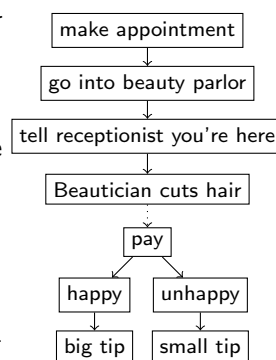
- ▷ **Idea:** organize typical event sequences, actors and props into representation structure

- ▷ **Example 14.3.1** 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 15

# 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)



©: Michael Kohlhase

165



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.

## 15.1 Propositional Logic as a Set Description Language

Before we look at “real” description logics in Chapter 16, 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.

### ▷ Propositional Logic as Set Description Language

- ▷ **Idea:** use propositional logic as a set description language
- ▷ variant syntax:  $\sqcap \triangleq \wedge$  (intersection),  $\sqcup \triangleq \vee$  (union),  $\neg \triangleq \neg$  (complement),  $\sqsubseteq \triangleq \Rightarrow$  (subsumption)
- ▷ **Example 15.1.1**

| Example                                                                                                                                                                                      | Set Semantics |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| $\text{son} \sqsubseteq \text{child}$<br>$\text{daughter} \sqsubseteq \text{child}$<br>$\text{son} \sqcap \text{daughter}$<br>$\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})$ |               |

- ▷ **Definition 15.1.2 (Formal Semantics)** let  $\mathcal{D}$  be a given set (called the **domain**) and  $\varphi: \mathcal{V}_o \rightarrow \wp(\mathcal{D})$ , then
  - ▷  $\llbracket P \rrbracket := \varphi(P) \subseteq \mathcal{D}$ ,
  - ▷  $\llbracket \mathbf{A} \sqcup \mathbf{B} \rrbracket = \llbracket \mathbf{A} \rrbracket \cup \llbracket \mathbf{B} \rrbracket$  and  $\llbracket \neg \mathbf{A} \rrbracket = \mathcal{D} \setminus \llbracket \mathbf{A} \rrbracket \dots$



### Effects of the Axioms in this example

- ▷ **Idea:** use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)
- ▷ **Example 15.1.3**

| Axioms                                                                                                | Effect |
|-------------------------------------------------------------------------------------------------------|--------|
| $\text{son} \sqsubseteq \text{child}$<br>$\text{daughter} \sqsubseteq \text{child}$                   |        |
| $\text{son} \sqcap \text{daughter}$<br>$\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})$ |        |



## Predicate-Logic Formulation

| Propositional Logic                                            | Predicate Logic                                                                |
|----------------------------------------------------------------|--------------------------------------------------------------------------------|
| $\text{son} \sqsubseteq \text{child}$                          | $\forall x. \text{son}(x) \Rightarrow \text{child}(x)$                         |
| $\text{daughter} \sqsubseteq \text{child}$                     | $\forall x. \text{daughter}(x) \Rightarrow \text{child}(x)$                    |
| $\text{son} \sqcap \text{daughter}$                            | $\forall x. \neg(\text{son}(x) \wedge \text{daughter}(x))$                     |
| $\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})$ | $\forall x. \text{child}(x) \Rightarrow \text{son}(x) \vee \text{daughter}(x)$ |



## Set-Theoretic Semantics

▷ **Definition 15.1.4** A model  $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$  consists of a **interpretation**  $\mathcal{I}$  over a non-empty domain  $\mathcal{D}$  is a mapping  $\llbracket \cdot \rrbracket$ :

|   | Operator Meaning                                       | formula semantics                                                                                                                                                           |
|---|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 |                                                        | $\llbracket p \rrbracket \subset \mathcal{D}$                                                                                                                               |
| 2 | $\llbracket \neg \cdot \rrbracket = \text{complement}$ | $\llbracket \neg A \rrbracket = \overline{\llbracket A \rrbracket} := \mathcal{D} \setminus \llbracket A \rrbracket$                                                        |
| 3 | $\llbracket \sqcap \rrbracket = \cap$                  | $\llbracket A \sqcap B \rrbracket = \llbracket A \rrbracket \cap \llbracket B \rrbracket$                                                                                   |
| 4 | $\llbracket \sqcup \rrbracket = \cup$                  | $\llbracket A \sqcup B \rrbracket = \llbracket A \rrbracket \cup \llbracket B \rrbracket$                                                                                   |
| 5 | $\llbracket \sqsubseteq \rrbracket = \subseteq_R$      | $\llbracket A \sqsubseteq B \rrbracket = \llbracket A \rrbracket \subseteq \llbracket B \rrbracket$                                                                         |
| 6 | $\llbracket \equiv \rrbracket = \text{set equality}$   | $\llbracket A \equiv B \rrbracket = \llbracket A \rrbracket \cap \llbracket B \rrbracket \cup (\overline{\llbracket A \rrbracket} \cap \overline{\llbracket B \rrbracket})$ |

▷ Justification for 5:  $A \Rightarrow B = \neg(A) \vee B$

▷ Justification for 6:  $A \Leftrightarrow B = \forall A \wedge B, \neg A \wedge \neg B = \forall A \wedge B, \neg \forall A, B$



## Set-Theoretic Semantics of Axioms

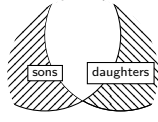
▷ Set-Theoretic Semantics of 'true' and 'false'

$$(\top = \varphi \sqcup \overline{\varphi} \quad \perp = \varphi \sqcap \overline{\varphi})$$

$$\llbracket \perp \rrbracket = \llbracket p \rrbracket \cup \llbracket \overline{p} \rrbracket = \llbracket p \rrbracket \cup \overline{\llbracket p \rrbracket} = \mathcal{D}$$

$$\text{Analogously: } \llbracket \top \rrbracket = \emptyset$$

▷ Set-Theoretic Semantics of Axioms:  $A$  is true in  $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ , iff  $\llbracket A \rrbracket = \mathcal{D}$

| Axioms                                                                                                                                                                                                                                       | Semantics                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| $\text{son} \sqsubseteq \text{child} \text{ is true iff } \llbracket \text{son} \rrbracket \cup \llbracket \text{child} \rrbracket = \mathcal{D} \text{ iff } \llbracket \text{son} \rrbracket \subseteq \llbracket \text{child} \rrbracket$ |                                          |
| $\text{son} \sqsubseteq \text{child}$                                                                                                                                                                                                        | $\llbracket \text{son} \rrbracket \subseteq \llbracket \text{child} \rrbracket$                                            |
| $\text{daughter} \sqsubseteq \text{child}$                                                                                                                                                                                                   | $\llbracket \text{daughter} \rrbracket \subseteq \llbracket \text{child} \rrbracket$                                       |
| $\text{son} \sqcap \text{daughter}$                                                                                                                                                                                                          | $\llbracket \text{son} \rrbracket \cap \llbracket \text{daughter} \rrbracket = \mathcal{D}$                                |
| $\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})$                                                                                                                                                                               | $\llbracket \text{child} \rrbracket \subseteq \llbracket \text{son} \rrbracket \cup \llbracket \text{daughter} \rrbracket$ |



## Set-Theoretic Semantics and Predicate Logic

▷ use logical operators  $\sqcap, \sqcup, \sqsubseteq, \equiv$  instead of  $\wedge, \vee, \Rightarrow, \Leftrightarrow$  if we are using  $\text{PL}^0$  with set-theoretic semantics.

▷ Translation into  $\text{PL}^1$

- ▷ 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                         |
|-------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| $\overline{p}^{fo(x)} := p(x)$                                                                                                      |                                 |
| $\overline{(\mathbf{A})}^{fo(x)} := \neg \mathbf{A}^{fo(x)}$                                                                        |                                 |
| $\overline{(\mathbf{A} \sqcap \mathbf{B})}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \wedge \overline{\mathbf{B}}^{fo(x)}$           | $\wedge$ 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                    |



## Translation Examples

▷ **Example 15.1.5**

$$\begin{aligned} \overline{\text{son} \sqsubseteq \text{child}}^{fo} &= \forall x. \text{son}(x) \Rightarrow \text{child}(x) \\ \overline{\text{daughter} \sqsubseteq \text{child}}^{fo} &= \forall x. \text{daughter}(x) \Rightarrow \text{child}(x) \\ \overline{(\text{son} \sqsubseteq \text{daughter})}^{fo} &= \forall x. \overline{\text{son}(x) \wedge \text{daughter}(x)} \\ \overline{\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})}^{fo} &= \forall x. \text{child}(x) \Rightarrow \text{son}(x) \vee \text{daughter}(x) \end{aligned}$$

▷ What are the advantages of translation to  $\text{PL}^1$ ?

▷ **theoretically**: A better understanding of the semantics

▷ **computationally**: **NOTHING**

many tests are decidable for  $\text{PL}^0$ , but not for  $\text{PL}^1$

(Description Logics?)





## 15.2 Description Logics and Inference

### 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



©: Michael Kohlhase

173



### Consistency Test

- ▷ **Example 15.2.1** T-Box

|               |   |                       |                             |
|---------------|---|-----------------------|-----------------------------|
| woman         | = | person $\sqcap$ has_Y | person without y-chromosome |
| man           | = | person $\sqcap$ has_Y | person with y-chromosome    |
| hermaphrodite | = | man $\sqcap$ woman    | man and woman               |

- ▷ This specification is inconsistent, i.e.  $\llbracket \text{hermaphrodite} \rrbracket = \emptyset$  for all  $\mathcal{D}, \varphi$ .
- ▷ **Algorithm:** propositional satisfiability test (NP-complete) we know how to do this, e.g. tableau, resolution



©: Michael Kohlhase

174



### Subsumption Test

- ▷ **Example 15.2.2** in this case trivial

| Axioms                        | entailed subsumption relation |
|-------------------------------|-------------------------------|
| woman = person $\sqcap$ has_Y | woman $\sqsubseteq$ person    |
| man = person $\sqcap$ has_Y   | man $\sqsubseteq$ person      |

**Reduction to consistency test:** (need to implement only one)  $Axioms \Rightarrow A \Rightarrow B$  is valid iff  $Axioms \wedge A \wedge \neg B$  is inconsistent.

- ▷ **Definition 15.2.3** A **subsumes** B (modulo an axiom set  $\mathcal{A}$ )
  - iff  $\llbracket B \rrbracket \subseteq \llbracket A \rrbracket$  for all interpretations  $\mathcal{D}$ , that satisfy  $\mathcal{A}$
  - iff  $Axioms \Rightarrow B \Rightarrow A$  is valid

- ▷ **in our example:** person subsumes woman and man



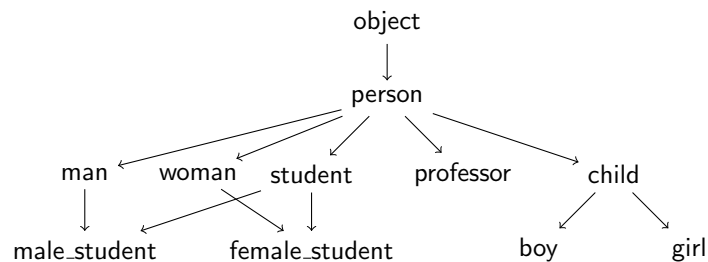
©: Michael Kohlhase

175



## Classification

- ▷ The subsumption relation among **all** concepts (subsumption graph)
- ▷ Visualization of the Subsumption graph for inspection (plausibility)
- ▷ **Definition 15.2.4** **Classification** is the computation of the subsumption graph.
- ▷ **Example 15.2.5** (not always so trivial)



©: Michael Kohlhase

176



## Instance Test

- ▷ **Example 15.2.6** (will explain TBox and ABox with ALC later)

| T-Box (terminological Box) |                         | A-Box (assertional Box, data base) |                         |
|----------------------------|-------------------------|------------------------------------|-------------------------|
| woman                      | = person $\sqcap$ has_Y | tony: person                       | Tony is a person        |
| man                        | = person $\sqcap$ has_Y | tony: has_Y                        | Tony has a y-chromosome |

- ▷ This entails: tony: man (Tony is a man).



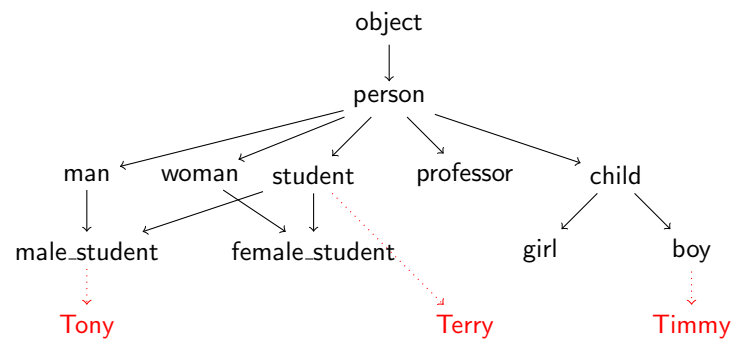
©: Michael Kohlhase

177



## Realization

- ▷ **Definition 15.2.7** **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.



## Chapter 16

# Description Logics and the Semantic Web

### Resource Description Framework

- ▷ **Definition 16.0.8** The **Resource Description Framework** (RDF) is a framework for describing resources on the web. It is a XML vocabulary developed by the W3C.
- ▷ **Note:** RDF is designed to be read and understood by computers, not to be being displayed to people
- ▷ **Example 16.0.9** 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



©: Michael Kohlhase

179



### Resources and URIs

- ▷ RDF describes resources with properties and property values.
- ▷ RDF uses Web identifiers (URIs) to identify resources.
- ▷ **Definition 16.0.10** A **resource** is anything that can have a URI, such as `http://www.jacobs-university.de`
- ▷ **Definition 16.0.11** 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 16.0.12** The combination of a resource, a property, and a property value forms a **statement** (known as the **subject**, **predicate** and **object** of a statement).

▷ **Example 16.0.13** Statement: *The [author]<sup>pred</sup> of [this slide]<sup>subj</sup> is [Michael Kohlhase]<sup>obj</sup>*



## XML Syntax for RDF

▷ RDF is a concrete XML vocabulary for writing statements

▷ **Example 16.0.14** 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://svn.kwarc.info/.../slides/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.

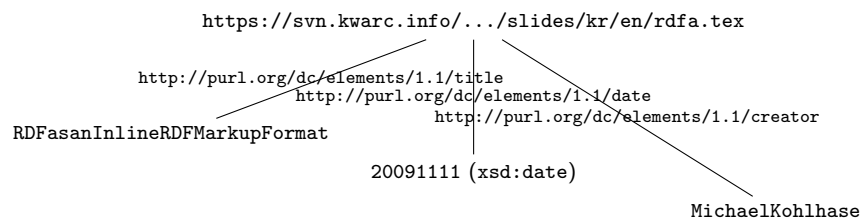
*Intuitively:* RDF is a way to write down ABox information in a web-scalable way.



## ▷ RDFa as an Inline RDF Markup Format

▷ **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/">
  <h2 property="dc:title">RDF as an Inline RDF Markup Format</h2>
  <h3 property="dc:creator">Michael Kohlhase</h3>
  <em property="dc:date" datatype="xsd:date"
    content="20091111">November 11., 2009</em>
</div>
```



## OWL as an Ontology Language for the Semantic Web

- ▷ **Idea:** Use Description Logics to talk about RDF triples.
- ▷ An RDF triple is an ABox entry for a role constraint  $hRs$
- ▷ **Example 16.0.16**  $h$  is the resource for Ian Horrocks,  $s$  is the resource for Ulrike Sattler, and  $R$  is the relation "hasColleague" in

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

**Idea:** Now collect similar resources in *classes*, and state rules about them in a way, so that we can use inference to make knowledge explicit that was implicit before (saves us lots of work!)

- ▷ **Idea:** We know how to do this, this is just  $\mathcal{ALC}+!!!$



## The OWL Language

- ▷ Three species of OWL
  - ▷ OWL Full is union of OWL syntax and RDF
  - ▷ OWL DL restricted to FOL fragment
  - ▷ OWL Lite is "easier to implement" subset of OWL DL
- ▷ Semantic layering
  - ▷ OWL DL  $\triangleq$  OWL Full within DL fragment
  - ▷ DL semantics officially definitive
  - ▷ OWL DL based on SHIQ Description Logic ( $\mathcal{ALC}$  + number restrictions, transitive roles, inverse roles, role inclusions)
  - ▷ OWL DL benefits from many years of DL research
  - ▷ Well defined semantics, formal properties well understood (complexity, decidability)
  - ▷ Known reasoning algorithms, Implemented systems (highly optimized)







# Chapter 17

## A simple Description Logic: ALC

### Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

- ▷ Propositional logic ( $PL^0$ ) is not expressive enough
- ▷ **Example 17.0.17** “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 ( $PL^1$ )  
$$\forall x.mother(x) \Leftrightarrow woman(x) \wedge (\exists y.has\_child(x,y))$$
- ▷ **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 *has\_child*.



©: Michael Kohlhase

185



### Syntax of $\mathcal{ALC}$

- ▷ **Definition 17.0.18 (Concepts)** (aka. “predicates” in  $PL^1$  or “propositional variables” in  $PL^0$ )  
concepts in DLs name classes of objects like in OOP.
- ▷ **Definition 17.0.19 (Special concepts)** The **top concept**  $\top$  (for “true” or “all”) and the **bottom concept**  $\perp$  (for “false” or “none”).
- ▷ **Example 17.0.20** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, ...
- ▷ name binary relations (like in  $PL^1$ )
- ▷ **Example 17.0.21** has\_child, has\_son, has\_daughter, loves, hates gives\_course, executes\_computer\_program, has\_leg\_of\_table, has\_wheel, has\_motor, ...



## Syntax of $\mathcal{ALC}$ : Formulae $F_{\mathcal{ALC}}$

▷  $F_{\mathcal{ALC}} ::= C \mid \top \mid \perp \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid (\exists R.F_{\mathcal{ALC}}) \mid (\forall R.F_{\mathcal{ALC}})$

▷ **Example 17.0.22** ▷  $\text{person} \sqcap (\exists \text{has\_child}.\text{student})$  (parents of students)

(The set of persons that have a child which is a student)

▷  $\text{person} \sqcap (\exists \text{has\_child}.\exists \text{has\_child}.\text{student})$  (grandparents of students)

▷  $\text{person} \sqcap (\exists \text{has\_child}.\exists \text{has\_child}.\text{student} \sqcup \text{teacher})$  (grandparents of students or teachers)

▷  $\text{person} \sqcap (\forall \text{has\_child}.\text{student})$  (parents whose children are all students)

▷  $\text{person} \sqcap (\forall \text{haschild}.\exists \text{has\_child}.\text{student})$  (grandparents, whose children all have at least one child that is a student)



## More $\mathcal{ALC}$ Examples

▷  $\text{car} \sqcap (\exists \text{has\_part}.\exists \text{made\_in}.\overline{\text{EU}})$  (cars that have at least one part that has not been made in the EU)

▷  $\text{student} \sqcap (\forall \text{audits\_course}.\text{graduatelevelcourse})$  (students, that only audit graduate level courses)

▷  $\text{house} \sqcap (\forall \text{has\_parking}.\text{off\_street})$  (houses with off-street parking)

▷ **Note:**  $p \sqsubseteq q$  can still be used as an abbreviation for  $\overline{p} \sqcup q$ .

▷  $\text{student} \sqcap (\forall \text{audits\_course}.\exists \text{has\_tutorial}.\top) \sqsubseteq (\forall \text{has\_TA}.\text{woman})$  (students that only audit courses that either have no t



## $\mathcal{ALC}$ Concept Definitions

▷ Define new concepts from known ones: ( $KD_{\mathcal{ALC}} ::= C = F_{\mathcal{ALC}}$ )

Definition	rec?
$\text{man} = \text{person} \sqcap (\exists \text{has\_chrom}.\text{Y\_chrom})$	-
$\text{woman} = \text{person} \sqcap (\forall \text{has\_chrom}.\overline{\text{Y\_chrom}})$	-
$\text{mother} = \text{woman} \sqcap (\exists \text{has\_child}.\text{person})$	-
$\text{father} = \text{man} \sqcap (\exists \text{has\_child}.\text{person})$	-
$\text{grandparent} = \text{person} \sqcap (\exists \text{has\_child}.\text{mother} \sqcup \text{father})$	-
$\text{german} = \text{person} \sqcap (\exists \text{has\_parents}.\text{german})$	+
$\text{number\_list} = \text{empty\_list} \sqcup (\exists \text{is\_first}.\text{number}) \sqcap (\exists \text{is\_rest}.\text{number\_list})$	+



## Concept Axioms

▷ **Definition 17.0.23** General DL formulae that are not concept definitions are called **Concept Axioms**.

- ▷ They normally contain additional information about concepts
- ▷ **Example 17.0.24** ▷  $\overline{\text{person}} \sqcap \text{car}$  (persons and cars are disjoint)
- ▷  $\text{car} \sqsubseteq \text{motor\_vehicle}$  (cars are motor vehicles)
- ▷  $\text{motor\_vehicle} \sqsubseteq (\text{car} \sqcup \text{truck} \sqcup \text{motorcycle})$  (motor vehicles are cars, trucks, or motorcycles)



## TBoxes: “terminological Box”

- ▷ **Definition 17.0.25** finite set of concept definitions + finite set of concept axioms
- ▷ **Definition 17.0.26** **Acyclic TBox** (mostly treated)  
TBox does not contain recursive definitions
- ▷ **Definition 17.0.27** (Normalized wrt. TBox) A formula **A** is called **normalized** wrt. **T**, iff it does not contain concept names defined in **T**. (convenient)
- ▷ **Definition 17.0.28** (Algorithm) Input: A formula **A** and a TBox **T**. (for arbitrary DLs)
  - ▷ **While** [**A** contains concept name **c** and **T** concept definition  $c = \mathbf{C}$ ]
    - ▷ substitute **c** by **C** in **A**.
- ▷ **Lemma 17.0.29** *this algorithm terminates for acyclic TBoxes*



## Normalization Example (normalizing grandparent)

**grandparent**

- ⇒  $\text{person} \sqcap (\exists \text{has\_child}.\text{mother} \sqcup \text{father})$
- ⇒  $\text{person} \sqcap (\exists \text{has\_child}.\text{woman} \sqcap (\exists \text{has\_child}.\text{person}), \text{man}, \exists \text{has\_child}.\text{person})$
- ⇒  $\text{person} \sqcap (\exists \text{has\_child}.\text{person} \sqcap (\exists \text{has\_chrom}.\text{Y\_chrom}) \sqcap (\exists \text{has\_child}.\text{person}) \sqcap \text{person} \sqcap (\exists \text{has\_chrom}.\text{Y\_chrom}) \sqcap (\exists \text{has\_child}.\text{person}))$

- ▷ **Observation:** normalization result can be exponential and redundant
- ▷ **Observation:** need not terminate on cyclic TBoxes

**german** ⇒  $\text{person} \sqcap (\exists \text{has\_parents}.\text{german})$

⇒  $\text{person} \sqcap (\exists \text{has\_parents}.\text{person} \sqcap (\exists \text{has\_parents}.\text{german}))$

⇒ ...



## Semantics of $\mathcal{ALC}$

- ▷  $\mathcal{ALC}$  semantics is an extension of the set-semantics of propositional logic.

▷ **Definition 17.0.30** An **Interpretation**  $\mathcal{I}$  over a non-empty domain  $\mathcal{D}$  is a mapping  $\llbracket \cdot \rrbracket$ :

Op.	formula semantics
	$\llbracket c \rrbracket \subseteq \mathcal{D} = \llbracket \top \rrbracket \quad \llbracket \perp \rrbracket = \emptyset \quad \llbracket r \rrbracket \subseteq \mathcal{D} \times \mathcal{D}$
$\neg$	$\llbracket \neg \varphi \rrbracket = \mathcal{D} \setminus \llbracket \varphi \rrbracket$
$\sqcap$	$\llbracket \varphi \sqcap \psi \rrbracket = \llbracket \varphi \rrbracket \cap \llbracket \psi \rrbracket$
$\sqcup$	$\llbracket \varphi \sqcup \psi \rrbracket = \llbracket \varphi \rrbracket \cup \llbracket \psi \rrbracket$
$\exists R.$	$\llbracket \exists R.\varphi \rrbracket = \{x \in \mathcal{D} \mid \exists y. \langle x, y \rangle \in \llbracket R \rrbracket \text{ and } y \in \llbracket \varphi \rrbracket\}$
$\forall R.$	$\llbracket \forall R.\varphi \rrbracket = \{x \in \mathcal{D} \mid \forall y. \text{ if } \langle x, y \rangle \in \llbracket R \rrbracket \text{ then } y \in \llbracket \varphi \rrbracket\}$



## Propositional Identities

Name	for $\sqcap$	for $\sqcup$
Idempot.	$\varphi \sqcap \varphi = \varphi$	$\varphi \sqcup \varphi = \varphi$
Identity	$\varphi \sqcap \top = \varphi$	$\varphi \sqcup \perp = \varphi$
Absorpt.	$\varphi \sqcup \top = \top$	$\varphi \sqcap \perp = \perp$
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.	$\varphi \sqcap (\varphi \sqcup \theta) = \varphi$	$\varphi \sqcup (\varphi \sqcap \theta) = \varphi$
Morgan	$\overline{\varphi \sqcap \psi} = \overline{\varphi} \sqcup \overline{\psi}$	$\overline{\varphi \sqcup \psi} = \overline{\varphi} \sqcap \overline{\psi}$
dneg	$\overline{\overline{\varphi}} = \varphi$	



## More $\mathcal{ALC}$ Identities

▷ $\overline{\exists R.\varphi} = \forall R.\overline{\varphi}$	$\overline{\forall R.\varphi} = \exists R.\overline{\varphi}$
$\overline{\forall R.\varphi \sqcap \psi} = \forall R.\varphi \sqcap (\forall R.\overline{\psi})$	$\overline{\exists R.\varphi \sqcup \psi} = \exists R.\varphi \sqcup (\exists R.\overline{\psi})$

▷ Proof of 1

$$\begin{aligned}
 \llbracket \overline{\exists R.\varphi} \rrbracket &= \mathcal{D} \setminus \llbracket (\exists R.\varphi) \rrbracket &= \mathcal{D} \setminus \{x \in \mathcal{D} \mid \exists y. \langle x, y \rangle \in \llbracket R \rrbracket \text{ and } y \in \llbracket \varphi \rrbracket\} \\
 &= \{x \in \mathcal{D} \mid \text{not } \exists y. \langle x, y \rangle \in \llbracket R \rrbracket \text{ and } y \in \llbracket \varphi \rrbracket\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } \langle x, y \rangle \in \llbracket R \rrbracket \text{ then } y \notin \llbracket \varphi \rrbracket\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } \langle x, y \rangle \in \llbracket R \rrbracket \text{ then } y \in (\mathcal{D} \setminus \llbracket \varphi \rrbracket)\} \\
 &= \{x \in \mathcal{D} \mid \forall y. \text{if } \langle x, y \rangle \in \llbracket R \rrbracket \text{ then } y \in \llbracket \overline{\varphi} \rrbracket\} \\
 &= \llbracket \forall R.\overline{\varphi} \rrbracket
 \end{aligned}$$



## Negation Normal Form

▷ **Definition 17.0.31 (NNF)**  $\neg$  directly in front of concept names in  $\mathcal{ALC}$  formulae

▷ Use the  $\mathcal{ALC}$  rules to compute it.

(in linear time)

example	by rule
$\overline{\exists R.(\forall S.e) \sqcap (\forall S.d)}$	$\overline{\exists R.\varphi} \mapsto \forall R.\overline{\varphi}$
$\mapsto \forall R.(\overline{\forall S.e} \sqcap \overline{\forall S.d})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall R.(\overline{\forall S.e} \sqcup \overline{\forall S.d})$	$\overline{\forall R.\varphi} \mapsto \forall R.\overline{\varphi}$
$\mapsto \forall R.(\exists S.\overline{e}) \sqcup (\forall S.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$
$\mapsto \forall R.(\exists S.\overline{e}) \sqcup (\forall S.d)$	



## $\mathcal{T}_{\mathcal{ALC}}$ : A Tableau-Calculus for $\mathcal{ALC}$

$\frac{x: c \quad x: \overline{c}}{\perp} *$	$\frac{x: \varphi \sqcap \psi}{x: \varphi \quad x: \psi} \sqcap$	$\frac{x: \varphi \sqcup \psi}{x: \varphi \mid x: \psi} \sqcup$	$\frac{x: \forall R.\varphi \quad x R y}{y: \varphi} \forall$	$\frac{x: \exists R.\varphi}{x R y \quad y: \varphi} \exists$
----------------------------------------------	------------------------------------------------------------------	-----------------------------------------------------------------	---------------------------------------------------------------	---------------------------------------------------------------

▷ The tableau calculus acts on judgments of the form

▷  $x: \varphi$ :

( $x$  is in the set  $\varphi$ )

▷  $x R y$ :

( $x$  and  $y$  are in relation  $R$ )



## Examples

1	$x: \forall \text{has\_child.man} \sqcap \exists \text{has\_child.man}$	initial	$x: \forall \text{has\_child.man} \sqcap \exists \text{has\_child.man}$	initial
2	$x: \forall \text{has\_child.man}$	$\sqcap_l$	$x: \forall \text{has\_child.man}$	$\sqcap_l$
3	$x: \exists \text{has\_child.man}$	$\sqcap_r$	$x: \exists \text{has\_child.man}$	$\sqcap_r$
4	$x \text{ has\_child } y$	$\exists_r$	$x \text{ has\_child } y$	$\exists_r$
5	$y: \text{man}$	$\exists_s$	$y: \text{man}$	$\exists_s$
6	$y: \text{man}$	$\forall$	<b>open</b>	
7	$*$	$\perp$		
	<b>inconsistent</b>			

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

▷ **Example 17.0.32**

1	$x : (\forall \text{has\_child.ugrad} \sqcup \text{grad}) \sqcap (\exists \text{has\_child.ugrad})$	
2	$x : \forall \text{has\_child.ugrad} \sqcup \text{grad}$	$\sqcap_l$
3	$x : \exists \text{has\_child.ugrad}$	$\sqcap_r$
4	$x \text{ has\_child } y$	$\exists_s$
5	$y : \text{ugrad}$	$\exists_r$
6	$y : \text{ugrad} \sqcup \text{grad}$	$\forall$
7	$y : \text{ugrad}$ $y : \text{grad}$	$\sqcup$
8	$*$ <b>open</b>	

The left branch is closed, the right one represents a model:  $y$  is a child of  $x$ ,  $y$  is a graduate student,  $x$  has exactly one child:  $y$ .



## 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  $\varphi$  is consistent, then  $\mathcal{C}$  terminates with an open branch.

**Completeness** If  $\varphi$  is inconsistent, then  $\mathcal{C}$  terminates and all branches are closed.

**Complexity** of the algorithm

**Complexity** of the satisfiability itself



## Termination

▷ **Theorem 17.0.33** *The Tableau Algorithm for ALC terminates*

To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

▷ **Proof:** Sketch (full proof very technical)

**P.1** any rule except  $\forall$  can only be applied once to  $x : \psi$ .

**P.2** rule  $\forall$  applicable to  $x : \forall R.\psi$  at most as the number of R-successors of  $x$ . (those  $y$  with  $x R y$  above)

**P.3** the R-successors are generated by  $x : \exists R.\theta$  above, (number bounded by size of input formula)

**P.4** every rule application to  $x : \psi$  generates constraints  $z : \psi'$ , where  $\psi'$  a proper sub-formula of  $\psi$ . □



## Correctness

▷ **Lemma 17.0.34** *If  $\varphi$  consistent, then  $\mathcal{T}$  terminates on  $x : \varphi$  with open branch.*

▷ **Proof:** Let  $\mathcal{M}$  be a model for  $\varphi$  and  $w \in \llbracket \varphi \rrbracket$ .

**P.1** we define  $\llbracket x \rrbracket := w$  and

$$\begin{aligned} \mathfrak{S} \models x: \psi & \text{ iff } \llbracket x \rrbracket \in \llbracket \psi \rrbracket \\ \mathfrak{S} \models x R y & \text{ iff } \langle x, y \rangle \in \llbracket R \rrbracket \\ \mathfrak{S} \models S & \text{ iff } \mathfrak{S} \models c \text{ for all } c \in S \end{aligned}$$

**P.2** This gives us  $\mathfrak{S} \models x: \varphi$  (base case)

**P.3 case analysis:** if branch consistent, then either

- ▷ no rule applicable to leaf (open branch)
- ▷ or rule applicable and one new branch satisfiable (green inductive case)

**P.4 consequence:** there must be an open branch (by termination)

□



## Case analysis on the rules

$\sqcap$  **applies**, then  $\mathfrak{S} \models x: \varphi \sqcap \psi$ , i.e.  $\llbracket x \rrbracket \in \llbracket (\varphi \sqcap \psi) \rrbracket$   
so  $\llbracket x \rrbracket \in \llbracket \varphi \rrbracket$  and  $\llbracket x \rrbracket \in \llbracket \psi \rrbracket$ , thus  $\mathfrak{S} \models x: \varphi$  and  $\mathfrak{S} \models x: \psi$ .

$\sqcup$  **applies**, then  $\mathfrak{S} \models x: \varphi \sqcup \psi$ , i.e.  $\llbracket x \rrbracket \in \llbracket (\varphi \sqcup \psi) \rrbracket$   
so  $\llbracket x \rrbracket \in \llbracket \varphi \rrbracket$  or  $\llbracket x \rrbracket \in \llbracket \psi \rrbracket$ , thus  $\mathfrak{S} \models x: \varphi$  or  $\mathfrak{S} \models x: \psi$ ,  
wlog.  $\mathfrak{S} \models x: \varphi$ .

$\forall$  **applies**, then  $\mathfrak{S} \models x: \forall R. \varphi$  and  $\mathfrak{S} \models x R y$ , i.e.  $\llbracket x \rrbracket \in \llbracket (\forall R. \varphi) \rrbracket$  and  $\langle x, y \rangle \in \llbracket R \rrbracket$ , so  
 $\llbracket y \rrbracket \in \llbracket \varphi \rrbracket$

$\exists$  **applies**, then  $\mathfrak{S} \models x: \exists R. \varphi$ , i.e.  $\llbracket x \rrbracket \in \llbracket (\exists R. \varphi) \rrbracket$ ,  
so there is a  $v \in D$  with  $\langle \llbracket x \rrbracket, v \rangle \in \llbracket R \rrbracket$  and  $v \in \llbracket \varphi \rrbracket$ .  
We define  $\llbracket y \rrbracket := v$ , then  $\mathfrak{S} \models x R y$  and  $\mathfrak{S} \models y: \varphi$



## Completeness of the Tableau Calculus

▷ **Lemma 17.0.35** *Open saturated tableau branches for  $\varphi$  induce models for  $\varphi$ .*

▷ **Proof:** construct a model for the branch and verify for  $\varphi$

**P.1 (Model Construction)** Let  $\mathcal{B}$  be an open saturated branch

▷ we define

$$\begin{aligned} D &:= \{x \mid x: \psi \in \mathcal{B} \text{ or } z R x \in \mathcal{B}\} \\ \llbracket c \rrbracket &:= \{x \mid x: c \in \mathcal{B}\} \\ \llbracket R \rrbracket &:= \{\langle x, y \rangle \mid x R y \in S_n\} \end{aligned}$$

- ▷ well-defined since never  $x: c, x: \bar{c} \in \mathcal{B}$  (otherwise  $\perp$  applies)
- ▷  $\mathfrak{S}$  satisfies all constraints  $x: c, x: \bar{c}$  and  $x R y$ , (by construction)

**P.2 (Induction)**  $\mathfrak{S} \models y: \psi$ , for all  $y: \psi \in \mathcal{B}$  (on  $k = \text{size}(\psi)$  next slide)

**P.3 (Consequence)**  $\mathfrak{S} \models x: \varphi$ . □



## Case Analysis for Induction

- case**  $y: \psi = y: \psi_1 \sqcap \psi_2$  Then  $\{y: \psi_1, y: \psi_2\} \subseteq \mathcal{B}$  (□-rule, saturation)  
 so  $\mathfrak{S} \models y: \psi_1$  and  $\mathfrak{S} \models y: \psi_2$  and  $\mathfrak{S} \models y: \psi_1 \sqcap \psi_2$  (IH, Definition)
- case**  $y: \psi = y: \psi_1 \sqcup \psi_2$  Then  $y: \psi_1 \in \mathbf{B}$  or  $y: \psi_2 \in \mathbf{B}$  (∪-rule, saturation)  
 so  $\mathfrak{S} \models y: \psi_1$  or  $\mathfrak{S} \models y: \psi_2$  and  $\mathfrak{S} \models y: \psi_1 \sqcup \psi_2$  (IH, Definition)
- case**  $y: \psi = y: \exists R.\theta$  then  $\{y R z, z: \theta\} \subseteq \mathbf{B}$  ( $z$  new variable) (∃\*-rules, saturation)  
 so  $\mathfrak{S} \models z: \theta$  and  $\mathfrak{S} \models y R z$ , thus  $\mathfrak{S} \models y: \exists R.\theta$ . (IH, Definition)
- case**  $y: \psi = y: \forall R.\theta$  Let  $\langle \llbracket y \rrbracket, v \rangle \in \llbracket R \rrbracket$  for some  $r \in \mathfrak{S}_D$   
 then  $v = z$  for some variable  $z$  with  $y R z \in \mathbf{B}$  (construction of  $\llbracket R \rrbracket$ )  
 So  $z: \theta \in \mathcal{B}$  and  $\mathfrak{S} \models z: \theta$ . (∀-rule, saturation, Def)  
 Since  $v$  was arbitrary we have  $\mathfrak{S} \models y: \forall R.\theta$ .



## Complexity

- ▷ **Idea:** We can organize the tableau procedure, so that the branches are worked off one after the other. Therefore the size of the branches is relevant of the (space)-complexity of the procedure.
- ▷ The size of the branches is *polynomial* in the size of the input formula (same reasons as for termination)
  - ▷ every rule except  $\forall$  is only applied to a constraint  $x: \psi$ .
  - ▷ The  $\forall$  is applied to constraints of the form  $x: \forall R.\psi$  at most as often as there are R-successors of  $x$ .
  - ▷ The R-successors of  $x$  are generated by constraints  $x: \exists R.\theta$ , whose number is bounded by the size of the input formula.
  - ▷ Each application to a constraint  $x: \psi$  generates constraints  $z: \psi'$  where  $\psi'$  is a proper subformula of  $\psi$ .

The total size is the size of the input formula plus number of  $\exists$ -formulae times number of  $\forall$ -formulae.

▷ **Theorem 17.0.36** *The consistency problem for  $\mathcal{ALC}$  is in PSPACE.*

▷ **Theorem 17.0.37** *The consistency problem for  $\mathcal{ALC}$  is PSPACE-Complete.*

▷ **Proof Sketch:** r □

duce a PSPACE-complete problem to  $\mathcal{ALC}$ -consistency

▷ **Theorem 17.0.38 (Time Complexity)** *The  $\mathcal{ALC}$ -consistency problem is in EXPTIME*



- ▷ **Proof Sketch:** There can be exponentially many branches (already for propositional logic)  $\square$



## The functional Algorithm for $\mathcal{ALC}$

- ▷ **Observation:** leads to treatment for  $\exists$

- ▷ the  $\exists$ -rule generates the constraints  $x R y$  and  $y: \psi$  from  $x: \exists R. \psi$
- ▷ this triggers the  $\forall$ -rule for  $x: \forall R. \theta_i$ , which generate  $y: \theta_1, \dots, y: \theta_n$
- ▷ for  $y$  we have  $y: \psi$  and  $y: \theta_1, \dots, y: \theta_n$ . (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  $\{c, \bar{c}\} \subseteq S$  then false\lec{inconsistent}
  elseif  $'(\varphi \sqcap \psi)' \in S$ 
    and  $(' \varphi ' \notin S$  or  $' \psi ' \notin S)$ 
  then consistent( $S \cup \{\varphi, \psi\}$ )
  elseif  $'(\varphi \sqcup \psi)' \in S$ 
    and  $\{\varphi, \psi\} \notin S$ 
  then
    consistent( $S \cup \{\varphi\}$ ) or consistent( $S \cup \{\psi\}$ )
  elseif forall  $'(\exists R. \psi)' \in S$ 
    consistent( $\{\psi\} \cup \{\theta \mid '(\forall R. \theta)' \in S\}$ )
  else true
```

- ▷ relatively simple to implement (good implementations optimized)
- ▷ but: this is restricted to  $\mathcal{ALC}$ . (extension to other DL difficult)



## Extending the Tableau Algorithm by Concept Axioms

- ▷ Concept axioms, e.g.  $\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})$  could not be handled in tableau calculi
- ▷ **Idea:** Whenever a new variable  $y$  is introduced (by  $\exists$ -rule) add the information that axioms hold for  $y$ .

- ▷ initialize tableau with  $\{x: \varphi\} \cup \mathcal{CA}$  ( $\mathcal{CA}$ : = set of concept axioms)
- ▷ new  $\exists$ -rule: 
$$\frac{x: \exists R. \varphi \quad \alpha \in \mathcal{CA}}{y: \alpha} \exists_{\mathcal{CA}}$$
 (apply-co-exhaustively to  $\exists$ )

**Problem:**  $\mathcal{CA} := \{\exists R. c\}$  and start tableau with  $x: d$  (non-termination)



### ▷ Non-Termination of Tableau with Concept Axioms

$x : d$	start
$x : \exists R.c$	in $\mathcal{CA}$
$x R y_1$	$\exists$
$y_1 : c$	$\exists$
$y_1 : \exists R.c$	$\exists_{\mathcal{CA}}$
$y_1 R y_2$	$\exists$
$y_2 : c$	$\exists$
$y_2 : \exists R.c$	$\exists_{\mathcal{CA}}$
...	

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  $\forall$ -rule has been exhaustively applied.



### ABoxes (Database Component of DL)

▷ Formula:  $a : \varphi$  ( $a$  is a  $\varphi$ )     $aRb$  ( $a$  stands in relation  $R$  to  $b$ )

property	example
internally inconsistent	tony: student, tony: student
inconsistent with a TBox	TBox: student $\sqcap$ prof ABox: tony: student, tony: prof
implicit info that is not explicit	Abox: tony: $\forall \text{has\_grad.genius}$ tonyhas_gradmary $\models$ mary: genius
info that can be combined with TBox info	TBox: cont_prof = prof $\sqcap$ ( $\forall \text{has\_grad.genius}$ ) ABox: tony: cont_prof, tonyhas_gradmary $\models$ mary: genius



### Tableau-based Instance Test and Realization

▷ Query: do the ABox and TBox together entail  $a : \varphi$  ( $a \in \varphi?$ )

▷ Algorithm: test  $a : \overline{\varphi}$  for consistency with ABox and TBox.<sup>10</sup> (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 mary: genius to determine $ABox, TBox \models \text{mary: genius}$			
TBox	cont_prof = prof $\sqcap$ ( $\forall \text{has\_grad.genius}$ )	tony: prof $\sqcap$ ( $\forall \text{has\_grad.genius}$ )	Norm
		tonyhas_gradmary	Norm
		tony: prof	$\sqcap$
ABox	tony: cont_prof	tony: $\forall \text{has\_grad.genius}$	$\sqcap$
	tonyhas_gradmary	mary: genius	$\forall$
		*	*

- ▷ **Note:** The instance test is the base for the **realization** (remember?)
- ▷ extend to more complex ABox queries: (give me all instances of  $\varphi$ )



©: Michael Kohlhase

211



<sup>J</sup>EdNOTE: need to unify abox and tbox judgments.



# Chapter 18

## 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?



©: Michael Kohlhase

212



### 18.1 Functional Roles and Number Restrictions

#### Functional Roles

- ▷ **Example 18.1.1**  $CSR \hat{=}$  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 18.1.2** 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 18.1.3** Teenager = human between 13 and 19

- ▷ teenager = human  $\sqcap$  (age < 20)age > 12 (not covered by  $\mathcal{ALC}$ )
- ▷ **Solution:** concrete domains (outside the scope of this course)



©: Michael Kohlhase

213



## Number Restrictions

- ▷ **Example 18.1.4** 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. $\overline{p_1}$   $\sqcap$   $p_2$ )
- ▷ **Problem:** city = town with at least 1,000,000 inhabitants (oh boy)
- ▷ **Alternative:** Operators for number restrictions.



©: Michael Kohlhase

214



## (Unqualified) Number Restrictions

- ▷  $\mathcal{ALC}$  plus operators  $\exists_{\geq}^n R$  and  $\forall_{\leq}^n R$  ( $R$  role,  $n \in \mathbb{N}$ )

- ▷ **Example 18.1.5**

$$\text{car} = \text{vehicle} \sqcap \exists_{\geq}^4 \text{has\_wheel} \quad (18.1)$$

$$\text{city} = \text{town} \sqcap \exists_{\geq}^{1,000,000} \text{has\_inhabitants} \quad (18.2)$$

$$\text{small\_family} = \text{family} \sqcap \forall_{\leq}^2 \text{has\_child} \quad (18.3)$$

▷

$$\llbracket \exists_{\geq}^n R \rrbracket = \{x \in \mathcal{D} \mid \#(\{y \mid \langle x, y \rangle \in \llbracket R \rrbracket\}) \geq n\} \quad (18.4)$$

$$\llbracket \forall_{\leq}^n R \rrbracket = \{x \in \mathcal{D} \mid \#(\{y \mid \langle x, y \rangle \in \llbracket R \rrbracket\}) \leq n\} \quad (18.5)$$

- ▷ **Intuitively:**  $\exists_{\geq}^n R$  is the set of objects that have at least  $n$   $R$ -successors.
- ▷ **Example 18.1.6**  $\exists_{\geq}^{1,000,000} \text{has\_inhabitants}$  is the set of objects that have at least 1,000,000 inhabitants.



©: Michael Kohlhase

215



## Translation into Predicate Logic

- ▷ Two extra rules for number restrictions: (very cumbersome)

$\overline{\exists_{\geq}^n R}^{fo(x)}$	$\overline{\forall_{\leq}^n R}^{fo(x)}$
$\exists y_1. R(x, y_1) \wedge \dots \wedge \exists y_n. R(x, y_n)$ $\wedge y_1 \neq y_2 \wedge \dots \wedge y_1 \neq y_n$ $\wedge y_2 \neq y_3 \wedge \dots \wedge y_2 \neq y_n \wedge$ $y_{n-1} \neq y_n$	$\neg \exists y. R(x, y) \vee$ $(\exists y_1. R(x, y_1) \wedge \dots \wedge \exists y_n. R(x, y_n)$ $\forall y. R(x, y) \Rightarrow (y = y_1 \vee \dots \vee y = y_n))$

▷ **Definable Operator:**  $\stackrel{n}{=} R := \exists_{\geq}^n R \sqcap \forall_{\leq}^n R$   
 defines the set of objects that have exactly  $n$   $R$ -successors.

▷ **Example 18.1.7**  $\text{car} = \text{vehicle} \sqcap \stackrel{4}{=} \text{has\_wheel}$  (vehicles with exactly 4 wheels)



## Functional Roles

▷ **Example 18.1.8**  $\text{CSR} = \text{car} \sqcap \stackrel{1}{=} \text{has\_sun\_roof}$  (CSR = car with sun roof)  
 $\text{has\_sun\_roof}$  is a relation, but restricted to CSR it is a total function.

▷ **Partial functions:**  $\text{Chd} = \text{computer} \sqcap \forall_{\leq}^1 \text{has\_hd}$  (computer with at most one hard drive)  
 $\text{has\_hd}$  is a partial function on the set  $\text{Chd}$

▷ **Intuition:** number restrictions can be used to encode partial and total functions, but not to specify the range type.



## Negation Rules

▷ **Observation:** to compute the negation normal form, need the rules for the new operators

$$\boxed{\overline{\exists_{\geq}^n R} \mapsto \forall_{\leq}^{n-1} R \quad \overline{\forall_{\leq}^n R} \mapsto \exists_{\geq}^{n+1} R}$$

▷ **Proof Sketch:** b □

y the semantics of the operators

▷ **Example 18.1.9**

- 1:  $\overline{\exists_{\geq}^5 \text{has\_child}} = \forall_{\leq}^4 \text{has\_child}$
- 2:  $\overline{\forall_{\leq}^5 \text{has\_child}} = \exists_{\geq}^6 \text{has\_child}$



## Tableaux Rules (without ABox Information)

$$\begin{array}{c}
 x R a_1 \\
 \vdots \\
 x R a_{n-k} \quad y_1, \dots, y_k \text{ new} \\
 x : \exists_{\geq}^n R \\
 \hline
 x R y_1 \\
 \vdots \\
 x R y_k
 \end{array}
 \qquad
 \begin{array}{c}
 x R a_1 \\
 \vdots \\
 x R a_m \quad m > n \\
 x : \forall_{\leq}^n R \quad 1 \leq i, j \leq m \\
 \hline
 [a_j/a_i] \text{ everywhere}
 \end{array}$$

▷ Basic Intuition (but when do we fail? Can we always identify)

▷  $\exists_{\geq}^n R$ : Introduce as many R-successors as necessary

▷  $\forall_{\leq}^n R$ : Identify two R-successors if there are too many (repeat as needed)



## 18.2 Unique Names

### Unique Name Assumption

▷ **Problem**: assuming UNA for ABox constants (but not always)

▷ **Definition 18.2.1 (Unique Name Assumption)** (UNA)

Different names for objects denote different objects, (cannot be equated)

▷ **Example 18.2.2**

Bob: gardener

Bob: gardener

UNAbomber: 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 ' $\overline{\text{UNA}}$ '



### Tableau Rules (with ABox Information)



$$\begin{array}{c}
 x R a_1 \\
 \vdots \\
 x R a_{n-k} \\
 x : \exists_{\geq}^n R
 \end{array}
 \quad
 \begin{array}{c}
 y_1, \dots, y_k : \text{UNA}_{\text{new}} \\
 a_1, \dots, a_{n-k} : \text{UNA}
 \end{array}$$


---


$$\begin{array}{c}
 x R y_1 \\
 \vdots \\
 x R y_k
 \end{array}$$

$$\begin{array}{c}
 x R a_1 : \\
 x R a_m : \\
 x : \forall_{\leq}^n R
 \end{array}
 \quad
 \begin{array}{c}
 m > n \\
 1 \leq i, j \leq m \\
 a_i : \text{UNA}
 \end{array}$$


---


$$[a_j/a_i] \text{ everywhere}$$

▷ **Definition 18.2.3**

$$\begin{array}{c}
 x R a_1 \\
 \vdots \\
 x R a_m \\
 x : \forall_{\leq}^n R
 \end{array}
 \quad
 \begin{array}{c}
 m > n \\
 a_1, \dots, a_m : \text{UNA}
 \end{array}$$


---


$$\perp$$

©: Michael Kohlhase
221

### Example: Solving a Crime with Number Restrictions

- ▷ **Example 18.2.4** 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, murderer:  $\overline{\text{UNA}}$

tony: $\forall_{\leq}^2$ observes
tony observes Bill
tony observes Bob
tony observes murderer
murderer: black_hair
Bill: $\overline{\text{black\_hair}}$
Bob: black_hair
tony observes Bill
Bill: black_hair
tony observes Bob
Bob: black_hair
*

©: Michael Kohlhase
222

## 18.3 Qualified Number Restrictions

### Qualified Number Restrictions

- ▷  $\mathcal{ALC}$  plus operators  $\exists_{\geq}^n R.\varphi$  and  $\forall_{\leq}^n R.\varphi$  (R role,  $n \in \mathbb{N}$ ,  $\varphi$  formula)
- ▷ **Example 18.3.1** person  $\sqcap \forall_{\leq}^2 \text{has\_child.blond}$  (persons with  $\leq 2$  blond kids)
- ▷ **Example 18.3.2** comp  $\sqcap \exists_{\geq}^5 \text{has\_client.car.comp}$  (company with at least 5 clients in the automobile industry)
- ▷ **Special case:** Unqualified Number restrictions ( $\exists_{\geq}^n R.T$ ,  $\forall_{\leq}^n R.T$ )

▷

$$[[\exists_{\geq}^n R.\varphi]] = \{x \in \mathcal{D} \mid \#(\{y \mid \langle x, y \rangle \in [R] \text{ and } y \in [[\varphi]]\}) \geq n\} \quad [[\forall_{\leq}^n R.\varphi]] = \{x \in \mathcal{D} \mid \#(\{y \mid \langle x, y \rangle \in [R] \text{ and } y \in [[\varphi]]\}) \leq n\}$$



©: Michael Kohlhase

223



## Negation and Quantifier Elimination

$$\triangleright \boxed{\exists_{\geq}^n R.\varphi = \forall_{\leq}^{n-1} R.\varphi \quad \forall_{\leq}^n R.\varphi = \exists_{\geq}^{n+1} R.\varphi}$$

$$\triangleright \textbf{Example 18.3.3} \quad \overline{\exists_{\geq}^3 \text{has\_child.teacher}} = \forall_{\leq}^2 \text{has\_child.teacher}$$

$$\triangleright \textbf{Example 18.3.4} \quad \overline{\forall_{\leq}^3 \text{has\_child.teacher}} = \exists_{\geq}^4 \text{has\_child.teacher}$$

▷ Quantifier elimination (regular quantifiers no longer necessary)

$$\triangleright \exists R.\varphi = \exists_{\geq}^1 R.\varphi$$

$$\triangleright \forall R.\varphi = \overline{\exists R.\overline{\varphi}} = \overline{\exists_{\geq}^1 R.\overline{\varphi}} = \forall_{\leq}^0 R.\overline{\varphi}$$



©: Michael Kohlhase

224



## Optimized Tableau Rules [Tob00]

▷ **Definition 18.3.5**  $\mathcal{T}_{ALC}$  rules plus:

$$\frac{\mathcal{B} \quad x: \exists_{\geq}^n r.\varphi \quad \#(\{y \mid x R y, y: \varphi \in \mathcal{B}\}) < n \quad y \text{ new}}{x R y \quad y: \varphi \quad y: \xi_1 \quad \vdots \quad y: \xi_k}$$

where  $\{\psi_1, \dots, \psi_k\} = \{\psi \mid x: \exists_{\geq}^m R.\psi \in \mathcal{B} \text{ or } x: \forall_{\leq}^m R.\psi \in \mathcal{B}\}$  and  $\xi_i = \psi$  or  $\xi_i = \overline{\psi}$ .

$$\frac{\mathcal{B} \quad x: \forall_{\leq}^n r.\varphi \quad \#(\{y \mid x R y, y: \varphi \in \mathcal{B}\}) > n}{*}$$



©: Michael Kohlhase

225



## Example Tableau

▷ **Example 18.3.6**

$$\begin{array}{c}
x: \exists_{\geq 3} R.\varphi \sqcap \forall_{\leq 1} R.\psi \sqcap \forall_{\leq 1} R.\bar{\psi} \\
x: \exists_{\geq 3} R.\varphi \\
x: \forall_{\leq 1} R.\psi \\
x: \forall_{\leq 1} R.\bar{\psi} \\
x R y_1 \\
y_1: \varphi \\
\begin{array}{c|c}
\begin{array}{c} y_1: \psi \\ x R y_2 \\ y_2: \varphi \\ y_2: \psi \\ * \end{array} & \begin{array}{c} y_1: \bar{\psi} \\ x R y_2 \\ y_2: \varphi \\ y_2: \bar{\psi} \\ * \end{array} \\
\hline
\begin{array}{c|c} \begin{array}{c} y_3: \psi \\ y_3: \varphi \\ * \end{array} & \begin{array}{c} y_3: \bar{\psi} \\ y_3: \varphi \\ * \end{array} \end{array} & \begin{array}{c|c} \begin{array}{c} y_2: \psi \\ x R y_3 \\ y_2: \varphi \\ y_3: \psi \\ * \end{array} & \begin{array}{c} y_2: \bar{\psi} \\ x R y_3 \\ y_2: \varphi \\ y_3: \bar{\psi} \\ * \end{array} \end{array}
\end{array}
\end{array}$$

▷ **Problem:** Naive Implementation: exponential path lengths

## Implementation by “Traces”

▷ Algorithm  $\text{SAT}(\varphi) = \text{sat}(x_0, \{x_0: \varphi\})$  $\text{sat}(x, S)$ :allocate counter  $\#r^S(x, \psi) := 0$  for all roles  $R$  and positive or negative subformulae  $\psi$  in  $S$ .apply rules  $\sqcap$  and  $\sqcup$  as long as possibleIf  $S$  contains an inconsistency, RETURN  $*$ .while( $\mapsto_{\geq}$  is applicable to  $x$ ) do: $S_{\text{neu}} := \{\mathcal{T}_{\text{ALC}} Rxy, y: \varphi, y: \xi_1, \dots, y: \xi_k\}$ 

where

 $y$  is a new variable, $x: \exists_{\geq n} R.\varphi$  triggers rule  $\mapsto_{\geq}$ , $\{\psi_1, \dots, \psi_k\} = \{\psi \mid x: \exists_{\geq m} R.\psi \in \mathcal{B} \text{ or } x: \forall_{\leq m} R.\psi \in \mathcal{B}\}$  and $\xi_i = \psi$  oder  $\xi_i = \neg\psi$ .For each  $y: \psi \in S_{\text{neu}}$ :  $\#r^S(x, \psi) + 1$  If  $x: \forall_{\leq m} R.\psi \in \mathcal{B}$  and  $\#r^S(x, \psi) > m$   
RETURN  $*$ If  $\text{sat}(y, S_{\text{neu}}) = *$  RETURN  $*$  od

RETURN “consistent”.



## Analysis

▷ **Idea:** Each  $R$ -successor of  $x$  triggers a recursive call of  $\text{sat}$ .

- ▷ 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 18.3.7** *Space consumption is polynomial.*
- ▷ **Lemma 18.3.8** *This algorithm is complete.*
- ▷ **Proof Sketch:** T □  
 he global counters  $\#r^S(x, \psi)$  count the R-successors and trigger rule  $\mapsto_{\leq}$ .
- ▷ **Theorem 18.3.9** *The algorithm is correct, complete and terminating, and PSPACE(no worse than ALC)*



## 18.4 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.  $\mathbb{N}, \mathbb{Z}, \text{trees}$ )
- ▷ external data structures (for programming)
- ▷ epistemic operators (belief, . . .)
- ▷ . . .



### Role Hierarchies

- ▷ **Idea:** specification of subset relations among relations.
- ▷ **Example 18.4.1** role hierarchy as a directed graph  $\mathcal{R}$

$\text{has\_daughter} \sqsubseteq \text{has\_child}$   
 $\text{has\_son} \sqsubseteq \text{has\_child}$   
 $\text{talks\_to} \sqsubseteq \text{communicates\_with}$   
 $\text{calls} \sqsubseteq \text{communicates\_with}$   
 $\text{buys} \sqsubseteq \text{obtains}$   
 $\text{steals} \sqsubseteq \text{obtains}$



## $\mathcal{ALC}$ with Role hierarchies (without role operators)

▷ **Definition 18.4.2**  $\mathcal{T}_{\mathcal{ALC}}$  + complex roles instead of role names

$$\frac{x: \exists R.\varphi \quad \frac{x S y \quad x: \forall R.\varphi \quad S \sqsubseteq R \in \mathcal{R}}{y: \varphi}}{y: \varphi} \exists \quad \forall \sqsubseteq$$

The  $\exists$  rule is the same as before



©: Michael Kohlhase

231

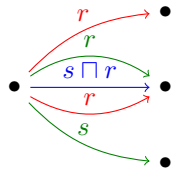


## Operators on Roles: Role Conjunction

▷ **Example 18.4.3**  $\text{person} \sqcap (\exists(\text{has\_teacher} \sqcap \text{has\_friend}), \text{swiss})$  (persons that have a Swiss teacher that is also their friend)

▷ **Example 18.4.4**  $\text{com} \sqcap (\exists(\text{has\_employee} \sqcap \text{has\_attorney}), \text{lawyer})$  (companies that have an employed attorney that is a lawyer)

▷  $\llbracket R \sqcap S \rrbracket = \llbracket R \rrbracket \cap \llbracket S \rrbracket = \{ \langle x, y \rangle \in \mathcal{D} \mid \langle x, y \rangle \in \llbracket R \rrbracket \text{ and } \langle x, y \rangle \in \llbracket S \rrbracket \}$



**Inference Rules**  
 $(\forall R \sqcap S.\varphi) \sqsubseteq (\forall R.\varphi) \sqcap (\forall S.\varphi)$   
 $(\exists R \sqcap S.\varphi) \sqsubseteq (\exists R.\varphi) \sqcap (\exists S.\varphi)$   
 $\exists_{\geq}^n R \sqcap S.\varphi \sqsubseteq \exists_{\geq}^n R.\varphi \sqcap \exists_{\geq}^n S.\varphi$   
 $\forall_{\leq}^{n+m} R \sqcap S.\varphi \sqsubseteq \forall_{\leq}^n R.\varphi \sqcap \forall_{\leq}^m S.\varphi$



©: Michael Kohlhase

232

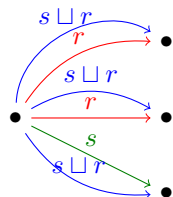


## Role Disjunction $\sqcup$

▷ **Example 18.4.5**  $\text{person} \sqcap (\forall \text{has\_child} \sqcup \text{has\_friend}, \text{teacher})$  (persons whose children and friends are all teachers)

▷ **Example 18.4.6**  $\text{com} \sqcap (\exists \text{has\_employee} \sqcup \text{has\_consultant}, \text{member\_of\_congress})$  (companies with an employee or consultant that is a member of congress)

▷  $\llbracket R \sqcup S \rrbracket = \llbracket R \rrbracket \cup \llbracket S \rrbracket = \{ \langle x, y \rangle \in \mathcal{D} \mid \langle x, y \rangle \in \llbracket R \rrbracket \text{ or } \langle x, y \rangle \in \llbracket S \rrbracket \}$



**Inference Rules**  
 $\forall R \sqcup S.\varphi = (\forall R.\varphi) \sqcup (\forall S.\varphi)$   
 $\exists R \sqcup S.\varphi = (\exists R.\varphi) \sqcup (\exists S.\varphi)$   
 $\exists_{\geq}^n R \sqcup S.\varphi = ??$   
 $\forall_{\leq}^n R \sqcup S.\varphi \sqsubseteq \forall_{\leq}^n R.\varphi \sqcap \forall_{\leq}^n S.\varphi$   
 $\forall_{\leq}^{n+m} R \sqcup S.\varphi \sqsubseteq \forall_{\leq}^n R.\varphi \sqcap \forall_{\leq}^m S.\varphi$   
 $\exists_{\geq}^{max(n,m)} R \sqcup S.\varphi \sqsubseteq (\exists_{\geq}^n R.\varphi) \sqcup (\exists_{\geq}^m S.\varphi)$



©: Michael Kohlhase

233



## Role Complement $\bar{\phantom{x}}$

▷ **Example 18.4.7**  $\text{univ} \sqcap (\forall \text{has\_employee} \sqcap \overline{\text{has\_prof}}, \text{unionized})$  (universities whose employees that are not professors are unionized)

▷ **Example 18.4.8**  $\text{house} \sqcap (\exists \text{resident} \sqcap \overline{\text{owner.swiss}})$  (houses whose residents that are not owners are Swiss)

▷  $\llbracket \overline{R} \rrbracket = \mathcal{D}^2 \setminus \llbracket R \rrbracket = \{ \langle x, y \rangle \in \mathcal{D}^2 \mid \langle x, y \rangle \notin \llbracket R \rrbracket \}$

▷ **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 18.4.9**  $\forall \overline{R} \sqcap \overline{S}. \varphi = \forall \overline{R} \sqcup \overline{S}. \varphi$

▷ **more rules:** if  $R \sqsubseteq S$  is a tautology, then  $(\forall S. \varphi) \sqsubseteq (\forall R. \varphi)$  and  $(\exists R. \varphi) \sqsubseteq (\exists S. \varphi)$



## Special Relations 0 and 1

$R \sqcap \overline{R} = 0$	empty relation
$R \sqcup \overline{R} = 1$	universal relation

▷ **Question:** what does  $\forall 1. \varphi$  mean?

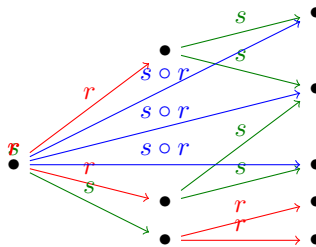


## Role composition $\circ$

▷ **Example 18.4.10**  $\text{person} \sqcap (\exists \text{has\_child} \circ \text{has\_child.prof})$  (persons that have grandchild that is a professor)

▷ **Example 18.4.11**  $\text{univ} \sqcap (\forall \text{has\_student} \circ \text{has\_Partner} \circ \text{lives\_in}). \text{Texas}$  (universities whose students all have partners that live in Texas)

▷  $\llbracket R \circ S \rrbracket = \llbracket R \rrbracket \llbracket S \rrbracket = \{ \langle x, z \rangle \in \mathcal{D}^2 \mid \exists y. \langle x, y \rangle \in \llbracket S \rrbracket \text{ and } \langle y, z \rangle \in \llbracket R \rrbracket \}$



## Converse Roles $(\cdot^{-1})$

▷ **Example 18.4.12** (set of objects whose parents are teachers)

$$\begin{aligned}
\llbracket \forall \text{has\_child}^{-1}.\text{teacher} \rrbracket &= \{x \mid \forall y, \langle x, y \rangle \in \llbracket \text{has\_child}^{-1} \rrbracket \Rightarrow y \in \llbracket \text{teacher} \rrbracket\} \\
&= \{x \mid \forall y, \langle y, x \rangle \in \llbracket \text{has\_child} \rrbracket \Rightarrow y \in \llbracket \text{teacher} \rrbracket\} \\
&= \{x \mid \forall y, \langle x, y \rangle \in \llbracket \text{has\_parents} \rrbracket \Rightarrow y \in \llbracket \text{teacher} \rrbracket\}
\end{aligned}$$

▷ **Definition 18.4.13**  $\llbracket R^{-1} \rrbracket = \llbracket R \rrbracket^{-1} = \{\langle y, x \rangle \in \mathcal{D}^2 \mid \langle x, y \rangle \in \llbracket R \rrbracket\}$

▷ **Example 18.4.14**

$$\begin{aligned}
\text{has\_child}^{-1} &= \text{has\_parents} \\
\text{is\_part\_of}^{-1} &= \text{contains\_as\_part} \\
\text{owns}^{-1} &= \text{belongs\_to} \\
&\dots
\end{aligned}$$



## Translation of Role Terms

▷ **Definition 18.4.15** Translation Rules:

$\text{tr}(R) := R(x, y)$	
$\text{tr}(R \sqcap S) := \text{tr}(R) \wedge \text{tr}(S)$	$\text{tr}(R \sqcup S) := \text{tr}(R) \vee \text{tr}(S)$
$\text{tr}(R \sqsubseteq S) := \text{tr}(R) \Rightarrow \text{tr}(S)$	$\text{tr}(R \circ S) := (\exists z. \text{tr}(R), \text{tr}(S))$
$\text{tr}(R^{-1}) := \text{tr}(R)$	$\text{tr}(\bar{R}) := \neg \text{tr}(R)$
$\overline{\forall R. \varphi}^{fo(x)} := (\forall y. \text{tr}(R)) \Rightarrow \bar{\varphi}^{fo(y)}$	$\overline{\exists R. \varphi}^{fo(x)} := (\exists y. \text{tr}(R), \bar{\varphi}^{fo(y)})$

▷ **Example 18.4.16**

$$\begin{aligned}
&\overline{\overline{\forall R \circ S \sqcap T^{-1}. c}}^{fo(x)} \\
&= \forall y. \text{tr}(\overline{R \circ S \sqcap T^{-1}}) \Rightarrow \bar{c}^{fo(y)} \\
&= \forall y. \neg \text{tr}(R \circ S \sqcap T^{-1}) \Rightarrow c(y) \\
&= \forall y. \neg (\exists z. R(x \wedge z) \wedge \text{tr}(S \sqcap T^{-1})) \Rightarrow c(y) \\
&= \forall y. \neg (\exists z. R(x \wedge z) \wedge \text{tr}(S \sqcap T)) \Rightarrow c(y) \\
&= \forall y. \neg (\exists z. R(x \wedge z) \wedge S(y \wedge z) \wedge T(y \wedge z)) \Rightarrow c(y)
\end{aligned}$$



## 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 ( $\llbracket \forall R.\varphi \rrbracket =$  “in all states after executing R,

$R \sqcap S$	parallel execution of R and S
$R \sqcup S$	execution of R or S (nondeterministically)
$R \circ S$	execution of S after R
$\bar{R}$	execution of a program that is not R
$R^{-1}$	execution of an undo operation
$?\psi$	test whether $\psi$ holds (not in $\mathcal{ALC}$ )



## Tableaux Calculus: $\mathcal{ALC} +$ Role Terms

- ▷ **Definition 18.4.17** complex roles instead of role names

$$\frac{x: \exists R.\varphi \quad x R z}{x R y} \exists \quad \frac{\mathcal{B} \quad x: \forall R.\varphi \quad \mathcal{B} \models x R y}{y: \varphi} \forall_R$$

- ▷ **Problem:** What is  $\mathcal{B} \models x R y$  ( $\mathcal{B}$  is the current branch)
- ▷ **Simple case:** no role composition  $\circ$  and no converse roles  $\cdot^{-1}$ .  
 ▷ then  $\mathcal{B} \models x R y$ , iff  $\{S \mid x S y \in \mathcal{B}\} \cup \{\bar{R}\}$  inconsistent in  $PL^0$  (decidable)
- ▷ **General case:**  $\mathcal{B} \models x R y$ , iff  $\{\text{tr}(S) \mid u S v \in \mathcal{B}\} \cup \{\text{tr}(\bar{R})\}$  inconsistent in  $PL^1$  (undecidable in general)



## Special Cases for $\mathcal{B} \models x R y$

- ▷ no role composition  $\circ$  (decidable)  
 ▷ then  $\mathcal{B} \models x R y$ , iff  $\{\text{tr}(S) \mid x S y \in \mathcal{B}\} \cup \{\text{tr}(\bar{R})\}$  inconsistent in  $PL^1$  (as set of ground formulae).
- ▷ role complement only for role names (decidable)  
 ▷ then  $\{\text{tr}(S) \mid u S v \in \mathcal{B}\}$  is a set of ground formulae and  $\text{tr}(\bar{R})$  only contains constants and variables in the clause normal form.
- ▷ The general case is undecidable, therefore the naive tableau approach is unsuitable



## 18.5 Role Axioms



### General Role Axioms

has_daughter $\sqsubseteq$ has_child	daughters are children
has_son $\sqsubseteq$ has_child	sons are children
has_daughter $\sqcap$ 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  $\rho$ :  $\text{trr}(\rho) = \forall x, y. \text{tr}(\rho)$

$$\begin{aligned}
 & \text{trr}(\text{has\_child} \sqsubseteq (\text{has\_son} \sqcup \text{has\_daughter})) \\
 = & \forall x, y. \text{tr}(\text{has\_child} \sqsubseteq \text{has\_son} \sqcup \text{has\_daughter}) \\
 = & \forall x, y. \text{has\_child}(x \Rightarrow y) \Rightarrow \text{has\_son}(x \vee y) \vee \text{has\_daughter}(x \vee y)
 \end{aligned}$$



### $\mathcal{ALC}$ + Role Terms + Role Axioms $\rho$

- ▷ **Idea**: Tableau like for  $\mathcal{ALC}$  + role terms ( $\mathcal{B}, \rho \models x R y$  instead of  $\mathcal{B} \models x R y$ )
- ▷ **Simple case**: no role composition  $\circ$  and no converse roles  $\cdot^{-1}$ . (decidable)
  - ▷ then  $\mathcal{B}, \rho \models x R y$  iff  $\{S \mid x S y \in \mathcal{B}\} \cup \rho \cup \{\bar{R}\}$  inconsistent in  $\text{PL}^0$
- ▷ **General case**:  $\mathcal{B}, \rho \models x R y$ , iff  $\{\text{tr}(S) \mid u S v \in (\mathcal{B} \cup \text{trr}(\rho) \cup \{\text{tr}(\bar{R})\})\}$  inconsistent in  $\text{PL}^1$  (undecidable in general)
- ▷ no role composition  $\circ$  (decidable)
  - ▷ then  $\mathcal{B}, \rho \models x R y$ , iff  $\{\text{tr}(S) \mid x S y \in (S \cup \text{trr}(\rho) \cup \{\text{tr}(\bar{R})\})\}$  inconsistent in  $\text{PL}^1$  (as set of formulae without functions).
- ▷ role complement only for role names (decidable)
  - ▷ then  $\{\text{tr}(S) \mid u S v \in \mathcal{B}\}$  is a set of ground formulae and both  $\text{tr}(\rho)$  and  $\text{tr}(\bar{R})$  only contain constants and variables in CNF



## 18.6 Features

### $\mathcal{ALCF}$ : Features

- ▷ **Features** are partial functions.
- ▷ **Idea**:  $\mathcal{ALCF}$  is  $\mathcal{ALC}$  + features + special constraints on feature paths
- ▷ **Definition 18.6.1** Let  $\mathcal{F} := \{f, g, f_1, \dots\}$  be a set of **features**, then we define the  $\mathcal{ALCF}$

formulae by

$$F_{\mathcal{ALF}} ::= F_{\mathcal{AL}} \mid R.F_{\mathcal{ALF}} \mid (\pi)^\uparrow \mid \pi = \pi \mid \pi \neq \pi \text{ where } \pi ::= f \mid f \circ \pi$$

▷ **Definition 18.6.2** The semantics of the  $\mathcal{ALF}$  part is as always.

1. The meaning of a feature  $f$  is a partial function  $\llbracket f \rrbracket : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ .
2.  $\llbracket f \circ \pi \rrbracket(x) := \llbracket \pi \rrbracket(\llbracket f \rrbracket(x))$
3.  $\llbracket (\pi)^\uparrow \rrbracket := \mathcal{D} \setminus \text{dom}(\llbracket \pi \rrbracket)$
4.  $\llbracket f.\varphi \rrbracket := \{x \in \text{dom}(\llbracket \pi \rrbracket) \mid \llbracket f \rrbracket(x) \in \llbracket \varphi \rrbracket\}$
5.  $\llbracket \varphi = \omega \rrbracket := \{x \in (\text{dom}(\llbracket \pi \rrbracket) \cap \text{dom}(\llbracket \omega \rrbracket)) \mid \llbracket \pi \rrbracket(x) = \llbracket \omega \rrbracket(x)\}$
6.  $\llbracket \varphi \neq \omega \rrbracket := \{x \in (\text{dom}(\llbracket \pi \rrbracket) \cap \text{dom}(\llbracket \omega \rrbracket)) \mid \llbracket \pi \rrbracket(x) \neq \llbracket \omega \rrbracket(x)\}$



## Examples

- ▷ **Example 18.6.3** *persons, whose father is a teacher*:  $\text{person} \sqcap \text{had\_father.teacher}$
- ▷ **Example 18.6.4** *persons that have no father*:  $\text{person} \sqcap (\text{had\_father})^\uparrow$
- ▷ **Example 18.6.5** *companies, whose bosses have no company car*:  $\text{company} \sqcap (\text{has\_boss} \circ \text{has\_comp\_car})^\uparrow$
- ▷ **Example 18.6.6** *cars whose exterior color is the same as the interior color*:  $\text{car} \sqcap \text{color\_exterior} = \text{color\_interior}$
- ▷ **Example 18.6.7** *cars whose exterior color is different from the interior color*:  $\text{car} \sqcap \text{color\_exterior} \neq \text{color\_interior}$
- ▷ **Example 18.6.8** *companies whose Bosses and Vice Presidents have the same company car*:  $\text{company} \sqcap \text{has\_boss} \circ \text{has\_comp\_car} = \text{has\_VP} \circ \text{has\_comp\_car}$



## Normalization

▷ Normalization rules

$$\begin{aligned} \overline{f.\varphi} &\rightarrow (f)^\uparrow \sqcup f.\varphi \\ \overline{\pi = \omega} &\rightarrow ((\pi)^\uparrow)(\omega)^\uparrow \sqcup \pi \neq \omega \\ \overline{\pi \neq \omega} &\rightarrow ((\pi)^\uparrow)(\omega)^\uparrow \sqcup \pi = \omega \\ \overline{(f \circ \pi)^\uparrow} &\rightarrow (f)^\uparrow \sqcup f \circ (\pi)^\uparrow \end{aligned}$$

▷ **Example 18.6.9** (for the last transformation)

$$(\text{has\_boss} \circ \text{has\_comp\_car} \circ \text{has\_sun\_roof})^\uparrow = \dots$$

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 18.6.10** The calculus is an extension of  $\mathcal{T}_{\text{ALC}}$ .

$$\begin{array}{c}
 \frac{x: f.\varphi}{x f y} \quad \frac{x: \pi = \omega}{x \pi y} \quad \frac{x: \pi \neq \omega}{x \pi y} \quad \frac{x f \circ \pi y}{x f y} \quad \frac{\mathcal{B} \quad x f y \neq y, z}{x f z} \\
 y: \varphi \quad x \omega y \quad x \omega z \quad z \pi y \quad \frac{[y/z](\mathcal{B})}{x f z} \\
 y \neq z
 \end{array}$$

$$\begin{array}{c}
 \frac{x: \perp}{*} \quad \frac{x: c}{*} \quad \frac{x: \bar{c}}{*} \quad \frac{x f y}{*} \quad \frac{x: (f)\uparrow}{*} \quad \frac{x \neq x}{*}
 \end{array}$$

▷ **Theorem 18.6.11** *The calculus is correct, complete and terminating.*

▷ **Theorem 18.6.12** *It can be implemented in PSPACE*



## Example

▷ **Example 18.6.13**  $(\text{has\_boss} \circ \text{has\_comp\_car})\uparrow \sqcap \text{has\_boss.has\_comp\_car.has\_sun\_roof}.\top$  is inconsistent.

▷ **Normalize:**  $((\text{has\_boss})\uparrow \sqcap \text{has\_boss}.\text{has\_comp\_car})\uparrow \sqcap \text{has\_boss.has\_comp\_car.has\_sun\_roof}.\top$

▷ **Tableau**

$$\begin{array}{c}
 x: (\text{has\_boss})\uparrow \sqcap \text{has\_boss}.\text{has\_comp\_car}\uparrow \\
 x: \text{has\_boss.has\_comp\_car.has\_sun\_roof}.\top \\
 x \text{ has\_boss } y \\
 y: \text{has\_comp\_car.has\_sun\_roof}.\top \\
 y \text{ has\_comp\_car } z \\
 z: \text{has\_sun\_roof}.\top \\
 x: (\text{has\_boss})\uparrow \quad x: \text{has\_boss}.\text{has\_comp\_car}\uparrow \\
 * \quad \quad \quad x \text{ has\_boss } v \\
 \quad \quad \quad v: (\text{has\_comp\_car})\uparrow \\
 \quad \quad \quad y: (\text{has\_comp\_car})\uparrow \quad (y = v) \\
 \quad \quad \quad *
 \end{array}$$



## 18.7 Concrete Domains

### $\mathcal{ALC}$ with “concrete Domains” (Examples)

Formula	Concrete Domain
person $\sqcap$ age $< 20$	real numbers
persons younger than 20	
company $\sqcap$ has_CEO $\circ$ has_comp_car $\circ$ price $> \$100000$	natural numbers
companies with CEOs with expensive car	
car $\sqcap$ height $> \text{width}$	natural numbers
cars that are higher than wide	
person $\sqcap$ first_name $< \text{last\_name}$	strings
persons whose first name is lexicographically smaller than their last name	
person $\sqcap$ has_father $\circ$ studiesbefore(has_mother $\circ$ studies	temporal interval logic
persons whose fathers have studied before their mothers	



## Concrete Domain

▷ **Definition 18.7.1** 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 18.7.2** ▷  $\mathcal{C} = \mathbb{N}$  and  $\mathcal{P} = \{=, <, \leq, >, \geq\}$  (natural numbers)

▷  $\mathcal{C} = \mathbb{R}$  and  $\mathcal{P} = \{=, <, \leq, >, \geq\}$  (real numbers)

▷  $\mathcal{C} = \text{temporal intervals}$ ,  $\mathcal{P} = \{\text{before, after, overlaps, } \dots\}$  (Allen's interval logic)

▷  $\mathcal{C} = \text{facts in a relational data base}$ ,  $\mathcal{P} = \text{SQL relations}$



## Admissible Concrete Domains

▷ **Idea:** concrete domains are admissible, iff  $\mathcal{P}$  is decidable.

▷ **Definition 18.7.3** Let  $\{P_1, \dots, P_n\} \subseteq \mathcal{P}$ , then conjunctions  $P_1(x_1, \dots) \wedge \dots \wedge P_n(x_n, \dots)$  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 18.7.4**  $\mathcal{C} = \text{real numbers}$

$P_1(x, y) = \exists z. (x + z^2 = y)$	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 18.7.5** 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  $\mathcal{C}$ .



## $ALC(\mathcal{C})$

▷  $F_{ALC(\mathcal{C})} := F_{ALC} \mid P(\pi, \dots, \pi)$

▷ **Example 18.7.6** a female human under 21 can become a woman by having a child

$$\begin{aligned}\text{mother} &= \text{human} \sqcap \text{female} \sqcap (\exists \text{has\_child}.\text{human}) \\ \text{woman} &= \text{human} \sqcap \text{female} \sqcap (\text{mother} \sqcup \text{age} \geq 21)\end{aligned}$$

here  $\text{age} \geq 21 \in \mathbf{F}_{\mathcal{ALC}}$ , 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.

▷  $P(\pi_1, \dots, \pi_n) = x \in \mathcal{D}$  there are  $y_1 = \llbracket \pi_1 \rrbracket(x), \dots, y_n = \llbracket \pi_n \rrbracket(x) \in \mathcal{C}$ , with  $\langle y_1, \dots, y_n \rangle \in [P]$

Warning:  $\llbracket \bar{\varphi} \rrbracket = \mathcal{D} \setminus \llbracket \varphi \rrbracket$ , but not  $\llbracket \bar{\varphi} \rrbracket = (\mathcal{D} \cup \mathcal{C}) \setminus \llbracket \varphi \rrbracket$



!

## ▷ Negation Rules and Tableau Calculus

▷ Let  $\top_{\mathcal{C}}$  be the name for the concrete domain (as a set) and  $\bar{P}$  the negated predicate for  $P$  ( $\mathcal{C}$  is admissible)

▷ New negation rule:  $\overline{P(\pi_1, \dots, \pi_n)} \rightarrow \bar{P}(\pi_1, \dots, \pi_n) \sqcup (\forall \pi_1. \top_{\mathcal{C}}) \sqcup \dots \sqcup (\forall \pi_n. \top_{\mathcal{C}})$

▷ New tableau rule

$$\frac{\begin{array}{c} P_1(x_{11}, \dots, x_{1n_1}) \\ \vdots \\ P_k(x_{k1}, \dots, x_{kn_k}) \end{array} \quad \bigwedge_{1 \leq i \leq k} P_i(x_{i1}, \dots, x_{in_i}) \text{ inconsistent}}{*} \perp^p$$



Example:  $\text{car} \sqcap \text{height} = 2 \sqcap \text{width} = 1 \sqsubseteq \text{car} \sqcap \text{height} > \text{width}$

$$\begin{aligned}x: \text{car} \sqcap \text{height} = 2 \sqcap \text{width} = 1 \\ x: \bar{\text{car}} \sqcap \text{width} \leq \text{height} \\ x: \text{car} \\ x: \text{height} = 2 \\ x: \text{width} = 1\end{aligned}$$

$$\begin{array}{c|l} x: \bar{\text{car}} & x: \text{width} \leq \text{height} \\ * & x \text{height } y_1 \\ & y_1 = 2 \\ & x: \text{width} = y_2 \\ & y_2 = 1 \\ & x: \text{width } y_3 \\ & x: \text{height} = y_4 \\ & y_3 \leq y_4 \\ & y_1 \leq y_2 \\ & * \end{array}$$



## 18.8 Nominals

### Nominals

- ▷ **Definition 18.8.1 (Idea)** *nominal* are names for domain elements that can be used in the T-Box.
- ▷ **Example 18.8.2** *Students that study on Bremen or Hamburg*:  $\text{student} \sqcap (\exists \text{studies\_in}.\{\text{Bremen}, \text{Hamburg}\})$
- ▷ **Example 18.8.3** *Students that have a friend with name Eva*:  $\text{student} \sqcap (\exists \text{has\_friend} \circ \text{has\_name}.\{\text{Eva}\})$
- ▷ **Example 18.8.4** *persons that have phoned Bill, Bob, or the murderer*:  $\text{person} \sqcap (\exists \text{has\_phoned}.\{\text{Bill}, \text{Bob}, \text{murderer}\})$
- ▷ **Example 18.8.5** *friends of Eva*:  $\text{person} \sqcap \text{has\_friend} : \text{Eva}$
- ▷ **Example 18.8.6** *companies whose employees all bank at Sparda Bank*:  $\text{company} \sqcap (\forall \text{has\_empl}.\text{has\_bank} : \text{Sparda})$
- ▷ **Example 18.8.7** *employees of Jacobs that bank at Sparda*:  $\text{employed\_at} : \text{Jacobs} \sqcap \text{has\_bank} : \text{Sparda}$



### Semantics

- ▷ **Definition 18.8.8**  $\llbracket \{a_1, \dots, a_n\} \rrbracket$  is the set of objects with names  $a_1, \dots, a_n$ .
- ▷ **Definition 18.8.9**  $\llbracket R : a \rrbracket$  is the set of objects that have  $\llbracket a \rrbracket$  as R-successor

$$\begin{aligned} \llbracket \{a_1, \dots, a_n\} \rrbracket &= \{\llbracket a_1 \rrbracket, \dots, \llbracket a_n \rrbracket\} \\ \llbracket R : a \rrbracket &= \{x \in \mathcal{D} \mid \langle x, \llbracket a \rrbracket \rangle \in R\} \end{aligned}$$

- ▷ **Definition 18.8.10 (Negation Rules)**

$$\begin{aligned} \overline{\{a_1, \dots, a_n\}} &= \text{invariant} \\ \overline{R : a} &= \forall R.\overline{\{a\}} \end{aligned}$$

- ▷ **Example 18.8.11**  $\overline{\text{had\_friend} : \text{Eva}}$  (the complement of the set of friends of Eva)  
 $= \forall \text{had\_friend}.\overline{\{\text{Eva}\}}$  (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 ( $\exists_{\geq}^n R, \forall_{\geq}^n R$ ), some role operators ( $\sqcap, \circ, \cdot^{-1}$ ),  $\{a_1, \dots, a_n\}$ ,  $R : a$
- ▷ **Example 18.8.12** *persons that have at most two friends among their neighbors and whose neighbors are Bill, Bob, or the gardener*  $\text{person} \sqcap \forall_{\leq}^2 (\text{has\_friend} \sqcap \text{had\_neighbor}) \sqcap (\forall \text{had\_neighbor}.\{\text{Bill}, \text{Bob}, \text{Gardener}\})$

- ▷ **Example 18.8.13** *companies with at least 100 employees that have a car and live in Bremen* company  $\sqcap \exists_{\geq 100} \text{has\_empl} \circ \text{has\_comp\_car} \sqcap \text{has\_empl} \circ \text{lives\_in} : \text{Bremen}$



## Tableaux Calculus (only T-Box)

- ▷ **Definition 18.8.14** The calculus consists of the  $\mathcal{T}_{\text{ALC}}$  rules together with:

$$\begin{array}{c}
 \frac{a : \{\dots, a, \dots\}}{*} \qquad \frac{\mathcal{B} \quad x : \{a_1, \dots, a_n\}}{[x/a_1](\mathcal{B}) \mid \dots [x/a_n](\mathcal{B})} \qquad \frac{x : R : a}{x R a} \\
 \\
 \frac{x R^{-1} y}{y R x} \qquad \frac{x R \sqcap S y}{x R y \quad x S y} \qquad \frac{x R \circ S y}{x R z \quad z S y}
 \end{array}$$

- ▷ **Theorem 18.8.15** *The calculus is correct, complete, and terminating*

- ▷ **Proof Sketch:**  $\forall$

□

ery technical but not terribly difficult using the techniques developed so far.







# 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.
- [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.
- [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].
- [HASB13] Ivan Herman, Ben Adida, Manu Sporny, and Mark Birbeck. RDFa 1.1 primer – second edition. W3C Working Group Note, World Wide Web Consortium (W3C), 2013.
- [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 L<sup>A</sup>T<sub>E</sub>X as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.
- [Koh12] Michael Kohlhase. sTeX: Semantic markup in T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X. Technical report, Comprehensive T<sub>E</sub>X Archive Network (CTAN), 2012.
- [OWL09] OWL Working Group. OWL 2 web ontology language: Document overview. W3C recommendation, World Wide Web Consortium (W3C), October 2009.
- [PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen*. Gabler Verlag, 4 (2003) edition, 1997.
- [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, 3<sup>rd</sup> printing, 1997 edition, 1967.
- [WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, UK, 2 edition, 1910.

# Index

- abstract
  - consistency
    - class, 28
- admissible, 20
- admits
  - weakening, 19
- $\alpha$ -equal, 16
- alphabetical
  - variants, 16
- assignment
  - variable, 12
- assignment
  - variable, 51
- assumption, 19
- Axiom
  - Extensionality, 54
- axiom, 19
- base
  - type, 50
- $\beta$ -equality
  - Axiom of, 54
- Axiom of
  - $\beta$ -equality, 54
- $\beta$  normal form, 65
- $\beta$  normal form of **A**, 65
- binary
  - conditional, 76
- binder, 57
- binding
  - imitation, 69
  - projection, 69
- bound, 55
- calculus, 19
- calculus
  - sequent, 26
- carrier, 51
- $\mathcal{C}$ -consistent, 28
- choice
  - operator, 76
- closed, 11
- closed under
  - subsets, 28
- collection
  - typed, 63
- compact, 29
- complete, 20
- complete, 41
- complete
  - set of unifiers, 67
- set of unifiers
  - complete, 67
- comprehension-closed, 63
- conclusion, 19
- conditional
  - binary, 76
  - unary, 76
- confluent, 62
- confluent
  - weakly, 62
- congruence, 65
- connective, 10
- connective, 74
- connectives, 50
- consistency
  - abstract (class), 28
- constant
  - Skolem, 10
  - function, 10
  - predicate, 10
- constants
  - interpretation of, 51
- contant
  - Skolem, 55
- contradiction, 27
- correct, 41
- $\mathcal{C}$ -refutable, 27
- Currying, 50
- DAG
  - solved
    - form, 44
- $\mathcal{C}$ -derivation, 19
- derivation
  - relation, 19
- derives, 37
- description
  - operator, 76
- discharge, 13

- discharged, 23
- domain
  - type, 50
- entailment
  - relation, 18
- entails, 18
- Equality
  - Leibniz, 49
- equational
  - system, 40
- $\eta$ -Expansion, 57
- $\eta$ -long
  - form, 57
- form
  - $\eta$ -long, 57
- extension, 13
- Extensionality
  - Axiom, 54
- Extensionality, 49
- extensionality, 53
- falsified by  $\mathcal{M}$ , 18
- falsifiable, 18
- first-order
  - signature, 10
- valuation, 31
- first-order
  - natural deduction
    - calculus, 24
- natural deduction
  - first-order (calculus), 24
- form
  - normal, 57
- form
  - solved, 68
- form
  - pre-solved, 73
- form
  - solved, 40
- formal
  - system, 20
- formal
  - system, 19
- formula, 18
- formula
  - well-typed, 51
- formula
  - well-typed, 55
- formula
  - quantified, 74
- $\mathcal{U}$ -reducible, 43
- frame, 63
- free, 55
- free
  - variable, 11
- function
  - typed, 63
  - value, 63
- function
  - value, 12
- function
  - constant, 10
- function
  - type, 50
- function
  - universe, 51
  - value, 51
- general
  - more, 67
- general
  - more, 39
- ground, 11
- grounding
  - substitution, 66
- Head
  - Reduction, 57
- head
  - symbol, 57
  - syntactic, 57
- higher-order
  - simplification
    - transformations, 68
- hypotheses, 19
- hypothetical
  - reasoning, 23
- imitation
  - binding, 69
- individual, 10
- individual
  - variable, 10
- individuals, 11
- inference
  - rule, 19
- interpretation, 11
- interpretation of
  - constants, 51
- introduced, 13
- judgment, 25
- Leibniz
  - Equality, 49
- logical
  - relation, 58
- logical

- system, 18
- $\beta\eta$ -normal
  - Long (form), 57
- Long
  - $\beta\eta$ -normal
    - form, 57
- matrix, 57
- measure
  - unification, 72
- most general
  - unifier, 67
- unifier
  - most general, 67
- most general
  - unifier, 39
- unifier
  - most general, 39
- Model, 12
- model, 18
- more
  - general, 67
- more
  - general, 39
- multiset
  - ordering, 42
- $\nabla$ -Hintikka Set, 30
- negative, 37
- normal
  - form, 57
- operator
  - choice, 76
  - description, 76
- ordering
  - multiset, 42
- pre-solved, 73
- pre-solved
  - form, 73
- predicate
  - constant, 10
- projection, 57
- projection
  - binding, 69
- proof, 20
- proof
  - tableau, 37
- proof-reflexive, 19
- proof-transitive, 19
- natural deduction
  - propositional (calculus), 22
- propositional
  - natural deduction
    - calculus, 22
  - proposition, 10
  - $\mathcal{T}_0$ -theorem, 37
  - quantified
    - formula, 74
  - quantifier, 74
  - range
    - type, 50
  - reasonable, 28
  - reasoning
    - hypothetical, 23
  - reducing
    - strongly, 58
  - Reduction
    - Head, 57
  - refutation
    - tableau, 37
  - relation
    - logical, 58
  - relation
    - derivation, 19
  - relation
    - entailment, 18
    - satisfaction, 18
  - rule
    - inference, 19
  - $\Sigma$ -algebra, 63
  - satisfaction
    - relation, 18
  - satisfiable, 18
  - satisfied by  $\mathcal{M}$ , 18
  - saturated, 37
  - sentence, 11
  - sequent, 25
  - sequent
    - calculus, 26
  - individuals
    - set of, 51
  - set of
    - individuals, 51
  - set of
    - truth values, 51
  - truth values
    - set of, 51
  - signature, 50
  - signature
    - first-order, 10
  - simplification
    - higher-order (transformations), 68
  - Skolem

- contant, 55
- Skolem
  - constant, 10
- solved
  - form, 68
- solved
  - DAG (form), 44
- solved
  - form, 40
- i, 40
- correct, 20
- sound, 20
- strongly
  - reducing, 58
- subsets
  - closed under, 28
- substitutable, 14
- substitution, 13
- substitution
  - grounding, 66
- support, 13
- symbol
  - head, 57
- syntactic
  - head, 57
- system
  - formal, 20
- system
  - formal, 19
- system
  - logical, 18
- system
  - equational, 40
- tableau
  - proof, 37
  - refutation, 37
- term, 10
- test calculi, 37
- theorem, 20
- truth
  - value, 11
- truth
  - value, 10
- type, 50
- type
  - base, 50
  - domain, 50
  - function, 50
  - range, 50
- truth values
  - type of, 50
- type of
  - truth values, 50
- typed
  - collection, 63
  - function, 63
- individuals
  - type of, 50
- type of
  - individuals, 50
- unary
  - conditional, 76
- unification
  - measure, 72
- unifier, 39
- unitary, 41
- Universe, 11
- universe, 11
- universe
  - function, 51
- unsatisfiable, 18
- valid, 18
- valid, 51
- valid, 74
- value
  - function, 63
- value
  - truth, 11
- value
  - function, 12
- value
  - truth, 10
- value
  - function, 51
- variable, 50
- variable
  - free, 11
- variable
  - assignment, 12
- variable
  - individual, 10
- variable
  - assignment, 51
- variants
  - alphabetical, 16
- weakening
  - admits, 19
- weakly
  - confluent, 62
- well-typed
  - formula, 51
- well-typed
  - formula, 55