

Logic for Computer Science

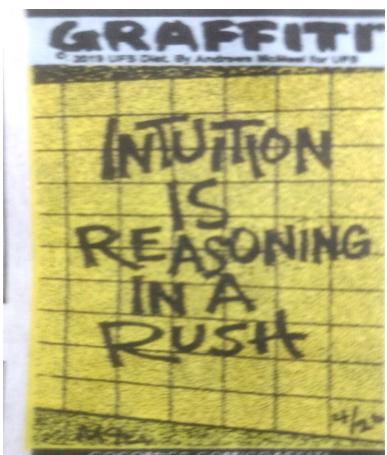
<http://www.cse.iitd.ac.in/~sak/courses/ilcs/2021-22/ilcs.pdf>

S. Arun-Kumar

*Department of Computer Science and Engineering
I. I. T. Delhi, Hauz Khas, New Delhi 110 016.*

© S. Arun-Kumar

August 5, 2021



Contents

-5 Motivation for the Study of Logic	29
-4 Mathematical Preliminaries	33
-4.1 Sets	35
-4.2 Relations and Functions	40
-4.3 Sequences	49
-4.4 Operations on Binary Relations	53
-4.5 Ordering Relations	58
-4.6 Partial Orders and Equivalences	62
-4.7 Equivalences induced by functions	66
-4.8 Well-founded Sets and Well-ordered Sets	68
-4.9 Infinite Sets: Countability and Uncountability	72
-3 Induction Principles	90
-3.1 Mathematical Induction	90
-3.2 Complete Induction	99

-3.3 Structural Induction	106
-3.3.1 Inductive definitions of functions	119
-3.4 Simultaneous Induction	122
-3.5 Well-ordered Induction	125
-3.6 Noetherian Induction	129
-2 Rooted trees	132
-1 Introduction to Universal Algebra	142
-1.1 Data types in functional and declarative languages	143
-1.2 Terms and Term Algebras	154
-1.3 Semantics and Meaning	168
-1.3.1 Semantics of ground terms	172
-1.3.2 Semantics of Open terms	174
0 Substitutions	178
0.1 Introduction to syntactic substitutions	178
0.2 Substitutions and Instantiations	179

0.3	The Composition of Substitutions	184
0.4	Pure-variable Substitutions	188
1	Introduction	192
2	Propositional Logic Syntax	209
2.1	Propositions in Natural Languages	217
2.2	Translation of Natural Language Statements	224
2.3	Associativity and Precedence of Operators	233
3	Semantics of Propositional Logic	245
4	Logical and Algebraic Concepts	261
4.1	Some Meta-Logical Concepts	263
5	Identities and Normal Forms	277
5.0.1	Adequacy	280
5.1	Duality	288
5.2	Normal Forms	293

6 Tautology Checking**7 Propositional Unsatisfiability & Resolution** 321

7.1 Space Complexity of Propositional Resolution	338
7.2 Time Complexity of Propositional Resolution	338
7.3 Unit Resolution	339

8 Binary Decision Diagrams 342

8.1 Remarks on Tautology checking and Resolution	345
--	-----

9 Analytic Tableaux 383**10 Consistency & Completeness** 401

10.1 Consistency and Completeness	403
---	-----

11 The Compactness Theorem 421**12 Maximally Consistent Sets** 431**13 Formal Theories** 450

14 Proof Theory: Hilbert-style	
15 Derived Rules	486
16 The Hilbert System: Soundness	510
16.1 Soundness	513
17 The Hilbert System: Completeness	522
17.1 Completeness of a proof system	524
18 Introduction to Predicate Logic	536
18.1 Predicates in Natural Languages	551
19 The Semantics of Predicate Logic	567
20 Substitutions	584
21 Models, Satisfiability and Validity	600
21.1 An excursion into Ordinals	606
21.2 Meta-operators and overloading	619

21.3 Some more Model Theory	625
22 Structures and Substructures	628
23 Predicate Logic: Proof Theory	646
23.1 The System \mathcal{H}_1 explained	650
24 Predicate Logic: Proof Theory (Contd.)	669
25 Existential Quantification	682
26 Normal Forms and Skolemization	700
27 Herbrand's theorem	718
28 Substitutions and Unification	738
28.1 Syntactic Unification as equation solving	741
29 Resolution in FOL	770
30 More on Resolution in FOL	784

31 Resolution: Soundness and Completeness	
32 Resolution and Tableaux	816
32.1 First-order Analytic Tableau with unification	828
33 Completeness of First-order Tableaux	835
34 Completeness of the Hilbert System	844
34.1 Model-theoretic and Proof-theoretic Consistency	847
35 First-Order Theories	856
35.1 First-order Theories and Models	858
36 Towards Logic Programming	884
37 Verification of Imperative Programs	913
37.1 Programs as Predicate Transformers	922
38 Verification of WHILE Programs	934
38.1 Nested Loops	966

39 Many-sorted FOL

40 Abstract Data Types	983
40.1 Introduction: Compound Data Structures	986
40.2 Arrays as functions	991
40.3 Inductively Defined Data-types	996
40.4 Lists: The First-order Theory	997
40.5 Stacks: The First-order Theory	1011
40.6 Queues: The First Order Theory	1017
40.7 Binary Trees: The First-order Theory	1024

List of Slides

- 1: Introduction
 - 1. What is Logic?
 - 2. Reasoning, Truth and Validity
 - 3. Examples
 - 4. Objectivity in Logic
 - 5. Formal Logic
 - 6. Formal Logic: Applications
 - 7. Form and Content
 - 8. Facets of Mathematical Logic
 - 9. Logic and Computer Science
- 2: Propositional Logic Syntax
 - 1. Truth and Falsehood: 1
 - 2. Truth and Falsehood: 2
 - 3. Extending the Boolean Algebra
 - 4. Sums & Products
 - 5. Propositional Logic: Syntax
 - 6. Propositional Logic: Syntax - 2
 - 7. Natural Language equivalents
 - 8. Some Remarks

- 9. Associativity and Precedence
- 10. Syntactic Identity
- 11. Abstract Syntax Trees
- 12. Subformulae
- 13. Atoms in a Formula
- 14. Degree of a Formula
- 15. Size of a Formula
- 16. Height of a Formula

- 3: Semantics of Propositional Logic

- 1. Semantics of Propositional Logic: 1
- 2. Semantics of Propositional Logic: 2
- 3. A 1-1 Correspondence
- 4. Models and Satisfiability
- 5. Example: Abstract Syntax trees
- 6. Tautology, Contradiction, Contingent

- 4: (Meta-)Logical and Algebraic Concepts

- 1. Unsatisfiability and Inconsistency
- 2. Logical Consequence: 1
- 3. Logical Consequence: 2
- 4. Other Theorems
- 5. Logical Implication and Equivalence

- 6. Implication & Equivalence
- 7. Propositional Logic is An Algebra
- 8. Logical Equivalence as a Congruence

- Lecture 5.1: Binary Decision Diagrams

- 1. Binary Decision Diagrams
- 2. The if-then-else Operator 1
- 3. The if-then-else truth table
- 4. The if-then-else Operator 2
- 5. Boolean Expressions: Variables
- 6. Boolean Expressions: terms
- 7. if-then-else: test, high and low
- 8. Examples: Boolean expressions
- 9. More on if-then-else
- 10. Functional Completeness with the new operator
- 11. Another Normal Form: INF
- 12. The Shannon Expansion
- 13. Example: Truth table to Decision tree
- 14. Example:1
- 15. Example:2
- 16. Example:3
- 17. Example:4,5

- 18. BDDs
- 19. OBDDs
- 20. ROBDDs
- 21. Order of Variables
- 22. ROBDDs: Representing Boolean functions
- 23. The Canonicity Lemma
- 24. Consequences of Canonicity

- 5: Identities and Normal Forms

- 1. Adequacy
- 2. Adequacy: Examples
- 3. Functional Completeness
- 4. Expressively adequate sets
- 5. Duality
- 6. Dual Formulae
- 7. Dual Operators
- 8. Principle of Duality
- 9. Literals: Positive and Negative
- 10. Negation Normal Forms: 1
- 11. Negation Normal Forms: 2
- 12. Conjunctive Normal Forms
- 13. CNF and DNF

- 6: Tautology Checking

1. Arguments
2. Arguments: 2
3. Validity & Falsification
4. Translation into propositional Logic
5. Atoms in Argument
6. The Representation
7. Propositional Rendering
8. The Strategy
9. Checking Tautology
10. Computing the CNF
11. Rewriting Conditionals, Biconditionals
12. Convert to NNF
13. Convert to CNF
14. Falsifying CNF

- 7: Propositional Unsatisfiability & Resolution

1. Tautology Checking
2. CNFs: Set of Sets of Literals
3. Propositional Resolution
4. Clean-up
5. The Resolution Method

- 6. The Resolution Algorithm
 - 7. Resolution Examples: Biconditional
 - 8. Resolution Examples: Exclusive-Or
 - 9. Resolution Refutation: 1
 - 10. Resolution Refutation: 2
 - 11. Resolvent as Logical Consequence
 - 12. Logical Consequence by Refutation
- 8: Binary Decision Diagrams
 - 1. Binary Decision Diagrams
 - 2. The if-then-else Operator 1
 - 3. The if-then-else truth table
 - 4. The if-then-else Operator 2
 - 5. Boolean Expressions: Variables
 - 6. Boolean Expressions: terms
 - 7. if-then-else: test, high and low
 - 8. Examples: Boolean expressions
 - 9. More on if-then-else
 - 10. Functional Completeness with the new operator
 - 11. Another Normal Form: INF
 - 12. The Shannon Expansion
 - 13. Example: Truth table to Decision tree

- 14. Example:1
- 15. Example:2
- 16. Example:3
- 17. Example:4,5
- 18. BDDs
- 19. OBDDs
- 20. ROBDDs
- 21. Order of Variables
- 22. ROBDDs: Representing Boolean functions
- 23. The Canonicity Lemma
- 24. Consequences of Canonicity

- **9: Analytic Tableaux**

- 1. Against Resolution
- 2. Against BDDs
- 3. The Analytic Tableau Method
- 4. Basic Tableaux Facts
- 5. Example: AST
- 6. Example: AST with Complements
- 7. Tableaux Rules
- 8. Structure of the Rules
- 9. Tableaux

10. An Example Tableau Proof

11. Tableaux Heuristics

12. Slim & Short Tableaux

- **10: Consistency & Completeness**

- 1. Tableaux Rules: Restructuring

- 2. Tableaux Rules: 2

- 3. Tableau Proofs

- 4. Consistency

- 5. Satisfiability Preservation

- 6. Unsatisfiability

- 7. Derived Formulae

- 8. Hintikka Sets

- 9. Degree of a Formula

- 10. Degree of a Derived Formula

- 11. Hintikka's Lemma

- 12. Tableaux and Hintikka sets

- 13. Soundness of the Tableau Method

- 14. Completeness of the Tableau Method

- **11: The Compactness Theorem**

- 1. Satisfiability of Infinite Sets

- 2. The Compactness Theorem

- 3. Inconsistency
- 4. Consequences of Compactness
- 12: Maximally Consistent Sets
 - 1. Consistent Sets
 - 2. Maximally Consistent Sets
 - 3. Properties of Finite Character: 1
 - 4. Properties of Finite Character: Examples:1
 - 5. Properties of Finite Character: Examples:2
 - 6. Properties of Finite Character: Compactness
 - 7. Properties of Finite Character: Tukey's Lemma
 - 8. Lindenbaum's Theorem
- 13: Formal Theories
 - 1. Introduction to Reasoning
 - 2. Proof Systems: 1
 - 3. Requirements of Proof Systems
 - 4. Proof Systems: Desiderata
 - 5. Formal Theories
 - 6. Formal Language
 - 7. Axioms and Inference Rules
 - 8. Axiomatic Theories
 - 9. Syntax and Decidability

- 10. A Hilbert-style Proof System
- 11. Axiom Schemas and Rules
- 12. Rule Patterns
- 14: Proof Theory: Hilbert-style
 - 1. More About Formal Theories
 - 2. The Law of The Excluded Middle
 - 3. An Example Proof:1
 - 4. An Example Proof:2
 - 5. An Example Proof:3
 - 6. An Example Proof:4
 - 7. An Example Proof:5
 - 8. Formal Proofs: 1
 - 9. Formal Proofs: 2
 - 10. Provability and Formal Proofs
 - 11. The Deduction Theorem
 - 12. About Formal Proofs
- 15: Derived Rules
 - 1. Simplifying Proofs
 - 2. Derived Rules
 - 3. The Sequent Form
 - 4. Proof trees in sequent form

- 5. Transitivity of Conditional
- 6. Derived Double Negation Rules
- 7. Derived Operators
- 8. Rules for Derived Operators
- 9. Gentzen's System
- 10. Natural Deduction: 1
- 11. Natural Deduction: 2
- 12. Natural Deduction: 3
- 13. Natural Deduction: 4
- 14. Natural Deduction: 5
- 16: The Hilbert System: Soundness
 - 1. Formal Theory: Issues
 - 2. Soundness of Formal Theories
 - 3. Soundness of the Hilbert System
 - 4. Soundness of the Hilbert System
- 17: The Hilbert System: Completeness
 - 1. Formal Theory: Incompleteness
 - 2. Towards Completeness
 - 3. Towards Proofs of Truth-tables
 - 4. The Truth-table Lemma
 - 5. The Completeness Theorem

- 18: Introduction to Predicate Logic

1. Predicate Logic: Introduction-1
2. Predicate Logic: Introduction-2
3. Internal Structure of Sentences
4. Internal Structure of Sentences
5. Parameterisation-1
6. Parameterisation-2
7. Predicate Logic: Introduction-3
8. Predicate Logic: Symbols
9. Predicate Logic: Signatures
10. Predicate Logic: Syntax of Terms
11. Predicate Logic: Syntax of Formulae
12. Precedence Conventions
13. Predicates: Abstract Syntax Trees
14. Subterms: Depth and Size
15. Variables in a Term
16. Bound Variables And Scope
17. Bound Variables And Scope: Example
18. Scope Trees
19. Free Variables
20. Bound Variables

21. Closure

• 19: The Semantics of Predicate Logic

1. Structures
2. Notes on Structures
3. Infix Convention
4. Expansions and Reducts
5. Valuations and Interpretations
6. Evaluating Terms
7. Coincidence Lemma for Terms
8. Valuation Variants
9. Variant Notation
10. Semantics of Formulae
11. Notes on the Semantics
12. Coincidence Lemma for Formulae

• 20: Substitutions

1. Substitutions
2. Instantiation of Terms
3. The Substitution Lemma for Terms
4. Admissibility
5. Instantiations of Formulae
6. The Substitution Lemma for Formulae

- 21: Models, Satisfiability and Validity

1. Satisfiability
2. Models and Consistency
3. Examples of Models:1
4. Other Models
5. Examples of Models:2
6. Examples of Models:3
7. Logical Consequence
8. Validity
9. Validity of Sets of Formulae
10. Negations of Semantical Concepts

- 22: Structures and Substructures

1. Satisfiability and Expansions
2. Distinguishability
3. Evaluations under Different Structures
4. Isomorphic Structures
5. The Isomorphism Lemma
6. Substructures
7. Substructure Facts
8. Substructure Examples
9. Quantifier-free Formulae

- 10. Lemma on Quantifier-free Formulae
- 11. Universal and Existential Formulae
- 12. The Substructure Lemma
- 23: Predicate Logic: Proof Theory
 - 1. Proof Theory: First-Order Logic
 - 2. Proof Rules: Hilbert-Style
 - 3. The Mortality of Socrates
 - 4. The Mortality of the Greeks
 - 5. Faulty Proof:2
 - 6. A Correct Proof
 - 7. The Sequent Forms
 - 8. The Case of Equality
 - 9. Semantics of Equality
 - 10. Theories of Predicate Calculus
 - 11. Axioms for Equality
 - 12. Symmetry and Transitivity
 - 13. Symmetry of Equality
 - 14. Transitivity of Equality
- 24: Predicate Logic: Proof Theory (Contd.)
 - 1. Alpha Conversion
 - 2. The Deduction Theorem for Predicate Calculus

- 3. Useful Corollaries
- 4. Soundness of Predicate Calculus
- 5. Soundness of The Hilbert System
- 25: Existential Quantification
 - 1. Existential Quantification
 - 2. Existential Elimination
 - 3. Remarks on Existential Elimination
 - 4. Restrictions on Existential Elimination
 - 5. Equivalence of Proofs
 - 6. Natural Deduction: 6
- 26: Normal Forms and Skolemization
 - 1. Moving Quantifiers
 - 2. Quantifier Movement
 - 3. More on Quantifier Movement
 - 4. Prenex Normal Forms
 - 5. The Prenex Normal Form Theorem
 - 6. Prenex Conjunctive Normal Form
 - 7. Skolem's Theorem
 - 8. Equisatisfiability
 - 9. Skolem Normal Forms
 - 10. SCNF

- 27: Herbrand's theorem

1. The Herbrand Algebra
2. Terms in a Herbrand Algebra
3. Herbrand Interpretations
4. Herbrand Models
5. Ground Quantifier-free Formulae
6. Ground Instances
7. Herbrand's Theorem
8. The Herbrand Tree of Interpretations
9. Compactness of Sets of Ground Formulae
10. Compactness of Closed Formulae
11. The Löwenheim-Skolem Theorem

- 28: Substitutions and Unification

1. Ground Substitutions
2. Unifiability
3. Unification Examples:1
4. Unification Examples:2
5. Elementary Facts
6. Generality of Unifiers
7. Generality: Facts
8. Most General Unifiers

- 9. More on Positions
 - 10. Disagreement Set
 - 11. Example: Disagreement 1
 - 12. Example: Occurs Check
 - 13. Example: Disagreement 3
 - 14. Example: Disagreement 4
 - 15. Disagreement and Unifiability
 - 16. Example: Unify 1
 - 17. Example: Unify 2
 - 18. The Unification Theorem
 - 19. Unification: consequences
- 29: Resolution in FOL
 - 1. Recapitulation
 - 2. SCNFs and Models
 - 3. SCNFs and Unsatisfiability
 - 4. Representing SCNFs
 - 5. Clauses: Terminology
 - 6. Clauses: Ground Instances
 - 7. Facts about Clauses
 - 8. Clauses: Models
 - 9. Clauses and Herbrand's Theorem

- 10. Resolution in FOL
- 11. The Resolution Rule for FOL
- 30: More on Resolution in FOL
 - 1. Standardizing Variables Apart
 - 2. Factoring
 - 3. Example: 1
 - 4. Example: 2
 - 5. Refutation
 - 6. Refutations: Using the Herbrand Universe
- 31: Resolution: Soundness and Completeness
 - 1. Soundness of FOL Resolution
 - 2. Ground Clauses
 - 3. The Lifting Lemma
 - 4. Lifting Lemma: Figure
 - 5. Completeness of Resolution Refutation: 1
 - 6. Completeness of Resolution Refutation: 2
 - 7. Completeness of Resolution Refutation: 3
- 32: Resolution and Tableaux
 - 1. FOL: Tableaux
 - 2. FOL: Tableaux Rules

- 3. FOL Tableaux: Example 1
 - 4. FOL Tableaux: Example 1 Contd
 - 5. First-Order Tableaux
 - 6. First-Order Tableaux: Heuristics
 - 7. FOL Tableaux: Example 2
 - 8. FOL Tableaux: Example 2 Contd
- 33: Completeness of First-order Tableaux
 - 1. First-order Hintikka Sets
 - 2. Hintikka's Lemma for FOL
 - 3. First-order tableaux and Hintikka sets
 - 4. Soundness of First-order Tableaux
 - 5. Completeness of First-order Tableaux
 - 34: Completeness of the Hilbert System
 - 1. Deductive Consistency
 - 2. Models of Deductively Consistent Sets
 - 3. Deductive Completeness
 - 4. The Completeness Theorem
 - 35: First-order Theories
 - 1. (Simple) Directed Graphs
 - 2. (Simple) Undirected Graphs

- 3. Irreflexive Partial Orderings
- 4. Irreflexive Linear Orderings
- 5. (Reflexive) Preorders
- 6. (Reflexive) Partial Orderings
- 7. (Reflexive) Linear Orderings
- 8. Equivalence Relations
- 9. Peano's Postulates
- 10. The Theory of The Naturals
- 11. Notes and Explanations
- 12. Finite Models of Arithmetic
- 13. A Non-standard Model of Arithmetic
- 14. Z-Chains
- 15. First-order Mathematical Induction
- 16. First-order Arithmetic
- 17. Extending First-order Arithmetic
- 36: Towards Logic Programming
 - 1. Logic Programming
 - 2. Reversing the Arrow
 - 3. Arrow Reversal
 - 4. Horn Clauses
 - 5. Program or Rule Clause

- 6. Facts: Unit Clauses
- 7. Goal clauses
- 8. Logic Programs
- 9. Prolog: Abstract Interpreter 0
- 10. Sorting in Logic
- 11. Prolog: Abstract Interpreter 1
- 37: Verification of Imperative Programs
 - 1. The WHILE Programming Language
 - 2. Programs As State Transformers
 - 3. The Semantics of WHILE
 - 4. Programs As Predicate Transformers
 - 5. Correctness Assertions
 - 6. Total Correctness of Programs
 - 7. Total Correctness = Partial Correctness + Termination
 - 8. Examples: Factorial 1
 - 9. Examples: Factorial 1'
 - 10. Examples: Factorial 2
- 38: Verification of WHILE Programs
 - 1. Proof Rule: Epsilon
 - 2. Proof Rule: Assignment
 - 3. Proof Rule: Composition

- 4. Proof Rule: The Conditional
- 5. Proof Rule: The While Loop
- 6. The Consequence Rule
- 7. Proof Rules for Partial Correctness
- 8. Example: Factorial 1
- 9. Towards Total Correctness
- 10. Termination and Total Correctness
- 11. Example: Factorial 2
- 12. Notes on Example: Factorial
- 13. Example: Factorial 2 Made Complete
- 14. An Open Problem: Collatz
- 39: Many-sorted FOL

- 1. Sortedness
- 2. Many-Sorted Logic: Symbols
- 3. Many-Sorted Signatures
- 4. Many-Sorted Signature: Terms
- 5. Many-Sorted Predicate Logic
- 6. Reductions: Guardedness
- 7. Reductions: Bounded Quantification

- 40: Abstract Data Types

- 1. Pairs

2. The Array Structure: Signature
3. The Array Structure: Typing Axioms
4. The Array Structure: Access and Update Axioms
5. The List Structure: Signature
6. The List Structure Axioms: 1
7. The List Structure Axioms: 2
8. The List Structure: Induction
9. The List Structure: Acyclicity 1
10. The List Structure: Acyclicity 2
11. The Stack Structure: Signature
12. The Stack Structure Axioms: 1
13. The Stack Structure Axioms: 2
14. The Stack Structure: Induction
15. The Queue Structure: Signature
16. The Queue Structure Axioms: 1
17. The Queue Structure Axioms: 2
18. The Queue Structure: Induction
19. Binary Trees: Signature
20. The Binary Tree Axioms: 1
21. The Binary Tree Axioms: 2
22. The Binary Tree Axioms: 3
23. The Binary Tree Induction Rule

- 41: Abstract Data Types

1. Pairs
2. The Array Structure: Signature
3. The Array Structure: Typing Axioms
4. The Array Structure: Access and Update Axioms
5. The List Structure: Signature
6. The List Structure Axioms: 1
7. The List Structure Axioms: 2
8. The List Structure: Induction
9. The List Structure: Acyclicity 1
10. The List Structure: Acyclicity 2
11. The Stack Structure: Signature
12. The Stack Structure Axioms: 1
13. The Stack Structure Axioms: 2
14. The Stack Structure: Induction
15. The Queue Structure: Signature
16. The Queue Structure Axioms: 1
17. The Queue Structure Axioms: 2
18. The Queue Structure: Induction
19. Binary Trees: Signature
20. The Binary Tree Axioms: 1

- 21. The Binary Tree Axioms: 2
- 22. The Binary Tree Axioms: 3
- 23. The Binary Tree Induction Rule
- 42: Miscellaneous Examples
 - 1. Random Examples

-5. Motivation for the Study of Logic

“Where shall I begin, please your Majesty?” he asked.

“Begin at the beginning,” the King said gravely, “and go on till you come to the end: then stop.”

Lewis Carroll, *Alice's Adventures in Wonderland*

In the early years of the twentieth century symbolic or formal logic became quite popular with philosophers and mathematicians because they were interested in the concept of what constitutes a correct proof in mathematics. Over the centuries mathematicians had pronounced various mathematical proofs as correct which were later disproved by other mathematicians. The whole concept of logic then hinged upon what is a correct argument as opposed to a wrong (or faulty) one. This has been amply illustrated by the number of so-called proofs that have come up for Euclid’s parallel postulate and for Fermat’s last theorem. There have invariably been “bugs” (a term popularised by computer scientists for the faults in a program) which were often very hard to detect and it was necessary therefore to find infallible methods of proof. For centuries (dating back at least to Plato and Aristotle) no rigorous formulation was attempted to capture the notion of a correct argument which would guide the development of all mathematics.

The early logicians of the nineteenth and twentieth centuries hoped to establish formal logic as a foundation for mathematics, though that never really happened. But mathematics does rest on one firm foundation, namely set theory. But Set theory itself has been expressed in first order logic. What really needed to be answered were questions relating to the automation or mechanizability of proofs. These questions are very relevant and important for the development of present-day computer

science and form the basis of many developments in automatic theorem proving. David Hilbert asked the important question, as to whether all mathematics, if reduced to statements of symbolic logic, can be derived by a machine. Can the act of constructing a proof be reduced to the manipulation of statements in symbolic logic? Logic enabled mathematicians to point out why an alleged proof is wrong, or where in the proof, the reasoning has been faulty. A large part of the credit for this achievement must go to the fact that by symbolising arguments rather than writing them out in some natural language (which is fraught with ambiguity), checking the correctness of a proof becomes a much more viable task. Of course, trying to symbolise the whole of mathematics could be disastrous as then it would become quite impossible to even read and understand mathematics, since what is presented usually as a one page proof could run into several pages. But at least in principle it can be done.

Since the latter half of the twentieth century logic has been used in computer science for various purposes ranging from program specification and verification to theorem-proving. Initially its use was restricted to merely specifying programs and reasoning about their implementations. This is exemplified in the some fairly elegant research on the development of correct programs using first-order logic in such calculi such as the weakest-precondition calculus of Dijkstra. A method called Hoare Logic which combines first-order logic sentences and program phrases into a specification and reasoning mechanism is also quite useful in the development of small programs. Logic in this form has also been used to specify the meanings of some programming languages, notably Pascal.

The close link between logic as a formal system and computer-based theorem proving is proving to be very useful especially where there are a large number of cases (following certain patterns) to be analysed and where quite often there are routine proof techniques available which are more easily and accurately performed by theorem-provers than by humans. The case of the four-colour theorem which until fairly recently remained a unproved conjecture is an instance of how human ingenuity

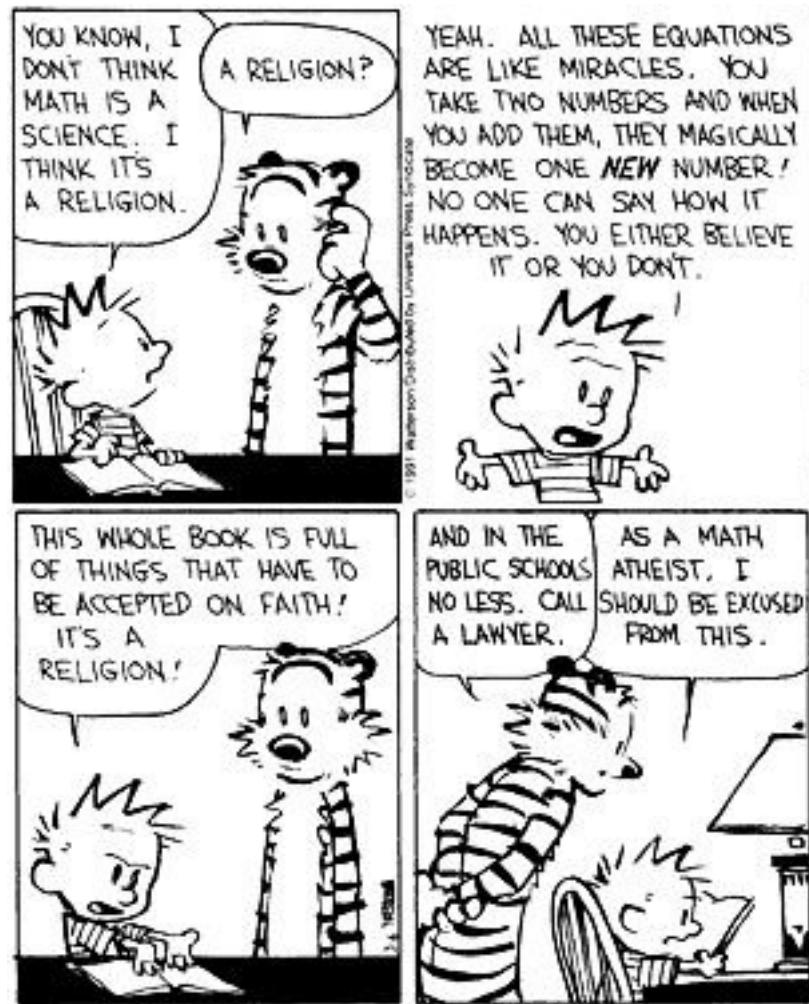
and creativity may be used to divide up proof into a few thousand cases and where machines may be used to perform routine checks on the individual cases. Another use of computers in theorem-proving or model-checking is the verification of the design of large circuits before a chip is fabricated. Analysing circuits with a billion transistors in them is at best error-prone and at worst a drudgery that few humans would like to do. Such analysis and results are best performed by machines using theorem proving techniques or model-checking techniques.

A powerful programming paradigm called declarative programming has evolved since the late seventies and has found several applications in computer science and artificial intelligence. Most programmers using this logical paradigm use a language called Prolog which is an implemented form of logic¹. More recently computer scientists are working on a form of logic called constraint logic programming.

Today a number of fields within computer science require a solid foundation in logic. These include automatic theorem proving, logic programming, abstract data types, program verification, program synthesis, reasoning in Artificial Intelligence etc. None of these studies is possible without a solid foundation in mathematical logic. Further mathematical logic is studied as a sub-field of mathematics with applications to mathematics and computer science.

¹actually a subset of logic called Horn-clause logic

-4. Mathematical Preliminaries



In this section we will discuss sets, relations, functions and other background material required for the rigorous study of mathematical logic. Though most of these topics are covered in the high school curriculum this section also establishes the notational conventions that will be used throughout. Even a confident reader may wish to browse this section to get familiar with the notation.

Mathematics of course is all about solving problems, and mathematical logic is no different. With every hard problem there are also a number of solutions proposed – some may be correct and others may be fallacious. So as a warm-up to the study of logic we ask the reader to solve the following problem which requires nothing more than middle school mathematics.

Exercise -4.1

- Find the fallacy in the proof of the following purported theorem.

Theorem: If $x = y$ then $2 = 1$.

Proof:

1.	$x = y$	Given
2.	$x^2 = xy$	Multiply both sides by x
3.	$x^2 - y^2 = xy - y^2$	Subtract y^2 from both sides
4.	$(x + y)(x - y) = y(x - y)$	Factorize
5.	$x + y = y$	Cancel out $(x - y)$
6.	$2y = y$	Substitute x for y , by equation 1.
7.	$2 = 1$	Divide both sides by y

QED

-4.1. Sets

A *set* is a collection of *distinct* objects. The class of students in a course is a set. So is the group of all first year students at IITD. We will use the notation $\{a, b, c\}$ to denote the collection of the objects a , b and c . The elements in a set are not ordered in any fashion. Thus the set $\{a, b, c\}$ is the same as the set $\{b, a, c\}$. Further, repetitions of elements in a set do not change it in any way. Two sets are *equal* if they contain exactly the same elements. Hence the sets $\{a, b, c\}$, $\{a, b, c, a\}$, $\{b, a, c\}$, $\{c, b, a, c\}$ are all equal. We write $|A|$ to denote the number of elements in a finite set A .

We can describe a set either by enumerating all the elements of the set or by stating the properties that uniquely characterize the elements of the set. Thus, the set of all even positive integers not larger than 10 can be described either as $S = \{2, 4, 6, 8, 10\}$ or, equivalently, as $S = \{x \mid x \text{ is an even positive integer not larger than } 10\}$

A set can have another set as one of its elements. For example, the set $A = \{\{a, b, c\}, d\}$ contains two elements $\{a, b, c\}$ and d ; and the first element is itself a set. We will use the notation $x \in S$ to denote that x is an *element of* (or *belongs to*) the set S .

A set A is a *subset* of another set B , denoted as $A \subseteq B$, if $x \in B$ whenever $x \in A$. Equivalently B is a *superset* of A , denoted $B \supseteq A$ if $A \subseteq B$.

An *empty set* is one which contains no elements and we will denote it with the symbol \emptyset . For example, let S be the set of all students who fail this course. S might turn out to be empty (hopefully; if everybody studies hard). By definition, the empty set \emptyset is a subset of all sets. We will also assume an *Universe of discourse* \mathbb{U} , and every set that we will consider is a subset of \mathbb{U} . Thus we have

1. $\emptyset \subseteq A$ for any set A .
2. $A \subseteq \mathbb{U}$ for any set A .
3. $A \supseteq B$ if and only if $A \subseteq B$.
4. $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.
5. A is a *proper subset* of B , denoted $A \subset B$, if and only if $A \subseteq B$ and $A \neq B$.

The *union* of two sets A and B , denoted $A \cup B$, is the set whose elements are exactly the elements of either A or B (or both). The *intersection* of two sets A and B , denoted $A \cap B$, is the set whose elements are exactly the elements that belong to *both* A and B . The *difference* of B from A , denoted $A - B$, is the set of all elements of A that do not belong to B . The *complement* of A , denoted $\sim A$ is the difference of A from the universe \mathbb{U} . Thus, we have

1. $A \cup B = \{x \mid (x \in A) \text{ or } (x \in B)\}$
2. $A \cap B = \{x \mid (x \in A) \text{ and } (x \in B)\}$
3. $A - B = \{x \mid (x \in A) \text{ and } (x \notin B)\}$
4. $\sim A = \mathbb{U} - A$

We also have the following named identities that hold for all sets A , B and C .

Theorem -4.1: Some set identities

(Compare with the Boolean identities)

$$A \cup \emptyset = A$$

$$A \cup \mathbb{U} = \mathbb{U}$$

$$A \cup A = A$$

$$A \cup B = B \cup A$$

$$(A \cup B) \cup C = A \cup (B \cup C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$\sim (A \cup B) = \sim A \cap \sim B$$

$$A \cup \sim A = \mathbb{U}$$

$$\sim \emptyset = \mathbb{U}$$

$$A \cup (A \cap B) = A$$

Negation $\sim \sim A = A$

Identity $A \cap \mathbb{U} = A$

Zero $A \cap \emptyset = \emptyset$

Idempotence $A \cap A = A$

Commutativity $A \cap B = B \cap A$

Associativity $(A \cap B) \cap C = A \cap (B \cap C)$

Distributivity $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

De Morgan $\sim (A \cap B) = \sim A \cup \sim B$

Simplification $A \cap \sim A = \emptyset$

Inversion $\sim \mathbb{U} = \emptyset$

Absorption $A \cap (A \cup B) = A$

Notation -4.1: Some Standard sets

- The empty set: \emptyset
- The Universe: \mathbb{U}
- The set of Natural Numbers: $\mathbb{N} = \{0, 1, 2, \dots\}$. We will include 0 in the set of Natural numbers. After all, it is quite natural to score a 0 in an examination!
- The set of positive integers: $\mathbb{P} = \{1, 2, 3, \dots\}$
- The two-element set: $\mathbb{2} = \{0, 1\}$. More generally for any natural number n we let $\mathbb{n} = \{0, 1, \dots, n - 1\}$ the set of all naturals less than n . By convention \emptyset is the set of all naturals less than 0.
- The set of integers: $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
- The set of rational numbers: \mathbb{Q}
- The set of real numbers: \mathbb{R}
- The Boolean set: $\mathbb{B} = \{\text{false}, \text{true}\}$
- The Powerset of a set A : $\mathbb{2}^A$ is the set of all subsets of the set A .

Exercise -4.2: Set Operations

Let A , B and C be subsets of some non-empty universe \mathbb{U} .

1. Prove or disprove the following. To disprove a statement find an instance of the statement which is false.
 - (a) $A \cup (B - C) = (A \cup B) - (A \cup C)$
 - (b) $A \cap (B - C) = (A \cap B) - (A \cap C)$
 - (c) $A - B = A - (A \cap B)$
2. Prove that if $A \subseteq B$ then $\mathcal{P}^A \subseteq \mathcal{P}^B$.
3. Let $A \oplus B$ be defined as the set of elements that occur in exactly one of the two sets A or B but not in both. Consider the following statements and in each case prove or disprove the statement.
 - (a) \oplus is a commutative operation on sets.
 - (b) \oplus is an associative operation on sets.
 - (c) \cup distributes over \oplus .
 - (d) \cap distributes over \oplus .
 - (e) Set difference distributes over \oplus .
 - (f) \oplus distributes over \cup .
 - (g) \oplus distributes over \cap .
 - (h) \oplus distributes over set difference.
 - (i) $\sim(A \oplus B) = (\sim A) \oplus (\sim B)$.

-4.2. Relations and Functions

Definition -4.1: Cartesian product

The **Cartesian product** of two sets A and B , denoted by $A \times B$, is the set of all ordered pairs (a, b) such that $a \in A$ and $b \in B$. Thus,

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

Given another set C we may form the following different kinds of cartesian products (which are not at all the same!).

$$(A \times B) \times C = \{((a, b), c) \mid a \in A, b \in B \text{ and } c \in C\}$$

$$A \times (B \times C) = \{(a, (b, c)) \mid a \in A, b \in B \text{ and } c \in C\}$$

$$A \times B \times C = \{(a, b, c) \mid a \in A, b \in B \text{ and } c \in C\}$$

The last cartesian product gives the construction of *triples*. Elements of the set $A_1 \times A_2 \times \dots \times A_n$ for given sets A_1, A_2, \dots, A_n are called *ordered n-tuples*.

A^n is the set of all ordered n -tuples (a_1, a_2, \dots, a_n) such that $a_i \in A$ for all i . i.e.,

$$A^n = \underbrace{A \times A \times \cdots \times A}_{n \text{ times}}$$

Definition -4.2: Binary relation

A **binary relation** \mathcal{R} from A to B is a subset of $A \times B$. When A and B are the same set, \mathcal{R} is said to be a **binary relation on A** .

It is a characterization of the intuitive notion that some of the elements of A are related to some of the elements of B .

Notation -4.2

Let $\mathcal{R} \subseteq A \times B$ be a binary relation

1. We also use the *infix* notation $a\mathcal{R}b$ to mean $(a, b) \in \mathcal{R}$.
2. $\mathcal{D}\text{om}\mathcal{R} = \{a \in A \mid (a, b) \in \mathcal{R} \text{ for some } b \in B\}$
3. $\mathcal{R}\text{an}\mathcal{R} = \{b \in B \mid (a, b) \in \mathcal{R} \text{ for some } a \in A\}$
4. If \mathcal{R} is a binary relation on A , $\mathcal{F}\text{ield}\mathcal{R} = \mathcal{D}\text{om}\mathcal{R} \cup \mathcal{R}\text{an}\mathcal{R}$.

Familiar binary relations from \mathbb{N} to \mathbb{N} are $=, \neq, <, \leq, >, \geq$. Thus the elements of the set $\{(0, 0), (0, 1), (0, 2), \dots, (1, 1), (1, 2), \dots\}$ are all members of the relation \leq which is a subset of $\mathbb{N} \times \mathbb{N}$.

In general, an *n-ary relation* among the sets A_1, A_2, \dots, A_n is a subset of the set $A_1 \times A_2 \times \dots \times A_n$.

Definition -4.3: Relations

Let $\mathcal{R} \subseteq A \times B$ be a binary relation from A to B . Then

1. For any set $A' \subseteq A$ the **image** of A' under \mathcal{R} is the set defined by

$$\mathcal{R}(A') = \{b \in B \mid a\mathcal{R}b \text{ for some } a \in A'\}$$

2. For every subset $B' \subseteq B$ the **pre-image** of B' under \mathcal{R} is the set defined by

$$\mathcal{R}^{-1}(B') = \{a \in A \mid a\mathcal{R}b \text{ for some } b \in B'\}$$

3. \mathcal{R} is **onto** (or **surjective**) with respect to A and B if $\mathcal{R}(A) = B$.

4. \mathcal{R} is **total** with respect to A and B if $\mathcal{R}^{-1}(B) = A$.

Definition -4.4: Functions

Let $\mathcal{R} \subseteq A \times B$ be a binary relation from A to B . Then

1. \mathcal{R} is **one-to-one** (or **injective**) with respect to A and B if for every $b \in B$ there is at most one $a \in A$ such that $(a, b) \in \mathcal{R}$.
2. \mathcal{R} is a **partial function** from A to B , usually denoted $\mathcal{R} : A \rightharpoonup B$, if for every $a \in A$ there is at most one $b \in B$ such that $(a, b) \in \mathcal{R}$.
3. \mathcal{R} is a **total function** from A to B , usually denoted $\mathcal{R} : A \longrightarrow B$ if \mathcal{R} is a partial function from A to B and is total. A is called the **domain** and B the **co-domain**. The **range** of the function is $\mathcal{R}(A)$.
4. \mathcal{R} is a **one-to-one correspondence** (or **bijection**) if it is an injective and surjective total function.

Notation -4.3: Graph, domain and range of a function

Given a (partial or total) function $f : A \rightharpoonup B$,

- The **graph** of f , $\text{Graph}(f) = \{(a, b) \in A \times B \mid f(a) = b\}$ is the binary relation it corresponds to.
- $\text{Dom}(f) = \text{Dom } \text{Graph}(f)$ is the **domain** of f .
- $\text{Ran}(f) = \text{Ran } \text{Graph}(f)$ is the **range** of f .

Notes.

1. Every total function is also a partial function.
2. A binary relation $\mathcal{R} \subseteq A \times B$ may also be thought of as a total function $\mathcal{R} : \mathcal{P}(A) \rightarrow \mathcal{P}(B)$. Likewise \mathcal{R}^{-1} the converse of the relation \mathcal{R} , may be thought of as a total function $\mathcal{R}^{-1} : \mathcal{P}(B) \rightarrow \mathcal{P}(A)$ (c.f. parts 1 and 2 of definition -4.4 where the relation symbol has been “overloaded”).
3. Similarly every partial function $f : A \rightharpoonup B$ may be “overloaded” to mean the *total* function $f : \mathcal{P}(A) \rightarrow \mathcal{P}(B)$, which yields the image of A' for each $A' \subseteq A$.

Notation -4.4: Injective, surjective and bijective functions

Let f be a total function from set A to set B . Then

- $f : A \xrightarrow{1-1} B$ will denote that f is injective,
- $f : A \xrightarrow{\text{onto}} B$ will denote that f is surjective, and
- $f : A \xrightarrow[{\text{onto}}]{1-1} B$ will denote that f is bijective,

Example -4.1

The following are some examples of familiar binary relations along with their properties.

1. The \leq relation on \mathbb{N} is a relation from \mathbb{N} to \mathbb{N} which is total and onto. That is, both the image and pre-image of \leq under \mathbb{N} are \mathbb{N} itself. What are image and the pre-image respectively of the relation $<$?
2. The binary relation which associates key sequences from a computer keyboard with their respective 8-bit ASCII codes is an example of a relation which is total and injective.
3. The binary relation which associates 7-bit ASCII codes with their corresponding ASCII characters is a bijection.

The figures 1, 2, 3, 4 and 5 respectively illustrate the concepts of partial, injective, surjective, bijective and inverse of a bijective function on finite sets. The directed arrows go from elements in the domain to their images in the codomain.

We may equivalently define partial and total functions as follows.

Definition -4.5: Total and partial functions

A **function** (or a **total function**) f from A to B is a binary relation $f \subseteq A \times B$ such that for *every* element $a \in A$ there is a *unique* element $b \in B$ so that $(a, b) \in f$ (usually denoted $f(a) = b$ and sometimes $f : a \mapsto b$). We will use the notation $R : A \rightarrow B$ to denote a function R from A to B . The set A is called the **domain** of the function R and the set B is called the **co-domain** of the function R . The **range** of a function $R : A \rightarrow B$ is the set $\{b \in B \mid \text{for some } a \in A, R(a) = b\}$. A **partial function** f from A to B , denoted $f : A \rightharpoonup B$ is a total function from some subset of A to the set B . Clearly every total function is also a partial function.

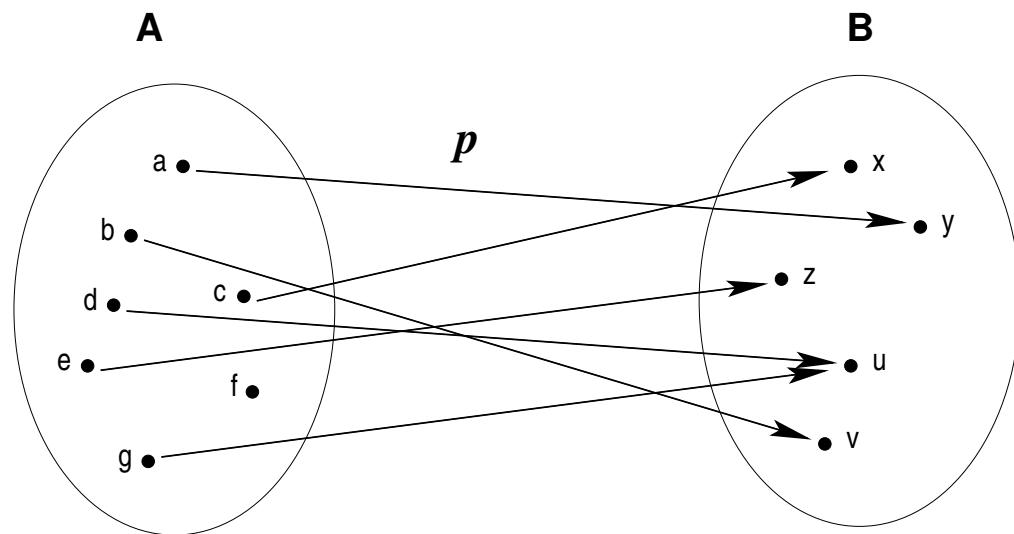


Figure 1: A partial function (*Why is it partial?*)

The word “function” unless otherwise specified is taken to mean a “total function”. Some familiar examples of partial and total functions are

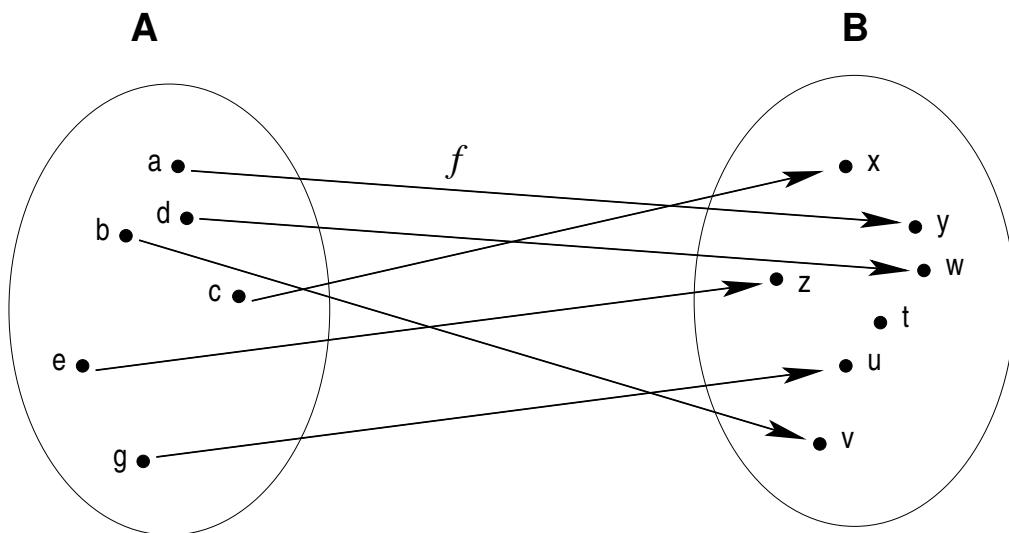


Figure 2: An injective function (*Why is it injective?*)

Example -4.2: Familiar functions

1. + and \times (addition and multiplication) on the natural numbers are total functions of the type $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
2. $-$ (subtraction) on the natural numbers is a partial function of the type $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$.
3. *div* and *mod* are total functions of the type $f : \mathbb{N} \times \mathbb{P} \rightarrow \mathbb{N}$. If $a = q * b + r$ such that $0 \leq r < b$ and $a, b, q, r \in \mathbb{N}$ then the functions *div* and *mod* are defined as $\text{div}(a, b) = q$ and $\text{mod}(a, b) = r$. We will often write these binary functions as $a * b$, $a \text{ div } b$, $a \text{ mod } b$ etc. Note that *div* and *mod* are also partial functions of the type $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$.
4. The binary relations $=, \neq, <, \leq, >, \geq$ may also be thought of as functions of the type $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$ where $\mathbb{B} = \{\text{false}, \text{true}\}$.

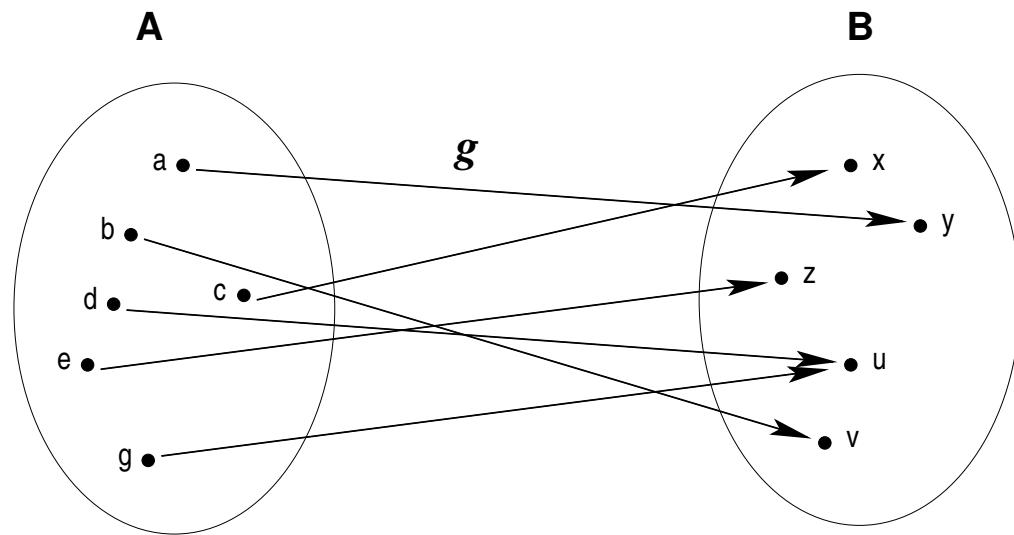


Figure 3: A surjective function (*Why is it surjective? Why is it not injective?*)

Exercise -4.3: Bijections

1. Prove that the composition of bijections is a bijection. That is, prove that for any bijective total functions $f : A \xrightarrow[\text{onto}]{1-1} B$ and $g : B \xrightarrow[\text{onto}]{1-1} C$, their composition is the function $h = g \circ f : A \longrightarrow C$ defined as $h(a) = g(f(a))$.
2. Is the composition of injective functions also injective? Is the composition of surjective functions also surjective? Prove or disprove the two statements.
3. Prove that the inverse of a bijective function is also a bijective function (*Hint: Since the inverse of a function is not guaranteed to be a function, you need to prove that first.*)

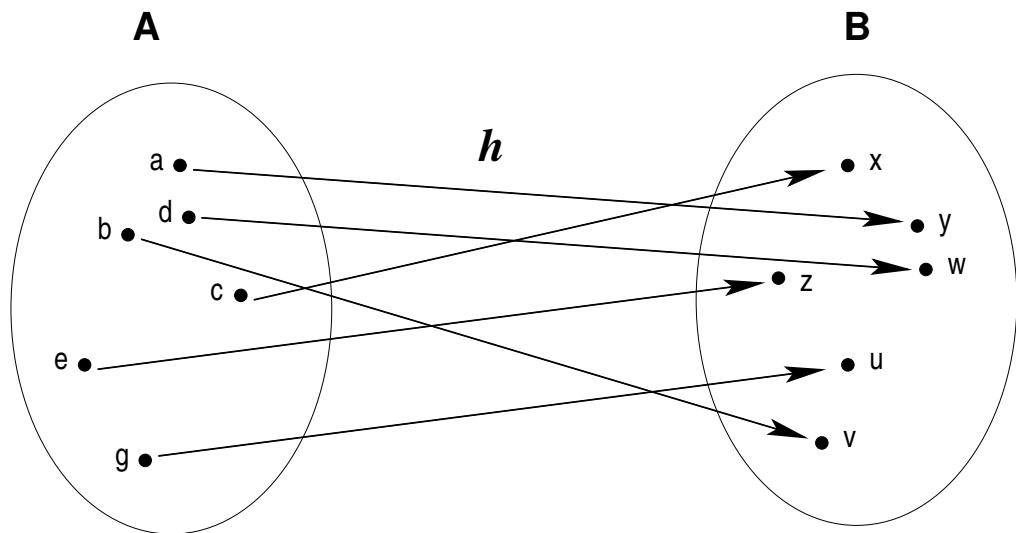


Figure 4: An bijective function (*Why is it bijective?*)

-4.3. Sequences

Sequences may be regarded as functions over an indexing set.

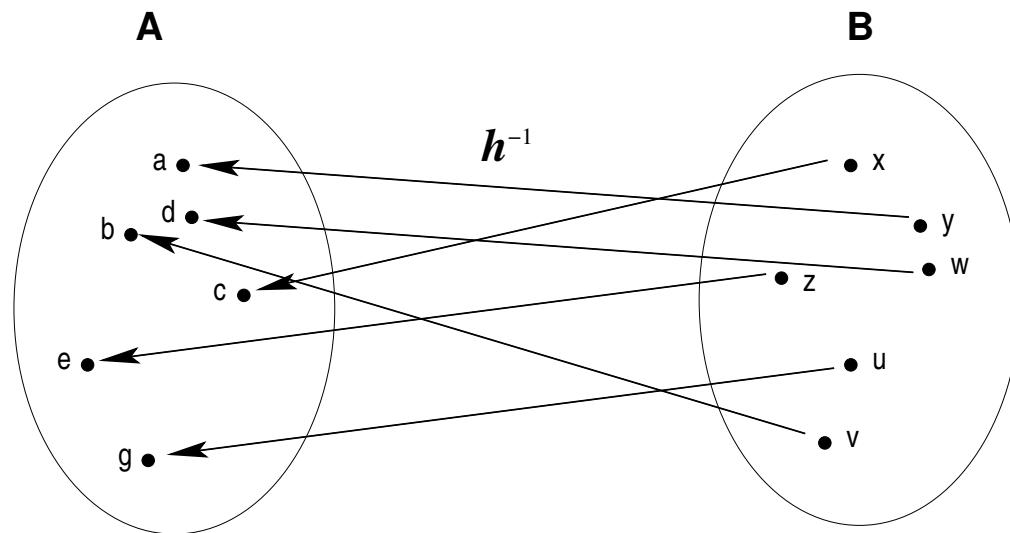


Figure 5: The inverse of the bijective function in Fig 4 (*Is it bijective?*)

Definition -4.6: Finite sequences

Given a set A , a **finite sequence** s of length $n \geq 0$ of elements from A , is a (total) function of the type $s : \{1, 2, \dots, n\} \rightarrow A$, where the domain of the function is also called the **index set**. We normally denote such a sequence of length n by $[a_1, a_2, \dots, a_n]$. Alternatively, s may be regarded as a total function from $\{0, \dots, n-1\}$ to A and may be denoted by $[a_0, a_1, \dots, a_{n-1}]$. The *empty sequence*, denoted $[]$, is also such a function $[] : \emptyset \rightarrow A$ and denotes a sequence of length 0.

Notation -4.5: Sequences

The sequence $[a_0, a_1, \dots, a_{n-1}]$ may also be written $[a_i | 0 \leq i < n]$. Similarly the sequence $[a_1, a_2, \dots, a_n]$ may be written $[a_i | 1 \leq i \leq n]$ or $[a_i | 0 < i \leq n]$. Contrast this with the notation for tuples viz. $(a_1, a_2, \dots, a_{n-1}) \in A^n$.

It is also very common in computer science to distinguish between the notion of a sequence and that of a string or a word over an alphabet.

Definition -4.7: Letter, alphabet, word, string, (con)catenation

An **alphabet** is a nonempty finite set of symbols called **letters**. Any finite sequence of letters from an alphabet A is called a **string** or a **word**. A string of length $n \in \mathbb{N}$ is usually written $a_1a_2\dots a_n$ or “ $a_1a_2\dots a_n$ ”, where each $a_i \in A$, $1 \leq i \leq n$. The operation of juxtaposing two strings s and t to form a new string $s.t$ is called **(con)catenation** i.e. if $s = a_1a_2\dots a_m$ and $t = b_1b_2\dots b_n$ then $s.t = a_1a_2\dots a_mb_1b_2\dots b_n$.

Notation -4.6

1. A^* is the set of all strings from the alphabet A .
2. The unique empty string (of length 0) is usually denoted ε .
3. The “.” denoting the concatenation operation is often omitted.

Definition -4.8: Prefix and prefix ordering

Given a string (see definition [-4.7](#)) w of symbols from an alphabet, a string u is called a **prefix** of w if there is a string v such that $w = uv$. u is called a **proper** prefix of w if v is a nonempty string.

Notation -4.7

If u and v are strings from an alphabet,

1. $u \preceq v$ if u is a prefix of v .
2. $u \prec v$ if u is a proper prefix of v .

Fact -4.1

Given an alphabet A ,

1. The empty string ε is a prefix of every string.
2. Every string is a prefix of itself.
3. The prefix ordering is a partial order on A^* .

It is quite clear that there exists a simple bijection from the set A^n (which is the set of all n -tuples of elements from the set A) and the set of all sequences of length n of elements from A . We will often identify the two as being the same set even

though they are actually different by definition².

Notation -4.8: Sets of finite sequences

1. The set of all finite sequences of elements from A is denoted A^* , where $A^* = \bigcup_{n \geq 0} A^n$.
2. The set of all *non-empty* sequences of elements from A is denoted A^+ and is defined as $A^+ = \bigcup_{n > 0} A^n$.

Definition -4.9: Infinite sequences

An **infinite** sequence of elements from A is a total function from \mathbb{P} to A . Alternatively it is a total function from \mathbb{N} to A . The set of all such infinite sequences is denoted A^ω .

-4.4. Operations on Binary Relations

In this section we will consider various operations on binary relations.

²In a programming language like ML, the difference is evident from the notation and the constructor operations for tuples and lists

Definition -4.10: Identity, converse and composition of relations

1. Given a set A , the **identity** relation over A , denoted \mathcal{I}_A , is the set $\{(a, a) \mid a \in A\}$. Notice that $\mathcal{I}_A \subseteq A \times A$.
2. Given a binary relation \mathcal{R} from A to B , the **converse** of \mathcal{R} , denoted \mathcal{R}^{-1} is the relation from B to A defined as $\mathcal{R}^{-1} = \{(b, a) \mid (a, b) \in \mathcal{R}\}$.
3. Given binary relations $\mathcal{R} \subseteq A \times B$ and $\mathcal{S} \subseteq B \times C$, the **composition** of \mathcal{R} with \mathcal{S} is denoted $\mathcal{R}; \mathcal{S}$ and defined as $\mathcal{R}; \mathcal{S} = \{(a, c) \mid a \mathcal{R} b \text{ and } b \mathcal{S} c, \text{ for some } b \in B\}$.

Exercise -4.4: Relational operations and set-theory

1. Prove that for any binary relations $\mathcal{R}, \mathcal{R}'$ from A to B and $\mathcal{S}, \mathcal{S}'$ from B to C , if $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{S} \subseteq \mathcal{S}'$ then $\mathcal{R}; \mathcal{S} \subseteq \mathcal{R}'; \mathcal{S}'$
2. Prove or disprove^a that relational composition satisfies the following distributive laws for relations, where $\mathcal{R} \subseteq A \times B$ and $\mathcal{S}, \mathcal{T} \subseteq B \times C$.
 - (a) $\mathcal{R}; (\mathcal{S} \cup \mathcal{T}) = (\mathcal{R}; \mathcal{S}) \cup (\mathcal{R}; \mathcal{T})$
 - (b) $\mathcal{R}; (\mathcal{S} \cap \mathcal{T}) = (\mathcal{R}; \mathcal{S}) \cap (\mathcal{R}; \mathcal{T})$
 - (c) $\mathcal{R}; (\mathcal{S} - \mathcal{T}) = (\mathcal{R}; \mathcal{S}) - (\mathcal{R}; \mathcal{T})$
3. Prove that for $\mathcal{R} \subseteq A \times B$ and $\mathcal{S} \subseteq B \times C$, $(\mathcal{R}; \mathcal{S})^{-1} = (\mathcal{S}^{-1}); (\mathcal{R}^{-1})$.

^athat is, find an example of appropriate relations which actually violate the equality

Note that unlike in the case of functions (where for any function $f : A \rightarrow B$ its inverse $f^{-1} : B \rightarrow A$ may not always be defined), the converse of a relation is always defined. Hence $f^{-1} : \mathcal{P}^B \rightarrow \mathcal{P}^A$ is always well-defined and for each $B' \subseteq B$,

yields the pre-image of f . Hence it is used as the inverse function of f whenever required. Given functions (whether partial or total) $f : A \rightarrow B$ and $g : B \rightarrow C$, their composition is the function $g \circ f : A \rightarrow C$ defined simply as the relational composition $\text{graph}(f); \text{graph}(g)$. Hence $(g \circ f)(a) = g(f(a))$.

Exercise -4.5: Operations on relations

1. Prove that $(\mathcal{R}^{-1})^{-1} = \mathcal{R}$ for any $\mathcal{R} \subseteq A \times B$.
2. Prove that for any relation $\mathcal{R} \subseteq A \times B$, $\mathcal{I}_A; \mathcal{R} = \mathcal{R} = \mathcal{R}; \mathcal{I}_B$.
3. Prove that relational composition is associative i.e. given relations $\mathcal{R} \subseteq A \times B$, $\mathcal{S} \subseteq B \times C$ and $\mathcal{T} \subseteq C \times D$, $\mathcal{R}; (\mathcal{S}; \mathcal{T}) = (\mathcal{R}; \mathcal{S}); \mathcal{T}$.
4. Prove that for any binary relations \mathcal{R} and \mathcal{S} on a set A ,
 - (a) $(\mathcal{R} \cap \mathcal{S})^{-1} = \mathcal{R}^{-1} \cap \mathcal{S}^{-1}$
 - (b) $(\mathcal{R} \cup \mathcal{S})^{-1} = \mathcal{R}^{-1} \cup \mathcal{S}^{-1}$
 - (c) $(\mathcal{R} - \mathcal{S})^{-1} = \mathcal{R}^{-1} - \mathcal{S}^{-1}$
5. Give examples of binary relations \mathcal{R}, \mathcal{S} on a set A to show that relational composition is not in general commutative.

The following is an important theorem with various applications in section -4.9.

Theorem -4.2: The Schroeder-Bernstein Theorem

Let A and B be sets and let $f : A \xrightarrow{1-1} B$ and $g : B \xrightarrow{1-1} A$ be injective functions. Then there exists a bijection between A and B .

Proof: Since f and g are both injective (1-1), they are both total functions, but their inverses may not be total. By injectivity (part 1 of definition -4.4), for any $a \in A$, $f(a) = b$ implies that b cannot be the image under f of any other member of A . Likewise for any $b \in B$, $g(b) \in A$ and for every other $b' \in B$ we have $g(b') \neq g(b)$. Hence $f^{-1} : B \rightharpoonup A$ and $g^{-1} : A \rightharpoonup B$ are both partial functions.

For any $a_0 \in A$ we define the origin of a_0 as a_0 itself if $g^{-1}(a_0)$ is undefined i.e. if a_0 is not the image of any $b \in B$ under g . (Likewise for any $b_0 \in B$, the origin of b_0 is b_0 itself if $f^{-1}(b_0)$ is undefined). Otherwise $g^{-1}(a_0) = b_1$ for a unique $b_1 \in B$. To define the origin of such an element a_0 we consider the maximal (possibly infinite) sequence of elements

$$\begin{array}{lll}
 & a_0 & \in A, \\
 g^{-1}(a_0) & = b_1 & \in B, \\
 f^{-1}(b_1) & = a_2 & \in A, \\
 g^{-1}(a_2) & = b_3 & \in B, \\
 \vdots & \vdots & \vdots, \\
 f^{-1}(b_{2k-1}) & = a_{2k} & \in A, \\
 g^{-1}(a_{2k}) & = b_{2k+1} & \in B, \\
 \vdots & \vdots & \vdots
 \end{array}$$

such that for each $k > 0$, $a_{2k} = f^{-1}(b_{2k-1})$ and $b_{2k+1} = g^{-1}(b_{2k})$. We then have the following cases for each $a_0 \in A$.

- Case A_A . a_{2m} is the origin of a_0 for some $m \geq 0$. That is, the sequence $a_0, b_1, a_2, b_3, \dots, a_{2m}$ is finite and $g^{-1}(a_{2m})$ is undefined. In this case a_{2m} is the origin of a_0 and $a_0 \in A_A$.
- Case A_B . b_{2m+1} is the origin of a_0 for some $m \geq 0$. That is, the sequence $a_0, b_1, a_2, b_3, \dots, a_{2m}, b_{2m+1}$ is finite and $f^{-1}(b_{2m+1})$ is undefined. Then b_{2m+1} is the origin of a_0 and $a_0 \in A_B$.
- Case A_U . The origin of a_0 is undefined. That is, the sequence $a_0, b_1, a_2, b_3, \dots, a_{2m}, b_{2m+1}, \dots$ is infinite. Then $a_0 \in A_U$.

Hence A may be partitioned into three (mutually disjoint) sets A_A, A_B, A_U depending upon the origins of the elements of A . (Analogously, B may be partitioned into B_A, B_B and B_U).

Now we may define the total function $h : A \longrightarrow B$ such that

$$h(a) = \begin{cases} f(a) & \text{if } a \in A_A \cup A_U \\ g^{-1}(a) & \text{if } a \in A_B \end{cases}$$

Claim. $h : A \xrightarrow[\text{onto}]{1-1} B$ i.e. h is a bijection from A to B .

⊤ The proof of the claim is easy and is left to the interested reader. In fact, we may show that the following hold

$$\begin{aligned} h(A_U) &= B_U , \quad h^{-1}(B_U) = A_U \\ h(A_A) &= B_A , \quad h^{-1}(B_A) = A_B \\ h(A_B) &= B_B , \quad h^{-1}(B_B) = A_B \end{aligned}$$

⊠

QED

Example -4.3: A bijection

We show using the Schroeder-Bernstein theorem -4.2 that there exists a bijection h between the sets $2^{\mathbb{P}}$ and the real closed-open interval $[0, 1)$.

We construct two injective mappings $f : 2^{\mathbb{P}} \xrightarrow{1-1} [0, 1)$ and $g : [0, 1) \xrightarrow{1-1} 2^{\mathbb{P}}$ as follows: For any $A \subseteq \mathbb{P}$ let $f(A) = 0.d_1d_2d_3\dots$ such that $d_i = 1$ if $i \in A$ and $d_i = 2$ otherwise. Clearly for every A there exists a unique image in $[0, 1)$ and no two distinct subsets of \mathbb{P} would have identical images. Hence f is injective, though not surjective.

To define g we consider only *normal binary* representations of real numbers. That is, we consider only binary representations which do not have an infinite sequence of trailing 1s, since any number of the form $0.b_1b_2\dots b_{i-1}\bar{1}$ equals the real number $0.b_1b_2\dots b_{i-1}\bar{0}$ which is normal. Every real number in $[1, 0)$ has a unique normal representation. Now consider the function defined by $g(0.b_1b_2b_3\dots) = \{i \in \mathbb{P} \mid b_i = 1\}$. g is clearly a well-defined function and it is injective as well. Again g is not surjective. However the Schroeder-Bernstein theorem -4.2 guarantees the existence of a bijection $h : 2^{\mathbb{P}} \longrightarrow [0, 1)$.

-4.5. Ordering Relations

We consider certain important kinds of binary relations that are subsets of $A \times A$ for any (nonempty) set A . These are the binary relations on a set A . In other words, we now consider $2^{A \times A}$ the powerset (see -4.1) of $A \times A$.

Definition -4.11: Binary relations on a set

A binary relation \mathcal{R} on a set A i.e. $\mathcal{R} \subseteq A \times A$ is

1. **reflexive** if and only if $\mathcal{I}_A \subseteq \mathcal{R}$.
2. **irreflexive** if and only if $\mathcal{I}_A \cap \mathcal{R} = \emptyset$.
3. **symmetric** if and only if $\mathcal{R} = \mathcal{R}^{-1}$.
4. **asymmetric** if and only if $\mathcal{R} \cap \mathcal{R}^{-1} = \emptyset$.
5. **antisymmetric** if and only if $\mathcal{R} \cap \mathcal{R}^{-1} \subseteq \mathcal{I}_{\text{Dom}\mathcal{R}}$.
6. **transitive** if and only if $\mathcal{R}; \mathcal{R} \subseteq \mathcal{R}$.
7. **connected** if and only if $(\text{Field}\mathcal{R} \times \text{Field}\mathcal{R}) - \mathcal{I}_{\text{Field}\mathcal{R}} \subseteq \mathcal{R} \cup \mathcal{R}^{-1}$.
8. **strongly connected** if and only if $(\text{Field}\mathcal{R} \times \text{Field}\mathcal{R}) = \mathcal{R} \cup \mathcal{R}^{-1}$.

Some of these definitions look very complicated. But the following exercise gives intuitive clarifications of some of them.

Exercise -4.6: Relations on a set

1. Show that a relation \mathcal{R} on a set A is
 - (a) antisymmetric if and only if for all a and b , $a \neq b$, $(a, b) \in \mathcal{R}$ implies $(b, a) \notin \mathcal{R}$.^a
 - (b) transitive if and only if for all $a, b, c \in A$, $(a, b), (b, c) \in \mathcal{R}$ implies $(a, c) \in \mathcal{R}$.
 - (c) connected if and only if for all $a, b \in A$, if $a \neq b$ then $a\mathcal{R}b$ or $b\mathcal{R}a$.
 - (d) strongly connected if and only if for all $a, b \in A$, $a\mathcal{R}b$ or $b\mathcal{R}a$.
2. Consider any reflexive relation \mathcal{R} on a set A . Does it necessarily follow that A is not asymmetric? Prove or disprove your answer.
3. If \mathcal{R} is asymmetric does it necessarily follow that it is irreflexive? Prove your answer.

^aAn equivalent definition used in most books is: \mathcal{R} is **antisymmetric** if and only if $(a, b), (b, a) \in \mathcal{R}$ implies $a = b$.

We may define the n -fold composition of a relation \mathcal{R} on a set A by induction as follows

$$\mathcal{R}^0 = \mathcal{I}_A$$

$$\mathcal{R}^{n+1} = \mathcal{R}^n; R$$

We may take the union these n -fold compositions to yield the **reflexive-transitive closure** of \mathcal{R} , denoted \mathcal{R}^* , as the relation

$$\mathcal{R}^* = \bigcup_{n \geq 0} \mathcal{R}^n \tag{1}$$

Sometimes it is also useful to consider merely the **transitive closure** \mathcal{R}^+ of \mathcal{R} which is defined as

$$\mathcal{R}^+ = \bigcup_{n>0} \mathcal{R}^n \quad (2)$$

Example -4.4: Example relations

1. The edge relation on an undirected graph is an example of a symmetric relation.
2. In any directed acyclic graph the edge relation is asymmetric.
3. Consider the reachability relation on a directed graph defined as: *A pair of vertices (A, B) is in the reachability relation, if either $A = B$ or there exists a vertex C such that both (A, C) and (C, B) are in the reachability relation.* The reachability relation is the reflexive transitive closure of the edge relation.
4. The reachability relation on directed graphs is also an example of a relation that need not be either symmetric or asymmetric. The relation need not be antisymmetric either.

Exercise -4.7: Relational closures

In the following assume \mathcal{R} is a binary relation on a non-empty set A .

1. Prove that \mathcal{R}^+ is transitive.
2. Prove that \mathcal{R}^* is reflexive and transitive.
3. Prove that if \mathcal{R} is reflexive then $\mathcal{R}^+ = \mathcal{R}^*$.
4. If \mathcal{R} is symmetric, is \mathcal{R}^* symmetric? Prove or disprove your answer.
5. We may define the *symmetric closure* of \mathcal{R} as the relation $\mathcal{R}^= = \mathcal{R} \cup \mathcal{R}^{-1}$.
 - (a) Prove that if \mathcal{R} is symmetric then $\mathcal{R} = \mathcal{R}^=$.
 - (b) Is $(\mathcal{R}^*)^= = (\mathcal{R}^=)^*$ always? Prove or disprove your answer.

-4.6. Partial Orders and Equivalences

Definition -4.12: Orders and equivalences

A binary relation \mathcal{R} on a set A is

1. a **preorder** if it is reflexive and transitive;
2. a **strict preorder** if it is irreflexive and transitive;
3. a **partial order** if it is an antisymmetric preorder;
4. a **strict partial order** if it is irreflexive, asymmetric and transitive;
5. a **linear order** or a **total order** if it is a connected partial order;
6. a **strict linear order** or a **strict total order** if it is connected, irreflexive and transitive;
7. an **equivalence** if it is reflexive, symmetric and transitive.

Fact -4.2: Preorders, partial order and equivalences

1. Every partial order is also a preorder.
2. Every strict partial order is also a strict preorder.
3. Every equivalence is a preorder.

Definition -4.13: Posets

A **partially ordered set**, or **poset** $\langle A, \leq \rangle$ consists of a set A together with a partial order relation \leq on A .

Fact -4.3: Partial order and strict partial order

1. If $\langle A, \preceq \rangle$ is a poset, then $\langle A, \prec \rangle$, where $\prec = \preceq - \mathcal{I}_A$, is a strict partial order.
2. If $\langle A, \prec \rangle$ is a strict partial order, then $\langle A, \preceq \rangle$, where $\preceq = \prec \cup \mathcal{I}_A$, is a poset.

By fact -4.3, any strict partial order $\langle A, \prec \rangle$ may be extended to a poset $\langle A, \preceq \rangle$ and any poset $\langle A, \preceq \rangle$ may be converted into a strict partial order $\langle A, \prec \rangle$.

Fact -4.4: Dual poset

If $\langle A, \leq \rangle$ is a poset, then so is $\langle A, \geq \rangle$ where $\geq = \leq^{-1}$.

Exercise -4.8: Lexicographic orderings

Let $\langle A, \preceq \rangle$ be a poset.

1. Prove that \prec_{lex2} on $A \times A$ defined by $(a, a') \prec_{lex2} (b, b')$ iff $[(a \prec b) \text{ or } (a = b \text{ and } a' \prec b')]$. Then $\langle A \times A, \prec_{lex} \rangle$ is a strict partial order.
2. Extend the definition of \prec_{lexn} to sets A^n for any $n \geq 0$ and show that it is a strict partial ordering on A^n and for $n = 2$, $\prec_{lexn} = \prec_{lex2}$
3. How will you extend \prec_{lexn} to a relation \prec_{lex} on A^* so that the following conditions hold.
 - (a) \prec_{lex} is a strict partial ordering relation on A^* ,
 - (b) For each $n \geq 0$ and $s, t \in A^n$, $s \prec_{lex} t$ iff $s \prec_{lexn} t$ and
 - (c) For any $s, t \in A^*$, if s is a proper prefix of t then $s \prec_{lex} t$.

Notation -4.9

Given a poset $\langle A, \leq \rangle$ and $a, b \in A$, we sometimes write

- $b \geq a$ to mean $a \leq b$,
- $a < b$ to mean $a \leq b$ and $a \neq b$ and
- $b > a$ to mean $a < b$.

Fact -4.5

If $\langle A, \leq \rangle$ is a poset, then $<$ and $>$ are strict partial orders.

For any set A (empty or non-empty) we have that $\langle 2^A, \subseteq \rangle$ is also a poset and in fact, the partial ordering relation \leq on A can be characterised (upto isomorphism) by the subset relation.

Definition -4.14: Order isomorphism

Given two posets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$, a function $f : A \longrightarrow B$ is said to be **order-preserving** if and only if for all $a, a' \in A$, $a \leq_A a'$ implies $f(a) \leq_B f(a')$. The two posets are said to be **(order-) isomorphic** if there exists an order-preserving bijection between them. We denote this fact by $\langle A, \leq_A \rangle \cong \langle B, \leq_B \rangle$.

Lemma -4.1: Poset isomorphism

For each poset $\langle A, \leq \rangle$ there exists a set $\mathcal{A} \subseteq 2^A$ such that $\langle A, \leq \rangle \cong \langle \mathcal{A}, \subseteq \rangle$.

Proof: For each $x \in A$ let $A_x = \{a \in A \mid a \leq x\}$. Define the set $\mathcal{A} = \{A_x \mid x \in A\} \subseteq 2^A$ and the function $f : A \longrightarrow \mathcal{A}$ such that $f(x) = A_x$ for each $x \in A$. It is easy to see that f is bijective and order-preserving i.e. for all $x, y \in A$, $x \leq y$ if and only if $A_x \subseteq A_y$. QED

Exercise -4.9: Preorders, partial orders and equivalences

1. Prove or disprove that for any nonempty relation \mathcal{R} on a nonempty set A ,
 - (a) $\mathcal{S} = \mathcal{R}^* \cup (\mathcal{R}^*)^{-1}$ is an equivalence relation,
 - (b) $\mathcal{T} = (\mathcal{R} \cup \mathcal{R}^{-1})^*$ is an equivalence relations.
 - (c) $\mathcal{S} = \mathcal{T}$.
2. Given any preorder \mathcal{R} on a set A , prove that the *kernel* of the preorder defined as $\mathcal{R} \cap \mathcal{R}^{-1}$ is an equivalence relation.
3. Consider any preorder \mathcal{R} on a set A . We give a construction of another relation as follows. For each $a \in A$, let $[a]_{\mathcal{R}}$ be the set defined as $[a]_{\mathcal{R}} = \{b \in A \mid a\mathcal{R}b \text{ and } b\mathcal{R}a\}$. Now consider the set $B = \{[a]_{\mathcal{R}} \mid a \in A\}$. Let \mathcal{S} be a relation on B such that for every $a, b \in A$, $[a]_{\mathcal{R}}\mathcal{S}[b]_{\mathcal{R}}$ if and only if $a\mathcal{R}b$. Prove that \mathcal{S} is a partial order on the set B .
4. Prove that a partially ordered set is totally ordered if and only if every one of its non-empty finite subsets is totally ordered.

-4.7. Equivalences induced by functions

We consider equivalences that may be induced by (partial) functions. This is of some importance in the semantics of languages.

Lemma -4.2: Equivalence induced by functions

Let $f : A \rightarrow B$ be a total function. Then the relation $=_f \subseteq A \times A$ defined by $a =_f a'$ if and only if $f(a) = f(a')$ is an equivalence relation.

Lemma -4.3: Weak equivalence induced by partial functions

Let $\mathcal{F}_{A'} = \{f \mid f : A \rightharpoonup B, A' \subseteq \text{Dom}(f) \subseteq A\}$ be a set of partial functions which are defined for every element of $A' \subseteq A$. Then the relation $=_{A'} \subseteq \mathcal{F}_{A'} \times \mathcal{F}_{A'}$ defined below is an equivalence relation on $\mathcal{F}_{A'}$.

$$f =_{A'} g \text{ iff for each } a' \in A', f(a') = g(a') \quad (3)$$

f and g are then said to be **weakly equivalent upto A'** .

Lemma -4.4: Strong equivalence induced by partial functions

Let $\mathcal{F} = \{f \mid f : A \rightharpoonup B\}$ be the set of partial functions from A to B . Then for any $A' \subseteq A$ the relation $\equiv'_A \subseteq \mathcal{F} \times \mathcal{F}$ defined below is an equivalence relation on $\mathcal{F}_{A'}$.

$$f \equiv_{A'} g \text{ iff for each } a' \in A', \text{ either both } f(a') \text{ and } g(a') \text{ are undefined or } f(a') = g(a'). \quad (4)$$

f and g are then said to be **strongly equivalent upto A'** .

The proofs of the above lemmata (-4.2, -4.3 and -4.4) are left as exercises for the interested reader.

Exercise -4.10: Ordering relations induced by functions

1. Prove lemma -4.2.
2. Prove lemma -4.3.
3. Prove lemma -4.4.
4. For any two sets A and B , $A \preceq B$ if there exists an injective function $f : A \xrightarrow{1-1} B$. Prove that \preceq is a preorder on any collection of sets.
5. Prove that any bijection between sets defines an equivalence relation on the collection of sets.

-4.8. Well-founded Sets and Well-ordered Sets

We discuss well-orders since an important induction principle (theorem -3.6) depends upon the notion of a well-ordering and generalises the principle of mathematical induction. In addition certain principles of proving the correctness of programs require well-foundedness.

Definition -4.15: Well-foundedness

Let $\langle A, \prec \rangle$ be a nonempty set A with an irreflexive, asymmetric binary relation $\prec \subseteq A \times A$. Let $\emptyset \neq B \subseteq A$. An element $b \in B$ is said to be **minimal** if there exists no $a \in B$ such that $a \prec b$. $\langle A, \prec \rangle$ is called **well-founded** if every nonempty subset of A has a minimal element. Equivalently we say that \prec is a **well-founded** relation on A .

Note that in general, \prec need not be transitive. The “immediate-predecessor-of” relation on the naturals given by $\prec_{pred} = \{(n, n + 1) \mid n \in \mathbb{N}\}$ is an example of an irreflexive, asymmetric and intransitive relation which is well-founded. However there are also transitive relations like $<$ on the naturals which are well-founded.

Fact -4.6

Every well-founded set has at least one minimal element.

Definition -4.16: Descending chain

Given a set $\langle A, \prec \rangle$ as in definition -4.15, an **infinite descending chain** is a nonempty subset of elements $\{a_i \in A \mid i \geq 0, a_{i+1} \prec a_i\}$. We normally write this as $a_0 \succ a_1 \succ a_2 \succ \dots$ where $\succ = \prec^{-1}$.

Lemma -4.5: Well-foundedness and descending chains

A set $\langle A, \prec \rangle$ is well-founded if and only if it has no infinite descending chain.

Proof:

(\Rightarrow). Assume $\langle A, \prec \rangle$ is well-founded and there is a an infinite descending chain $A' = \{a_i \in A \mid i \geq 0, a_i \succ a_{i+1}\} \subseteq A$. Clearly A' contains no minimal element, which is a contradiction.

(\Leftarrow). Assume there is no subset $A' = \{a_i \in A \mid i \geq 0, a_i \succ a_{i+1}\} \subseteq A$. If $\langle A, \prec \rangle$ is not well-founded, there exists a nonempty subset $B \subseteq A$ which has no minimal element. Consider any $b_0 \in B$. Since b_0 is not minimal there exists $b_1 \in B$ such that $b_0 \succ b_1$. Again b_1 is not minimal, so there must be a $b_2 \in B$ with $b_1 \succ b_2$. Proceeding in this fashion we find that for each $b_i \prec \dots \prec b_1 \prec b_0$, there exists a $b_{i+1} \in B$ such that $b_{i+1} \prec b_i \prec \dots \prec b_1 \prec b_0$. We may thus construct a set $B' = \{b_i \in B \mid i \geq 0, b_i \succ b_{i+1}\} \subseteq B \subseteq A$ which contradicts the assumption that there

is no such subset.

QED

The set $B' = \{b_i \in B \mid i \geq 0, b_i \succ b_{i+1}\}$ in the proof of lemma -4.5 is an example of an “*infinite descending chain*”

$$\cdots \prec b_{i+1} \prec b_i \prec \cdots \prec b_1 \prec b_0$$

We usually say that a well-founded set has no “*infinite descending chain*”.

We have already seen in definition -4.12 that a linear or total order is a connected partial order. We now use this and definition -4.15 to define well-ordered sets. In general, given a partial ordering \leq on a set with minimal elements we could use the irreflexive, asymmetric subset $<$ of the partial order defined by $x < y$ iff $x \leq y$ and $y \not\leq x$ to define a well-founded set wherever possible. For example, $<$ on the set \mathbb{N} is one such well-order. However, $<$ on the \mathbb{Z} is not a well-order because there are subsets of the integers which have no minimal element. Notice that the real interval $[0, 1]$ even though totally ordered and having a minimal element, is not well-founded because there are subsets which have infinite descending chains such as the open interval $(0, 1)$. Similarly for any two rational numbers $a < b$, the closed interval of rationals $[a, b]$ is also not well-founded.

Example -4.5: Lexicographic ordering of well-founded sets

Let $\langle A, \prec \rangle$ be a well founded strict partial order. Consider the *lexicographic ordering* \prec_{lex} on $A \times A$ defined by $(a, a') \prec_{lex} (b, b')$ iff $[(a < b) \text{ or } (a = b \text{ and } a' \prec b')]$. Then $\langle A \times A, \prec_{lex} \rangle$ is also well-founded.

Exercise -4.11: well-founded lexicographic ordering

Prove that $\langle A \times A, \prec_{lex} \rangle$ in example -4.5 is well-founded.

By fact -4.3 we simply claim a poset is well-founded if its strict version is well-founded. For instance we could claim that $\langle \mathbb{N}, \leq \rangle$ is well-founded.

Definition -4.17: Well-ordering

A **well-ordering** of a set is a well-founded total ordering of the set. A well-ordered set is sometimes called a **woset**.

Fact -4.7: Well-ordered sets

Let $\langle A, \prec \rangle$ be a (nonempty) well-ordered set.

1. Every nonempty subset of A has a unique least element.
2. A has a unique least element.

-4.9. Infinite Sets: Countability and Uncountability

Definition -4.18: Finite sets

A set is **finite** if it cannot be placed in bijection with any of its proper subsets.

However we will always use the following informal definition which is more useful for our purposes and it is more intuitive.

A set A is **finite** if it is either empty or can be placed in bijection with a set $n = \{0, \dots, n-1\}$ for some $n \in \mathbb{N}$.

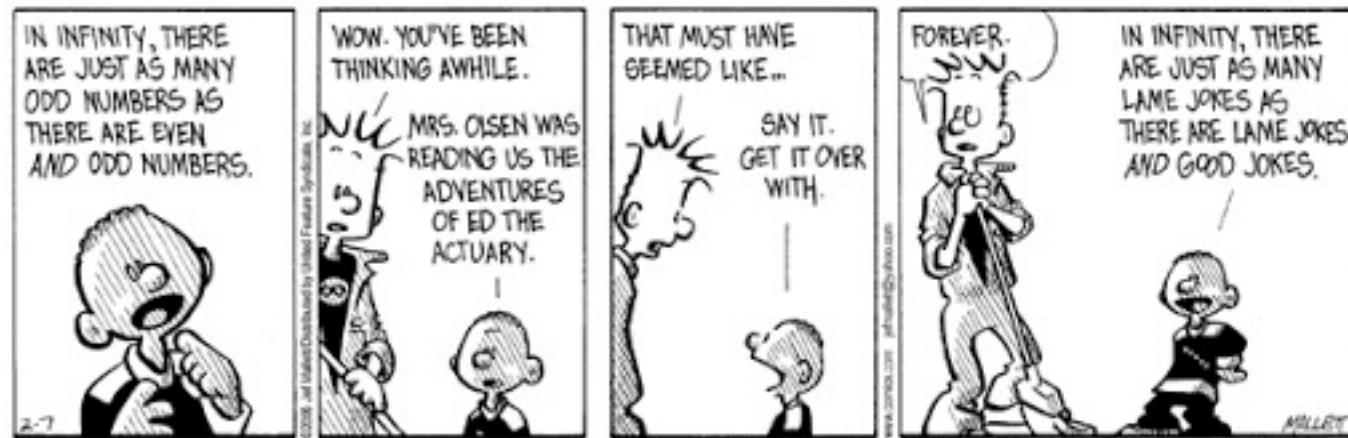
The above definition embodies the usual notion of counting.

Fact -4.8: Finite unions of finite sets

A finite union of finite sets is finite. That is, for any $n \in \mathbb{N}$, if $\{A_i \mid 0 \leq i < n, A_i \text{ is a finite set}\}$ is a finite collection of finite sets, then $A = \bigcup_{i=0}^{n-1} A_i$ is a finite set.

Exercise -4.12

1. Consider the set $\mathbb{2}$.
 - (a) How many 0-ary total functions can be defined on $\mathbb{2}$?
 - (b) How many unary total functions can be defined on $\mathbb{2}$?
 - (c) How many binary total functions can be defined on $\mathbb{2}$?
 - (d) How many k -ary ($k \geq 0$) total functions can be defined on $\mathbb{2}$? Does your answer coincide with your answers for $k = 0$, $k = 1$ and $k = 2$?
 - (e) How many k -ary ($k \geq 0$) partial functions can be defined on $\mathbb{2}$?
2. Let A and B be nonempty finite sets. What is the size of the following sets?
 - (a) $A \times B$.
 - (b) The set of all total functions from A to B .
 - (c) The set of all total functions from B to A .
 - (d) The set of all partial functions from A to B .



© UFS, Inc.

Definition -4.19: Infinite sets

A set A is called **infinite** if there exists a bijection between A and some proper subset of itself.

This definition begs the question, “If a set is not infinite, then is it necessarily finite?”. It turns out that indeed it is. Further it is also true that if a set is not finite then it can be placed in bijection with a proper subset of itself. But rigorous proofs of these statements are beyond the scope of this course and hence we shall not pursue them.

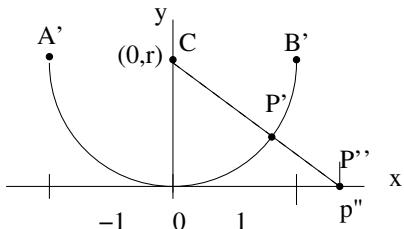


Figure 6: Bijection between the arc $\widehat{A'B'}$ and the real line

Example -4.6: Some infinite sets

We give appropriate bijections to show that various sets are infinite. In each case, note that the codomain of the bijection is a proper subset of the domain.

1. The set \mathbb{N} of natural numbers is infinite because we can define the 1-1 correspondence $p : \mathbb{N} \xrightarrow[\text{onto}]{1-1} \mathbb{P}$, with $p(m) \stackrel{df}{=} m + 1$.
2. The set E of even natural numbers is infinite because we have the bijection $e : E \xrightarrow[\text{onto}]{1-1} F$ where F is the set of all multiples of 4.
3. The set of odd natural numbers is infinite. (*Why?*)
4. The set \mathbb{Z} of integers is infinite because we have the following bijection $z : \mathbb{Z} \xrightarrow[\text{onto}]{1-1} \mathbb{N}$ by which the negative integers have unique images among the odd numbers and the non-negative integers have unique images among the even numbers. More specifically,

$$z(m) = \begin{cases} 2m & \text{if } m \in \mathbb{N} \\ -2m - 1 & \text{otherwise} \end{cases}$$

Example -4.7: Infinitude of \mathbb{R}

The set \mathbb{R} of reals is infinite. We outline the proof by considering the nonempty open interval $(a, b) = \{p \mid a < p < b\}$ and use figure 6 as a guide to understand the mapping.

Take any line-segment \overline{AB} of length $b - a \neq 0$ and “bend” it into the semi-circle $\widehat{A'B'}$ and place it tangent to the x -axis at the point $(0, 0)$ (as shown in the figure). The bijection between the points on the semi-circle and the real numbers p , $a < p < b$ is “obvious”^a. This semicircle has a radius $r = \frac{b-a}{\pi}$. The centre C of this semi-circle is then located at the point $(0, r)$ on the 2-dimensional plane.

Consider an arbitrary point P' on the semi-circle, which corresponds to a real number p , $a < p < b$. The ray $\overrightarrow{CP'}$ intersects the x -axis at some point P'' which has the coordinates $(p'', 0)$. Since $A' \neq P' \neq B'$, the ray cannot be parallel to the x -axis. Similarly from every point P'' on the x -axis there exists a unique point P' on the semi-circle such that C , P' and P'' are collinear. Each point P' such that $A' \neq P' \neq B'$ on this semi-circle corresponds exactly to a unique real number p in the open interval (a, b) and vice-versa. Hence there exists a 1-1 correspondence between the points on the semicircle (excluding the end-points of the semi-circle) and those on the x -axis. Let p'' be the x -coordinate of the point P'' . Since the composition of bijections is a bijection (see problem 1 in exercise -4.3), we may compose all these bijections to obtain a 1-1 correspondence between each p in the interval (a, b) and the real numbers.

^aMany of the logical problems with proofs in mathematics also occur because of the use of loose and improperly defined terms such as “bend” and “obvious” even though their meaning might appear to be intuitively clear. A more precise statement might be that for each line segment \overline{AB} of non-zero length, there exists a semicircle $\widehat{A'B'}$ (with radius $\frac{|AB|}{\pi}$) of the same length with the “obvious” 1-1 correspondence between them which maps any point P on \overline{AB} with $0 < AP < b - a$ to a point P' on $\widehat{A'B'}$ such that the length of the arc $\widehat{A'P'}$ is AP . Words like “obvious” and “trivial” are also standard terms used by students in examinations when they know fully well that there are gaps in their proof which they are unable to fill satisfactorily.

Definition -4.20: Countable sets

An infinite set is said to be **countable** (or **countably infinite** or **denumerable**) if it can be placed in bijection with the set \mathbb{P} . Otherwise, it is said to be **uncountable**.

The above definition essentially says that a countably infinite set may be enumerated by selecting a unique “first element”, a unique “second” element and so on. Countability of an infinite set therefore implies that for any positive integer n , it should be possible to obtain the unique designated n -th element from the set and also for any element in the set, it should be possible to obtain its position in the enumeration. The following are easy to prove.

Fact -4.9: Countability

1. An infinite set A is countable if and only if there is a bijection between A and \mathbb{N} .
2. Every infinite subset of \mathbb{N} is countable.
3. If A is a finite set and B is a countable set, then $A \cup B$ is countable.
4. If A and B are countable sets, then $A \cup B$ is also countable.

Definition -4.21: At most countable sets

A set is **at most countable** if it is finite or countable.

The following is an interesting characterisation of at most countable sets.

Theorem -4.3: Equivalents of at most countability

The following statements are equivalent for any nonempty set A .

1. A is at most countable.
2. There is a surjective map $f : \mathbb{N} \xrightarrow{\text{onto}} A$.
3. There is an injective map $g : A \xrightarrow{1-1} \mathbb{N}$.

Proof: We prove it by showing that

1. statement 1 implies statement 2
 2. statement 2 implies statement 3
 3. statement 3 implies statement 1
1. statement 1 implies statement 2. If A is countable it is clear that there exists a bijection $f : \mathbb{N} \xrightarrow{\text{onto}} A$ which is surjective. On the other hand if $A = \{a_0, \dots, a_{n-1}\}$ is finite and has $n > 0$ elements then the function $f : \mathbb{N} \longrightarrow A$ such that $f(m) = \begin{cases} a_m & \text{if } 0 \leq m < n \\ a_0 & \text{if } m \geq n \end{cases}$ is clearly surjective.
 2. statement 2 implies statement 3. Let $f : \mathbb{N} \xrightarrow{\text{onto}} A$ be surjective. We then define the following injective function $g : A \xrightarrow{1-1} \mathbb{N}$ such that $g(a) = \text{the least } m \text{ such that } f(m) = a$.
 3. statement 3 implies statement 1. Let $g : A \xrightarrow{1-1} \mathbb{N}$ be an injective map. We may use this injective map to well-order the elements of A , so that $a \sqsubset b$ if and only if $g(a) < g(b)$. If A is finite there is nothing to prove. If it is not finite then let $A_0 = A - \emptyset = A$ has a (unique) least element

a_0 under the ordering \sqsubseteq . For each positive integer $j > 0$, let $A_j = A - \{a_0, \dots, a_{j-1}\} \subseteq A$, let a_j be the (unique) least element of A_j . The function $h : a_j \mapsto j$ is a bijection between A and \mathbb{N} , showing that A is countable.

QED

Corollary -4.1: Subsets of at most countable sets

1. Any subset of a finite set is finite.
2. Any subset of an at most countable set is at most countable.
3. Any subset of a countable set is at most countable.

There are of course various kinds of infinitudes, of which countability is one. $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ all turn out to be only countable. By contrast the set \mathbb{R} is uncountable as we shall see. In fact it turns out that the set of irrational numbers is also uncountable and hence is (by sheer abuse of language) “more infinite” than \mathbb{Q} while being (again by abuse of language) “only as infinite as” the set \mathbb{R} .

The interesting question is where do sets such as $\mathbb{N}^2, \mathbb{N}^*$ lie in this spectrum of infinitude.

Theorem -4.4: Countability of \mathbb{N}^2

\mathbb{N}^2 is a countable set.

Proof:

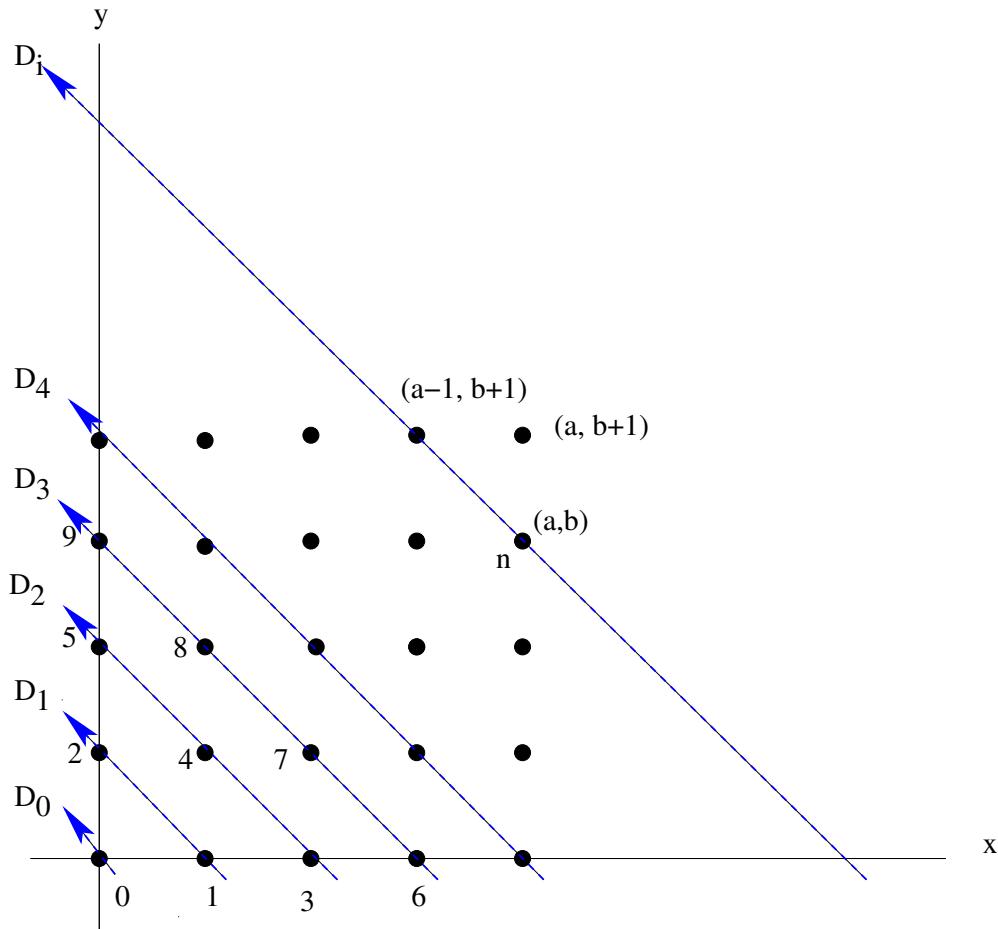


Figure 7: Counting “lattice-points” on the “diagonals”

We show that \mathbb{N}^2 is countably infinite by devising a way to order the elements of \mathbb{N}^2 which guarantees that there is indeed a 1-1 correspondence between the ordered pairs of naturals and the naturals themselves. For instance, an obvious ordering such as

(0, 0)	(0, 1)	(0, 2)	(0, 3)	...
(1, 0)	(1, 1)	(1, 2)	(1, 3)	...
(2, 0)	(2, 1)	(2, 2)	(2, 3)	...
⋮	⋮	⋮	⋮	⋮

is not such a 1-1 correspondence because we cannot answer the following questions with (unique) answers.

1. *What is the n -th element in the ordering for any given n ?*
2. *What is the position in the ordering of the pair (a, b) for arbitrary naturals a and b ?*

So it is necessary to construct a more rigorous and ingenious device to ensure a bijection. So we consider the ordering implicitly defined in figure 7. By traversing the blue rays $\overrightarrow{D_0}, \overrightarrow{D_1}, \overrightarrow{D_2}, \dots$ in order, we get an obvious ordering on the elements of \mathbb{N}^2 . However it should be possible to give unique answers to the above questions.

Claim. $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ defined by $f(a, b) = \frac{(a+b)(a+b+1) + 2b}{2}$ is the required bijection.

⊤ The function f defines essentially the traversal of the rays $\overrightarrow{D_0}, \overrightarrow{D_1}, \overrightarrow{D_2}, \dots$ in order as we shall prove. It is easy to verify that $\overrightarrow{D_0}$ contains only the pair $(0, 0)$ and $f(0, 0) = 0$. Now consider any pair $(a, b) \neq (0, 0)$. If (a, b) lies on the ray $\overrightarrow{D_i}$, then it is clear that $i = a + b$. Now consider all the pairs that lie on the rays $\overrightarrow{D_0}, \overrightarrow{D_1}, \dots, \overrightarrow{D_{i-1}}$ ³

³Under the usual (x, y) coordinate system, these are all the *lattice points* on and inside the right triangle defined by the three points $(i-1, 0), (0, 0)$ and $(0, i-1)$. A *lattice point* in the (x, y) -plane is a point whose x - and y - coordinates are both integers.

The number of such pairs is given by the “triangular number”

$$i + (i - 1) + (i - 2) + \dots + 1 = \frac{i(i + 1)}{2}$$

Since we started counting from 0 this number is also the value of the lattice point $(i, 0)$ under the function f . This brings us to the starting point of the ray D_i and after crossing b lattice points along the ray D_i we arrive at the point (a, b) . Hence

$$\begin{aligned} f(a, b) &= \frac{i(i + 1)}{2} + b \\ &= \frac{(a + b)(a + b + 1) + 2b}{2} \end{aligned}$$

⊣

We leave it as an exercise to the reader to define the inverse of this function. (*Hint: Use “triangular numbers”!*)

QED

Exercise -4.13: Tuples and sequences of naturals

1. Using theorem -4.4 prove that \mathbb{N}^3 and \mathbb{N}^4 are also countable sets.
2. Prove that \mathbb{N}^n , for any $n > 0$ is a countable set,

Theorem -4.5: Countable union of countable sets

The countable union of countable sets is countable, i.e. given a family $\mathcal{A} = \{A_i \mid A_i \text{ is countable}, i \in \mathbb{N}\}$ of countable sets, their union $A_\infty = \bigcup_{i \in \mathbb{N}} A_i$ is also countable.

Proof: For simplicity we assume that the sets are all pairwise disjoint i.e. $A_i \cap A_j = \emptyset$ for each $i \neq j$. Hence for each element $a \in A_\infty$, there exists a unique $i \in \mathbb{N}$ such that $a \in A_i$. This implies there exists a bijection $h : A_\infty \xrightarrow[\text{onto}]{1-1} \{(i, a) \mid a \in A_i, i \in \mathbb{N}\}$. Since each A_i is countable, there exists a bijection $f_i : A_i \xrightarrow[\text{onto}]{1-1} \mathbb{N}$ for each $i \in \mathbb{N}$. Define the bijection $g : A_\infty \xrightarrow[\text{onto}]{1-1} \mathbb{N}^2$ such that $g(a) = (i, f_i(a))$. By theorem -4.4 it follows that A_∞ is countable. QED

Exercise -4.14: Countability

1. Suppose the sets $A_i \in \mathcal{A}$ are not pair-wise disjoint. Is their union still countable? Prove or disprove.
2. Prove that \mathbb{N}^* the set of all finite sequences of natural numbers is countable.
3. Prove that the set of all infinite sequences of natural numbers is uncountable.
4. Prove that the set of total functions $f : \mathbb{N} \longrightarrow \mathbb{N}$ is uncountable.

Example -4.8: Logic of negation and implication

Let the language \mathcal{M}_0 be “generated” by the following process from a countably infinite set of “atoms” \mathbb{A}^a , such that \mathbb{A} does not contain any of the symbols \neg , \rightarrow , (and).

1. $\mathbb{A} \subseteq \mathcal{M}_0$,
2. If μ and ν are any two elements of \mathcal{M}_0 then $(\neg\mu)$ and $(\mu \rightarrow \nu)$ also belong to \mathcal{M}_0 , and
3. No string other than those obtained by a finite number of applications of the above rules belongs to \mathcal{M}_0 .

There are at least two possible ways of proving that \mathcal{M}_0 is countable. The first one simply encodes formulas into unique natural numbers. The second uses induction on the structure of formulas and the fact that a countable union of countable sets yields a countable set. We postpone the second proof to the chapter on induction.

^aWe use this colour to distinguish strings from an alphabet or terms from a language from the normal words used in our descriptions.

Solution There are at least two possible proofs. The first one simply encodes formulas into unique natural numbers. The second uses induction on the structure of formulas and the fact that a countable union of countable sets yields a countable set. We postpone the second proof to the chapter on induction. So here goes!

Proof: Since \mathbb{A} is countably infinite, there exists a $1 - 1$ correspondence $ord : \mathbb{A} \rightarrow \mathbb{P}$ which uniquely enumerates the atoms in some order. This function may be extended to a function ord' which includes the symbols “ \neg ”, “ $($ ”, “ $)$ ”, “ \rightarrow ”, such that $ord'(\neg) = 1$, $ord'(() = 2$, $ord'(\") = 3$, $ord'(\rightarrow) = 4$, and $ord'(A) = ord(A) + 4$, for every $A \in \mathbb{A}$. Let $Syms = \mathbb{A} \cup \{\neg, (,), \rightarrow\}$. Clearly $ord' : Syms \rightarrow \mathbb{P}$ is also a $1 - 1$ correspondence. Hence there also exist inverse functions ord^{-1} and ord'^{-1} which for any positive integer identify a unique symbol from the domains of the two functions respectively.

Now consider any string⁴ belonging to $Syms^*$. It is possible to assign a unique positive integer to this string by using powers of primes. Let $p_1 = 2$,

⁴This includes even arbitrary strings which are not part of the language. For example, you may have strings such as “ $)\neg($ ”.

$p_2 = 3, \dots, p_i, \dots$ be the infinite list of primes in increasing order. Let the function $\text{encode} : \text{Syms}^* \rightarrow \mathbb{P}$ be defined by induction on the lengths of the strings in Syms^* , as follows. Assume $s \in \text{Syms}^*$, $a \in \text{Syms}$ and “” denotes the empty string.

$$\begin{aligned}\text{encode}(“”) &= 1 \\ \text{encode}(sa) &= \text{encode}(s) \times p_m^{\text{ord}'(a)}\end{aligned}$$

where s is a string of length $m - 1$ for $m \geq 1$.

It is now obvious from the unique prime-factorization of positive integers that every string in Syms^* has a unique positive integer as its “encoding” and from any positive integer it is possible to get the unique string that it represents. Hence Syms^* is a countably infinite set. Since the language of minimal logic is a subset of the Syms^* it cannot be an *uncountable*. Hence there are only two possibilities: either it is finite or it is countably infinite.

Claim. The language of minimal logic is not finite.

⊤ Suppose the language were finite. Then there exists a formula ϕ in the language such that $\text{encode}(\phi)$ is the maximum possible positive integer. This $\phi \in \text{Syms}^*$ and hence is a string of the form $a_1 \dots a_m$ where each $a_i \in \text{Syms}$. Clearly

$$\text{encode}(\phi) = \prod_{i=1}^m p_i^{\text{ord}'(a_i)}$$

Now consider the longer formula $\psi = (\neg\phi)$. It is easy to show that

$$\text{encode}(\psi) = 2^{\text{ord}'(“(“)} \times 3^{\text{ord}'(“-“)} \times \prod_{i=1}^m p_{i+2}^{\text{ord}'(a_i)} \times p_{m+3}^{\text{ord}'(“)”)}$$

and $\text{encode}(\psi) > \text{encode}(\phi)$ contradicting the assumption of the claim. ⊢

Hence the language is countably infinite.

QED

We have seen from theorem -4.4 that the set \mathbb{N}^2 is countable. It is easy to show that for any $n \geq 2$, \mathbb{N}^n is also countable and by theorem -4.5 it follows that the set \mathbb{N}^* of all finite length sequences of naturals is also countable.

However, not all infinite sets are countable. In other words even among infinite sets there are some sets that are “more infinite than others”. The following theorem and the form of its proof was first given by Georg Cantor and has been used to prove several results in logic, mathematics and computer science. The theorem embodies an important proof technique called “diagonalization”.

Theorem -4.6: Cantor's diagonalization

The powerset of \mathbb{N} (i.e. $\mathcal{P}(\mathbb{N})$, the set of all subsets of \mathbb{N}) is an uncountable set.

Proof: Firstly, it should be clear that $\mathcal{P}(\mathbb{N})$ is not a finite set, since it can be placed in bijection with $\mathcal{P}(\mathbb{P})$ which is a proper subset of $\mathcal{P}(\mathbb{N})$. A simple bijection extends the mapping $i \mapsto i + 1$ to sets in $\mathcal{P}(\mathbb{N})$ i.e. define $f : \mathcal{P}(\mathbb{N}) \xrightarrow[\text{onto}]{1-1} \mathcal{P}(\mathbb{P})$ such that for each $A \in \mathcal{P}(\mathbb{N})$, $f(A) = \{a + 1 \mid a \in A\}$.

Consider any subset $A \subseteq \mathbb{N}$. We may represent this set as an infinite sequence σ_A composed of 0's and 1's such that $\sigma_A(i) = 1$ if $i \in A$, otherwise $\sigma_A(i) = 0$. Let $\Sigma = \{\sigma_A \mid A \subseteq \mathbb{N}, \text{for each } i \in \mathbb{N}, \sigma_A(i) \in \{0, 1\}\}$ be the set of all such sequences. It is easy to show that there exists a bijection $g : \mathcal{P}(\mathbb{N}) \xrightarrow[\text{onto}]{1-1} \Sigma$ such that $g(A) = \sigma_A$, for each $A \subseteq \mathbb{N}$. Clearly, therefore $\mathcal{P}(\mathbb{N})$ is countable if and only if Σ is countable. Hence, if there exists a bijection $f : \Sigma \xrightarrow[\text{onto}]{1-1} \mathbb{N}$, then $g \circ f$ is the required bijection from $\mathcal{P}(\mathbb{N})$ to \mathbb{N} . On the other hand, if there is no bijection f then $\mathcal{P}(\mathbb{N})$ is uncountable if and only if Σ is uncountable. We make the following claim which we prove by Cantor's diagonalization.

Claim. The set Σ is uncountable.

↪ Suppose Σ is countable then there exists a bijection $h : \mathbb{N} \xrightarrow[\text{onto}]{1-1} \Sigma$. In fact let $h(i) = \sigma_i \in \Sigma$, for each $i \in \mathbb{N}$. Now consider the sequence

ρ constructed in such a manner that for each $i \in \mathbb{N}$, $\rho(i) \neq \sigma_i(i)$. In other words,

$$\rho(i) = \begin{cases} 0 & \text{if } \sigma_i(i) = 1 \\ 1 & \text{if } \sigma_i(i) = 0 \end{cases}$$

Since ρ is an infinite sequence of 0's and 1's, $\rho \in \Sigma$. But from the above construction it follows that since ρ is different from every sequence in Σ it cannot be a member of Σ , leading to a contradiction. Hence the assumption that the bijection h exists is wrong. Hence the assumption that Σ is countable must be wrong. \dashv

QED

It is possible to generalize the above theorem and the proof to all powersets as follows.

Theorem -4.7: The Powerset theorem

There is no 1-1 correspondence between a set and its powerset.

Proof: Let A be any set and let \mathcal{P}^A be its powerset. Assume that $g : A \xrightarrow[\text{onto}]{} \mathcal{P}^A$ is a 1-1 correspondence between A and \mathcal{P}^A . This implies for every $a \in A$, $g(a) \subseteq A$ is uniquely determined and further for each $B \subseteq A$, $g^{-1}(B)$ exists and is uniquely determined.

For any $a \in A$, a is called an *interior* member if $a \in g(a)$ and otherwise a is an *exterior* member. Consider the set

$$X = \{x \in A \mid x \notin g(x)\}$$

which consists of exactly the exterior members of A . Since g is a 1-1 correspondence, there exists a unique $x \in A$ such that $X = g(x)$. Note that X could be the empty set.

x is either an interior member or an exterior member. If x is an interior member then $x \in g(x) = X$ which contradicts the assumption that X contains

only exterior members. If x is an exterior member then $x \notin g(x) = X$. But then since x is an exterior member $x \in X$, which is a contradiction. Hence the assumption that there exists a 1-1 correspondence g between A and 2^A must be false. QED

Exercise -4.15: Uncountability

1. Prove that \mathbb{N}^ω the set of all *infinite* sequences of natural numbers is uncountable.
2. Prove that the set of all binary relations on a countable set is uncountable.
3. Prove that the set of all total functions from \mathbb{N} to \mathbb{N} is uncountable.
4. Prove that there exists a bijection between the set $2^{\mathbb{N}}$ and the open interval $(0, 1)$ of real numbers. What can you conclude about the cardinality of the set $2^{\mathbb{N}}$ in relation to the set \mathbb{R} ?

Example -4.9: Uncountability of Power set of \mathbb{P}

By the bijection h defined in example -4.3 it follows that that $2^{\mathbb{P}}$ is an uncountable set. Further since $2^{\mathbb{P}} \subset 2^{\mathbb{N}}$ it confirms that $2^{\mathbb{N}}$ is an uncountable set.

Exercise -4.16: Uncountables

1. Prove that the set \mathbb{R} is uncountable.
2. Prove that every real interval of the form (a, b) , $[a, b)$, $(a, b]$, $[a, b]$ where $a < b$ is uncountable.
3. Prove that the set \mathbb{C} of complex numbers is uncountable. Can you define a bijection between \mathbb{C} and \mathbb{R} ?

So whereas \mathbb{N} is countable, $2^{\mathbb{N}}$ is uncountable. There is a hierarchy of infinitudes defined by the Powerset theorem ([-4.7](#)) which shows, loosely speaking, that $2^{2^{\mathbb{N}}}$ is “even more uncountable” (again by abuse of language) than $2^{\mathbb{N}}$. Similarly $2^{\mathbb{R}}$ is “more uncountable” than \mathbb{R} .

-3. Induction Principles

Theorem: All natural numbers are equal.

Proof: Given a pair of natural numbers a and b , we prove they are equal by performing complete induction on the maximum of a and b (denoted $\max(a, b)$).

Basis. For all natural numbers less than or equal to 0, the claim holds.

Induction hypothesis. For any a and b such that $\max(a, b) \leq k$, for some natural $k \geq 0$, $a = b$.

Induction step. Let a and b be naturals such that $\max(a, b) = k + 1$. It follows that $\max(a - 1, b - 1) = k$. By the induction hypothesis $a - 1 = b - 1$. Adding 1 on both sides we get $a = b$

QED.

Fortune cookie on Linux

-3.1. Mathematical Induction

Anyone who has had a good background in school mathematics must be familiar with two uses of induction.

1. definition of functions and relations by mathematical induction, and
2. proofs by the principle of mathematical induction.

Example -3.1: Definitions by induction

We present below some familiar examples of definitions by mathematical induction.

1. The factorial function on natural numbers is defined as follows.

Basis. $0! = 1$

Induction step. $(n + 1)! = n! \times (n + 1)$

2. The n -th power (where n is a natural number) of a real number x is often defined as

Basis. $x^1 = x$

Induction step. $x^{n+1} = x^n \times x$

3. The n -fold composition of a binary relation \mathcal{R} on a set A may be defined as follows.

Basis. $\mathcal{R}^1 = \mathcal{R}$

Induction step. $\mathcal{R}^{n+1} = \mathcal{R}; \mathcal{R}^n$. Equivalently we have $\mathcal{R}^{n+1} = \mathcal{R}^n; \mathcal{R}$

Similarly the principle of mathematical induction is the means by which we have often *proved* (as opposed to *defining*) properties about numbers, or statements involving the natural numbers. There are several versions of this principle and we will consider a few of them that are important. Often it is just a matter of taste or convenience which version is used. As we shall see they are all mutually equivalent. One version of the principle may be stated as follows.

Principle of Mathematical Induction – Version 1 (PMI1)

A property \mathbb{P} holds for all natural numbers provided

Basis. \mathbb{P} holds for 0, and

Induction step. For arbitrarily chosen $n > 0$,

\mathbb{P} holds for $n - 1$ implies \mathbb{P} holds for n .

The underlined portion, called the **induction hypothesis**, is an assumption that is necessary for the conclusion to be proved. Intuitively, the principle captures the fact that in order to prove that a property holds for all natural numbers, it suffices to do it in two steps. The first step is the basis and needs to be proved. The proof of the induction step essentially tells us that the reasoning involved in proving the statement for all other natural numbers is the same. Hence instead of an infinitary proof (one for each natural number) we have a compact finitary proof which exploits the similarity of the proofs for all the naturals except the basis. Alternatively the principle could be expressed as follows, by substituting " $n \geq 0$ " for " $n > 0$ ", " P holds for n " for " P holds for $n - 1$ " and " P holds for $n + 1$ " for " P holds for n " in the induction step.

Normally this principle is merely stated and applied to various problems in school mathematics, somehow giving the impression that it needs to be taken for granted. However we shall actually prove it.

Theorem -3.1: The Principle of Mathematical Induction Version 1 (PMI1)

A property \mathbb{P} holds for all natural numbers provided

Basis. \mathbb{P} holds for 0, and

Induction step. For arbitrarily chosen $n > 0$,

\mathbb{P} holds for $n - 1$ implies \mathbb{P} holds for n .

Proof: We give a proof by contradiction. Let \mathbb{P} be a property of natural numbers such that both the basis and the induction step hold but \mathbb{P} does not hold for all natural numbers. Since the naturals are well-ordered under the ordering $<$, there must be a smallest natural number n for which the property \mathbb{P} does not hold. Clearly $n \neq 0$ since by assumption \mathbb{P} holds for 0. Hence $n > 0$. However \mathbb{P} does hold for $n - 1$ since $n > 0$ is the smallest natural for which \mathbb{P} does not hold. By the induction step it follows that if \mathbb{P} holds for $n - 1$, \mathbb{P} must also hold for n , which contradicts the assumption that \mathbb{P} does not hold for n . QED

Exercise -3.1: Use of PMI1

Prove the following using PMI1.

1. The powerset 2^A of a finite set A of $n \geq 0$ elements has 2^n subsets.
2. $(a + b)^n = \sum_{k=0}^n {}^n C_k a^{n-k} b^k$ where ${}^n C_k$ denotes the number of combinations of n objects taken k at a time.
3. The sum of the first n odd numbers is n^2 .

There are other versions we shall consider now, but shall not prove any since the proofs are similar and can be handled by the intelligent reader. Here is the first variation which is most commonly used in proofs.

Principle of Mathematical Induction – Version 1' (PMI1')

A property \mathbb{P} holds for all natural numbers provided

Basis. \mathbb{P} holds for 0, and

Induction step. For arbitrarily chosen $n \geq 0$,

\mathbb{P} holds for n implies \mathbb{P} holds for $n + 1$.

Example -3.2

We prove that all natural numbers of the form $n^3 + 2n$ are divisible by 3.

Proof:

Basis. For $n = 0$, we have $n^3 + 2n = 0$ which is divisible by 3.

Induction step. Assume for an arbitrarily chosen $n \geq 0$, $n^3 + 2n$ is divisible by 3. Now consider $(n + 1)^3 + 2(n + 1)$. We have

$$\begin{aligned}(n + 1)^3 + 2(n + 1) &= (n^3 + 3n^2 + 3n + 1) + (2n + 2) \\ &= (n^3 + 2n) + 3(n^2 + n + 1)\end{aligned}$$

which clearly is divisible by 3.

QED

Several versions of this principle exist. We state some of the most important ones. In such cases, the underlined portion is the induction hypothesis. For example it is not necessary to consider 0 (or even 1) as the basis step. Any integer k could be considered the basis, as long as the property is to be proved for all $n \geq k$.

Principle of Mathematical Induction – Version 2

A property \mathfrak{P} holds for all natural numbers $n \geq k$ for some natural number k , provided

Basis. \mathfrak{P} holds for k , and

Induction step. For arbitrarily chosen $n > k$,
 \mathbb{P} holds for $n - 1$ implies \mathbb{P} holds for n .

Such a version seems very useful when the property to be proved is either not true or is undefined for all naturals less than k . The following example illustrates this.

Example -3.3

Every positive integer $n \geq 8$ is expressible as $n = 3i + 5j$ where $i, j \geq 0$.

Proof:

Basis. For $n = 8$, we have $n = 3 + 5$, i.e. $i = j = 1$.

Induction step. Assuming for an arbitrary $n \geq 8$, $n = 3i + 5j$ for some naturals i and j . Now consider $n + 1$. If $j = 0$ then clearly $i \geq 3$ and we may write $n + 1$ as $3(i - 3) + 5(j + 2)$. Otherwise $n + 1 = 3(i + 2) + 5(j - 1)$.

QED

However it is not necessary to have this new version of the Principle at all as the following reworking of the previous example shows.

Example -3.4

The property of the previous example could be equivalently reworded as follows. “*For every natural number n , $n + 8$ is expressible as $n + 8 = 3i + 5j$ where $i, j \geq 0$.*”

Proof: .

Basis. For $n = 0$, we have $n + 8 = 8 = 3 + 5$, i.e. $i = j = 1$.

Induction step. Assuming for an arbitrary $n \geq 0$, $n + 8 = 3i + 5j$ for some naturals i and j , consider $n + 1$. If $j = 0$ then clearly $i \geq 3$ and we may write $(n + 1) + 8$ as $3(i - 3) + 5(j + 2)$. Otherwise $(n + 1) + 8 = 3(i + 2) + 5(j - 1)$.

QED

In general any property \mathfrak{P} that holds for all naturals greater than or equal to some given k may be transformed equivalently into a property \mathfrak{Q} , which reads exactly like \mathfrak{P} except that all occurrences of “ n ” in \mathfrak{P} are systematically replaced by “ $n + k$ ”. We may then prove the property \mathfrak{Q} using the **PMI1** version of the principle.

What we have stated above informally is, in fact a proof outline of the following theorem.

Theorem -3.2: Equivalence of PMI1 and PMI2

The two principles of mathematical induction are equivalent. In other words, every application of **PMI – version 1** may be transformed into an application of **PMI – version 2** and vice-versa.

In the sequel we will assume that the principle of mathematical induction always refers to the first versions (**PMI1** or **PMI1'**).

Exercise -3.2

1. Find the fallacy in the proof by PMI of the following alleged theorem.

Theorem:

$$\frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \dots + \frac{1}{(n-1)n} = \frac{3}{2} - \frac{1}{n}$$

Proof: For $n = 1$ the LHS is $\frac{1}{2}$ and so is RHS. Assume the theorem is true for some $n \geq 1$. We then prove the induction step for $n + 1$.

$$\begin{aligned} & \text{LHS} \\ &= \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \dots + \frac{1}{(n-1)n} + \frac{1}{n(n+1)} \\ &= \frac{3}{2} - \frac{1}{n} + \frac{1}{n(n+1)} \\ &= \frac{3}{2} - \frac{1}{n} + \frac{1}{n} - \frac{1}{(n+1)} \\ &= \frac{3}{2} - \frac{1}{n+1} \end{aligned}$$

which was required to be proved.

2. Rectify the fallacy in problem 1 and prove it using PMI.
3. Let A be any set with a reflexive and transitive binary relation \preceq defined on it. That is to say, $\preceq \subseteq A \times A$ satisfies the following conditions.
- For every $a \in A$, $a \preceq a$.
 - For all a, b, c in A , $a \preceq b$ and $b \preceq c$ implies $a \preceq c$.

Then show by induction that $\preceq; R = \preceq^n; R$ for all $n \geq 1$.

-3.2. Complete Induction

Often in inductive definitions and proofs it seems necessary to work with an inductive hypothesis that includes not just the predecessor of a natural number, but some or all of their predecessors as well.

Example -3.5: Fibonacci

The definition of the following sequence is a case of precisely such a definition where the function $F(n)$ is defined for all naturals as follows.

Basis. $F(0) = 0$

Induction step

$$F(n + 1) = \begin{cases} 1 & \text{if } n = 0 \\ F(n) + F(n - 1) & \text{otherwise} \end{cases}$$

This is the famous Fibonacci^a sequence.

^anamed after Leonardo of Fibonacci.

One of the properties of the Fibonacci sequence is that the sequence converges to the “golden ratio”⁵. For any inductive proof of properties of the Fibonacci numbers, we would clearly need to assume that the property holds for the two preceding numbers in the sequence.

In the following, we present a principle that assumes a “stronger” induction hypothesis. And hence the principle itself seems

⁵one of the solutions of the equation $x^2 = x + 1$. It was considered an aesthetically pleasing aspect ratio for buildings in ancient Greek architecture.

“weaker” than the previous versions.

Principle of Complete Induction (PCI)

A property \mathbb{P} holds for all natural numbers provided

Basis. \mathbb{P} holds for 0.

Induction step. For an arbitrary $n > 0$

\mathbb{P} holds for every m , $0 \leq m < n$ implies \mathbb{P} holds for n .

This principle is also called *course of values induction* in some books. Notice the difference in the induction hypothesis in **PCI** and **PMI1**

Example -3.6: Fibonacci and the golden ratio

Let $F(0) = 0$, $F(1) = 1$, $F(2) = 1$, \dots , $F(n+1) = F(n) + F(n-1)$, \dots be the Fibonacci sequence. Let ϕ be the “golden ratio” $(1 + \sqrt{5})/2$. We now show that the property $F(n+1) \leq \phi^n$ holds for all n .

Proof: By the principle of complete induction on n .

Basis. For $n = 0$, we have $F(1) = \phi^0 = 1$.

Induction step. Assuming the property holds for all m , $0 \leq m \leq n-1$, for an arbitrarily chosen $n > 0$, we need to prove that $F(n+1) \leq \phi^n$.

$$\begin{aligned} F(n+1) &= F(n) + F(n-1) \\ &\leq \phi^{n-1} + \phi^{n-2} && \text{by the induction hypothesis} \\ &= \phi^{n-2}(\phi + 1) \\ &= \phi^n && \text{since } \phi^2 = \phi + 1 \end{aligned}$$

QED

The above example shows quite clearly that the induction hypothesis used in any application of complete induction though seemingly stronger, can also lead to the proof of seemingly stronger properties. On the other hand by appearing to use a stronger hypothesis, the principle PCI seems weaker. However, in the following example we again prove the property in example -3.6 but this time we use the principle of mathematical induction instead.

Example -3.7: Fibonacci and the golden ratio

Let $\mathbb{P}(n)$ denote the property

$$F(n+1) \leq \phi^n$$

Rather than prove the original statement **For all n , $\mathbb{P}(n)$** , we instead consider the property $\mathbb{Q}(n)$

$$\text{For every } m, 0 \leq m \leq n, \mathbb{P}(m).$$

which is equivalent to the statement

$$\text{For every } m, 0 \leq m \leq n, F(m+1) \leq \phi^m.$$

and prove the statement **For all n , $\mathbb{Q}(n)$** . Further notice that $\mathbb{Q}(n)$ is equivalent to the conjunction of $\mathbb{Q}(n-1)$ and $\mathbb{P}(n)$. This property \mathbb{Q} can now be proved by mathematical induction as follows. The reader is encouraged to study the following proof carefully as its ideas will be used in the proof of theorem [-3.3](#).

Proof: By the principle of mathematical induction on n .

Basis. For $n = 0$, we have $F(1) = \phi^0 = 1$.

Induction step. Assuming the property $\mathbb{Q}(n-1)$, holds for an arbitrarily chosen $n > 0$, we need to prove the property \mathbb{Q} for n . But for this it suffices to prove the property \mathbb{P} for n , since $\mathbb{Q}(n)$ is equivalent to the conjunction of $\mathbb{Q}(n-1)$ and $\mathbb{P}(n)$. Hence we prove the property $\mathbb{P}(n)$.

$$\begin{aligned} F(n+1) &= F(n) + F(n-1) \\ &\leq \phi^{n-1} + \phi^{n-2} && \text{by the induction hypothesis} \\ &= \phi^{n-2}(\phi + 1) \\ &= \phi^n && \text{since } \phi^2 = \phi + 1 \end{aligned}$$

QED

But in the end the proofs are almost identical. These proofs lead us then naturally into the next theorem. That is, even though the principle **PCI** seems weaker as compared to **PMI**, the two are equally powerful.

Theorem -3.3: Equivalence of **PMI1** and **PCI**

The two principles of mathematical induction are equivalent. In other words, every application of **PMI** may be transformed into an application of **PCI** and vice-versa.

Proof: We need to prove the following two claims.

1. *Any proof of a property using the principle of mathematical induction, is also a proof of the same property using the principle of complete induction.* This is so because the only possible change in the nature of two proofs could be because they use different induction hypotheses. Since the proof by mathematical induction uses a fairly weak assumption which is sufficient to prove the property, strengthening it in any way does not need to change the rest of the proof of the induction step.
2. *For every proof of a property using the principle of complete induction, there exists a corresponding proof of the same property using the principle of mathematical induction.* To prove this claim we resort to the same trick employed in example -3.7. We merely replace each occurrence of the original property in the form $\mathfrak{P}(n)$ by $\mathfrak{Q}(n)$, where the property \mathfrak{Q} is defined as

For every m , $0 \leq m \leq n$, $\mathfrak{P}(m)$.

Since $\mathfrak{Q}(0)$ is the same as $\mathfrak{P}(0)$ there is no other change in the basis step of the proof. In the original proof by complete induction the induction hypothesis would have read

For arbitrarily chosen $n > 0$, for all m , $0 \leq m \leq n - 1$, $\mathfrak{P}(m)$

whereas in the new proof by mathematical induction the induction hypothesis would read

For arbitrarily chosen $n > 0$, $\Phi(n - 1)$

Clearly the two induction hypotheses are logically equivalent. Hence the rest of the proof of the induction step would suffer no other change. The basis step and the induction step would together constitute a proof by *mathematical* induction of the property Φ for all naturals n . Since $\Phi(n)$ logically implies $\Psi(n)$ it follows that the proof of property Ψ for all naturals has been done by *mathematical* induction.

QED

The natural numbers are themselves defined as the smallest set \mathbb{N} such that $0 \in \mathbb{N}$ and whenever $n \in \mathbb{N}$, $n + 1$ also belongs to \mathbb{N} . Therefore we may state yet another version of PMI from which the other versions previously stated may be derived. The intuition behind this version is that a property Ψ may also be considered as defining a set $S = \{x \in \mathbb{N} \mid x \text{ satisfies property } \Psi\}$. Therefore if a property Ψ is true for all natural numbers the set defined by the property must be the set of natural numbers. This gives us the last version of the principle of mathematical induction.

Principle of Mathematical Induction – Version 0

A set $S = \mathbb{N}$ provided

Basis. $0 \in S$, and

Induction step. For arbitrarily chosen $n > 0$,

$n - 1 \in S$ implies $n \in S$.

Example -3.8

From theorem -4.4 we know there exists a bijection $f_2 : \mathbb{N}^2 \rightarrow \mathbb{N}$. We use this fact to prove by induction on n , that there exists a bijection $f_n : \mathbb{N}^n \rightarrow \mathbb{N}$, for all $n \geq 2$ and hence we may conclude that \mathbb{N}^n for each $n > 0$ is countable.

Proof: We know from exercise -4.3 problem 1 that if $F : A \rightarrow B$ and $G : B \rightarrow C$ are both bijections then their composition $G \circ F : A \rightarrow C$ is also a bijection. We now proceed to prove by induction on n .

Basis. For $n = 2$ it is given that f_2 is a bijection.

Induction step. Assume the induction hypothesis,

$$\boxed{\text{For some } n \geq 2 \text{ there exists a bijection } f_n : \mathbb{N}^n \rightarrow \mathbb{N}.}$$

We need to prove that there exists a 1-1 correspondence (bijection) between \mathbb{N}^{n+1} and \mathbb{N} . We prove this by constructing a function $f_{n+1} : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$. Let $h : \mathbb{N}^{n+1} \rightarrow (\mathbb{N} \times \mathbb{N})$ be defined by

$$h(x_1, \dots, x_n, x_{n+1}) = (f_n(x_1, \dots, x_n), x_{n+1})$$

Claim: h is a 1-1 correspondence. It can be proved from the fact that $f_n : \mathbb{N}^n \rightarrow \mathbb{N}$ is a bijection.

Since f_2 is also a bijection, it follows that the composition of h and f_2 , is also a bijection. Hence f_{n+1} defined as

$$f_{n+1}(x_1, \dots, x_n, x_{n+1}) = f_2(h(x_1, \dots, x_n, x_{n+1})) = f_2(f_n(x_1, \dots, x_n), x_{n+1})$$

is a bijection. ■

QED

Exercise -3.3: PMI0, PCI

1. Give a proof of **PCI** along the lines of the proof (-3.1) given for theorem **-3.1**. Do you see why the two principles are equivalent?
2. Modify the proof (-3.1) given for theorem **-3.1** to give a proof of **PMI version 0**.

-3.3. Structural Induction

In many situations involving the syntax and semantics of languages, the construction of “recursive” data types in languages such as ML and Java, it is helpful to consider a form of induction called structural induction. This form of induction enables us to prove fairly general properties about the datatypes so constructed and is a convenient tool for defining functions and proving properties about data types and programs.

Definition -3.1: Structural induction

Let \mathbb{U} be a set called the **Universe**, B a nonempty subset of \mathbb{U} called the **basis**, and let K called the **constructor** set be a nonempty set of functions, such that each function $f \in K$ has associated with it a unique natural number $n \geq 0$ called its **arity** and denoted by $\alpha(f)$. If a function f has an arity of $n \geq 0$, then $f : \mathbb{U}^n \rightarrow \mathbb{U}$. Let \mathcal{X} be the family of subsets of \mathbb{U} such that each $X \in \mathcal{X}$ satisfies the following conditions.

Basis. $B \subseteq X$

Induction step. if $f \in K$ is of arity $n \geq 0$ and $a_1, \dots, a_n \in X$ then $f(a_1, \dots, a_n) \in X$

A set A is said to be **inductively defined** from B , K , \mathbb{U} if it is the smallest set (under the subset ordering \subseteq) satisfying the above conditions i.e.

$$A = \bigcap_{X \in \mathcal{X}} X \quad (5)$$

The set A is also said to have been **generated** from the basis B and **the rules of generation** $f \in K$.

As in the other induction principles the underlined portion is the **induction hypothesis**. It may not be absolutely clear whether A defined as in equation (5) above satisfies the two conditions of definition -3.1.

Lemma -3.1: The smallest set

If A and \mathcal{X} are as in definition -3.1 then $A \in \mathcal{X}$.

Proof: We need to show that A satisfies the two conditions that each $X \in \mathcal{X}$ satisfies.

Basis. It is easy to see that since $B \subseteq X$ for each $X \in \mathcal{X}$, we have $B \subseteq A = \bigcap_{X \in \mathcal{X}} X$ and hence A does satisfy the basis condition [\(-3.3\)](#).

Induction step. Consider any $f \in K$ and elements $a_1, \dots, a_{\alpha(f)} \in A$. By equation [\(5\)](#) we have $a_1, \dots, a_{\alpha(f)} \in X$ for every $X \in \mathcal{X}$. Therefore $f(a_1, \dots, a_{\alpha(f)}) \in X$ for every $X \in \mathcal{X}$ which implies $f(a_1, \dots, a_{\alpha(f)}) \in A$. Hence $A \in \mathcal{X}$.

QED

We may also think of A as the smallest set (under the subset ordering) which satsfies the set equation

$$X = B \cup \bigcup \{f(X^n) \mid f \in K, \alpha(f) = n \geq 0, X \subseteq \mathbb{U}\} \quad (6)$$

in the unknown X , where $f(X^n) = \{a \mid a = f(a_1, \dots, a_n), \text{ for some } (a_1, \dots, a_n) \in X^n\}$. We show in lemma [-3.2](#) that such equations may be solved for their unique smallest solution.

Definition -3.2: Construction sequence

Let \mathbb{U}, B, K be as in definition [-3.1](#). Then a sequence $[a_1, \dots, a_m]$ of elements of \mathbb{U} is called a **construction sequence** for a_m if for all $i = 1, \dots, m$ either $a_i \in B$ or there exists a constructor $f \in K$, of arity $n > 0$, and $0 < i_1, \dots, i_n < i$ such that $f(a_{i_1}, \dots, a_{i_n}) = a_i$. a_i is said to **directly depend** on each of the elements a_{i_1}, \dots, a_{i_n} (denoted $a_{i_j} \prec^1 a_i$ for each $j \in \{1, \dots, n\}$). a_i **depends** on a_j , denoted $a_j \prec a_i$ if either $a_j \prec^1 a_i$ or there exists some i' such that $a_j \prec a_{i'}$ and $a_{i'} \prec^1 a_i$.

The set A contains exactly all those elements of \mathbb{U} which have a construction sequence. The basis along with the constructor functions are said to define the terms generated by the rules of construction of definition -3.1.

Example -3.9: am-expressions

Consider the following definition of a subclass of arithmetic expressions, called *am-expressions* generated only from natural numbers and the addition and multiplication operations. The rules may be expressed in English as follows.

Basis Every natural number n is an *am-expression*.

Induction step

- If e and e' are *am-expressions* then $\text{add}(e, e')$ is an *am-expression*,
- If e and e' are *am-expressions* then $\text{mult}(e, e')$ is an *am-expression*.

Initiality Nothing else is an *am-expression*

In the above definition of *am-expressions* \mathbb{N} is the basis B , $K = \{\text{add}, \text{mult}\}$ is the set of constructors each of arity 2 and the universe \mathbb{U} consists of all possible finite sequences of symbols drawn from the natural numbers and applications of the constructors. Only strings that are obtained by a finite number of applications of the above rules are *am-expressions*. The smallest set generated from finite sequences of applications of the basis and induction steps is the set of *am-expressions* involving only the naturals and the 2-ary constructors *add* and *mult* such that every application of a constructor has exactly two operands each of which in turn is either a natural number or constructed in a similar fashion. 0 , $\text{add}(0, 1)$, $\text{mult}(\text{add}(1, 0), \text{mult}(1, 1))$ are all am-expressions.

Example -3.10: Illegal am-expressions

On the other hand, (refer example -3.9),

1. $\text{add}(0)$ is not an am-expression since the arity of add is 2,
2. $0, 1$ is not an am-expression since it is not a natural number (it is actually a sequence of two natural numbers),
3. $\text{mult}(\infty, 0)$ is not an am-expression since ∞ is not a natural number.

Example -3.11: constructions sequences

The following are possible construction sequences of the am-expression $\text{mult}(\text{add}(1, 0), \text{mult}(1, 1))$ generated in example -3.9.

1. $[1, 0, \text{add}(1, 0), 1, 1, \text{mult}(1, 1), \text{mult}(\text{add}(1, 0), \text{mult}(1, 1))]$
2. $[1, 0, \text{add}(1, 0), \text{mult}(1, 1), \text{mult}(\text{add}(1, 0), \text{mult}(1, 1))]$ where replicas of *am-expressions* have been omitted.
3. $[1, 0, \text{mult}(1, 1), \text{add}(1, 0), \text{mult}(\text{add}(1, 0), \text{mult}(1, 1))]$ since it does not matter in which order the two operands of the last element in the sequence occur in the construction sequence as long as they precede the final *am-expression*.

Notation -3.1

A convenient shorthand notation called the *Backus-Naur Form (BNF)* is usually employed to express the rules of generation. For the set of am-expressions defined above the BNF is as follows.

$$e, e' ::= n \in \mathbb{N} \mid \text{add}(e, e') \mid \text{mult}(e, e')$$

Example -3.12

Consider the following BNF of arithmetic expressions

$$e, e' ::= n \in \mathbb{N} \mid (e + e') \mid (e \times e') \mid (e - e')$$

Here $\mathbb{U} = (B \cup \{(,)\}, +, \times, -)^*$, where $B = \mathbb{N}$, $K = \{f_+, f_\times, f_-\}$ and such that for any expressions e and e' and any operator $\odot \in \{+, \times, -\}$, $f_\odot(e, e') = (e \odot e')$.

It is possible to relate the notions of dependence in a construction sequence to the construction process by partially ordering the process of construction of elements in an inductively generated set as follows. Let B , K and \mathbb{U} be as in definition -3.1. Consider the infinite sequence of sets

$$[A_0 \subseteq A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots] \quad (7)$$

defined by mathematical induction as $A_0 = B$ and for each $i \geq 0$, $A_{i+1} = A_i \cup \{f(a_1, \dots, a_{\alpha(f)}) \mid f \in K, a_1, \dots, a_{\alpha(f)} \in A_i\}$. Now consider the set

$$A_\infty = \bigcup_{i \geq 0} A_i \quad (8)$$

The following lemma shows that $A_\infty = A$ and is indeed the smallest solution to the equation (6). Moreover (7) gives a construction of the smallest solution by mathematical induction.

Lemma -3.2: Equation solving

Let \mathbb{U} , B , K and A be as in definition -3.1 and A_∞ be as in equation (8). Then $A = A_\infty$.

Proof: By definition (8) $A_i \subseteq A_\infty$ for each $i \in \mathbb{N}$

$$A_0 \subseteq A_1 \subseteq A_2 \subseteq A_3 \subseteq \cdots \subseteq A_\infty$$

We structure the proof in the form of two claims. Firstly we show that $A_\infty \in \mathcal{X}$ i.e. A_∞ is a set that satisfies the conditions in definition -3.1. Secondly, we prove that $A_\infty \subseteq A$. It follows then that $A = A_\infty$ since $A \subseteq X$ for each $X \in \mathcal{X}$.

Claim. $A_\infty \in \mathcal{X}$.

⊤ Since $B = A_0 \subseteq A_\infty$, A_∞ does satisfy the basis condition. Consider any $f \in K$ and elements $a_1, \dots, a_{\alpha(f)} \in A_\infty$. By the definition of A_∞ , there exist indices $i_1, \dots, i_{\alpha(f)} \in \mathbb{N}$ such that $a_1 \in A_{i_1}, \dots, a_{\alpha(f)} \in A_{i_{\alpha(f)}}$. Let $i \geq \max(i_1, \dots, i_{\alpha(f)})$. It is clear from the definition of A_i that $a_1, \dots, a_{\alpha(f)} \in A_i$ and hence $f(a_1, \dots, a_{\alpha(f)}) \in A_{i+1} \subseteq A_\infty$. Hence A_∞ satisfies the second condition too and therefore $A_\infty \in \mathcal{X}$.
⊣

Claim. $A_\infty \subseteq A$.

⊤ We prove by mathematical induction on indices i , that $A_i \subseteq A$ for every index i . The basis is trivial since we know that $A_0 = B \subseteq A$. For the induction step assume for some $k \geq 0$, $A_k \subseteq A$. We have $A_{k+1} = A_k \cup \{f(a_1, \dots, a_{\alpha(f)}) \mid f \in K, a_1, \dots, a_{\alpha(f)} \in A_k\}$. For any $a \in A_k$ we already know by the induction hypothesis that $a \in A$. Any $a \in A_{k+1} - A_k$ is then of the form $a = f(a_1, \dots, a_{\alpha(f)})$ where $a_1, \dots, a_{\alpha(f)} \in A_k$. Again by the induction hypothesis we have $a_1, \dots, a_{\alpha(f)} \in A$ and since $A \in \mathcal{X}$, $a = f(a_1, \dots, a_{\alpha(f)}) \in A$. Hence $A_{k+1} \subseteq A$.
⊣

QED

Lemma -3.2 and its proof, in addition to showing us how to obtain least solutions to equations of the form (6) also show the relationship to the principle of mathematical induction. Further the lemma gives us a way to quantify dependence of elements in a construction sequence by assigning each element an order number.

Definition -3.3: Height of an element

The **height** of any element a (denoted $\triangle(a)$) in an inductively generated set A is the least index i in the monotonic sequence (7) such that $a \in A_i$.

In any construction sequence $[a_1, \dots, a_m]$, $a_i \prec a_j$ only if the height of a_i is less than the height of a_j .

Example -3.13: Inductive generation of \mathbb{N}

Let $S \subseteq \mathbb{N}$ be the smallest set of numbers defined by

$$n ::= 0 \mid n + 1$$

Clearly this defines the smallest set containing 0 and closed under the successor operation on the naturals. It follows that $S = \mathbb{N}$. Notice that the above BNF is merely a rewording of the [principle of mathematical induction version 0](#).

Example -3.13 shows that mathematical induction is merely a particular case of structural induction.

Example -3.14: Logic of negation and implication

The language \mathcal{M}_0 was defined in example -4.8. We may redefine the language by the following BNF

$$\mu, \nu ::= a \in \mathbb{A} \mid (\neg\mu) \mid (\mu \rightarrow \nu)$$

We prove that the language is countable.

Proof: We first begin by classifying the formulas of the language according to their height. Let M_k be the set of formulas of the language such that

each formula has a height at most k for $k \geq 0$. We assume that $M_0 = \mathbb{A}$ and $M_{k+1} = M_k \cup \{(\neg \mu_k), (\mu_k \rightarrow \nu_k) \mid \mu_k, \nu_k \in M_k\}$. Let \vec{M}_k be the set of all formulas of depth $k + 1$ of the form “ $(\neg \mu_k)$ ” and let $\vec{\vec{M}}_k$ be the set of all formulas of the form “ $(\mu_k \rightarrow \nu_k)$ ”, where $\mu_k, \nu_k \in M_k$. We then have

$$\begin{aligned} M_{k+1} &= M_k \cup \vec{M}_k \cup \vec{\vec{M}}_k \\ &= M_k \cup (\vec{M}_k - M_k) \cup (\vec{\vec{M}}_k - M_k) \\ &= M_k \cup N_{k+1} \cup \vec{N}_{k+1} \end{aligned}$$

Here $N_{k+1} = \vec{N}_{k+1} \cup \vec{\vec{N}}_{k+1}$ represents the set of all formulas of height exactly $k + 1$. \vec{N}_{k+1} consists of exactly those formulas of height $k + 1$ whose root operator is \neg . Similarly $\vec{\vec{N}}_{k+1}$ represents the set of all formulas of height exactly $k + 1$, whose root operator is \rightarrow . Hence the three sets are mutually disjoint.

$$M_k \cap N_{k+1} = \emptyset, \quad M_k \cap \vec{N}_{k+1} = \emptyset, \quad \vec{N}_{k+1} \cap \vec{\vec{N}}_{k+1} = \emptyset$$

The entire language may then be defined as the set $\mathcal{M}_0 = \bigcup_{k \geq 0} M_k = \mathbb{A} \cup \bigcup_{k > 0} \vec{N}_k \cup \bigcup_{k > 0} \vec{\vec{N}}_k$ which is a countable union of countable sets and is hence countable. QED

We present below a generalization of the principle of mathematical induction to arbitrary inductively defined sets. It provides us a way of reasoning about the properties of structures that are inductively defined (definition -3.1) or by a BNF (notation -3.1).

Theorem -3.4: Principle of structural induction

Let $A \subseteq \mathbb{U} \neq \emptyset$ be inductively defined by the basis $B \neq \emptyset$ and the constructor set $K \neq \emptyset$.

Principle of Structural Induction (PSI)

A property \mathfrak{P} holds for all elements of A provided

Basis. \mathfrak{P} is true for all basis elements.

Induction step. For each $f \in K$, \mathfrak{P} holds for elements $a_1, \dots, a_{\alpha(f)} \in A$ implies \mathfrak{P} holds for $f(a_1, \dots, a_{\alpha(f)})$.

Proof:

Let \mathfrak{P} be a property of the elements of \mathbb{U} that satisfies the conditions above. Let C be the set of all elements of A that satisfy the property \mathfrak{P} , i.e. $C = \{a \in A \mid \mathfrak{P} \text{ holds for } a\}$. It is clear that $B \subseteq C \subseteq A$. To show that $C = A$ it suffices to prove that $A - C = \emptyset$. Consider the sequence of sets defined in (7) and the set $A = \bigcup_{i \geq 0} A_i$ as given in equation (8). We prove that for all $i \geq 0$, $A_i \subseteq C$ by contradiction.

Assume that there is a smallest $i \geq 0$ such that $A_i \not\subseteq C$. Since $A_0 = B \subseteq C$, $A_i \not\subseteq C$ implies $i > 0$ and $A_{i-1} \subseteq C$. There exists $a \in A_i - A_{i-1}$ which does not satisfy the property \mathfrak{P} . Since $a \notin A_{i-1}$, we have $a = f(a_1, \dots, a_{\alpha(f)})$ for some $f \in K$ such that $a_1, \dots, a_{\alpha(f)} \in A_{i-1}$. Further since $A_{i-1} \subseteq C$ and hence property \mathfrak{P} must hold for each of $a_1, \dots, a_{\alpha(f)}$. This implies by the induction step that \mathfrak{P} holds for a , contradicting the assumption that a does not satisfy \mathfrak{P} . Hence there is no smallest i such that $A_i \not\subseteq C$. Therefore for all $i \geq 0$, $A_i \subseteq C$ and hence $A \subseteq C$ from which it follows that \mathfrak{P}

holds for every element of A .

Example -3.15

Let e be any expression generated by the BNF of example -3.12 and let e' be a prefix^a of e . Further, let $L(e')$ and $R(e')$ denote respectively the numbers of left $(()$ and right $)()$ parentheses in e' . Let $\mathfrak{P}(e)$ be For every prefix e' of e , $L(e') \geq R(e')$. We use the principle of structural induction (theorem -3.4) to prove that property \mathfrak{P} holds for all expressions in the language.

Basis. It holds for all $n \in \mathbb{N}$ because no natural number has any parentheses.

Induction Hypothesis (IH).

For any e of the form $(f \odot g)$, where $\odot \in \{+, \times, -\}$ and f, g are themselves expressions in the language

1. $\mathfrak{P}(f)$ holds i.e. For every prefix f' of f , $L(f') \geq R(f')$ and
2. $\mathfrak{P}(g)$ holds i.e. For every prefix g' of g , $L(g') \geq R(g')$

Induction Step. We do an exhaustive (and a rather exhausting!) case analysis of all the possible prefixes of e .

- Case $e' = \varepsilon$. $L(e') = R(e')$.
- Case $e' = (. L(e') = 1 > 0 = R(e')$.
- Case $e' = (f'$. By IH we have $L(f') \geq R(f')$ and hence $L(e') = 1 + L(f') > R(f') = R(e')$.
- Case $e' = (f'\odot$. By IH we have $L(f') \geq R(f')$ and hence $L(e') = 1 + L(f') > R(f') = R(e')$.
- Case $e' = (f\odot g'$. By IH we have $L(f) \geq R(f)$ and $L(g') \geq R(g')$. Hence $L(e') = 1 + L(f) + L(g') > R(f) + R(g') = R(e')$.
- Case $e' = (f\odot g$. By IH we have $L(f) \geq R(f)$ and $L(g) \geq R(g)$. Hence $L(e') = 1 + L(f) + L(g) > R(f) + R(g) = R(e')$.
- Case $e' = e = (f\odot g$. By IH we have $L(f) \geq R(f)$ and $L(g) \geq R(g)$. Hence $L(e') = L(e) = 1 + L(f) + L(g) \geq R(f) + R(g) + 1 = R(e') = R(e)$.

^a e' may or may not be an expression of the language.

Exercise -3.4

1. Use PSI to prove the property **For every e , $L(e) = R(e)$** for the language of example -3.12.

We leave it as an exercise for the reader to prove that every proof by the principle of mathematical induction may also be translated into a proof by the principle of structural induction (see example -3.13 and the equivalences between the various versions of the principle of mathematical induction and complete induction e.g theorem -3.3). We are now ready to show that even though structural induction seems to be more general than mathematical induction they are in fact equivalent. In other words, every proof by the principle of structural induction may also be rewritten as a proof by (some version) of the principle of mathematical induction or complete induction. To do this we need the height (see definition -3.3) of each element in an inductively defined set.

Theorem -3.5: Structural induction to complete induction

Every proof using the principle of structural induction (PSI) may be replaced by a proof using the principle of complete induction (PCI).

Proof: Let A be inductively defined by B , K , \cup and let \mathfrak{P} be a property of elements of A that has been proved by the principle of structural induction. Let the property $\mathfrak{Q}(n)$ for each natural number n be defined as

The property \mathfrak{P} holds for all elements of height n

Basis. The basis step $n = 0$ of the proof of property Φ proceeds exactly as the basis step of the proof by PSI with as many cases are required in the proof of by PSI.

The induction hypothesis. The induction hypothesis is the assumption that $\Phi(m)$ holds for all $0 \leq m < n$.

The induction step The induction step is simply that if the induction hypothesis holds for all $m < n$ then Φ holds for n . The proof by PSI for each constructor in the induction step is a case in the induction step of the proof $\Phi(n)$.

QED

-3.3.1. Inductive definitions of functions

Just as we define functions by induction we could also define functions by structural induction.

Definition -3.4: Definition by structural induction

Let A be inductively defined by B , K , and \cup and let V be any arbitrary set. Then $h : A \rightarrow V$ is said to be an **inductively defined** function if $h(b) = h_0(b)$ and $h(f(a_1, \dots, a_{\alpha(f)})) = h_f(h(a_1), \dots, h(a_{\alpha(f)}))$ where $h_0 : B \rightarrow V$ is a function and for every n -ary constructor $f \in K$, there is an n -ary function $h_f : V^n \rightarrow V$. A relation $\mathcal{R} \subseteq A \times V$ is said to be **inductively defined** if

$$\mathcal{R} = \{(b, h_0(b)) \mid b \in B\} \cup \{(f(a_1, \dots, a_{\alpha(f)}), h_f(v_1, \dots, v_{\alpha(f)})) \mid a_1 \mathcal{R} v_1, \dots, a_{\alpha(f)} \mathcal{R} v_{\alpha(f)}\}$$

Example -3.16: Semantics of Propositions

Consider the language \mathcal{P}_0 of propositional logic, where the basis is a countable set \mathbb{A} of atoms, and the language is defined by the BNF

$$\phi, \psi ::= p \in \mathbb{A} \mid (\neg\phi) \mid (\phi \vee \psi) \mid (\phi \wedge \psi)$$

Further let $\mathbf{2} = \langle \mathfrak{D}, \bar{}, +, \cdot \rangle$ (where $\mathfrak{D} = \{0, 1\}$) be the algebraic system of *truth values* whose operations are defined as follows.

$$\begin{aligned}\bar{0} &= 1 && \text{and } \bar{1} = 0 \\ 1 + 1 &= 0 + 1 = 1 + 0 = 1 && \text{and } 0 + 0 = 0 \\ 0 \times 0 &= 0 \times 1 = 1 \times 0 = 0 && \text{and } 1 \times 1 = 1\end{aligned}$$

Let $\tau_0 : \mathbb{A} \longrightarrow \{0, 1\}$ be an assignment of binary values to the propositional atoms and let the functions $\tau_{\bar{}} = \bar{}$, $\tau_{\vee} = +$ and $\tau_{\wedge} = \times$. The assignment τ_0 may be extended by structural induction to propositional formulas in \mathcal{P}_0 , by the function $\mathcal{T} : \mathcal{P}_0 \longrightarrow \{0, 1\}$ as follows

$$\begin{aligned}\mathcal{T}[p] &= \tau_0(p) && p \in \mathbb{A} \\ \mathcal{T}[(\neg\phi)] &= \overline{\mathcal{T}[\phi]} \\ \mathcal{T}[(\phi \vee \psi)] &= \mathcal{T}[\phi] + \mathcal{T}[\psi] \\ \mathcal{T}[(\phi \wedge \psi)] &= \mathcal{T}[\phi] \times \mathcal{T}[\psi]\end{aligned}$$

Exercise -3.5

1. Show using PSI that \mathcal{T} in example [-3.16](#) is well defined.
2. The language of numerals in the decimal notation may be defined by the following BNF (it is actually a regular grammar).

$$\begin{aligned} d & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ n & ::= d \mid nd \end{aligned}$$

Define the value of a string in this language, so that it conforms to the normally accepted meaning of a numeral

3. Assume the BNF in problem [2](#) is defined instead as

$$\begin{aligned} d & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ n & ::= d \mid dn \end{aligned}$$

- (a) Prove that the two grammars generate the same set of strings.
- (b) Define the value of a string in the grammar [3](#). How is it different from that in problem [2](#).

4. For the language \mathcal{P}_0 of propositions in example [-3.16](#) we may also call the function τ_0 a *state*. Let Σ denote the set of all possible states i.e. $\Sigma = \{\tau_0 \mid \tau_0 : \mathbb{A} \rightarrow \mathbb{2}\}$. The function \mathcal{T} defines the truth value of a proposition in a given state τ_0 . We may also define the meaning of a proposition as the set of states in which the proposition has a truth value of 1 i.e. $\mathcal{S} : \mathcal{P}_0 \longrightarrow \mathcal{P}^\Sigma$

- (a) Define the function \mathcal{S} by structural induction.
- (b) Prove using PSI that for any proposition ϕ , and a state τ_0 , $\mathcal{T}[\phi] = 1$ if and only if $\tau_0 \in \mathcal{S}(\phi)$.

-3.4. Simultaneous Induction

One question that naturally arises is that if induction can be viewed as equation solving (see equation 6), then are there may be problems or properties which require us to solve a *system of simultaneous equations*? We answer this question with the following example.

Example -3.17: Simultaneous set equations

Consider the following BNF which consists of the generation of three different sets of bit strings (i.e. sequences of 0s and 1s) which are all mutually dependent.

$$\begin{aligned} z &::= 0 \mid z0 \mid i1 \\ i &::= 1 \mid t0 \mid z1 \\ t &::= i0 \mid t1 \end{aligned}$$

In effect this BNF defines a system of three (simultaneous) set equations.

$$\begin{aligned} Z &= \{0\} \cup \{z0 \in 2^+ \mid z \in Z\} \cup \{i1 \in 2^+ \mid i \in I\} \\ I &= \{1\} \cup \{t0 \in 2^+ \mid t \in T\} \cup \{z1 \in 2^+ \mid z \in Z\} \\ T &= \{i0 \in 2^+ \mid i \in I\} \cup \{t1 \in 2^+ \mid t \in T\} \end{aligned}$$

For any bit string s let $(s)_2$ denote the non-negative integer j such that s is a binary representation of j (there could be leading zeroes). Suppose we need to prove the following property \mathfrak{P}_Z of the set Z .

$$\boxed{\mathfrak{P}_Z : Z = \{z \in 2^+ \mid (z)_2 \text{ is a non-negative multiple of } 3\}}$$

The intuition which guides the BNF above (or equivalently the above system of equations) is the following.

- Each $z \in Z$ is such that $(z)_2 = 3l$ for some $l \in \mathbb{N}$ i.e. $(z)_2$ is a multiple of 3. In particular, the bit-string $0 \in Z$. The other sets in the definition of Z are constructed so that if $(z)_2 = 3l$, for some $l \in \mathbb{N}$, then $(z0)_2 = 6l$ is also a multiple of 3 and for each $i \in I$, if $(i)_2 = 3m + 1$ for some $m \in \mathbb{N}$ then $i1 \in Z$ since $(i1)_2 = 2(3m + 1) + 1 = 3(2m + 1)$ is a multiple of 3.
- Likewise the definitions of the sets I and T are such that for each $i \in I$, $(i)_2 = 3m + 1$ for some natural m and for each $t \in T$, $(t)_2 = 3n + 2$ for some natural n .

The same intuition also guides the proof of property \mathfrak{P}_Z . However, the proof is dependent upon properties \mathfrak{P}_I for the set I and \mathfrak{P}_T for T where

$$\mathfrak{P}_I : I = \{i \in 2^+ \mid (i)_2 = 3m + 1, m \in \mathbb{N}\}$$

and

$$\mathfrak{P}_T : T = \{t \in 2^+ \mid (t)_2 = 3n + 2, n \in \mathbb{N}\}$$

Hence an inductive proof of property \mathfrak{P}_Z requires a simultaneous proofs of properties \mathfrak{P}_I and \mathfrak{P}_T as well.

Lemma -3.3: Simultaneous induction proof

The elements of the sets Z , I and T satisfy properties \mathfrak{P}_Z , \mathfrak{P}_I and \mathfrak{P}_T respectively.

Proof: We proceed by simultaneous induction on the lengths of strings of 2^+ ($|s|$ denotes the length of string s).

Basis. There are exactly two bit strings of length 1 viz, 0 and 1 and it follows easily that $0 \in Z$ and $1 \in I$.

Induction Hypothesis (IH).

For some $k > 0$,

0. each $z' \in Z$ such that $|z'| = k$ satisfies property \mathfrak{P}_Z ,
1. each $i' \in I$ such that $|i'| = k$ satisfies property \mathfrak{P}_I , and
2. each $t' \in T$ such that $|t'| = k$ satisfies property \mathfrak{P}_T ,

Induction Step. Consider strings of length $k + 1$. We need to show that

0. Property \mathfrak{P}_Z holds for $z = z'0$ and $z = i'1$, where $|z'| = k = |i'|$,
1. Property \mathfrak{P}_I holds for $i = t'0$ and $i = z'1$, where $|t'| = k = |z'|$,
2. Property \mathfrak{P}_T holds for $t = i'0$ and $t = t'1$, where $|i'| = k = |t'|$,

We may use the induction hypothesis then to prove the induction cases. We leave the rest of the proof to be completed by the interested reader.

QED

Exercise -3.6

1. Prove that $2^+ = Z \cup I \cup T$ in example -3.17.

-3.5. Well-ordered Induction

We have already defined the notions of well-founded (definition -4.15) relations and well-ordered (definition -4.17) sets. Well-ordered induction generalises the principle of mathematical induction to any set A which may be enumerated as a sequence $[a_i \mid i \in \mathbb{N}]$ indexed by the non-negative integers. There is an implicit total ordering in the elements such that $a_i < a_j$ if and only if $i < j$. Hence any property $\mathfrak{P}(a_i)$ of elements of the set may be regarded as a property $\mathfrak{Q}(i)$ of the index of the element in the set. What makes induction applicable here is the fact that the set \mathbb{N} is “well-ordered” (see definition -4.17) by the relation $<$. Hence were a property \mathfrak{P} to fail for some element a_j in the set A , there would be a *least element* $a_i \leq a_j$ for which it would fail and which in turn translates to the property $\mathfrak{Q}(j)$ and $\mathfrak{Q}(i)$ failing in such a fashion that for all (if any) $i' < i$, $\mathfrak{Q}(i')$ would hold.

Theorem -3.6: Principle of well-ordered induction

Let $\langle A, \prec \rangle$ be a well-ordered set.

Principle of Well-ordered Induction (PWI)

For any $X \subseteq A$, $X = A$ provided

Basis. The (unique) least element of A is in X and

Induction step. For each $a \in A$,

if (for every $a' \in A$, $[a' \prec a \text{ implies } a' \in X])$ then $a \in X^a$.

^aThe brackets and parentheses have been put in to avoid ambiguity in the statement.

Proof: Suppose $X \neq A$. Then since $X \subseteq A$, we have $B = A - X \neq \emptyset$. Since $B \subseteq A$, and A is well-ordered, B has a unique smallest element $b \in B$. But $b \notin X$, even though for every element $a \prec b$, $a \in X$. But by the induction step $b \in X$, which is a contradiction. QED

Note that in the statement of the above theorem, the basis is included (vacuously) in the induction step. Hence the basis statement in this principle is actually superfluous.

Example -3.18: The fundamental theorem of arithmetic

Let $A = \{n \in \mathbb{N} \mid n > 1\}$. Noting that $\langle A, < \rangle$ is well-ordered, we prove the property

$$\mathfrak{P}(n) : n \text{ is either a prime or is a product of primes}$$

using induction^a.

Basis. 2 is a prime. Hence $\mathfrak{P}(2)$ is true.

Induction step. Assume $a \in A$. If a is prime there is nothing to prove. If a is composite then $a = b \times c$ for some $b, c \in A$ and by the property of multiplication we know that $b < a$ and $c < a$. By the induction hypothesis, both $\mathfrak{P}(b)$ and $\mathfrak{P}(c)$ hold and since $a = bc$, it follows that $\mathfrak{P}(a)$ also holds.

^aBy now, it should not come as a surprise to the reader that this could equally well have been proved using PMI or PCI.

Example -3.19: The greatest common divisor

Consider the following function $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (equivalently it is a functional program) such that

$$g(a, b) = \begin{cases} b & \text{if } a = 0, b > 0 \\ g(b \% a, a) & \text{if } a \neq 0 \end{cases}$$

where $b \% a$ denotes the remainder obtained on dividing b by a . We prove that $g(a, b)$ computes the greatest common divisor (gcd) of a and b whenever at least one of a, b is non-zero^a. As a functional program we need to show that it terminates i.e. the function is well-defined for all $(a, b) \neq (0, 0)$.

Proof: Whenever $0 \leq b < a$ we have that $g(a, b) = g(b, a)$ since $b \% a = b$ for $b < a$. Hence it suffices to prove the following claim.

Claim. For all $0 \leq a < b$, $g(a, b) = \text{gcd}(a, b)$

↪ For $0 = a < b$ it is clear that $g(a, b) = g(0, b) = b = \text{gcd}(0, b)$. For $0 < a_0 < b_0$, we know that the set of arguments of the function g is given by $S_{(a_0, b_0)} = \{(a_i, b_i) \mid i \geq 0, a_{i+1} = b_i \% a_i, b_{i+1} = a_i\}$ This set $S_{(a_0, b_0)}$ is well-ordered by the $<_{lex}$ relation on ordered pairs of numbers which is defined as follows.

$$(a, b) <_{lex} (a', b') \text{ iff } a < a' \text{ or } (a = a' \text{ and } b < b')$$

Hence if $>_{lex}$ denotes the converse of $<_{lex}$ we have that the sequence

$$(a_0, b_0) >_{lex} (a_1, b_1) >_{lex} \dots$$

is a finite descending sequence bounded (from below) by $(0, b_n)$ for some $b_n, n \geq 0$. By the properties of gcd (from any standard book on Elementary Number theory) we know that $\text{gcd}(a_i, b_i) = \text{gcd}(a_{i+1}, b_{i+1})$, for each $i \geq 0$. Hence $g(a_0, b_0) = \text{gcd}(a_0, b_0)$ for all $0 < a_0 < b_0$. \dashv

QED

^a $\text{gcd}(0, 0)$ is undefined

-3.6. Noetherian Induction

We may extend most of our notions of induction to partially ordered sets as well, provided we can construct a well-founded relation from a partial order. Let $\langle A, < \rangle$ be a strict partial order (see definition -4.12) which is also well-founded (see definition -4.15). Then every subset of A has at least one minimal element with respect to the relation $<$. Further since $A \subseteq A$, there is a subset $M \subseteq A$ of minimal elements of A . We then have the following principle.

Theorem -3.7: Principle of Noetherian induction

Let $\langle A, < \rangle$ be a well-founded strict partial order.

Principle of Noetherian Induction (PNI)

For any $X \subseteq A$, $X = A$ provided

Basis. $M \subseteq X$ where $M \subseteq A$ is the set of all minimal elements of A and

Induction step. For each $a \in A$,

if (for every $a' \in A$, $[a' < a \text{ implies } a' \in X]$) then $a \in X^a$.

^aThe brackets and parentheses have been put in to avoid ambiguity in the statement.

The proof is a slight generalisation of the proof of PWI (page 126). Again as in the case of PWI the basis step is actually

redundant, since for all minimal elements of A , the induction hypothesis is vacuously true.

Proof: Suppose $X \neq A$. Then since $X \subseteq A$, we have $B = A - X \neq \emptyset$. Since $B \subseteq A$, and A is well-founded, there exists M' , $\emptyset \neq M' \subseteq B$ of minimal elements in B . For each $m \in M'$ and each $a < m$, $a \in X$. By the induction step therefore $m \in X$ which is a contradiction⁶. Hence the assumption $B \neq \emptyset$ is false. QED

Example -3.20: Ackermann's function

Ackermann's function is a (partial) function $ack : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ defined on pairs of naturals as follows.

$$ack(x, y) = \begin{cases} y + 1 & \text{if } x = 0 \\ ack(x - 1, 1) & \text{if } x > 0 \text{ and } y = 0 \\ ack(x - 1, ack(x, y - 1)) & x, y > 0 \end{cases}$$

It is not obvious that this function is total. We prove that it is defined over all pairs of naturals.

Proof: $\langle \mathbb{N} \times \mathbb{N}, <_{lex2} \rangle$ (see exercise -4.8) is well-ordered lexicographically and $(0, 0)$ is its least element. We prove the property

$$\mathbb{P}(m, n): \text{ack}(m, n) \text{ is well defined i.e. } ack(m, n) \in \mathbb{N}$$

for all $(m, n) \in \mathbb{N} \times \mathbb{N}$.

Basis. $ack(0, 0) = 1$ and is hence well-defined.

Induction step. Assume $ack(m', n')$ is well-defined for all $(m', n') <_{lex2} (m, n)$. Now consider the following cases.

- Case $m = 0$. Then $ack(0, n) = n + 1$ and is well defined.

⁶If m is also minimal in A then by the basis $m \in X$ which is again a contradiction

- Case $m > 0$ and $n = 0$. Then $\text{ack}(m, 0) = \text{ack}(m-1, 1)$ and since $(m-1, 1) <_{lex2} (m, 0)$, by the induction hypothesis it must be well-defined.
- Case $m > 0$ and $n > 0$. Then $\text{ack}(m, n) = \text{ack}(m-1, \text{ack}(m, n-1))$. Since $(m, n-1) <_{lex2} (m, n)$ by the induction hypothesis $\text{ack}(m, n-1)$ is well-defined and equals some $k \in \mathbb{N}$. Hence $\text{ack}(m, n) = \text{ack}(m-1, k)$. Since $(m-1, n) <_{lex2} (m, k)$ for any value of k again by the induction hypothesis $\text{ack}(m-1, k)$ is well-defined and hence $\mathfrak{P}(m, n)$ holds.

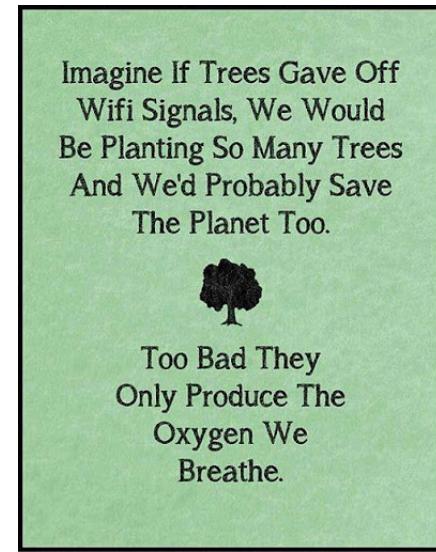
QED

Since $<_{lex2}$ is a total ordering, this proof is identical whether it is done by invoking **PWI** or **PNI**.

Exercise -3.7: Equivalence of induction principles

1. Prove that the **PNI** and **PMI0** are equivalent. *Hint: Use a ranking function $r : A \longrightarrow \mathbb{N}$ which assigns a rank 0 to all minimal elements of A with progressively increasing ranks for non-minimal elements.*
2. Use a similar idea of ranking functions to prove that **PNI** and **PSI** are equivalent.

-2. Rooted trees



The concept of trees and their manipulations is going to play a significant role in the rest of this course. Students of computer science would be familiar with the notions of binary trees and multi-way trees. They would also be familiar with the graph-theoretic structure called a tree. Trees in elementary expositions of graph theory are often finite undirected acyclic structures. Trees as a data-structure in early data-structures courses are usually finite, directed and possess a root. We try to provide a uniform and unambiguous treatment of rooted trees that is useful for the rest of the exposition of this course. Further we may also consider trees with an infinite number of nodes – something that is neither treated in elementary expositions of graph theory nor in computer science.

Definition -2.1: Directed graphs

- A **directed graph** is a pair $\langle N, \rightarrow \rangle$ consisting of a non-empty finite or a countably infinite set N of **nodes** and a set $\rightarrow \subseteq N \times N$ of **(directed) edges** such that for any edge $(s, t) \in \rightarrow$ (usually denoted $s \rightarrow t$ in infix notation), s is called the **source** and t the **target** of the edge. s is called a **predecessor** of t and t is called a **successor** of s .
- A **(directed) path** of length $k \geq 1$ in a directed graph is a finite sequence of nodes $n_0, n_1, \dots, n_{k-1}, n_k$ such that $n_i \rightarrow n_{i+1}$ for each $i \in \{0, \dots, k-1\}$. In addition if $n_k = n_0$ the path is called a **cycle** of length k , A cycle of length 1 is called a **self-loop**.
- A directed graph $\langle N', \rightarrow' \rangle$ is a **sub-graph** of a directed graph $\langle N, \rightarrow \rangle$ if $N' \subseteq N$ and $\rightarrow' \subseteq \rightarrow$.
- A directed graph is called a **directed acyclic graph (DAG)** if it has no cycles.
- A DAG is a **rooted DAG** if it has a distinguished node called the **root** from which there is a directed path to every other node.

Fact -2.1

The root node is not the successor of any node in a rooted DAG.

Definition -2.2: Trees

- An **unordered (directed) tree** is a directed acyclic graph such that there is at most one path between any pair of nodes.
- A **(rooted directed) tree** is a triple $\langle N, \rightarrow, r \rangle$ such that $\langle N, \rightarrow \rangle$ is an unordered (directed) tree and $r \in N$ is a distinguished node called the **root** of the tree and satisfying the following properties.
 - There exists a function $\ell : N \rightarrow \mathbb{N}$ called the **level** such that $\ell(r) = 0$ and for any $s \rightarrow t$, $\ell(t) = \ell(s) + 1$.
 - Every node in $N - \{r\}$ has a unique predecessor.
- A node with no successor is called a **leaf (node)**.

Fact -2.2: Level

The level of a node in a rooted tree is the length of the path from the root node.

We will be primarily interested in rooted directed trees and we will simply refer to them as trees. The main difference between a rooted DAG and rooted tree is that a node in a rooted DAG may not have unique predecessor. Rooted DAGs may then be transformed into rooted trees by replicating nodes which do not have unique predecessors. Similarly rooted trees may be transformed into rooted DAGs by coalescing nodes which are roots of isomorphic sub-graphs.

Notation -2.1

- We will often represent a tree \mathcal{T} specified only by a name and a root node r either as $\begin{array}{c} r \\ \swarrow \quad \searrow \\ \mathcal{T} \end{array}$ or “upside-down” as $\begin{array}{c} \nearrow \quad \nwarrow \\ \mathcal{T} \\ r \end{array}$ according to convenience.
- \rightarrow^+ is the transitive closure (page 60) of the \rightarrow relation and \rightarrow^* (page 59) is the reflexive-transitive closure of \rightarrow .

Fact -2.3: Elementary facts

Let $\langle N, \rightarrow, r \rangle$ be a rooted tree.

- Every node in N is source or a target of one or more edges,
- The \rightarrow relation is irreflexive (part 2 of definition -4.11) i.e. there is no edge whose source and target are the same node).
- The root node has no predecessor.

Let $\mathcal{T} = \langle N, \rightarrow, r \rangle$ be a rooted tree in the rest of this section.

Definition -2.3: Paths

- \mathcal{T} is also called a **tree rooted at r** .
- \mathcal{T} is infinite if N is an infinite set.
- A **path** in \mathcal{T} is a finite or (countably) infinite sequence $r = n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow \dots$ such that $(n_i, n_{i+1}) \in \rightarrow$ for each $i \geq 0$.
- A finite path $r = n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow \dots n_m$ is said to have a length $m \geq 0$. A path which is not finite is said to be **infinite**.
- A **branch** is a maximal path – it is either infinite or is finite $r = n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow \dots n_m$ such that n_m is a leaf.

Definition -2.4: Successors, predecessors, descendants, ancestors

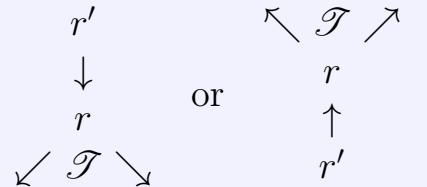
- The **successors** of a node $s \in N$ is the set $Succ(s) = \{t \in N \mid s \rightarrow t\}$ and **predecessors** of a node $t \in N$ is the set $Pred(t) = \{s \in N \mid s \rightarrow t\}$.
- The **descendants** of a node $n \in N$ is the set $Desc(n) = \{n\} \cup \bigcup_{s \in Succ(n)} Desc(s)$ and $Desc(n) - \{n\}$ is the set of **proper** descendants of n .
- The **ancestors** of a node n is the set $Ances(n) = \{n\} \cup \bigcup_{s \in Pred(n)} Ances(s)$ and the **proper** ancestors of n is the set $Ances(n) - \{n\}$.

Definition -2.5: Subtree

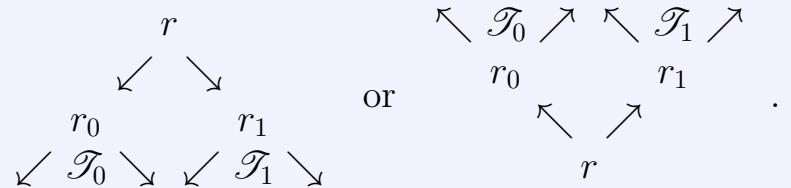
A **subtree** of a rooted tree $\mathcal{T} = \langle N, \rightarrow, r \rangle$ is a tree $\mathcal{T}' = \langle Desc(n), \rightarrow', n \rangle$ rooted at a node $n \in Desc(r)$ such that $s \rightarrow' t$ for $s, t \in Desc(n)$ if and only if $s \rightarrow t$.

Definition -2.6: Extensions

- A rooted tree $\mathcal{T} = \langle N, \rightarrow, r \rangle$ may be **extended at a leaf** n to another tree $\mathcal{T}' = \langle N'', \rightarrow', r \rangle$ by a set N' of nodes with edges $n \rightarrow' n'$ for each $n' \in N'$ provided $N' \cap N = \emptyset$, $N'' = N \cup N'$.
- A rooted tree $\mathcal{T} = \langle N, \rightarrow, r \rangle$ may be **extended at the root** r to another tree $\mathcal{T}' = \langle N', \rightarrow', r' \rangle$ by an edge $r' \rightarrow' r$ provided $r' \notin N$, $N' = N \cup \{r'\}$. \mathcal{T}' may be denoted



- Two trees $\mathcal{T}_0 = \langle N_0, \rightarrow_0, r_0 \rangle$ and $\mathcal{T}_1 = \langle N_1, \rightarrow_1, r_1 \rangle$ with $N_0 \cap N_1 = \emptyset$, may be **joined** together at a new node r to form a new tree $\mathcal{T} = \langle N, \rightarrow, r \rangle$ where $N = N_0 \cup N_1 \cup \{r\}$, $\rightarrow = \rightarrow_0 \cup \rightarrow_1 \cup \{r \rightarrow r_0, r \rightarrow r_1\}$. \mathcal{T} may be denoted



Fact -2.4

In any rooted tree $\mathcal{T} = \langle N, \rightarrow, r \rangle$,

1. $N = Desc(r)$
2. $Ances(n) = \{s \in N \mid s \rightarrow^* n\}$ for any $n \in N$.
3. $Desc(n) = \{t \in N \mid n \rightarrow^* t\}$ for any $n \in N$.
4. A rooted tree is acyclic i.e. for all nodes n , $n \not\rightarrow^+ n$.
5. If N is finite then \rightarrow is also finite.

Definition -2.7: Finite branching, infinitary node

- $\mathcal{T} = \langle N, \rightarrow, r \rangle$ is an **infinite** tree if N is infinite. Otherwise it is a **finite** tree.
- \mathcal{T} is said to be **finitely branching** if for each $n \in N$, $Succ(n)$ is a finite set.
- A node $n \in N$ is called **infinitary** if $Desc(n)$ is an infinite set otherwise it is called **finitary**.

Lemma -2.1: Infinitary-nodes

In a finitely branching tree, every infinitary node has an infinitary successor.

Proof: If not, then for some infinitary node n , $Succ(n)$ is finite and for each $s \in Succ(n)$, $Desc(s)$ is finite which implies $Desc(n) = \{n\} \cup \bigcup_{s \in Succ(n)} Desc(s)$ which by fact -4.8 is a finite union of finite sets. Then $Desc(n)$ would be finite leading to a contradiction. QED

Lemma -2.2: König's Lemma

Every finitely branching infinite tree has an infinite path.

Proof: Assume $\mathcal{T} = \langle N, \rightarrow, r \rangle$ is a finitely branching infinite tree which has no infinite path. Clearly since $N = Desc(r)$ is infinite, r is infinitary. r has an infinitary successor by lemma -2.1. Hence there exists a branch in \mathcal{T} all of whose nodes are infinitary. This path has to be infinite, otherwise there would be a last node in the path which is infinitary but has no successors, which is impossible. QED

Corollary -2.1

Every infinite tree is infinitely branching or finitely branching with at least one infinite path.

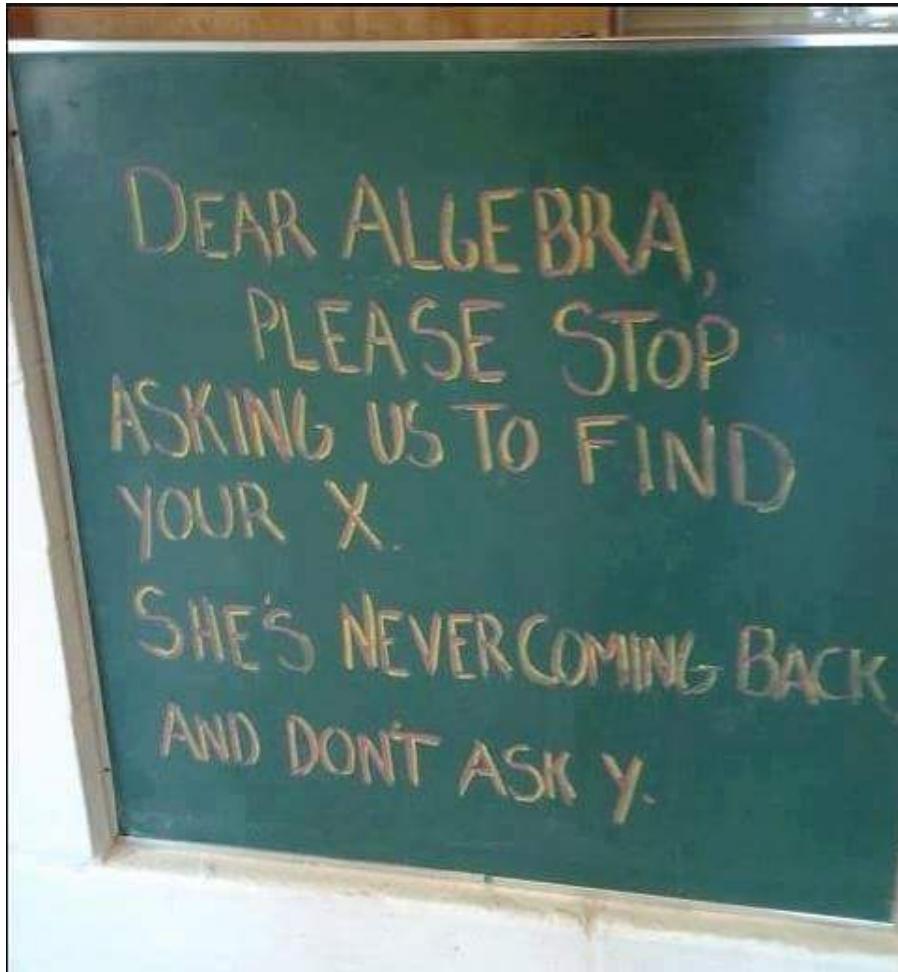
Corollary -2.2: Contrapositive of König's Lemma

A finitely branching tree is finite if and only if it has no infinite path.

Corollary -2.3: Infinitary root

The root node of any finitely branching infinite tree is infinitary.

-1. Introduction to Universal Algebra



From the internet

The basic notions of universal algebra are useful in understanding the essence of data types in object-oriented languages and modern functional languages like **ML** and logic programming languages like **Prolog**. This is especially so in the computations of values and functions in user-defined data-types, pattern-matching etc. Many of our examples in this section will use the data type facility in ML. We use the **typewriter** script for code written in a programming language and programming languages are colour-coded.

The few basic notions of universal algebra that we treat here will also be used in our expositions of model theory and proof theory which constitute the bulk of the course on mathematical logic. Further, since we are dealing with formal languages with certain fixed syntax, the issues related to compiling are better exposed in their generality using concepts from universal algebra. This enables us to treat syntax and semantics without worrying unduly about scanning and parsing techniques for them, by treating formal languages as algebraic systems.

-1.1. Data types in functional and declarative languages

We begin with a few examples of simple data types in **ML** and **Prolog**. The concepts that we treat here are however applicable to all formal languages. This includes all programming languages since any programming language (regardless of whether it is functional, declarative, imperative or object-oriented) can at its most basic level be considered a datatype that generates terms. Each program is simply a term generated by the data type. Each such term may be regarded as a linear representation of a syntax tree. The result of scanning and parsing a well-formed program is simply an abstract syntax tree.

Example -1.1: Simple data-type for naturals with addition

Consider the following **ML** data type which simulates the natural numbers^a through a data type definition.

```
datatype nat = z | s of nat | a of nat * nat
```

The data type **nat** is defined recursively to generate all patterns of terms which may be identified with the natural numbers. There may be more than one pattern to represent any given natural number. In this example we have a signature Ω consisting of a 0-ary function **z** (which is meant to represent the number 0), a 1-ary constructor **s** (meant to represent the successor function on the naturals) and a 2-ary **a** (meant to represent addition on natural numbers). This data type does not actually generate the naturals. It can be used to define patterns which satisfy some of the properties of naturals. Examples of patterns which belong to the data type **nat** are the following **z**, **s(z)**, **s(a(s(z), s(s(z))))** which may be thought of as representing the numbers 0, 1 and $4 = 1 + (1 + 2)$ respectively.

^aWe always include the number zero in the set of naturals

Example -1.2: Simple data-type for naturals with addition

The same definition in **Prolog** would look as follows:

```
nat(z).  
nat(s(X)) :- nat(X).  
nat(a(X, Y)) :- nat(X), nat(Y).
```

where **nat** is a property such that

- **nat** is true of **z**,
- **nat** is true of **s(X)** if **nat** is true of **X** for any **X** and
- **nat** is true of **a(X, Y)** if **nat** is true of **X** and **nat** is true of **Y**, for any **X** and **Y**.

Further in both languages it is understood that any object or entity which does not satisfy the formation rules in the data type definition does not belong to the datatype.

Example -1.3: Integer datatype

The following data type^a tries to simulate the notion of an integer (with 0, successor and predecessor functions) by means of patterns derived from the data type.

```
datatype integer = z | s of integer | p of integer
```

A pattern such as `p(s(s(p(s(z)))))` is meant to represent the integer value of the expression $0 + 1 - 1 + 1 + 1 - 1$ viz. the value 1. Notice that when we say “meant” to represent we are only expressing our intent about the use of the constructors. At the level of patterns the given pattern is completely different from other patterns which may be used to represent the value 1, such as `s(z)`, `p(s(s(z)))`, `s(p(s(z)))` etc. even though our intention may be to consider all these patterns as denoting the same number.

^ato be used quite independently of the data type `nat` defined in example -1.1, otherwise there will be name conflicts if they are both used in the same program.

Exercise -1.1

- Define the **Prolog** version of the **ML** data type `integer`.

Definition -1.1: Homogeneous algebra

A **homogeneous** or **1-sorted algebra A** is an ordered pair (A, \mathcal{F}) where A is called the **carrier set** and \mathcal{F} is the set of **operators/functions/constructors** (with each $f \in \mathcal{F}$ having a predefined arity) such that A is closed under the operators of \mathcal{F} . We will often refer to A as the algebra when \mathcal{F} is understood.

The term “1-sorted” or “homogeneous” refers to the fact that there is a single carrier set A and A is closed under all the operations of \mathcal{F} i.e. there is only one kind of element viz. elements of A and each operation $f \in \mathcal{F}$ of arity $n \geq 0$ is of the

kind $f : A^n \longrightarrow A$.

Example -1.4: Revisiting example -1.1

The data type defined in example -1.1 gives us a 1-sorted algebra (A, \mathcal{F}) where the carrier set is the set of all patterns involving **z**, **s** and **a** which respect the arities of the constructors. $\mathcal{F} = \{\mathbf{z}, \mathbf{s}, \mathbf{a}\}$. Here the notion of a sort refers to the carrier set A .

The set of patterns generated by a recursively defined data type could be infinite. Even if it is finite but large it is usually useful to introduce the concept of a *variable* as a means of naming either arbitrary patterns of certain kinds or as devices for representing and reasoning about unknown patterns. Since we do not want to be limited by the number of names we may require, we assume the existence of an infinite set V of names all distinct from any of the symbols of the data type.

Example -1.5: Patterns with variables

We might be interested in general patterns of the form $p(s(\dots))$ from the data type **integer** defined in example -1.3. We may represent them generally as terms of the form $p(s(x))$ where $x \in V$. A variable is in general a place-holder for patterns that we do not want to specify.

Example -1.6: Patterns of computation rules

Similarly we may want to specify a general computation rule which specifies that any pattern of the form $a(s(\dots), \dots)$ in example -1.1 should be replaced by $s(a(\dots, \dots))$ where the \dots and the \dots themselves represent arbitrary patterns from the data type **nat**. It is then more convenient to use different variables from V to represent them. We could then specify that terms of the form $a(s(x), y)$ should be replaced by terms of the form $s(a(x, y))$. The patterns represented by the pattern **x** and **y** are both arbitrary (they could therefore be completely different or even (coincidentally!) the same pattern).

Example -1.7: Patterns of computation rules

Note that in $s(a(x, x))$ where both place-holders have the same name, we are saying that even though x represents an arbitrary pattern, each instance of $s(a(x, x))$ must have identical copies in both places!

We may generalise these notions to a many-sorted algebra where there are a finite number of sorts and a collection of operations where each operation has a predefined type specifying its domain and co-domain.

Definition -1.2: Signature

A **signature** Ω is a pair $\langle S, \mathcal{F} \rangle$ where

- S is a finite nonempty set of **sorts** representing the various carrier sets and
- \mathcal{F} the (possibly empty) set of **function symbols** such that
- \mathcal{F} is equipped with a mapping $type : \mathcal{F} \longrightarrow (S^* \longrightarrow S)$, where each element of S^* is of the form $s_1 \times s_2 \times \cdots \times s_n$ for each $n \geq 0$.
- For each $f \in \mathcal{F}$, $type(f)$ is called the **type** of f ,
- $sorts(\Omega) = S$ and $opns(\Omega) = \mathcal{F}$.

Note that a constant $c \in s$ where $s \in S$ is a function symbol of type $\epsilon \longrightarrow s$ where $\epsilon \in S^*$ denotes the empty cartesian product of S .

Example -1.8: 1-sorted algebras

1. $\Omega_{\mathbb{B}_0} = \langle \{\{\text{true}, \text{false}\}\}, \emptyset \rangle$ is the signature of boolean values. It is a 1-sorted algebra with no operations at all.
2. $\Omega_{\mathbb{B}_1} = \langle \{\text{true}, \text{false}\}, \{-, ., +\} \rangle$ is the signature of a boolean algebra with the usual operations.
3. $\Omega_{\mathbb{Z}_0} = \langle \{\mathbb{Z}\}, \emptyset \rangle$ is the signature of integer values. it is a 1-sorted algebra with no operations.
4. $\Omega_{\mathbb{Z}_1} = \langle \{\mathbb{Z}\}, \{p, s\} \rangle$ is the 1-sorted algebra of integers where p and s stand for the unary *predecessor* and *successor* functions respectively.
The two functions satisfy the following identities for all integers x .

$$p(s(x)) = x = s(p(x)) \quad (9)$$

Definition -1.3: Homogeneous algebra

Let Ω be a 1-sorted signature. A structure $\mathbf{A} = \langle \mathbb{A}, \Omega \rangle$ is called a **1-sorted or homogeneous algebra** if the carrier set \mathbb{A} is closed under the operations in Ω i.e. every constant in Ω is present in \mathbb{A} and every operation on the elements of \mathbb{A} is a total function whose co-domain is \mathbb{A} .

Example -1.9: Homogeneous algebras

Let $+, \times, -$ denote the usual operations on numbers.

- $\mathbf{N} = \langle \mathbb{N}; \{+, \times\}; \{=\} \rangle$ is a homogeneous algebra since the set of naturals is closed under both addition and multiplication.
- $\mathbf{N}' = \langle \mathbb{N}; \{+, \times, -\}; \{=\} \rangle$ is however not closed under subtraction and hence \mathbf{N}' is not a homogeneous algebra.
- $\mathbf{Z} = \langle \mathbb{Z}; \{+, \times, -\}; \{=\} \rangle$ is a homogeneous algebra.
- $\mathbf{n} = \langle \mathbb{n}; \{+_n, \times_n, -_n\}; \{=\} \rangle$ is a homogeneous algebra for any positive integer n (see notation -4.1) with $+_n, \times_n, -_n$ denoting the corresponding arithmetic operations modulo n .

Definition -1.4: Precongruence and Congruence

Let $\mathbf{A} = \langle \mathbb{A}, \Omega \rangle$ be a homogeneous algebra.

- A preorder (see definition -4.12) $\leq \subseteq \mathbb{A} \times \mathbb{A}$ is called a **pre-congruence** if it is *preserved* by every operator in Ω i.e. for each operator $o \in \Omega$ of arity $n > 0$, $a_1 \leq b_1, \dots, a_n \leq b_n$ implies $o(a_1, \dots, a_n) \leq o(b_1, \dots, b_n)$.
- Likewise an equivalence relation $\approx \subseteq \mathbb{A} \times \mathbb{A}$ is a **congruence** if it is preserved by every operator i.e. for each operator $o \in \Omega$ of arity $n > 0$, $a_1 \approx b_1, \dots, a_n \approx b_n$ implies $o(a_1, \dots, a_n) \approx o(b_1, \dots, b_n)$.

Exercise -1.2

1. Prove that the equality relation $=$ in each of \mathbf{N} , \mathbf{Z} , \mathbf{n} of example [-1.9](#) is a congruence.
2. Prove that the usual \leq relation on \mathbb{Z} of the algebra \mathbf{Z} is a precongruence.
3. Prove that \subseteq is a precongruence in the algebra $\mathbf{U} = \langle 2^{\mathbb{U}}; \{\cup, \cap, -\}; \{\subseteq, =\} \rangle$
4. Consider the algebra \mathbf{Z} in example [-1.9](#). For any integer $n > 1$ let $=_n$ be the relation $a =_n b$ if and only if $a \bmod n = b \bmod n$.
 - (a) Prove that $=_n$ is an equivalence relation on \mathbb{Z} .
 - (b) Prove that $=_n$ is a congruence.
 - (c) Let $a \leq_n b$ if and only if $a \bmod n = b \bmod n$. Prove or disprove that \leq_n is a precongruence.
5. Let \mathbb{U} be a non-empty finite set containing $n > 1$ elements. Further for all $A, B \in 2^{\mathbb{U}}$ let $A \sim B$ if and only if $|A| = |B|$. Prove that \sim is an equivalence relation, but many not be a congruence in the algebra \mathbf{U} of problem 3.

Example -1.10: Many-sorted algebras

1. Consider $\Omega_{\mathbb{B}\mathbb{Z}_1} = \langle \{\mathbb{B}, \mathbb{Z}\}, \{p, s, \neg, ., +, <\} \rangle$ which is a 2-sorted algebra containing both booleans and integers and including all the operations as defined in $\Omega_{\mathbb{B}_1}$ and $\Omega_{\mathbb{Z}_1}$ (parts 2 and 4 respectively) of example -1.8. In addition, there is another operation $< : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$ which stands for the usual “less-than” relation on the integers. For example, then the following are (some of the) identities for all integers x .

$$p(x) < x = \text{true} \quad (10)$$

$$x < s(x) = \text{true} \quad (11)$$

$$(p(x) < x) . (x < s(x)) = \text{true} \quad (12)$$

$$x < p(x) = \text{false} \quad (13)$$

Definition -1.5: Terms and ground terms

Let Ω be a signature and let V be an infinite set of variables such that $\Omega \cap V = \emptyset$. The set $\mathbb{T}_\Omega(V)$ of all Ω -terms over V is defined inductively as the smallest set of terms such that

Basis $V \subseteq \mathcal{T}_\Omega(V)$

Induction Step If $f \in \Omega_n$ for $n \geq 0$ and $t_1, \dots, t_n \in \mathbb{T}_\Omega(V)$, then $f(t_1, \dots, t_n) \in \mathbb{T}_\Omega(V)$

The set of **ground terms** denoted \mathbb{T}_Ω is the set of all terms constructed which do not involve any variables.

The above definition is usually more concisely written as

$$s, t \in \mathcal{T}_\Omega(V) ::= x \in V \mid f(t_1, \dots, t_n)$$

Note.

- $\mathbb{T}_\Omega(V)$ is never empty.
- If Ω contains no constants then $\mathbb{T}_\Omega = \emptyset$

-1.2. Terms and Term Algebras

We assume a countably infinite set of *variables/names* V (ranged over by x, y, z possibly decorated) and a set Ω called the *signature* or the set of *operators* (ranged over by $o \in \Omega$) such that $V \cap \Omega = \emptyset$. Each *operator* $o \in \Omega$ has a fixed arity $\alpha(o) \in \mathbb{N}$. The operators with arity 0, if any, are the *constants*. $\Sigma = V \cup \Omega$ is the set of *symbols*⁷.

Definition -1.6: Terms

The set of **terms** is the set denoted $\mathbb{T}_\Omega(V)$ and defined inductively by the BNF

$$s, t, u \in \mathbb{T}_\Omega(V) ::= x \in V \mid o(t_1, \dots, t_n) \quad (14)$$

for each $o \in \Omega$ such that $\alpha(o) = n \geq 0$ and $t_1, \dots, t_n \in \mathbb{T}_\Omega(V)$.

- Each term t_i , $1 \leq i \leq n$ in $o(t_1, \dots, t_n)$ is called a **proper subterm** of $o(t_1, \dots, t_n)$.
- Every term is a **subterm** of itself.
- The set of **ground terms** is the set $\mathbb{T}_\Omega = \mathbb{T}_\Omega(\emptyset) \subseteq \mathbb{T}_\Omega(V)$. Equivalently, the set of ground terms is precisely the *variable-free* subset of $\mathcal{T}_\Omega(V)$.

\mathbb{T}_Ω is nonempty precisely when there is at least one constant in Ω .

⁷Besides these symbols it is usual to have grouping, punctuation, association and scoping symbols which facilitate a linear reading of the term. However since every term defines a unique abstract syntax tree which render these other symbols redundant we regard them purely as *lexical decorations* and not as symbols.

Notation -1.1

- Often a constant term $c() \in \mathbb{T}_\Omega(V)$ will be simply written as c .
- Syntactic identity of terms will be denoted by the symbol \equiv . In particular if references x and y refer to the same variable then $x \equiv y$ and if they refer to different variables then $x \not\equiv y$.
- Similar remarks apply for references to terms as well.

Notice that the set of terms $\mathbb{T}_\Omega(V)$ generated by the BNF (14) is closed under the operators of Ω , by definition (-1.6). This leads us to the notion of a term algebra

Definition -1.7: Term algebra

$\langle \mathbb{T}_\Omega(V), \Omega, \{\equiv\} \rangle$ is the **term algebra** of the language defined by the BNF (14).

Example -1.11: Term algebra of integers

Consider the term algebra $\langle \mathbb{T}_\Omega(V), \Omega, \{\equiv\} \rangle$ generated by the set of integers where each integer is a constant and the operators are the binary operators of addition, multiplication and subtraction i.e $\Omega = \mathbb{Z} \cup \{+, *, -\}$. In this example $\alpha(+)=\alpha(*)=\alpha(-)=2$. $(a+b)/(c*(d-5))$ is an example of a term generated by the BNF (14) where a, b, c and d are variables. Figure 8 depicts the abstract syntax tree of this expression.

In many term algebras such as the λ -calculus, first-order logic, the signature could be infinite because it may contain *operator-schemas* i.e. operators that are parameterized over variables (e.g. $\lambda x.$ and $\forall x$ etc.).

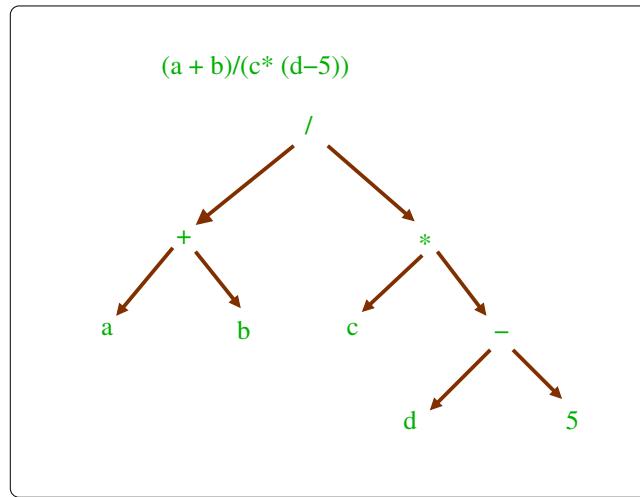


Figure 8: Abstract syntax tree of an integer arithmetic expression

Definition -1.8: Term algebras with operator schemas

Let Ω consist of operators (ranged over by o) and operator schemas (ranged over by O). Then the set of terms $\mathbb{T}_\Omega(V)$ is defined by the BNF.

$$s, t, u \in \mathbb{T}_\Omega(V) ::= x \in V \mid o(t_1, \dots, t_n) \mid O x [t] \quad (15)$$

for each $o \in \Omega$ such that $\alpha(o) = n \geq 0$ and $t_1, \dots, t_n \in \mathbb{T}_\Omega(V)$ and each $O \in \Omega$, where O is an operator schema and $x \in V$.

Definition -1.9: Free and bound variables in terms

Let $\mathbb{T}_\Omega(V)$ be the set of terms generated by the BNF (15)

- Ox is called a **binding** occurrence of the variable x ;
- the brackets [and] are used to delimit the **scope** of the bound variable x , and all occurrences of x in the subterm t of the term $Ox[t]$ are said to be **bound**.
- All occurrences of variables in the term $o(t_1, \dots, t_n)$ occur **free** unless they are bound in any of the sub-terms t_i .

In general we assume the usual rules of *binding* occurrences of variables and *lexical scoping* of bound variables i.e. any occurrence of a variable, if not free, is bound to the innermost enclosing scope which contains a binding occurrence of the variable. In terms of abstract syntax trees each occurrence of a bound variable refers to the binding occurrence that is closest to it in the path leading to the occurrence from the root of the tree (see examples -1.13, -1.14 and -1.15).

Each such operator schema along with its parameter (drawn from V) is regarded as a unary operator (operator schemas with multiple parameters can usually be redefined using one variable parameter at a time). Examples of such operator schemas in school mathematics include summations and products which usually use an indexing variable.

Example -1.12: Operator schemas

Consider the term $\sum_i (a_i \times b_i)$ where the indexing variable i is a parameter of operator (schema) \sum and is hence bound to it, whereas a_i and b_i are free variables. The same term would be rendered linearly in the notation of BNF (15) as $\sum i[a_i * b_i]$ where the scope of the binding of i is delimited by the brackets. However in the term $\sum i[a_i] * b_i$, i occurs *bound* to the operator \sum in a_i , but occurs *free* in b_i .

Definition -1.10: Free and bound variables

For any term $s \in \mathbb{T}_\Omega(V)$, $FV(s)$ denotes the set of *free* variables of s and $Var(s)$ denotes the set of all variables (free and bound) that occur in s . These functions may be defined by induction on the structure of terms as follows.

s	$FV(s)$	$Var(s)$
$c()$	\emptyset	\emptyset
x	$\{x\}$	$\{x\}$
$o(t_1, \dots, t_n)$	$\bigcup_{1 \leq i \leq n} FV(t_i)$	$\bigcup_{1 \leq i \leq n} Var(t_i)$
$Ox[t]$	$FV(t) - \{x\}$	$Var(t) \cup \{x\}$

where $c()$ is a constant.

In such term algebras the usual notions of α -conversion and α -equivalence (denoted \equiv_α) may be used to provide an unambiguous but equivalent rendering of the term so that free and bound variables are kept disjoint. Going back to the summation example -1.12 we have $\sum_i [a_i * b_i] \equiv_\alpha \sum_j [a_j * b_j]$ and $\sum_i [a_i] * b_i \equiv_\alpha \sum_j [a_j] * b_i$. Notice that in the latter case

since b_i lies outside the scope of the binding the subterm b_i remains unaltered.

What we have treated here and in the sequel are *homogeneous* term algebras, wherein it is assumed that every interpretation of the terms of the algebra refers to a singel carrier set. A more refined form of a term algebra may be one where the bound variables may be *constrained* in some way, either through a language of constraints such as types (e.g. the ML programming language) or some predicate that defines the constraints on bound variables and operators (such as bounded quantification logic). A BNF for bindings with constraints may be defined as follows:

$$s, t, u \in \mathbb{T}_\Omega(V) ::= x \in V \mid o(t_1, \dots, t_n) \mid Ox : C[t] \quad (16)$$

where C belongs to some language of constraints. One example of such constrained terms is the term $\sum_{i=1}^n a_i * b_i$ where the bound variable i is constrained to vary between the limits 1 and n , where n is a new free variable. This term would be rendered in the BNF (16) as $\sum i : 1 \leq i \leq n [a_i * b_i]$ where $1 \leq i \leq n$ is itself a term in some suitably defined language of constraints.

Notes.

- $\equiv \subseteq \equiv_\alpha$ in algebras with operator schemas, whereas in term algebras that have no operator schemas the two notions are identical (see examples 9 and 10 below).
- We have used names from the set V as place-holders for both free and bound variables. It is possible to use *name-free* representations as done in the lambda calculus. These representations have the following advantages:
 - they separate free variables from bound variables in a term and keep the two sets disjoint,

- they separate bound variables from other bound variables which in a named representation may have the same name⁸.
- they make the use of α -conversion redundant since any two terms are α -equivalent iff they have syntactically identical name-free representations. See figure 11 for a name-free representation of the lambda term in figure 10.
- In term algebras with operator schemas, the binding mechanism Ox is to be regarded as a single position in the term with a “single symbol” Ox .

The terms of any set $T_\Omega(V)$ may be regarded as abstract syntax trees (AST). The brackets are used merely to provide an unambiguous linear rendering of the AST.

Example -1.13: Abstract syntax tree of a FOL-predicate

(See figure 9). Let p , q and r be unary predicates. The first-order logic formula

$$\forall x[p(x) \wedge \exists x[q(x)]] \vee r(x)$$

has two distinct bound variables called x . In addition there is also a free variable x . All three of them are distinct. While the free variable is simply referred to as x the two bound variables may be referred to as $(x, 0)$ and $(x, 1)$ respectively where the second component in each reference refers to the depth of nesting of the bindings of the two variables in the term. Alternatively, the second component may refer to the unique positions at which they have their respective binding occurrences in the formula (see example -1.16 below).

⁸For instance we could have terms of the form $\sum_i (a_i \times (\prod_i b_i))$ which is α -equivalent to $\sum_j (a_j \times (\prod_k b_k))$.

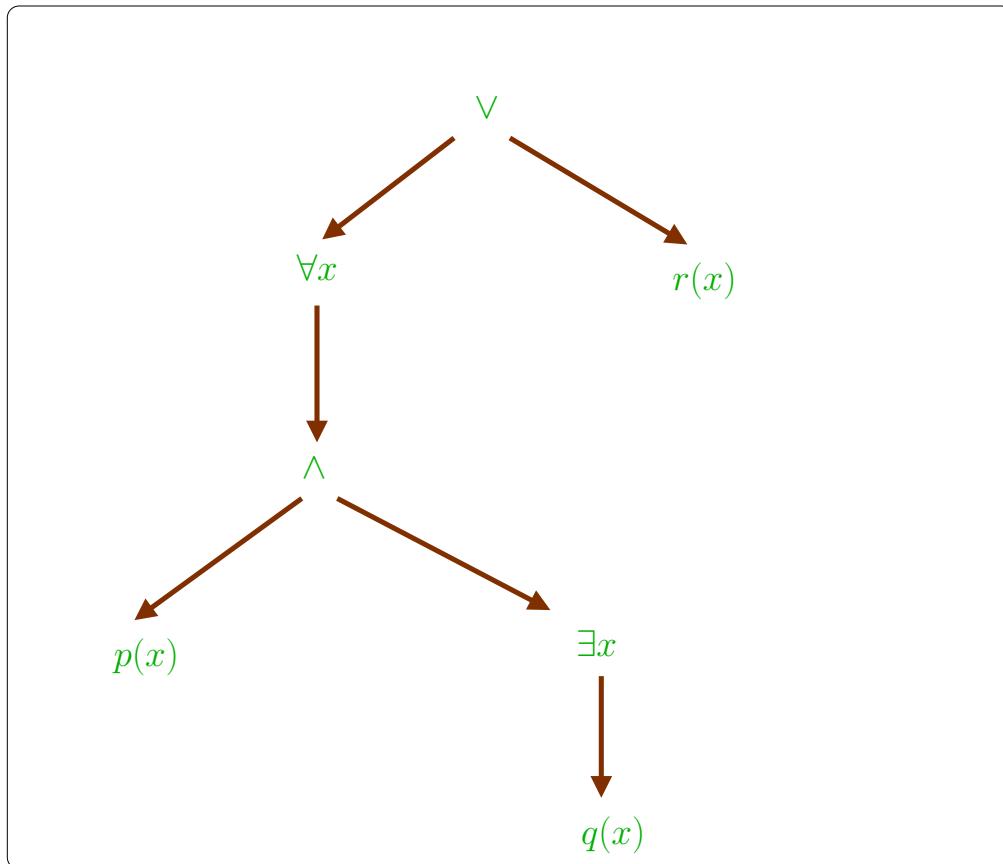


Figure 9: Abstract syntax tree of a FOL formula

Example -1.14

Figure 10 shows the abstract syntax tree of a term in the λ -calculus, where $()$ is used as the symbol for application.

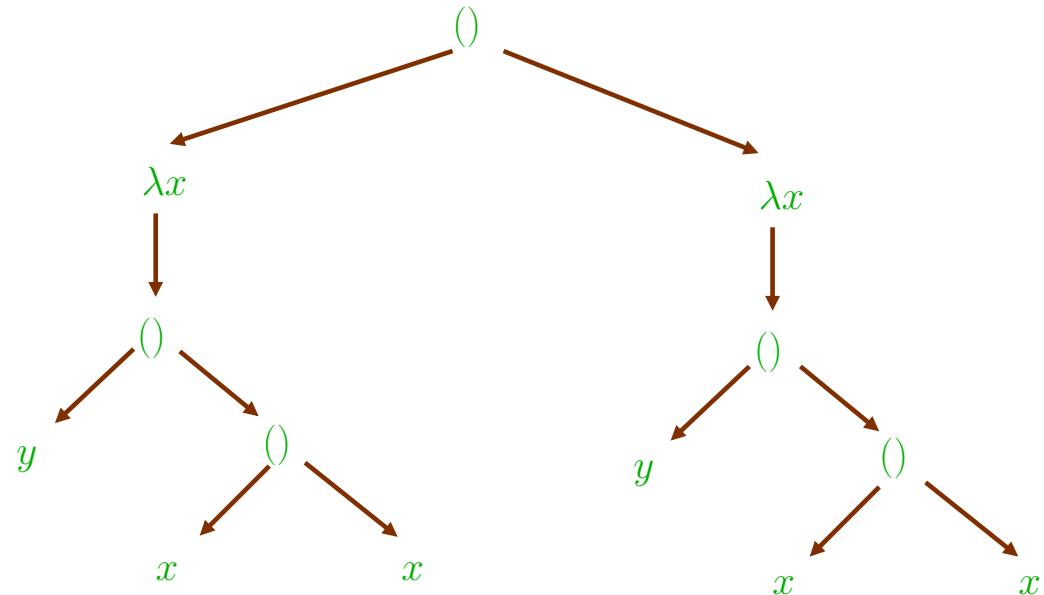


Figure 10: Abstract syntax tree of the λ -term $(\lambda x[(y (x x))] \lambda x[(y (x x))])$

Example -1.15

The name-free representation of the λ -term in example -1.14 is shown in figure 11. The red arrows indicate the variable to be filled in by referring to the position at which the variable is bound. In linear text the name-free representation would be something like $(\lambda[(y (0 0))]\ \lambda[(y (0 0))])$ where the integer 0 indicates that the bound variable is the one declared at the beginning of the current scope. A term such as $\lambda x[\lambda y[(y x)]]$ would have the representation $\lambda[\lambda[(0 1)]]$.

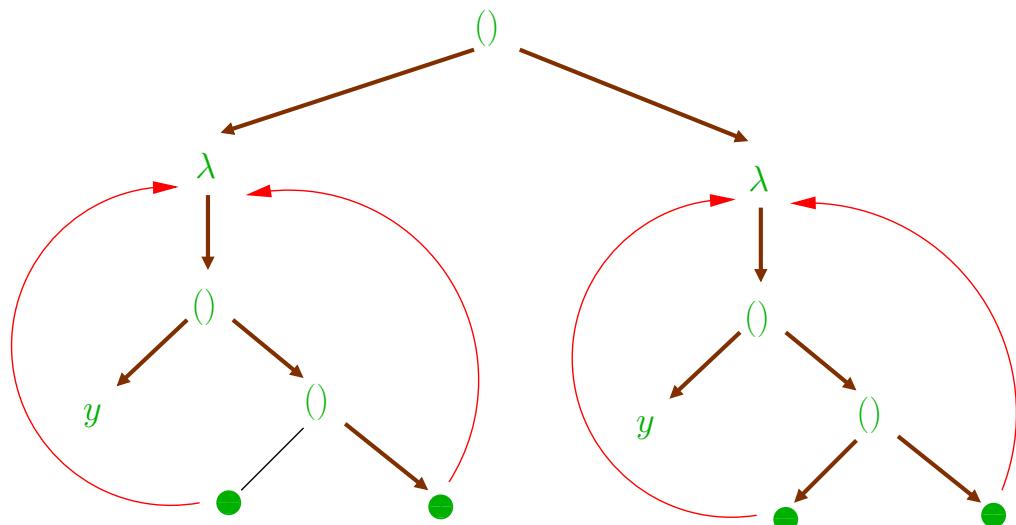


Figure 11: Abstract syntax tree of the name-free λ -term $(\lambda[(y (0 0))]\ \lambda[(y (0 0))])$

We will use words like “occurrence”, “sub-term”, “depth”, “size” quite liberally. In the light of the presence of several occurrences of operators, free variables and bound variables (including different bound variable occurrences signifying different

variables but possessing the same name e.g. $(\lambda x[(x\ x)]\ \lambda x[(x\ x)])$ in a term, it is useful to define a unique *position* for each symbol in a term t . For any term t we have a set of strings $Pos(t) \subseteq \mathbb{N}^*$ which is the set of positions occurring in t . Further for each $p \in Pos(t)$, there is a unique symbol occurring at that position denoted by $pos(p, t)$.

For each term t , $ST(t)$ denotes the set of subterms of t (including t itself). The set of *proper subterms* of t is the set $ST(t) - \{t\}$.

Definition -1.11: *depth*, *size*, *ST*, *Pos*

Let $t \in T_\Omega(V)$. The following are definitions by induction on the structure of t .

t	$depth(t)$	$size(t)$	$ST(t)$	$Pos(t)$
$c()$	1	1	$\{t\}$	$\{\epsilon\}$
x	1	1	$\{t\}$	$\{\epsilon\}$
$o(t_1, \dots, t_n)$	$1 + \text{Max}_{i=1}^n depth(t_i)$	$1 + \sum_{i=1}^n size(t_i)$	$\{t\} \cup \bigcup_{i=1}^n ST(t_i)$	$\{\epsilon\} \cup \bigcup_{i=1}^n i.Pos(t_i)$
$Ox[s]$	$1 + depth(s)$	$1 + size(s)$	$\{t\} \cup ST(s)$	$\{\epsilon\} \cup i.Pos(s)$

Notation -1.2

- The functions given in the table above are defined by induction on the structure of term t .
- ϵ is the empty word on strings, $.$ is the catenation operator on strings and $i.Pos(t_i) = \{i.p \mid p \in Pos(t_i)\}$.

Definition -1.12: Subterm ordering

$s \sqsubseteq t$ iff $s \in ST(t)$. \sqsubseteq is the **subterm** relation on terms. s is a **proper subterm** of t (denoted $s \sqsubset t$) iff $s \sqsubseteq t$ and $s \not\equiv t$.

Notation -1.3: Positions, subterms, occurrences

- For any t , the subterm at position $p \in Pos(t)$ is denoted $t|_p$ and defined by induction on p as follows: $t|_\epsilon \equiv t$, and for $t \equiv o(t_1, \dots, t_n)$, $t|_{i.p'} \equiv t_i|_{p'}$ if $p = i.p' \in Pos(t)$
- For any term t and any position $p \in Pos(t)$, $sym(p, t)$ yields the symbol at position p in the term t .
- The position ϵ is called the **root position** and the symbol at the root position is called the **root symbol**. Hence $rootsym(t) = sym(\epsilon, t)$ and for any position $p \in Pos(t)$, $sym(p, t) = rootsym(t|_p)$.
- The set of occurrences of a symbol $\sigma \in \Omega \cup V$ in a term is defined as the set $Occ(\sigma, t)$ of positions in which that symbol occurs i.e. $Occ(\sigma, t) = \{p \in Pos(t) \mid sym(p, t) = \sigma\}$. For any operator schema O and variable x , $Ox \in \Omega$ is considered a single operator symbol and therefore does not count as an occurrence of the variable x .

The *prefix* ordering on \mathbb{N}^* (defined as $p \preceq q$ iff there exists a string $r \in \mathbb{N}^*$ such that $p.r = q$) is a partial order on strings. The *proper* prefix ordering ($p \prec q$ iff $p \preceq q$ and $p \neq q$) is an irreflexive and transitive ordering.

Fact -1.1

1. For any term t , $\text{size}(t) = |\text{Pos}(t)|$.
2. \sqsubseteq is a partial order on $\mathcal{T}_\Omega(V)$ i.e. it is reflexive, transitive and anti-symmetric.
3. For any $p, q \in \text{Pos}(t)$,
 - (a) $p \preceq q$ iff $t|_q \sqsubseteq t|_p$ and
 - (b) $p \prec q$ iff $t|_q \subset t|_p$.

Example -1.16: *sym*

Let $t \equiv \lambda x[\lambda y[(x\ y)]]$ be a λ -term. Then

$$\begin{aligned}
 \text{Pos}(t) &= \{\epsilon, 1, 1.1, 1.1.1, 1.1.2\} \\
 \text{sym}(\epsilon, t) &= \lambda x \\
 \text{sym}(1, t) &= \lambda y \\
 \text{sym}(1.1, t) &= () \\
 \text{sym}(1.1.1, t) &= x@\epsilon \\
 \text{sym}(1.1.2, t) &= y@1
 \end{aligned}$$

We have used the string “ $y@1$ ” to represent the fact that the symbol at the given position is a variable named y bound at the position 1. Similarly “ $x@\epsilon$ ” denotes the variable x bound at the position ϵ .

Even though our examples will have operators of single digit arities, theoretically it is possible to have operators of arities of more than one digit. So we write “1.1”, instead of “11” using the catenation operator “.” as a separator between different

elements of \mathbb{N} in strings of \mathbb{N}^* .

Example -1.17: Pos

Consider the lambda term $u \equiv (\lambda x[(y(x\ x))]\ \lambda x[(y(x\ x))])$. We then have

$$Pos(u) = \{\epsilon, 1, 2, 1.1, 2.1, 1.1.1, 1.1.2, 2.1.1, 2.1.2, 1.1.2.1, 1.1.2.2, 2.1.2.1, 2.1.2.2\}$$

Example -1.18: Occurrences

In the previous example -1.17 we have

$$Occ(\textcolor{brown}{y}, u) = \{\epsilon, 1.1, 2.1, 1.1.2, 2.1.2\}$$

and

$$Occ(\textcolor{brown}{y}, u) = \{1.1.1, 2.1.1\}$$

However there are two different bound variables which happen to have the same name. Each of them is called " x " and they have binding occurrences at positions 1 and 2 respectively. We refer to them respectively as $x@1$ and $x@2$. Each of these variables has the occurrences given by

$$Occ(x@1, u) = \{1.1.2.1, 1.1.2.2\}$$

and

$$Occ(x@2, u) = \{2.1.2.1, 2.1.2.2\}$$

respectively.

Note. For any bound variable $x@p$ in a term t ,

1. $p \notin \text{Occ}(x@p, t)$ i.e. $\text{O}x$ is not considered an occurrence of the variable x though it is a binding of x
2. for any $q \in \text{Occ}(x@p, t)$, $p \prec q$, by the conventions governing lexical scoping

Exercise -1.3: Cardinality of term algebras

Let $\mathbb{T}_\Omega(V)$ be any term algebra.

1. Prove that \mathbb{T}_Ω is empty if Ω has no constants.
2. Prove that if Ω has at least one constant and one function symbol of arity at least 1 then \mathbb{T}_Ω is countable.
3. Prove that $\mathbb{T}_\Omega(V)$ is countable if Ω is at most countable.

-1.3. Semantics and Meaning

We normally communicate by using a language. In the case of mathematics, we often need a (formal) language to express various arguments about a mathematical structure. For simplicity we assume that our mathematical models are algebraic systems and there is a need to express the properties of such a system using a formal language. Clearly therefore every distinguished object in the model needs to have some symbol associated with it and every distinguished operation or relation needs to have a symbol of the same arity to be associated with it.

Conversely, to every symbol in a formal language we need to associate a meaning in the mathematical domain in which we wish to express and argue about its properties. We know that every formal language automatically defines a term algebra.

However the terms of such an algebra need to be given precise meanings. In general the term algebra would be some countably infinite set (see problem 3 in exercise -1.3) and to specify the meaning we would need to specify it inductively to be able to include all the terms. Further the meaning should preferably be specified using a model which in turn is an algebra (not necessarily a term algebra) with a “similar” structure.

Example -1.19: Plane geometry constructions of segments

The straight-edge and compass constructions in geometry may be specified algebraically as a term algebra with the following operators. Even though lengths of segments are (non-negative) real numbers (an uncountable set), for the purpose of describing constructions, it suffices to have a countable set of terms generated from a small finite set of positive real numbers. To keep it simple we use two positive real constants a and b and assume $a > b$. Now Consider the BNF

$$s, t, u ::= a \mid b \mid 0 \mid 1 \mid (s + t) \mid (s - t) \mid (s \times t) \mid (s/t) \mid (r.t) \quad (17)$$

where

- a, b are positive reals representing some given measures of lengths in some construction. These numbers could be irrational or transcendental as well.
- The constant 0 to represent the length of degenerate segments (i.e. segments whose end-points coincide),
- The constant 1 to represent a unit length,
- The binary operation $\cdot : \mathbb{Q}_+ \times \mathbb{R} \rightarrow \mathbb{R}$ to compute (improper) fractions of the given reals. For example we may need to construct segments of length $(\frac{3}{2}.a)$
- The binary $+, \times$ and $/$ operations on positive real numbers,
- The partial binary operation $- : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ to subtract smaller numbers from larger ones. Negative lengths do not occur in euclidean geometry.

Definition -1.13: Similar signatures

Two signatures Ω_1 and Ω_2 are said to be **similar** if there is an arity-preserving bijection between them. That is, there exists a bijection $\mu : \Omega_1 \xrightarrow[\text{onto}]{1-1} \Omega_2$ such that for every $o_1 \in \Omega_1$ with $\alpha(o_1) = n \geq 0$ there exists a unique $o_2 \in \Omega_2$ with $\alpha(o_2) = n \geq 0$ and vice-versa. We write $\mu(o_1) = o_2$ and $\mu^{-1}(o_2) = o_1$.

Definition -1.14: Similar algebras

Algebras $\mathbf{A}_1 = \langle \mathbb{A}_1, \Omega_1, \{=_1\} \rangle$ and $\mathbf{A}_2 = \langle \mathbb{A}_2, \Omega_2, \{=_2\} \rangle$ are said to be **similar algebras** if their signatures are similar.

In general, the bijection μ is understood or obvious since the operator symbols are usually “overloaded” to save on explicitly defining the correspondence.

Note that the carrier sets \mathbb{A}_1 and \mathbb{A}_2 may not be isomorphic even if the algebras are similar. We use this fact to define the notion of a semantics or meaning of the sentences of a formal language.

Definition -1.15: Homomorphism

Let $\mathbf{A}_1 = \langle \mathbb{A}_1, \Omega_1, \{=_1\} \rangle$ and $\mathbf{A}_2 = \langle \mathbb{A}_2, \Omega_2, \{=_2\} \rangle$ be similar algebras with $\mu : \Omega_1 \xrightarrow[\text{onto}]{1-1} \Omega_2$. A function $h : \mathbb{A}_1 \longrightarrow \mathbb{A}_2$ is called a **homomorphism** if for all $n \geq 0$, operators o_1 of arity n and elements $a_1, \dots, a_n \in \mathbb{A}_1$,

$$h(o_1(a_1, \dots, a_n)) =_2 \mu(o_1)(h(a_1), \dots, h(a_n))$$

A formal language is usually defined to linguistically represent or express objects of some intended model, along with the operations in the model. This needs to be done unambiguously and completely, so that

1. a sentence in the language represents or expresses a unique object in the model (unambiguously) in the context in which it is used, and
2. every sentence in the language does indeed represent an object in the model.

Exercise -1.4

Let $h : \mathbb{A}_1 \xrightarrow{\text{onto}} \mathbb{A}_2$ be a surjective homomorphism between two similar algebras \mathbf{A}_1 and \mathbf{A}_2 .

1. Prove that h induces an equivalence relation \sim_h on \mathbb{A}_1 .
2. Prove that the induced equivalence \sim_h is a congruence (see definition -1.4) on the algebra \mathbf{A}_1 .

-1.3.1. Semantics of ground terms

Let Ω be a signature with at least one constant symbol. Then the set of ground (or variable-free) terms, \mathbb{T}_Ω is non-empty. The meaning of a ground term is simply the value that the term is intended to represent. Hence we may define the meaning of a ground term by induction on the structure of the term.

It is usual in any discussion on semantics, to enclose purely syntactical elements within $\llbracket \rrbracket$ to distinguish them from the semantic elements. The reader may safely read it as simply pair of decorated brackets.

Definition -1.16: Semantics of ground terms

Given an algebra of terms with at least one constant symbol $\mathbf{T}_\Omega(V) = \langle \mathbb{T}_\Omega(V), \Omega, \{\equiv\} \rangle$ and another similar algebra $\mathbf{M} = \langle \mathbb{M}, \Omega', \{=\} \rangle$ with the bijection μ as in definition (-1.13) between their signatures. Then a function

$$\mathcal{V} : \mathbb{T}_\Omega \longrightarrow \mathbb{M}$$

is called a **meaning** or **interpretation** if the following conditions are satisfied.

1. For each constant $c \in \Omega$, $\mathcal{V}[c] = \mu[c]$ and
2. For each operator $o \in \Omega$ with $\alpha(o) = n > 0$ and ground terms $g_1, \dots, g_n \in \mathbb{T}_\Omega$,

$$\mathcal{V}[o(g_1, \dots, g_n)] = \mu(o)(\mathcal{V}[g_1], \dots, \mathcal{V}[g_n])$$

3. For each operator schema $O \in \Omega$, $x \in V$ and ground term $g \in \mathbb{T}_\Omega$,

$$\mathcal{V}[Ox[g]] = \mathcal{V}[g]$$

In the last part of definition -1.16 notice that the bound variable has no role to play in the meaning of $Ox[g]$ since it has no occurrences in g . In general we expect the semantic function to be a homomorphism between the two similar algebras (see definition -1.15).

Exercise -1.5: Semantics of term algebras

Refer to definition -1.16

1. Prove that if \mathcal{V} is surjective then it induces a congruence on ground terms.

-1.3.2. Semantics of Open terms

However while considering the meaning of open terms (i.e. those that are not ground and may have free variables), the value of the term would depend on the “context” in which the term occurs. This “context” refers to the values of the *free* variables used in the term. This requires us to first assign values to variables from a model which the language putatively represents.

Definition -1.17: Valuation

Let V be a set of variables and let \mathbb{M} be any set. A **valuation** $v : V \longrightarrow \mathbb{M}$ is a (total) function associating with each variable $x \in V$ a unique value $m \in \mathbb{M}$. For any variable x and valuation v' , v' is called an **x -variant** of v if for all $y \in V - \{x\}$, $v(y) = v'(y)$.

Notation -1.4

$v =_{\setminus \textcolor{teal}{x}} v'$ denotes that v and v' are $\textcolor{teal}{x}$ -variants. In general for any subset $X \subset V$, $v =_{\setminus X} v'$ if for all $\textcolor{teal}{y} \in V - X$, $v(\textcolor{teal}{y}) = v'(\textcolor{teal}{y})$.

Further such a language should be “compositional” in the sense that every operation in the term algebra $\mathbf{T}_\Omega(V) = \langle \mathbb{T}_\Omega(V), \Omega, \{\equiv\} \rangle$ needs to represent an appropriate operation on the object model $\mathbf{M} = \langle \mathbb{M}, \Omega', \{=\} \rangle$. To ensure that every term in the language be assigned a suitable value in the model, it is convenient to ensure that the model is itself a similar algebra where each operation in the language represents an operation (or function of the same arity) in the model.

In particular, this means that

1. Every constant in the term algebra also represents an object in the carrier set of the model and
2. For every operator $\textcolor{teal}{o}$ in the term algebra is associated with a unique operation (of exactly the same arity) in the model.
3. For every operator schema O in the term algebra the meaning of any term $Ox[t] \in \Omega$ is somehow independent of the value assigned to the variable $\textcolor{teal}{x}$.

Hence we use *induction on the structure of terms* (see section -3.3 and PSI) to define the semantics of terms.

Definition -1.18: Semantics of open terms

Given a term algebra $\mathbf{T}_\Omega(V) = \langle \mathcal{T}_\Omega(V), \Omega, \{\equiv\} \rangle$ and another similar algebra $\mathbf{M} = \langle \mathbb{M}, \Omega', \{=\} \rangle$ with the bijection μ as in definition (-1.13) between their signatures. Then a function \mathcal{M}

$$\mathcal{M}[\![\cdot]\!](v) : \mathcal{T}_\Omega(V) \longrightarrow \mathbb{M}$$

is called a **meaning** under a valuation v if for each valuation $v : V \longrightarrow \mathbb{M}$, the following conditions are satisfied.

1. For each variable $x \in V$, $\mathcal{M}[\![x]\!](v) = v[\![x]\!]$,
2. For each operator $o \in \Omega$ with $\alpha(o) = n \geq 0$ and terms t_1, \dots, t_n ,

$$\mathcal{M}[\![o(t_1, \dots, t_n)]\!](v) = \mu(o)(\mathcal{M}[\![t_1]\!](v), \dots, \mathcal{M}[\![t_n]\!](v))$$

3. For each operator schema $O \in \Omega$, variable $x \in V$ and term t there is a function \odot such that

$$\mathcal{M}[\![Ox[t]]\!](v) = \odot\{\mathcal{M}[\![t]\!](v') \mid v' =_{\setminus x} v\}$$

Given μ and v the function \mathcal{M} associates with each term t a unique value, called its **meaning** under the valuation v .

Exercise -1.6: Semantics of term algebras

Refer to definition [-1.18](#)

1. Prove that $\mathcal{M}[\cdot](v)$ is a surjective homomorphism if v is surjective.

0. Substitutions

0.1. Introduction to syntactic substitutions

Substitutions are so common in most of mathematics, that most mathematicians do not even realise that they are using them. Substitutions appear in various forms but most often they appear as instantiations of functions. For example, when you calculate the value of a function $f(x)$ at some particular point say 0, you first perform the substitution of “0” for all occurrences of the variable “ x ” in the expression for $f(x)$ by replacing all occurrences of x in the expression by 0. The result of this substitution is simply referred to as “ $f(0)$ ”. Other forms of substitutions occur when complex expressions are expressed more simply by using names for various subexpressions since it may be more convenient to define a function $h(x)$ as a composition of two functions $f(y)$ and $g(x)$. In this case $h(x) = f(g(x))$ is obtained by substituting $g(x)$ for y in the expression for $f(y)$. In all such cases the names and place-holders that are used in building the expressions are all variables.

Syntactic substitutions also play an important part in computations. Almost all computations can be described using syntactic substitutions of values for variables, expressions for variables or in the form of correct-answer substitutions in logic programming. The fundamental computational mechanism viz. application in the λ -calculus also uses syntactic substitution. It is therefore important to study syntactic substitutions more formally.

However syntactic substitution as an operation is usually *meta-syntactic* and hence belongs properly to the meta-language of any formal mechanism. We will look at it in general as an operation which will be required to be performed in any term algebra. Our main motivation for describing syntactic substitutions is because of their applications in logic and universal algebra.

0.2. Substitutions and Instantiations

We formally start by defining the notion of a substitution as a function. Throughout this section we will consider substitutions in the context of a term algebra $\mathbb{T}_\Omega(V)$ with signature Ω and a countable collection V of variables.

Definition 0.1: Substitution

A **substitution** θ is a (total) function $\theta : V \rightarrow \mathbb{T}_\Omega(V)$ which is almost everywhere identity. That is, there is only a finite set $X \subseteq V$, such that $\theta(x) = t \not\equiv x$ and for all $y \in V - X$, $\theta(y) = y$.

Notation 0.1

- We usually write θ as a finite set of the form $\theta = \{t/x \mid \theta(x) = t \not\equiv x\}$ where only the non-identical elements of the substitution are specified as pairs t/x .
- $dom(\theta) = \{x \in V \mid \theta(x) \not\equiv x\}$ refers to the set of variables whose image under θ is not the identity and $ran(\theta) = \{t \mid \theta(x) = t \not\equiv x\}$ ^a.
- $S_\Omega(V)$ is the set of all substitutions.

^aWe need to clearly distinguish between the domain of a (partial) function $f : A \rightharpoonup B$ given by $Dom(f) = \{a \in A \mid f(a) \in B\} \subseteq A$ (see -4.3) and $dom(\theta)$. Since by definition 0.1, θ is a total function we have $Dom(\theta) = V$ an infinite set whereas $dom(\theta) \subset_f Dom(\theta)$ is a finite subset of V . Similarly the range of a function given by (see -4.3) $Ran(f) = \{b \in B \mid \exists a \in A : b = f(a)\}$. In the case of θ , $Ran(\theta) = \{t \in \mathbb{T}_\Omega(V) \mid \exists x \in V : t = \theta(x)\}$ is different from $ran(\theta)$ which is a finite set.

Each such pair $\{t/x\}$ is read as “ t replaces x ” or more simply as “ t for x ”⁹. **1** is the **identity** substitution and $dom(1) =$

⁹Sounds like a campaign slogan e.g. “Obama for President”. But some authors prefer the notation x/t to be read as “ x is replaced by t ”. Usually there would be no confusion except when t is itself a

$\emptyset = \text{ran}(\mathbf{1})$. But θ is also a finite set, so we may use set theoretic operations on substitutions. However one needs to be careful while doing this, since the class $\mathbf{S}_\Omega(V)$ of all substitutions over a set of variables is not closed under set theoretic operations such as union.

α conversion and equivalence. The definition of α equivalence below firstly shows that we may use substitutions to rename bound variables in a term without changing the meaning.

Definition 0.2: α equivalence

For any term t ,

$$\mathcal{O}x[t] \equiv_\alpha \mathcal{O}y[\{y/x\}t]$$

provided

1. $y \notin FV(t)$ i.e. y does not occur free in t and
2. For every subterm $u \in ST(t)$ in which y occurs bound, $x \notin FV(u)$.

Notice that y could occur bound in some sub-term of t and yet the two bound variables would not clash with each other because of the conversion.

Free variable capture. But care is required to actually prevent the so-called “capture” of free variables, because it can alter the meaning of expressions to some wholly unintended one¹⁰. The following example illustrates the meaning of this

variable, as happens in section 0.4 on Pure-variable Substitutions.

¹⁰Loosely put, you might get wrong or wholly unexpected and unintended answers.

term.

Example 0.1: Free variable capture

Consider the expression $\sum_{z=1}^n y$ which may have been obtained as a result of some derivation or some calculation. Assuming all the symbols here have their usual meanings in the integers, we may safely assume that for any value of the free variables n and y , the value of this expression should be ny (i.e the product of n and y). So if we were to substitute any integer-valued function such as $y = f(x) = x^2$, we expect the result to be obtained by substituting $f(x)$ for y yielding $n.f(x) = nx^2$. However, if it so happened that f was defined by using the variable z instead of x as in $y = f(z) = z^2$, the corresponding result obtained is unfortunately not $n.f(z)$. Instead it becomes the sum of the first n squares of positive integers which is the unintended result of the substitution. This has happened because the variable z which is free in the expression $f(z)$ has got “captured” by the binding of z in $\sum_{z=1}^n y$. So whereas x^2/y is a permitted substitution, z^2/y should not be permitted because the free variable z may be captured by the binding $\sum_{z=1}^n$ in the host term.

Definition 0.3: Admissibility

An element t/x in a substitution is **admissible** for a *host* term s if no free occurrence of x in s occurs within the scope of a binding of any free variable in t . A substitution θ is **admissible** for an expression s if every element of θ is admissible for s .

So definition 0.4 carefully identifies substitution with only the syntactic replacement of *free* variables in a host term with their images under the substitution such that the *free variables in the image* do not get bound by binding occurrences in an enclosing scope of the host term.

Definition 0.4: Instantiation

Given a term s and a substitution $\theta = \{t_i/x_i \mid t_i \not\equiv x_i, 1 \leq i \leq n\}$ admissible for s , θs denotes the new term u obtained by **instantiation** of s by θ (also called **application** of θ to s). u is also called an **instance** of s . θs is defined by induction on the structure of s as follows^a.

Basis

Case $s \equiv c()$. $\theta c() = c()$

Case $s \equiv x \notin \text{dom}(\theta)$. $\theta x = x$

Case $s \equiv x_i \in \text{dom}(\theta)$. $\theta x_i = t_i$

Induction

Case $s \equiv o(s_1, \dots, s_m)$. $\theta o(s_1, \dots, s_m) = o(\theta s_1, \dots, \theta s_m).$

Case $s \equiv Ox[s']$, $x \notin \text{dom}(\theta)$. $\theta Ox[s'] = Ox[\theta s']$

Case $s \equiv Ox[s']$, $t/x \in \theta$. $\theta Ox[s'] = Ox[(\theta - \{t/x\})s']$

^aWe are defining this as a prefix operation, though many authors prefer to use the postfix notation “ $s\theta$ ”. There should be no confusion except possibly when dealing with compositions of substitutions (see subsection 0.3).

Intuitively θs is a new term u obtained by replacing each *free* occurrence in s of each variable $x_i \in \text{dom}(\theta) \cap FV(t)$ by t_i . There may be several occurrences of x_i in s . Our definition of instantiation is actually *total* instantiation of a substitution.

But in certain cases¹¹ one could also consider *partial* applications of substitutions wherein only certain free occurrences of the variable x_i defined by its position in t may be replaced.

Since in any substitution $\theta = \{t_i/x_i \mid t_i \not\equiv x_i, 1 \leq i \leq n\}$, $\text{depth}(t_i) \geq \text{depth}(x_i) = 1$ and $\text{size}(t_i) \geq \text{size}(x_i) = 1$ for all $1 \leq i \leq n$, the effect of applying a substitution is always depth and size (see definition -1.11) non-decreasing.

Fact 0.1

Let θ be a substitution and t a term. Then

1. $\text{depth}(\theta t) \geq \text{depth}(t)$
2. $\text{size}(\theta t) \geq \text{size}(t)$.

Exercise 0.1

Assume t/x is not admissible in some term s , yet it is necessary to perform the substitution. How would you perform it?

Definition 0.5: Ground instances

- A substitution $\theta = \{t_i/x_i \mid t_i \not\equiv x_i, 1 \leq i \leq n\}$ is a **ground substitution** if each t_i , $1 \leq i \leq n$, is a ground term.
- u is a **ground instance** of t if u is an instance of t and is ground.

¹¹e.g. instantiation rules in first-order logic

Definition 0.6: Common instances, Variants

- u is a **common instance** of two or more terms t_1, \dots, t_n if there exist substitutions $\theta_1, \dots, \theta_n$ such that

$$u \equiv \theta_1 t_1 \equiv \dots \equiv \theta_n t_n$$

- Terms t and u are called **variants** of each other if there exist substitutions θ and τ such that $\theta t \equiv u$ and $\tau u \equiv t$.

Exercise 0.2

Let $u \equiv \theta t$. Give examples of t , u and θ such that $FV(t) \neq \emptyset$, u is ground but θ is not a ground substitution.

0.3. The Composition of Substitutions

We will often require to perform substitutions in sequence i.e. it may be necessary to first apply a substitution θ on a term t yielding a term θt to which another substitution τ may be applied to yield a term $\tau(\theta t)$. We would like to answer the question of how to define a single substitution χ such that for every term t ,

$$\tau(\theta t) \equiv \chi t \tag{18}$$

Then χ is the *composition* of τ with θ . Before presenting the formal definition of composition we try to understand how such a composition must be defined to ensure that equation (18) holds. Let $\theta = \{s_1/x_1, \dots, s_k/x_k\}$ and $\tau = \{t_1/y_1, \dots, t_m/y_m\}$. We have $dom(\theta) = X = \{x_1, \dots, x_k\}$ and $dom(\tau) = Y = \{y_1, \dots, y_m\}$. The effect of θ on any term u is to replace each

free occurrence of each variable x_i by the term s_i simultaneously for $1 \leq i \leq k$. The terms s_i could contain (free) variables drawn from X and Y . It could also happen that some of the terms s_i may simply be variables themselves. Consider a single variable x_i . If $s_i \equiv z$ for some variable z , then θu would simply have z occurring free in all those positions of u where x_i occurs free. Of course, free occurrences of x_i could be present in θu because of some other variable substitution (say $s_{i'}/x_{i'}$ for some $i' \neq i$). Hence it is clear that all free occurrences of any $x \in X$ in θu are due to the application of the substitution θ . Further, for any $y_j \in Y$, we have the following possibilities.

1. Case $y_j \in Y - X$ and $y_j \in FV(u)$. All such free occurrences of y_j in u will be present in the same positions in θu as well. The effect of τ would be to replace them all with t_i .
2. Case $y_j \in Y - X$ and $y_j \notin FV(u)$. New free occurrences may arise due to the substitution θ . The effect of the application of τ will replace all of them by t_j .
3. Case $y_j \equiv x_i$ for some $x_i \in X$. In this case the only free occurrences of y_j possible are those which occur after applying θ .

Case 1 requires τ to be applied separately. The effect of τ on cases 2 and 3 may be captured by applying τ to the range of θ . Once that is done one may even remove the element t_j/y_j from the substitution, since it would have no effect. Further all elements such that $\tau s_i \equiv x_i$ are removed from χ , since we are interested in specifying the substitution as a finite set of non-identical replacements. With this understanding we are ready to tackle our definition of composition.

Definition 0.7: Composition of substitutions

Given substitutions $\theta = \{s_1/x_1, \dots, s_k/x_k\}$ and $\tau = \{t_1/y_1, \dots, t_m/y_m\}$, their **composition** $\tau \circ \theta$ is a new substitution χ such that

$$\chi = \{\tau s_i/x_i \mid 1 \leq i \leq k, \tau s_i \not\equiv x_i\} \cup \{\tau t_j/y_j \mid 1 \leq j \leq m, y_j \notin \text{dom}(\theta)\}$$

Exercise 0.3

Prove that for any substitutions θ and τ , $\tau \circ \theta = \tau \cup \theta$ iff $\text{dom}(\theta) \cap \text{dom}(\tau) = \emptyset$ and $\text{dom}(\tau) \cap \bigcup_{t \in \text{ran}(\theta)} FV(t) = \emptyset$

Lemma 0.1: Substitutions form a monoid

Given substitutions θ, τ, χ and a term t , we have

1. $\theta \circ \mathbf{1} = \mathbf{1} \circ \theta = \theta$
2. $(\tau \circ \theta)t \equiv \tau(\theta t)$
3. $\chi \circ (\tau \circ \theta) = (\chi \circ \tau) \circ \theta$

Proof: We assume $\theta = \{s_1/x_1, \dots, s_k/x_k\}$ and $\tau = \{t_1/y_1, \dots, t_m/y_m\}$ and $\rho = \tau \circ \theta$ as in definition 0.7. Then

1. Trivial.
2. We prove this by induction on the structure of terms. The case of constants is trivial. The induction case will also follow once the cases of

simple variables has been proven. So we simply prove this case for simple variables. For any variable x we have the following cases.

Case $x \notin \text{dom}(\theta)$. Then clearly $(\theta x) \equiv x$ and $\tau(\theta x) \equiv \tau x$. Since the first component of the union in the definition of ρ does not apply, we have $\rho x \equiv \tau x$.

Case $x \equiv x_i \in \text{dom}(\theta)$ for some i , $1 \leq i \leq m$. In this case $\rho x_i \equiv \tau s_i$ and since $\theta x_i \equiv s_i$ we have $\tau(\theta x_i) \equiv \tau s_i$.

3. For any term t we have from the previous proof

$$(\chi \circ (\tau \circ \theta))t \equiv \chi((\tau \circ \theta)t) \equiv \chi(\tau(\theta t)) \equiv (\chi \circ \tau)(\theta t) \equiv ((\chi \circ \tau) \circ \theta)t$$

QED

Exercise 0.4

A substitution θ is called **idempotent** if $\theta \circ \theta = \theta$. Now complete the statement of the following lemma and prove it.

Lemma 0.2

A substitution $\theta = \{t_i/x_i \mid 1 \leq i \leq m\}$ for some $m \geq 0$ is idempotent iff $\text{dom}(\theta) \dots$

0.4. Pure-variable Substitutions

Definition 0.8: Pure variable substitutions

- θ is a **pure-variable** substitution if $\text{ran}(\theta) \subseteq V$.
- A pure-variable substitution $\theta = \{y_1/x_1, \dots, y_n/x_n\}$ is a **renaming** substitution on a term t if
 - all the y_i are distinct and
 - $(FV(t) - \text{dom}(\theta)) \cap \text{ran}(\theta) = \emptyset$

Example 0.2

A pure-variable substitution may replace more than one distinct name by a single name e.g. $\theta = \{z/x, z/y\}$. A pure-variable substitution may also identify several names with a single name e.g. if $f(x, y)$ is a term and $\tau = \{x/y\}$ then $\{x/y\}f(x, y) \equiv f(x, x)$. However neither of these is a renaming substitution for $f(x, y)$ because in the case of θ the replacements are not all distinct and in the case of τ , the range of τ is not disjoint from $(FV(f(x, y)) - \text{dom}(\tau))$. If $FV(f(x, y)) = \{x, y\}$, then $\chi = \{u/x, v/y\}$ is a renaming substitution for $f(x, y)$. Now it is however clear that we may define an inverse substitution $\chi^{-1} = \{x/u, y/v\}$ which is a renaming substitution for the term $\chi f(x, y)$ and in fact $\chi \circ \chi^{-1} = \mathbf{1} = \chi^{-1} \circ \chi$.

Note that a renaming substitution always requires to be judged along with a term. A renaming substitution for one term may not be a renaming substitution for all terms. On the other hand, the notion of a pure-variable substitution is purely syntactic and independent of the terms to which it is applied.

Fact 0.2

1. $\mathbf{1}$ is a renaming substitution for all terms.
2. If θ is a pure-variable substitution then
 - (a) $\text{depth}(\theta t) = \text{depth}(t)$ for all terms t
 - (b) $\text{size}(\theta t) = \text{size}(t)$ for all terms t .
3. If $\theta t \not\equiv t$ and
 - (a) $\text{depth}(\theta t) = \text{depth}(t)$ for all terms t and
 - (b) $\text{size}(\theta t) = \text{size}(t)$ for all terms t .

then θ is a pure-variable substitution unless $\text{ran}(\theta)$ contains a constant.

Lemma 0.3: Variants and renaming

If t and u are variants (see definition 0.6), then there exist renaming substitutions θ and τ such that $\theta t \equiv u$ and $t \equiv \tau u$.

Proof: Without loss of generality we assume t and u are variants of each other and $t \not\equiv u$. Then there exist substitutions θ_1 and τ_1 such that $\theta_1 t \equiv u$ and $\tau_1 u \equiv t$. Define $\theta = \{s/x \in \theta_1 \mid x \in FV(t)\}$ and $\tau = \{t/y \in \tau_1 \mid y \in FV(u)\}$. Clearly then

$$\theta t \equiv u \text{ and } \tau u \equiv t$$

Hence $\tau(\theta t) \equiv t$ and $\theta(\tau u) \equiv u$. It is also clear that $\text{ran}(\theta)$ and $\text{ran}(\tau)$ contain no constants. Since substitutions are always depth and size

non-decreasing we have by fact 0.1

$$\text{depth}(\textcolor{violet}{t}) = \text{depth}(\tau(\theta\textcolor{violet}{t})) \geq \text{depth}(\theta\textcolor{violet}{t}) \geq \text{depth}(\textcolor{violet}{t})$$

and

$$\text{size}(\textcolor{violet}{t}) = \text{size}(\tau(\theta\textcolor{violet}{t})) \geq \text{size}(\theta\textcolor{violet}{t}) \geq \text{size}(\textcolor{violet}{t})$$

Hence both θ and τ are pure-variable substitutions and since $\theta\textcolor{violet}{t} \equiv \textcolor{violet}{u}$ and $\tau\textcolor{violet}{u} \equiv \textcolor{violet}{t}$, and $\textcolor{violet}{t} \not\equiv \textcolor{violet}{u}$ both of them are renaming substitutions.

QED

1. Introduction

1: Introduction

CALVIN & HOBBES by Bill Watterson



1. What is Logic?
2. Reasoning, Truth and Validity
3. Examples
4. Objectivity in Logic
5. Formal Logic
6. Formal Logic: Applications
7. Form and Content
8. Facets of Mathematical Logic
9. Logic and Computer Science

What is Logic?

- Logic is about *reasoning*:
 - *validity* of arguments
 - *consistency* among sets of statements
 - matters of *truth* and *falsehood*
- Logic is concerned only about the *form* of reasoning and not about the *content*

Reasoning, Truth and Validity

- Reasoning is sound only if it is impossible to draw false conclusions from true premises
- Sound reasoning can however produce false conclusions from false premises.
- Sound reasoning can also produce true conclusions from false premises.



©Bill Watterson

Examples

Consider the following arguments.

Example 1.1: Humans are forever

All humans live forever.

Socrates is human.

Hence Socrates lives forever.

Example 1.2

All humans are born with opposable toes

By adulthood opposable toes become non-opposable

John is an adult human

Hence John has no opposable toes.

NO RETRACTABLE CLAWS,
NO OPPOSABLE TOES,
NO PREHENSILE TAIL,
NO COMPOUND EYES,
NO FANGS, NO WINGS..

..SIGHHH...



©1995 Bill Watterson

©Bill Watterson

Objectivity in Logic

- The traditional logic of Aristotle and Leibniz are essentially philosophical in nature with its main purpose being to investigate the *objective* laws of thought.
- *Objectivity* implies essentially that arguments must be communicable to and verifiable by other people.
- *Objectivity* has always implied *formalizability*.

The colours and shades of Logic.

The name *mathematical logic* may be interpreted in two ways. We may interpret *logic* as a subject treated using mathematical methods. We may just as well, think of it as a subject which formalises the methods of reasoning used in mathematics. This leads us apparently to a paradox. How can *logic* be treated mathematically without using *logic* itself?

We resolve this paradox as follows. We simply separate out the *logic* that we are studying by expressing it formally through a language – the *object language* or the *target language* – from the logic that is used in reasoning about it. The latter logic that is used for reasoning is called the *meta-language*.

We will define the *object language* formally via a grammar (and for good measure we also colour-code the sentences of the object language in green). The meta-language however, is the usual “language of mathematics” – a mixture of natural language with mathematical symbols that is normally used in mathematical texts. The words of these sentences will usually appear in black and some times in other colours such as blue (e.g for emphasis) or red (e.g. when we want some concept or idea to stand out). However since the objects of our study are green, the objects when appearing in these sentences will still be green.

When treating any object language formally, it also becomes necessary to specify the meanings of phrases, expressions and sentences in the formal language. Since we will be expressing these meanings through objects

in mathematical structures, these objects will be coloured brown.

The study of logic promises to be pretty colourful, what?

Formal Logic

- Logic as a **formal language** with a *syntax* to which a *semantics* had to be attached.
- In turn using mathematical methods within logic, led to its formalization as **mathematical logic**
- The strict separation of **syntax** from **semantics** made logic a clean and elegant mathematical discipline which could be then applied to the foundational questions of mathematics.

Formal Logic: Applications

- Of great use in analysing questions concerning the foundations of mathematics
- In clarifying reasoning mechanisms within mathematics.
- Identifying various occurrences of *circular reasoning* which were previously very hard to identify.

Form and Content

- The separation of syntax and semantics also led to the separation of form and content and
- allowed the formalization of correct methods of reasoning purely in terms of the syntax.
- allowed the possibility of plugging in a semantics as demanded by an application whenever it was necessary
- allowed the possibility of *mechanizing* reasoning completely.

Facets of Mathematical Logic

1. A *formalization* language for mathematics,
2. A *calculus* clarifying the notion of a mathematical proof
3. *mechanization* of proof
4. A *sub-discipline* of mathematics itself
5. A *mathematical tool* applicable to various branches of mathematics.

Logic and Computer Science

1. *Mechanization* of reasoning within a formalized syntactic setup.
2. Applications to program and system specification and verification.
3. As a declarative programming language
4. Proving theorems within areas of mathematics where the number of cases is too high or the details of proof are too long and tedious.

2. Propositional Logic Syntax

2: Propositional Logic Syntax

“I suppose when you tell the tale, you deviate from the truth a lot?”

“Quite a good deal. I have always found the truth an excellent thing to deviate from.”

P. G. Wodehouse, *Sunset at Blandings*

1. Truth and Falsehood: 1
2. Truth and Falsehood: 2
3. Extending the Boolean Algebra
4. Sums & Products
5. Propositional Logic: Syntax
6. Propositional Logic: Syntax - 2
7. Natural Language equivalents
8. Some Remarks
9. Associativity and Precedence
10. Syntactic Identity
11. Abstract Syntax Trees
12. Subformulae
13. Atoms in a Formula
14. Degree of a Formula
15. Size of a Formula
16. Height of a Formula

Truth and Falsehood: 1

Our notion of meaning is restricted to absolute “truth” and absolute “falsehood” (with nothing else in between) of statements.

Definition 2.1: Truth and Falsehood

Let $\mathbf{2} = \langle \mathcal{2}, \{\neg, ., +\}, \{\leq, =\} \rangle$ be the 2-element boolean algebra where $\mathcal{2} = \{0, 1\}$.

- We use 1 to denote absolute “truth” and 0 to denote absolute “falsehood” and
- the boolean operations have their **usual meaning** in the booleans.

Table of Truth & Falsehood

a	\bar{a}
0	1
1	0

a	b	$a.b$	$a + b$	$a \leq b$	$a \doteq b$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Truth and Falsehood: 2

Alternatively if 0 and 1 are regarded as naturals, for any $a, b \in 2$,

- $\bar{a} = 1 - a$ is the unary boolean inverse operation,
- $a + b = \max(a, b)$ is the binary summation operation which yields the maximum of two boolean values regarded as natural numbers,
- $a.b = \min(a, b)$ is the binary product operation which yields the minimum of two boolean values regarded as natural numbers,
- $a \leq b$, and $a = b$ denote the usual binary relations “less-than-or-equal-to” and “equals” on natural numbers restricted to the set $\{0, 1\}$.

Extending the Boolean Algebra

While \leq and $=$ are binary relations, it is possible to define corresponding binary operations as shown below.

Definition 2.2: Extended boolean algebra

For $a, b \in \{0, 1\}$ define

$$a \leq b = \begin{cases} 1 & \text{if } a \leq b \\ 0 & \text{otherwise} \end{cases} \quad a = b = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

Let $2' = \langle \mathcal{D}, \{\neg, ., +, \leq, =\}, \{\leq, =\} \rangle$ denote the extended algebra.

Boolean identities

Inverse $\bar{\bar{a}} = a$

Comparison $a \leq b = \bar{a} + b$

Equality $a = b = (a \leq b).(b \leq a)$

$$a + 0 = a$$

Identity

$$a \cdot 1 = a$$

$$a + 1 = 1$$

Zero

$$a \cdot 0 = 0$$

$$a + a = a$$

Idempotence

$$a \cdot a = a$$

$$a + b = b + a$$

Commutativity

$$a \cdot b = b \cdot a$$

$$(a + b) + c = a + (b + c)$$

Associativity

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

Distributivity

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$\overline{a + b} = \bar{a} \cdot \bar{b}$$

De Morgan

$$\overline{a \cdot b} = \bar{a} + \bar{b}$$

$$a + \bar{a} = 1$$

Simplification

$$a \cdot \bar{a} = 0$$

$$\bar{0} = 1$$

Inversion

$$\bar{1} = 0$$

$$a + (a \cdot b) = a$$

Absorption

$$a \cdot (a + b) = a$$

Sums & Products

The Associativity and Commutativity identities allow us to define summations and products over sets of boolean values.

$$\sum_{0 < i \leq n} a_i \stackrel{df}{=} a_1 + \cdots + a_n$$

$$\prod_{0 < i \leq n} a_i \stackrel{df}{=} a_1 \cdot \cdots \cdot a_n$$

[Skip to Propositional Logic: Syntax](#)

2.1. Propositions in Natural Languages

We now begin our study of formal logic by studying statements in natural language. We will frequently appeal to reasoning methods employed in sentences (actually statements) in natural languages. We initially restrict our study to statements expressed in natural languages. This logic is often called *propositional* or *sentential* logic.

In general sentences in any natural language may be classified into the following forms (usually indicated by the *mood* of the sentence. A standard text book on English grammar defines a *mood* as *the mode or manner in which the*. It further illustrates three moods in the English language. [English Grammar 101](#) classifies the verbs in the English language into four moods (including the *infinitive* as a mood. But to get really confounded and confused, the reader needs to refer to [Wikipedia](#) where several moods are defined (e.g. *optative*, *jussive*, *potential*, *inferential*).

The following example sentences are taken from the references above.

Indicative Mood: expresses an assertion, denial, or question.

*Little Rock is the capital of Arkansas.
Ostriches cannot fly.
Have you finished your homework?*

Imperative Mood: expresses command, prohibition, entreaty, or advice.

*Don't smoke in this building.
Be careful!
Have mercy upon us.
Give us this day our daily bread.*

Subjunctive Mood: expresses a doubt, a wish or an improbability

*God bless you!
I wish I knew his name.
I would rather be paid by cheque.
He walks as though he were drunk.*

Loosely speaking natural language sentences which are assertions or denials in the indicative mood may be considered propositions. Questions however, cannot be considered propositions. More accurately only those

sentences to which one might (at least theoretically) ascribe a truth value (such as assertions and denials) may be considered to be propositions in our setting. Hence of the examples given above the only ones which are of interest to us would be

Little Rock is the capital of Arakansas.

(which is an assertion) and

Ostriches cannot fly.

which is a denial (it denies the statement *Ostriches can fly*). The last indicative statement

Have you finished your homework?

is a question for which no truth value can be assigned. It is therefore not a proposition in the sense that we understand propositions. Notice that a denial is an assertion too – it is simply the negation of an assertion and hence is a statement to which a truth value may be assigned. We will treat denials also as assertions and declare that assertions in natural language are all propositions in our sense of the term.

The examples that we have considered above are rather simple. We could consider more examples of assertions, denials and complex assertions which are made up of assertions and denials. Here are a few.

- *God is in his Heaven and all is right with the world.*
- *Time and tide wait for no man.*
- *You can fool some people some of the time, some people all of the time and all the people some of the time, but you cannot fool all the people all the time.*
- *Anybody who becomes the Prime Minister has a clear national and international agenda.*

Propositional Logic: Syntax

(see section -1 and -1.2)

Definition 2.3

- A : a countable collection of propositional atoms.
- $\Omega_0 = \{\perp, \top, \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$: the set of operators (also called connectives) disjoint from A such that their arities^a are as follows
 - $\alpha(\perp) = \alpha(\top) = 0$
 - $\alpha(\neg) = 1$ and
 - $\alpha(\odot) = 2$, for $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.
- Parentheses (and) of various kinds are used for grouping.
- $\mathcal{P}_0 = \mathbb{T}_{\Omega_0}(A)$ is the smallest set generated from A and Ω_0 .

^asee definition -1.6

Propositional Logic: Syntax - 2

Alternatively, we may define the language by the following grammar.

Definition 2.4: BNF of \mathcal{P}_0

$$\phi, \psi ::= \perp \mid \top \mid p \in A \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi) \quad (19)$$

Each expression of this language is also called a **well-formed formula** (wff) (or sentence) of \mathcal{P}_0 . Each atom is a **simple formula** or sentence. Each sentence having one or more occurrences of unary or binary operators is called a **compound formula**. \mathcal{P}_0 is the set of all sentences.

Natural Language equivalents

The (unary and binary) operators of the language \mathcal{P}_0 are chosen to denote constructs that allow the construction of compound statements from simple ones. The following table shows the intended meaning of each of the operators in the English language.

Operator	Name	English rendering
\perp	<i>bottom</i>	false
\top	<i>top</i>	true
\neg	<i>negation</i>	not
\wedge	<i>conjunction</i>	and
\vee	<i>disjunction</i>	or
\rightarrow	<i>conditional</i>	if...then
\leftrightarrow	<i>biconditional</i>	if and only if

Skip to [Some Remarks](#)

2.2. Translation of Natural Language Statements

We may translate natural language sentences (assertions and denials) into propositions by identifying the simplest sentences as atoms and connecting up the simple sentences using the connectives at our disposal. Naturally, many of the subtleties of the use of the connectives in natural languages would be lost in translation. For the purposes of logical reasoning, the table given in [Natural Language equivalents](#) is sufficient for propositional reasoning, if we ignore the subtleties in natural language such as tonal variations, implied tense and judgmental implications that often come loaded with the sentences. We will have more to say on this with respect to particular connectives in the following descriptions.

Negation (\neg). This connective is used to mean [not](#), [it is not the case that](#) and abbreviated negation prefixes such as [non-](#) and [un-](#). If the atom A denotes the simple statement [I am at work](#), then $(\neg A)$ denotes [I am not at work](#). In certain cases opposites could be translated using negation. For example, if the sentence

This dish is good.

is denoted by the atom B , the sentence

This dish is bad.

and the statement

This dish is *not good*.

are both translated as $(\neg B)$. Going further, the sentence

This dish is *not bad*.

would be translated as $(\neg\neg B)$. As we shall see both the statements B and $(\neg\neg B)$ would be considered *logically* (though not *syntactically*) *equivalent* and the subtle difference between the expressions *good* and *not bad* would be lost in translation.

Conjunction (\wedge). The usual meaning of conjunction as denoting *and* holds. In addition, the words *but*, *moreover*, *furthermore* and phrases such as *in addition* would all be considered synonymous with *and*. Notice that the sentence

Rama *and* Seeta went to the forest

would be considered synonymous with the sentence

Rama went to the forest *and* Seeta went to the forest

even though the first sentence has an implied meaning of “togetherness” or “simultaneity”.

The operator \wedge is commutative as we shall see later. Therefore for any formulae ϕ and ψ , $\phi \wedge \psi$ would be logically equivalent to $\psi \wedge \phi$. Hence the implied difference between the two sentences (from [8])

Jane got married *and* had a baby.

and

Jane had a baby *and* got married.

is usually lost in translation.

Disjunction (\vee). The usual meaning is an *or* in the *inclusive* sense although *or* is often used in English in the *exclusive* sense. Hence $\phi \vee \psi$ would have to be translated to mean *either ϕ or ψ or both ϕ and ψ* .

In our natural language arguments we will use *or* in the inclusive sense. *either ϕ or ψ* would refer to an *or* that is used in the *exclusive* sense. That is *either ϕ or ψ* would mean that exactly one of the two propositions holds and both cannot hold at the same time. Therefore the sentence

Ram *or* Shyam topped the class.

could be equivalently rendered in English as a compound sentence

Ram topped the class *or* Shyam topped the class.

Neither of the above sentences rules out the possibility that both Ram and Shyam topped the class, whereas

Either Ram *or* Shyam topped the class.

would mean that exactly one of them could have topped the class.

Analogously one might ask what is the correct rendering of the sentence

Neither Ram nor Shyam topped the class.

If r denotes the atomic statement *Ram topped the class* and s denotes the atomic statement *Shyam topped the class*, then $(\neg r) \wedge (\neg s)$ would be an accurate propositional rendering of the sentence.

Conditional (\rightarrow). Also called the *material conditional*, it comes pretty close to the English conditional statement of the form “*If ϕ then ψ* ”. Alternative translations are “ ψ only if ϕ ”, “ ψ provided ϕ ” and “ ψ whenever ϕ ”.

There are other conditionals that we frequently use in English such as “ ψ unless ϕ ” which may be rendered as “ ψ if not ϕ ” and would be represented by the formula $((\neg\phi) \rightarrow \psi)$.

Biconditional (\leftrightarrow) The translation “*If ϕ then ψ , not otherwise*” is reserved for the biconditional. Alternative translations for $\phi \leftrightarrow \psi$ are “ ϕ if and only if ψ ”, “ ϕ iff ψ ”, “ ϕ exactly when ψ ” and “ ϕ just in case ψ ”.

The following tables adapted from [8] summarise the various English renderings of each operator given arbitrary operands ϕ and ψ .

Operation	English rendering
$\neg\phi$	not ϕ ϕ does not hold It is not the case that ϕ holds

Operation	English rendering
$\phi \wedge \psi$	ϕ and ψ Both ϕ and ψ ϕ but ψ Not only ϕ but ψ ϕ although ψ ϕ despite ψ ϕ yet ψ ϕ while ψ

Operation	English rendering
$\phi \vee \psi$	ϕ or ψ
	ϕ or ψ or both
	ϕ and/or ψ
	ϕ unless ψ
	ϕ except when ψ

Operation	English rendering
$\phi \rightarrow \psi$	If ϕ then ψ ψ if ϕ ϕ only if ψ When ϕ then ψ ψ when ϕ ϕ only when ψ In case ϕ then ψ ψ in case ϕ ψ provided ϕ ϕ is a sufficient condition for ψ ψ is a necessary condition for ϕ

Operation	English rendering
$\phi \leftrightarrow \psi$	ϕ if and only if ψ ϕ iff ψ ϕ if ψ and ψ if ϕ If ϕ then ψ , and conversely ϕ exactly if ψ ϕ exactly when ψ ϕ just in case ψ ϕ is a necessary and sufficient condition for ψ

Some Remarks

- It is convenient to have a syntactic symbol for “absolute truth” and “absolute falsehood” though it is strictly not necessary.
- \perp and \top are **constants** and hence have no precedence (ref. section 2.3) associated with them.
- In a formula of the form $\phi \rightarrow \psi$, ϕ is called the *antecedent* and ψ the *consequent*.
- To maintain a pleasing symmetry one could have also defined an operator \leftarrow , where $\phi \leftarrow \psi$ is intended to be read as “ ϕ if ψ ”, but that would only reverse the arrow \rightarrow and not provide any additional power of expression.

[Skip to Associativity and Precedence](#)

2.3. Associativity and Precedence of Operators

Notice that we have defined the language to be fully parenthesized i.e. every *compound* formula is enclosed in a pair of parentheses $((\dots))$. However it may actually be very distracting and confusing to read formulae with too many parentheses. We will define **precedence and associativity conventions** to reduce the number of parentheses while reading and writing formulae.

Associativity conventions. This convention refers to the consecutive occurrences of the same binary operator. Simple examples from school arithmetic are used to illustrate the conventions used in disambiguating expressions which are not fully parenthesized.

Left Associativity is the convention that an expression on numbers like $6 - 3 - 2$ should be read as $((6 - 3) - 2)$ (which would yield the value 1). It should not be read as $(6 - (3 - 2))$ (which would yield the value 5). Here we say that *the subtraction operation associates to the left* or that *subtraction is a left associative operation*. Other binary operations such as addition, multiplication and division on numbers are also left associative.

Right Associativity is the convention used to group consecutive occurrences of powers of numbers. For example 4^{3^2} is to be read as $(4^{(3^2)})$ which would yield the result $4^9 = 262144$. It should not be read as

$((4^3)^2)$ which would yield the result $64^2 = 4096$. We say that *exponentiation is right associative*.

Precedence of operators If two different binary operators occur consecutively in an expression, we need some way to associate and group them unambiguously. This is usually specified for binary operators by a precedence relation. In school arithmetic this usually takes the form of the “bodmas” rule specifying that brackets have the highest precedence followed by division and multiplication which in turn are followed by addition and subtraction. Hence for example an expression such as $3 \times 4 + 5$ is to be read as $((3 \times 4) + 5)$ representing the value 17. It would be wrong to read $3 \times 4 + 5$ as $(3 \times (4 + 5))$ (representing the value $3 \times 9 = 27$) since *multiplication has a higher precedence than addition or multiplication precedes addition*. This is usually denoted $\times \prec +$.

Similarly the expression $3 + 4 \times 5$ is to be read as $(3 + (4 \times 5))$ yielding the value $3 + 20 = 23$ and it would be wrong to read it as $((3 + 4) \times 5)$ (which represents the value $7 \times 5 = 35$).

It is the usual convention in mathematical texts that unless otherwise specified, unary operators have a higher precedence than binary operators.

Precedence of operators with equal precedence When two distinct binary operators have equal precedence (e.g. addition and subtraction on numbers) then our convention dictates that in the absence of parentheses, the left operator has a higher precedence in the expression than the one on the right i.e. they should

be grouped from left to right. Thus $5 + 4 - 3$ is to be read as $((5 + 4) - 3)$ yielding 6 (even though reading it as $(5 + (4 - 3))$ yields the same result). Similarly $5 - 4 + 3$ should be read as $((5 - 4) + 3)$ (yielding the result 4) rather than as $(5 - (4 + 3))$ (which yields -2). For example $24/4 \times 3$ would yield 18 by our convention. While this convention is well-established for left associative operators, it is not clear what the convention is when there are consecutive occurrences of distinct right associative operators having equal precedence. However in case of any confusion we may always put in enough parentheses to disambiguate the expression.

Our interest in precedence, however is restricted to being able to translate an expression written in linear form unambiguously into its **abstract syntax tree**. The use of parentheses aids in unambiguously defining an unique abstract syntax tree. Even though we write formulae in linear form we will always implicitly assume that they represent the corresponding abstract syntax tree.

Associativity and Precedence

Our language has been defined to be **fully parenthesized**.

- The binary operators \wedge and \vee are **left associative** i.e. a formula written as $\phi \wedge \psi \wedge \chi$ should be read as $((\phi \wedge \psi) \wedge \chi)$.
- The binary operators \rightarrow and \leftrightarrow , on the other hand are **right associative** i.e. a formula written as $\phi \rightarrow \psi \rightarrow \chi$ should be read as $(\phi \rightarrow (\psi \rightarrow \chi))$.
- The operator precedence convention we follow is:

$\leftrightarrow \prec \rightarrow \prec \vee \prec \wedge \prec \neg$

i.e. \neg has the highest precedence and \leftrightarrow has the lowest.

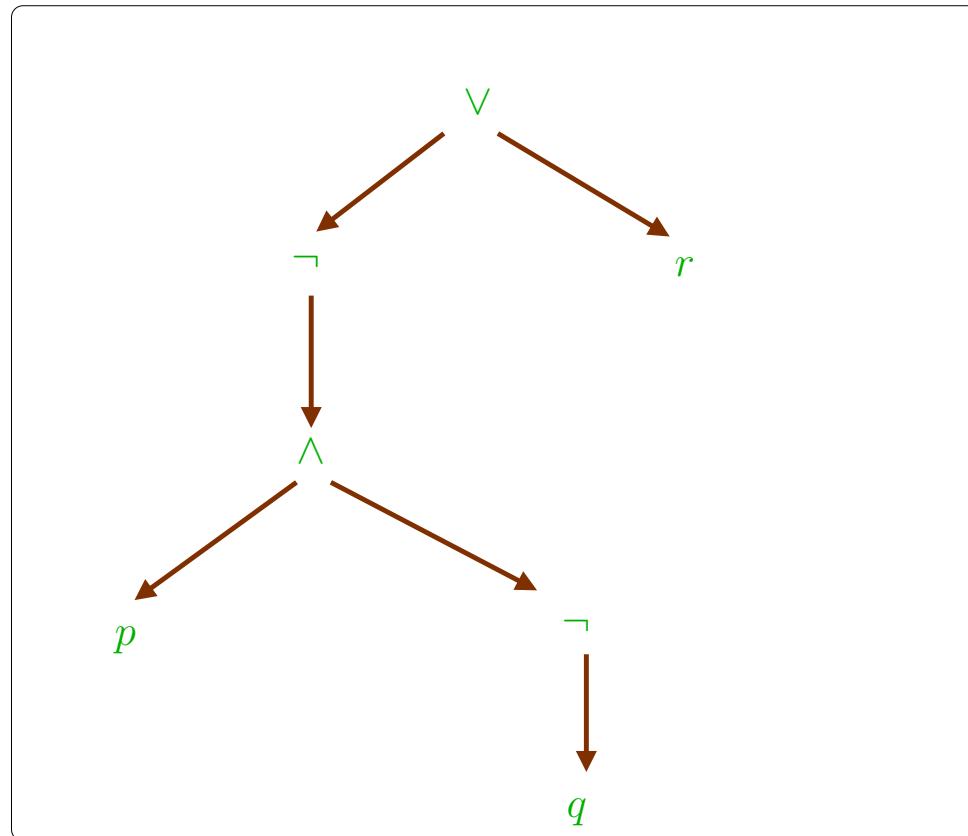
Syntactic Identity

- The precedence rules make a lot of parentheses redundant.
- Instead of propositions written linearly we will rather think of propositions as **abstract syntax trees (AST)**.
- Any two propositions ϕ and ψ which have the same AST will be considered *syntactically identical* and denoted $\phi \equiv \psi$.
- If $\phi \equiv \psi$ then ϕ and ψ differ only in the presence of redundant parentheses.

Abstract Syntax Trees

Abstract syntax trees are rooted directed trees (see definition -2.2)

Example 2.1 *The AST of the formula $(\neg(p \wedge \neg q) \vee r)$.*



Subformulae

Definition 2.5: Subformulae

The set of subformulae of any formula ϕ is the set $SF(\phi)$ defined by induction on the structure of formulae as follows:

$$SF(\perp) = \{\perp\}$$

$$SF(\top) = \{\top\}$$

$$SF(p) = \{p\}, \quad \text{for each atom } p$$

$$SF(\neg\psi) = SF(\psi) \cup \{\neg\psi\}$$

$$SF(\psi \odot \chi) = SF(\psi) \cup SF(\chi) \cup \{\psi \odot \chi\}, \quad \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

Atoms in a Formula

Definition 2.6: Atoms

The atoms of a formula is defined by induction on the structure of formulae.

$$\text{atoms}(\perp) = \{\perp\}$$

$$\text{atoms}(\top) = \{\top\}$$

$$\text{atoms}(p) = \{p\},$$

for each atom p

$$\text{atoms}(\neg\phi) = \text{atoms}(\phi)$$

$$\text{atoms}(\psi \odot \chi) = \text{atoms}(\psi) \cup \text{atoms}(\chi), \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

Size of a Formula

Definition 2.7: Size

The size of a formula is defined by induction on the structure of formulae.

$$\text{size}(\perp) = 1$$

$$\text{size}(\top) = 1$$

$$\text{size}(p) = 1, \quad \text{for each atom } p$$

$$\text{size}(\neg\phi) = 1 + \text{size}(\phi)$$

$$\text{size}(\psi \odot \chi) = 1 + \text{size}(\psi) + \text{size}(\chi), \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

Height of a Formula

Definition 2.8: height

The height of a formula is defined by induction on the structure of formulae.

$$\text{height}(\perp) = 0$$

$$\text{height}(\top) = 0$$

$$\text{height}(p) = 0, \quad \text{for each atom } p$$

$$\text{height}(\neg\phi) = 1 + \text{height}(\phi)$$

$$\text{height}(\psi \odot \chi) = 1 + \max(\text{height}(\psi), \text{height}(\chi)), \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

where *max* is the maximum of two numbers.

Exercise 2.1: Syntax

1. Prove that the set \mathcal{P}_0 is countably infinite. *Hint: Use the solution given in example -3.14.*
2. Let $\mathcal{T}(\mathcal{P}_0)$ be the set of all abstract syntax trees of the language \mathcal{P}_0 . Define a function $AST : \mathcal{P}_0 \longrightarrow \mathcal{T}(\mathcal{P}_0)$ which for any well-formed formula yields the corresponding unique abstract syntax tree of the formula.

3. Semantics of Propositional Logic

3: Semantics of Propositional Logic

"If there's no meaning in it," said the King, "that saves a world of trouble, you know, as we needn't try to find any."

Lewis Carroll, *Alice's Adventures in Wonderland*

1. Semantics of Propositional Logic: 1
2. Semantics of Propositional Logic: 2
3. A 1-1 Correspondence
4. Models and Satisfiability
5. Example: Abstract Syntax trees
6. Tautology, Contradiction, Contingent

Semantics of Propositional Logic: 1

Definition 3.1: Truth assignment/valuation

A **truth assignment** or **truth valuation** is a function τ which assigns to each atom a **truth value**.

$$\tau : A \rightarrow \mathcal{D}$$

Note:

- In definition 3.1 τ corresponds to the valuation v referred to in definition 19.2 in section -1.3.
- $\mathcal{T}[\cdot]_\tau$ corresponds to the meaning function \mathcal{M} in definition -1.18.
- We use $\stackrel{df}{=}$ to denote equality by definition.

Semantics of Propositional Logic: 2

The **truth value** of a proposition ϕ is defined by induction (see section -3.3) on the structure of propositions as a function $\mathcal{T}[\![\cdot]\!]_\tau : \mathcal{P}_0 \rightarrow \mathbb{2}$

Definition 3.2: Semantics of propositions

$$\mathcal{T}[\![\perp]\!]_\tau \stackrel{df}{=} 0 \quad , \quad \mathcal{T}[\![\top]\!]_\tau \stackrel{df}{=} 1$$

$$\mathcal{T}[\![p]\!]_\tau \stackrel{df}{=} \tau(p) \text{ if } p \in A \quad , \quad \mathcal{T}[\![\neg\phi]\!]_\tau \stackrel{df}{=} \overline{\mathcal{T}[\![\phi]\!]_\tau}$$

$$\mathcal{T}[\![\phi \wedge \psi]\!]_\tau \stackrel{df}{=} \mathcal{T}[\![\phi]\!]_\tau \cdot \mathcal{T}[\![\psi]\!]_\tau \quad , \quad \mathcal{T}[\![\phi \vee \psi]\!]_\tau \stackrel{df}{=} \mathcal{T}[\![\phi]\!]_\tau + \mathcal{T}[\![\psi]\!]_\tau$$

$$\mathcal{T}[\![\phi \rightarrow \psi]\!]_\tau \stackrel{df}{=} \mathcal{T}[\![\phi]\!]_\tau \leq \mathcal{T}[\![\psi]\!]_\tau \text{ , } \mathcal{T}[\![\phi \leftrightarrow \psi]\!]_\tau \stackrel{df}{=} \mathcal{T}[\![\phi]\!]_\tau \doteq \mathcal{T}[\![\psi]\!]_\tau)$$

A 1-1 Correspondence

Notice the 1-1 correspondence (see μ in definition -1.13) between the set of operators of the language of propositional logic and those of the algebra $2'$.

$$\begin{array}{lcl} \perp & \longleftrightarrow & 0 \\ \top & \longleftrightarrow & 1 \\ \neg & \longleftrightarrow & \bar{} \\ \wedge & \longleftrightarrow & . \\ \vee & \longleftrightarrow & + \\ \rightarrow & \longleftrightarrow & \leq \\ \leftrightarrow & \longleftrightarrow & \doteq \end{array}$$

Exercise 3.1

1. If we were to extend the language \mathcal{P}_0 to include the operator \leftarrow mentioned earlier. Define its meaning so that it is consistent with what was mentioned.
2. With the inclusion of \leftarrow , what operator needs to be added to the algebra $\mathbf{2}'$ so as to maintain and extend the 1-1 correspondence.
3. What boolean identities can you state for the new boolean operator defined above?

Partial truth assignment and truth tables.

Even though in our semantics of propositions we have used a truth assignment function that is total, it is strictly not necessary. Every formula ϕ in propositional logic (and later in predicate logic), however complex, is made up of only a finite number of atoms. Hence the truth of ϕ depends only on the truth assignments given to the non-constant atomic propositions $a\text{prop}(\phi) = \text{atoms}(\phi) - \{\perp, \top\}$. We need an infinite supply of atomic propositions only to ensure that even theoretically we have an inexhaustible supply.

In fact by lemma -4.4 it follows that if two truth assignments τ and τ' are strongly (or weakly) equivalent upto the set $a\text{prop}(\phi)$, i.e $\tau =_{a\text{prop}(\phi)} \tau'$ then $\mathcal{T}[\![\phi]\!]_\tau = \mathcal{T}[\![\phi]\!]_{\tau'}$. Further, it is obvious that since $a\text{prop}(\phi)$ is a finite set, the equivalence $=_{a\text{prop}(\phi)}$ partitions the set of all truth assignments into $2^{|a\text{prop}(\phi)|}$ equivalence classes and hence it is only necessary to consider $2^{|a\text{prop}(\phi)|}$ distinct partial truth assignments, in each of which every atomic proposition has a well-defined truth value. Given an ordering of the atomic propositions, these distinct

partial truth assignments may be enumerated in the form of a table, which we call a **truth table**. Similar remarks apply to any finite collection $\{\phi_i \mid 0 < i \leq n\}$ of atomic propositions. To consider the truth assigned by various truth assignments it would be necessary to consider only a single truth table with $2^{\sum_{0 < i \leq n} |aprop(\phi_i)|}$ partial truth assignments.

Models and Satisfiability

Definition 3.3: Model

A truth assignment τ is called a **model** of a formula ϕ (denoted $\tau \Vdash \phi$), if and only if $\mathcal{T}[\![\phi]\!]_{\tau} = 1$. τ is said to **satisfy** the formula ϕ .

Definition 3.4: Satisfiability

A formula is **satisfiable** if it has a model. Otherwise it is said to be **unsatisfiable**. A set S of formulae is **satisfiable** if there is a model τ which satisfies every formula in S (denoted $\tau \Vdash S$). Otherwise S is **unsatisfiable**.

Fact 3.1: Satisfiability of finite sets of formulae

For any finite set $S = \{\phi_i \mid 1 \leq i \leq n\}$ of formulae, $\tau \Vdash S$ if and only if $\tau \Vdash \phi_1 \wedge \cdots \wedge \phi_n$.

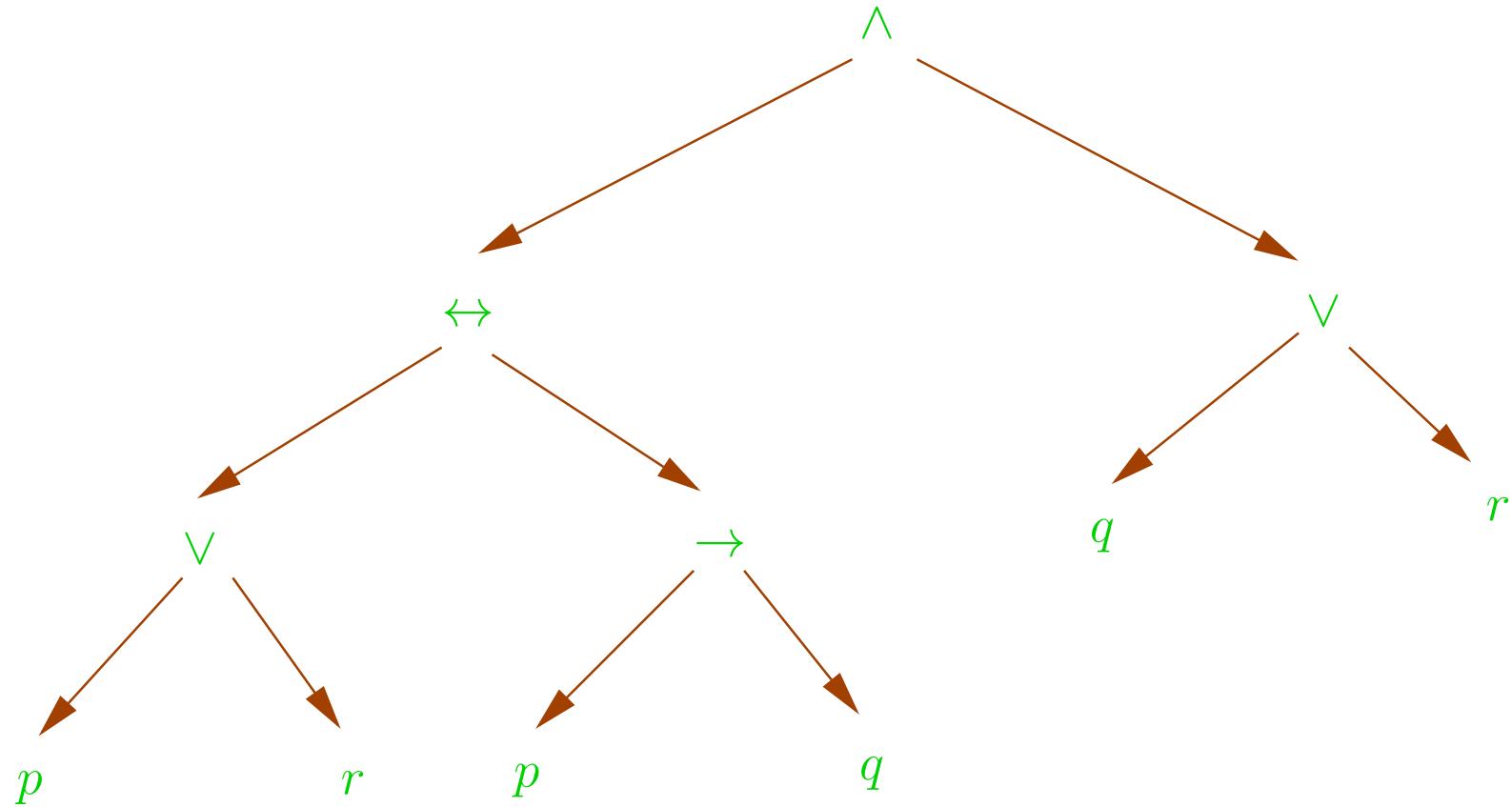
Example: Abstract Syntax trees

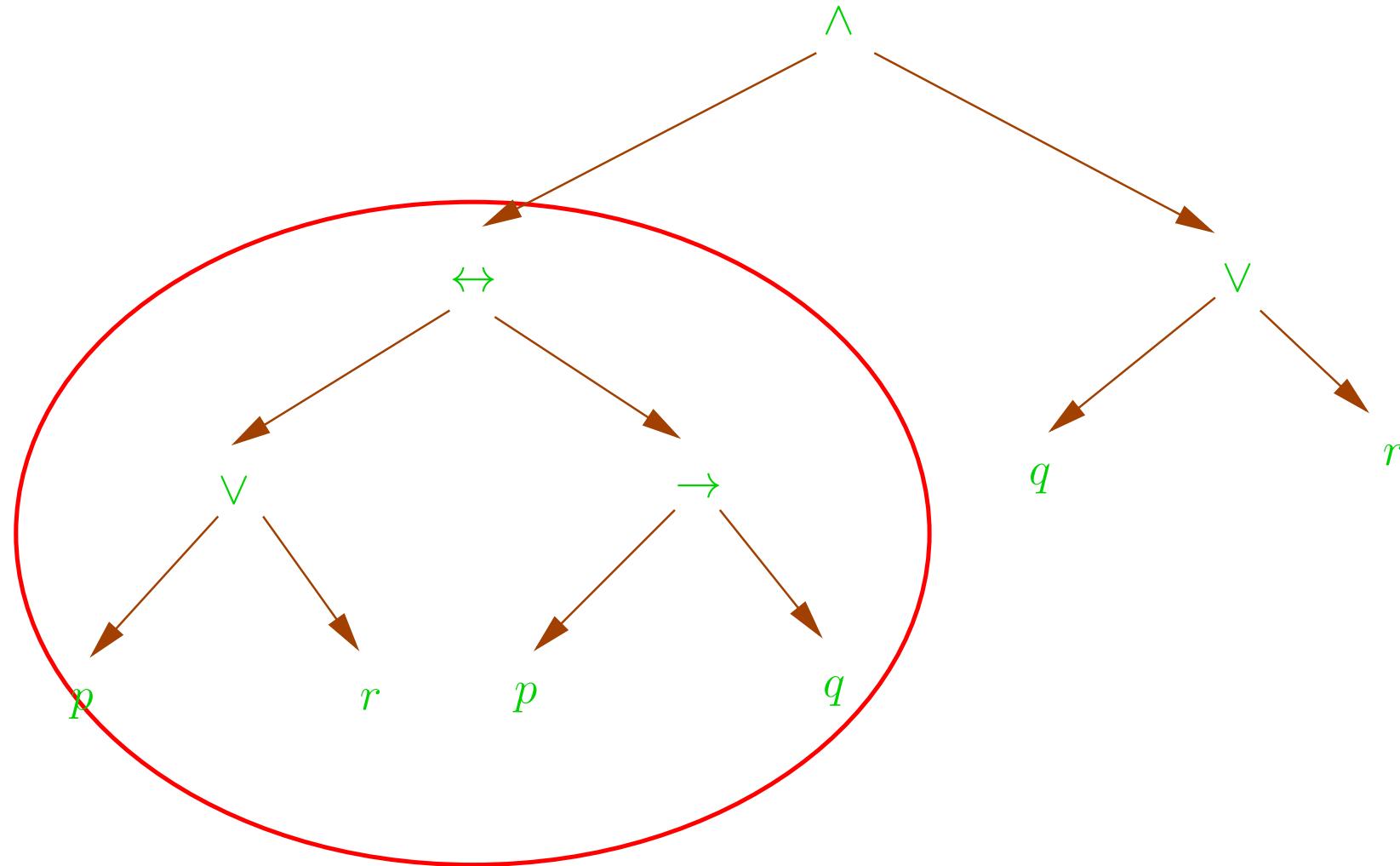
In the following example we illustrate the evaluation of the truth value of the proposition containing the atoms p , q and r

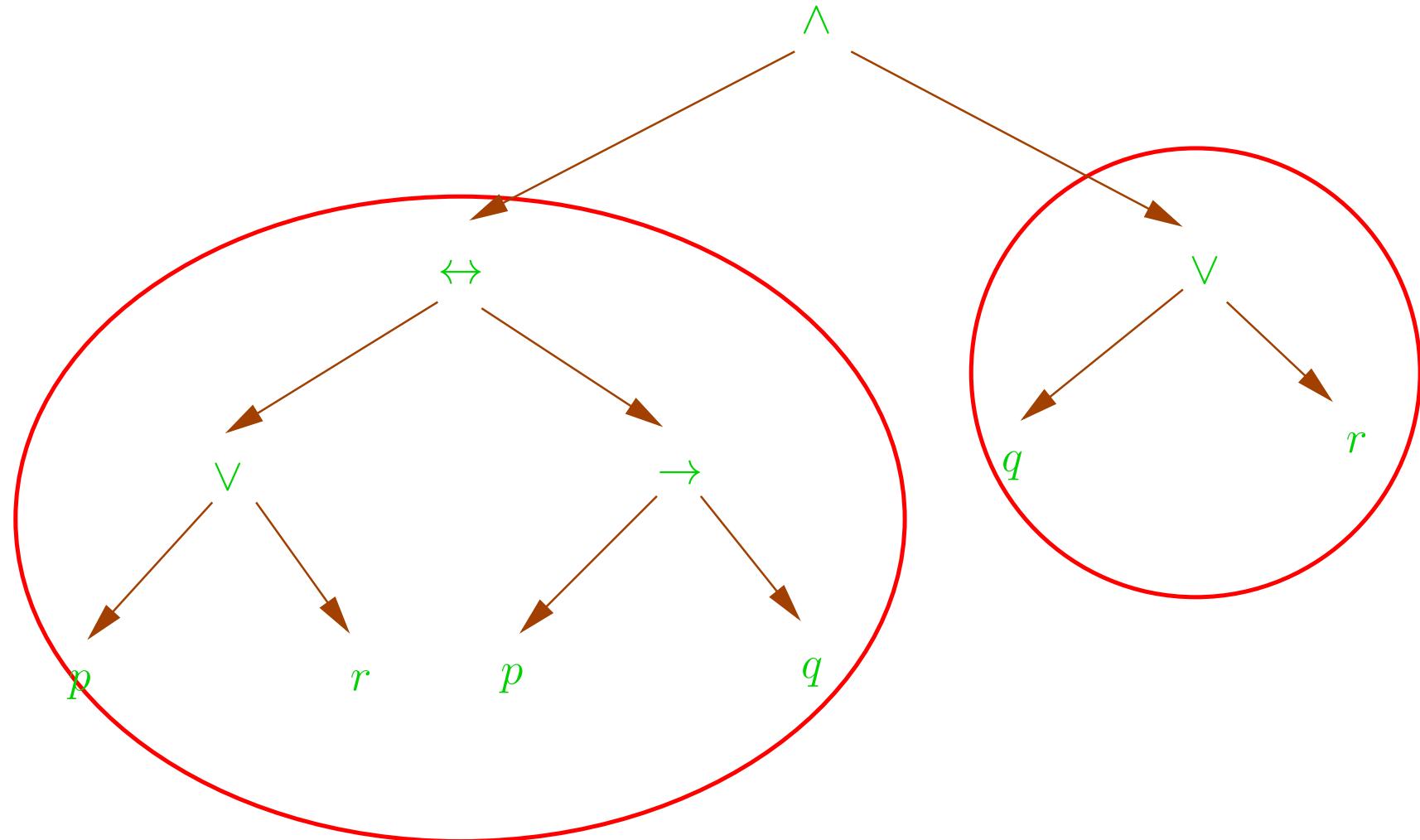
$$(((p \vee r) \leftrightarrow (p \rightarrow q)) \wedge (q \vee r))$$

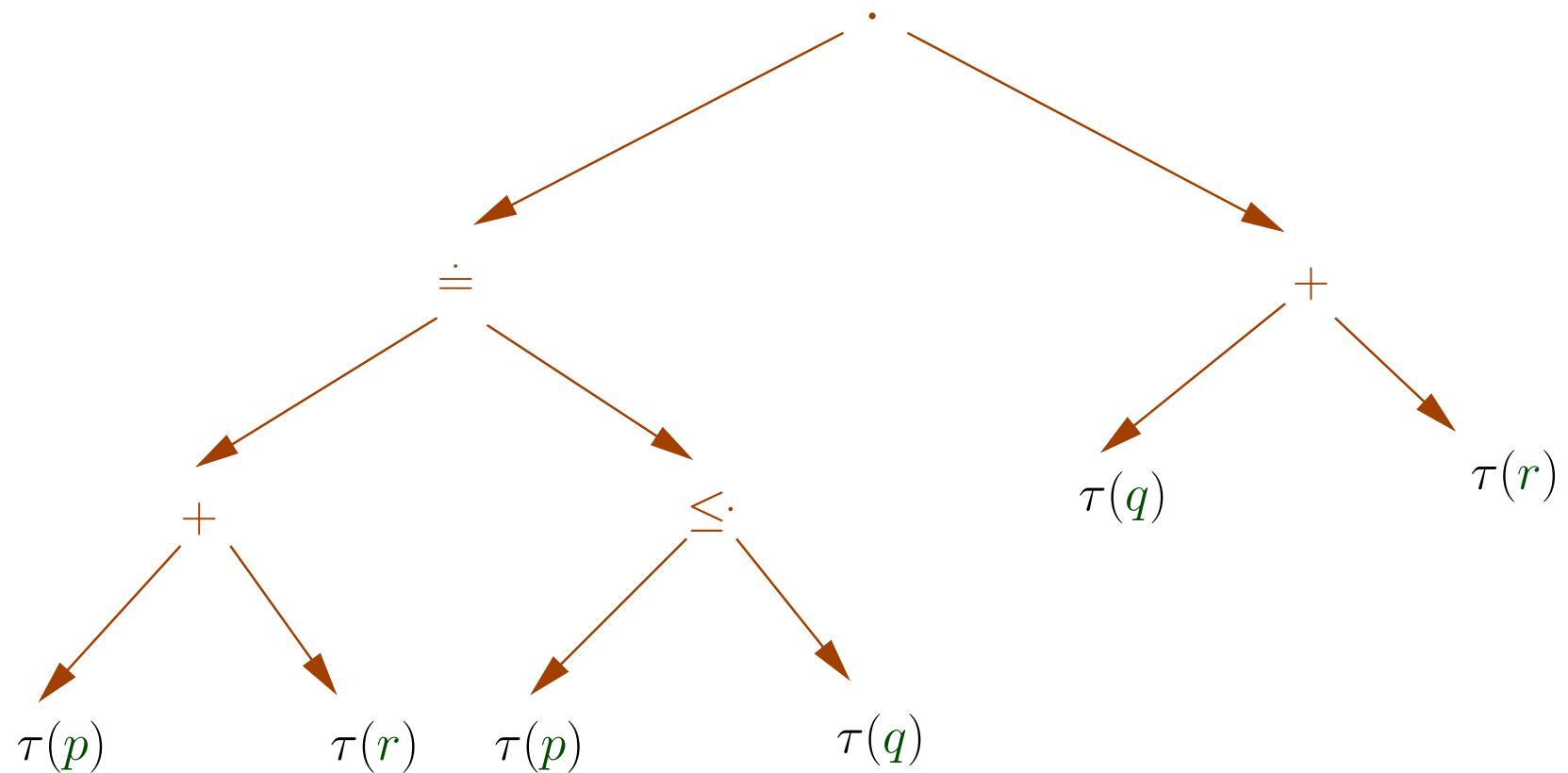
using the **semantics** to

$$(((\tau(p) + \tau(r)) \doteq (\tau(p) \leq \tau(q))).(\tau(q) + \tau(r)))$$









Tautology, Contradiction, Contingent

Definition 3.5: Tautology, Contradiction, Contingent

A proposition is said to be a **tautology** or logically valid if it is true under all truth assignments. Otherwise it is **invalid**.

contradiction or unsatisfiable if it is false under all truth assignments. Otherwise it is **satisfiable**.

contingent if it is neither a tautology nor a contradiction.

- A formula is **satisfiable** if it is not a contradiction.
- A formula is **falsifiable** if it is not a tautology.
- A contingent formula is both satisfiable and falsifiable.

Exercise 3.2: Semantics of Propositional logic

1. Prove that for each truth assignment τ , the function $\mathcal{T}[\cdot]_\tau$ is a homomorphism (see definition -1.13) between the algebras $\mathcal{P}_0 = \langle \mathcal{P}_0, \Omega_0, \{\equiv\} \rangle$ and $\mathbf{2}'$.
2. Prove that if two truth assignments τ and τ' are exactly the same for elements in $\text{atoms}(\phi)$ for any formula ϕ , then $\tau \Vdash \phi$ if and only if $\tau' \Vdash \phi$.
3. Any homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ from an algebra \mathcal{A} to another algebra \mathcal{B} (thus establishing a 1-1 correspondence between the signatures of the two algebras) induces an equivalence relation on \mathcal{A} . What is the nature of the equivalence relation $=_\tau$ induced by $\mathcal{T}[\cdot]_\tau$ for a truth assignment τ ?

4. Logical and Algebraic Concepts

4: (Meta-)Logical and Algebraic Concepts

Today you are you!
That is truer than true!
There is no one alive
who is you-er than you!

Dr. Seuss

1. Unsatisfiability and Inconsistency
2. Logical Consequence: 1
3. Logical Consequence: 2
4. Other Theorems
5. Logical Implication and Equivalence
6. Implication & Equivalence
7. Propositional Logic is An Algebra
8. Logical Equivalence as a Congruence

4.1. Some Meta-Logical Concepts

We now introduce some semantical concepts that will be used in the sequel to deal with various aspects of logic as a discipline for reasoning, as a theory of reasoning and as a mathematical discipline in its own right.

The main concepts that we need for both propositional logic and first-order logic are the following.

- Logical consequence
- Validity and invalidity of a formula
- Logical implication and Logical equivalence
- Satisfiability and unsatisfiability of a set of formulae

Many of these notions can be quite confusing especially when we restrict attention to propositional logic. They need to be regarded as concepts that are common to all logics. The confusion in the case of propositional logic is due to the following.

1. They have very similar or analogous concepts already present in the logical language (e.g. validity and tautologousness in propositional logic)
2. The term *tautology* refers to only an individual formula in propositional logic or as we shall see soon it refers to the shape and structure of its abstract syntax tree which makes it true independent of the truth assignment.
3. The corresponding logical concept viz. *logical validity* in first-order logic will not be referred to as a tautology unless it has the same shape and form as a tautology in propositional logic.

4. Different authors use different terms for synonymous concepts (*unsatisfiability* of a formula and *contradiction* in propositional logic).
5. Some of these concepts are for individual formulae (e.g. *validity*) while others are for sets of formulae (e.g. *satisfiability* and *unsatisfiability*).
6. The same term is overloaded for both single formulae and sets of formulae (e.g. *satisfiability*).
7. Most of the literature on logic tends to identify many of these concepts even though they are conceptually distinct (*logical consequence* and *logical implication* in propositional logic)

We have tried to clarify all these terms as far as possible. It is therefore important that the reader

- carefully follow the colour coding that distinguishes the object language from the language of reasoning about the objects (viz. the formulae) in the object language.
- understand the type of each concept – for example $\models \subseteq 2^{\mathcal{P}_0} \times \mathcal{P}_0$.
- recognize that the concepts may be somehow mutually related especially in propositional logic (see for example theorems 4.1, 4.2 and corollary 4.1).

Unsatisfiability and Inconsistency

Definition 4.1: Unsatisfiability and Inconsistency

Let Γ be a set of formulae. We say Γ is **unsatisfiable** or **inconsistent** if there is no truth assignment for which every formula in Γ is true simultaneously. Otherwise Γ is said to be **satisfiable** (and **consistent**).

Logical Consequence: 1

Definition 4.2: Logical consequence

A proposition $\phi \in \mathcal{P}_0$ is called a **logical consequence** of a set $\Gamma \subseteq \mathcal{P}_0$ of formulas (denoted $\Gamma \models \phi$) if any truth assignment that satisfies all formulas of Γ also satisfies ϕ .

- When $\Gamma = \emptyset$ then logical consequence reduces to **logical validity**.
- $\models \phi$ denotes that ϕ is logically valid i.e. a **tautology** in propositional logic.
- $\Gamma \not\models \phi$ denotes that ϕ is not a logical consequence of Γ .
- $\not\models \phi$ denotes that ϕ is logically invalid i.e. it may be **contingent** or a **contradiction** in propositional logic.

Logical Consequence: 2

Here is the first meta-logical theorem that relates the various concepts we have defined. It is proved using the **semantics** of propositional logic.

Theorem 4.1: Logical consequence and tautology

Let $\Gamma = \{\phi_i \mid 1 \leq i \leq n\}$ be a finite set of propositions, and let ψ be any proposition. Then $\Gamma \models \psi$ if and only if $((\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)$ is a **tautology**.



Proof of theorem 4.1.

Proof: Assume $\Gamma = \{\phi_i \mid 1 \leq i \leq n\}$. Since the theorem states an if and only if condition there are two parts to prove.

(\Rightarrow) Assume $\Gamma \models \psi$. To show that $((\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)$ is a tautology, we start by assuming that it is not a tautology and then arrive at a contradiction. If it is not a tautology, then there exists a truth assignment τ such that $\mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)]_{\tau} = 0$. By the definition of $\mathcal{T}[\cdot]_{\tau}$ we have

$$\begin{aligned}
 & \mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)]_{\tau} = 0 \\
 \text{iff } & (\mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n)]_{\tau} \leq \mathcal{T}[\psi]_{\tau}) = 0 \\
 \text{iff } & (\mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n)]_{\tau}) = 1 \text{ and } \mathcal{T}[\psi]_{\tau} = 0 \\
 \text{iff } & \mathcal{T}[\phi_1]_{\tau}, \dots, \mathcal{T}[\phi_n]_{\tau} = 1 \text{ and } \mathcal{T}[\psi]_{\tau} = 0 \\
 \text{iff } & \mathcal{T}[\phi_1]_{\tau} = \dots = \mathcal{T}[\phi_n]_{\tau} = 1 \text{ and } \mathcal{T}[\psi]_{\tau} = 0
 \end{aligned}$$

i.e.

$$\mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)]_{\tau} = 0 \text{ iff } \mathcal{T}[\phi_1]_{\tau} = \dots = \mathcal{T}[\phi_n]_{\tau} = 1 \text{ and } \mathcal{T}[\psi]_{\tau} = 0 \quad (20)$$

Hence τ is a truth assignment in which every formula in Γ is true but ψ is false. This contradicts the notion of logical consequence (definition 4.2) and hence the assumption that $((\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)$ is not a tautology is wrong.

(\Leftarrow) Assume $((\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)$ is a tautology. We prove $\Gamma \models \psi$ by assuming the contrary viz. assume $\Gamma \not\models \psi$. Then there exists a truth assignment τ such that

$$\mathcal{T}[\phi_1]_{\tau} = \dots = \mathcal{T}[\phi_n]_{\tau} = 1 \text{ and } \mathcal{T}[\psi]_{\tau} = 0$$

By (20), we obtain

$$\mathcal{T}[(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)]_{\tau} = 0$$

from which it follows that $((\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \rightarrow \psi)$ is not a tautology, contradicting our assumption. QED

The following results may also be proved using the semantics of propositional logic.

Other Theorems

Theorem 4.2: More on logical consequence

Let $\Gamma = \{\phi_i \mid 1 \leq i \leq n\}$ be a finite set of propositions, and let ψ be any proposition. Then

1. $\Gamma \models \psi$ if and only if $\models \phi_1 \rightarrow (\phi_2 \rightarrow \dots (\phi_n \rightarrow \psi) \dots)$
2. $\Gamma \models \psi$ if and only if $((\dots ((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \wedge \neg\psi)$ is a contradiction.

Corollary 4.1

A formula ϕ is a tautology iff $\neg\phi$ is a contradiction (unsatisfiable).

Logical Implication and Equivalence

Definition 4.3: Logical implication

A formula ϕ logically implies another formula ψ (denoted $\phi \Rightarrow \psi$) iff $\models \phi \rightarrow \psi$. \Rightarrow is called (logical) implication. We may also say ψ is logically implied by ϕ and denote it by $\psi \Leftarrow \phi$.

Definition 4.4: Logical equivalence

A formula ϕ is logically equivalent to another formula ψ (denoted $\phi \Leftrightarrow \psi$) iff $\models \phi \leftrightarrow \psi$. \Leftrightarrow is called (logical) equivalence.

Fact 4.1: Implication and Equivalence as tautologies

1. $\phi \Rightarrow \psi$ if and only if $\phi \rightarrow \psi$ is a tautology.
2. $\phi \Leftrightarrow \psi$ if and only if $\phi \leftrightarrow \psi$ is a tautology.

Implication & Equivalence

Fact 4.2

1. $\phi \Rightarrow \psi$ iff $\{\phi\} \models \psi$.
2. $\phi \Leftrightarrow \psi$ iff $\phi \Rightarrow \psi$ and $\psi \Rightarrow \phi$.
3. $\phi \Leftrightarrow \psi$ iff $\phi \Rightarrow \psi$ and $\phi \Leftarrow \psi$.
4. \Rightarrow is a preordering (reflexive and transitive) relation on \mathcal{P}_0 .
5. \Leftrightarrow is the kernel of \Rightarrow i.e. $\Leftrightarrow = \Rightarrow \cap \Rightarrow^{-1}$ and is hence indeed an equivalence relation on \mathcal{P}_0 .

Notes.

1. Many logicians and mathematicians tend to overload the same symbol \rightarrow for both the conditional operator and logical implication (\Rightarrow). However we have chosen to keep them separate since \rightarrow is an operator in the language \mathcal{P}_0 whereas logical implication (\Rightarrow) are both meta-logical concepts. Algebraically speaking, they are both binary relations on formulae i.e. $\Rightarrow, \Leftrightarrow \subseteq \mathcal{P}_0 \times \mathcal{P}_0$.
2. Analogous remarks as above also apply to the overloading of the symbols \leftrightarrow which is an operator of the language \mathcal{P}_0 and logical equivalence \Leftrightarrow .
3. Many logicians and mathematicians also tend to use the logical implication (\Rightarrow) to stand for logical consequence \models . However even though they are both related meta-logical concepts, they fundamentally differ in the fact that logical implication relates a pair of formulae while logical consequence relates a set (which could be empty, finite or even infinite) of formulae to a single formula.

Propositional Logic is An Algebra

Propositional Logic taken together with the concepts of logical implication and logical equivalence then becomes an algebraic structure whose ground terms are isomorphic to the boolean expressions of the **boolean algebra $2'$** .

$$P_0 = \langle P_0; \{\perp, \top, \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}; \{\Rightarrow, \Leftrightarrow\} \rangle$$

Fact 4.3: Algebra of propositions

1. $P_0 = T_{\Omega_0}(A)$ i.e. is the set of terms (section -1.2) inductively generated from the signature Ω_0 .
2. The set T_{Ω_0} of ground terms is the set of propositions which contain no atoms.

Logical Equivalence as a Congruence

Theorem 4.3: Logical Equivalence as congruence

Logical equivalence is a congruence relation on \mathcal{P}_0 i.e. if $\phi \Leftrightarrow \psi$ then

- $\neg\phi \Leftrightarrow \neg\psi$ and
- for each $* \in \{\wedge, \vee, \rightarrow, \Leftrightarrow\}$ and every formula χ we have

$$\phi * \chi \Leftrightarrow \psi * \chi$$

$$\chi * \phi \Leftrightarrow \chi * \psi$$

Exercise 4.1: Logical and algebraic concepts

1. Use the semantics of propositional logic to prove theorem 4.2 and corollary 4.1.
2. Prove the facts 4.2.
3. Prove that logical equivalence is indeed a congruence relation on \mathcal{P}_0 .
4. Is logical implication a precongruence on propositions? Which operators preserve logical implication?
5. For each truth assignment τ let $=_\tau$ denote the equivalence defined in exercise 3.2. What is the relationship between \Leftrightarrow and $=_\tau$?
6. Prove that $\phi_1, \dots, \phi_n \models \psi$ if and only if for each $i, 1 \leq i \leq n$, $\phi_1, \dots, \phi_{i-1}, \phi_{i+1}, \dots, \phi_n, \neg\psi \models \neg\phi_i$.

5. Identities and Normal Forms

5: Identities and Normal Forms

“Be what you would seem to be - or, if you'd like it put more simply - never imagine yourself not to be otherwise than what it might appear to others that what you were or might have been was not otherwise than what you had been would have appeared to them to be otherwise.”

Lewis Caroll, *Alice's Adventures in Wonderland*

1. Adequacy
2. Adequacy: Examples
3. Functional Completeness
4. Expressively adequate sets
5. Duality
6. Dual Formulae
7. Dual Operators
8. Principle of Duality
9. Literals: Positive and Negative
10. Negation Normal Forms: 1
11. Negation Normal Forms: 2
12. Conjunctive Normal Forms
13. CNF and DNF

Some identities (Compare with the Boolean identities)

Negation	$\neg\neg\phi \Leftrightarrow \phi$
Conditional	$\phi \rightarrow \psi \Leftrightarrow \neg\phi \vee \psi$
Biconditional	$\phi \leftrightarrow \psi \Leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$

$\phi \vee \perp \Leftrightarrow \phi$ $\phi \vee \top \Leftrightarrow \top$ $\phi \vee \phi \Leftrightarrow \phi$ $\phi \vee \psi \Leftrightarrow \psi \vee \phi$ $(\phi \vee \psi) \vee \chi \Leftrightarrow \phi \vee (\psi \vee \chi)$ $\phi \vee (\psi \wedge \chi) \Leftrightarrow (\phi \vee \psi) \wedge (\phi \vee \chi)$ $\neg(\phi \vee \psi) \Leftrightarrow \neg\phi \wedge \neg\psi$ $\phi \vee \neg\phi \Leftrightarrow \top$ $\neg\perp \Leftrightarrow \top$ $\phi \vee (\phi \wedge \psi) \Leftrightarrow \phi$	$\text{Identity} \quad \phi \wedge \top \Leftrightarrow \phi$ $\text{Zero} \quad \phi \wedge \perp \Leftrightarrow \perp$ $\text{Idempotence} \quad \phi \wedge \phi \Leftrightarrow \phi$ $\text{Commutativity} \quad \phi \wedge \psi \Leftrightarrow \psi \wedge \phi$ $\text{Associativity} \quad (\phi \wedge \psi) \wedge \chi \Leftrightarrow \phi \wedge (\psi \wedge \chi)$ $\text{Distributivity} \quad \phi \wedge (\psi \vee \chi) \Leftrightarrow (\phi \wedge \psi) \vee (\phi \wedge \chi)$ $\text{De Morgan} \quad \neg(\phi \wedge \psi) \Leftrightarrow \neg\phi \vee \neg\psi$ $\text{Simplification} \quad \phi \wedge \neg\phi \Leftrightarrow \perp$ $\text{Inversion} \quad \neg\top \Leftrightarrow \perp$ $\text{Absorption} \quad \phi \wedge (\phi \vee \psi) \Leftrightarrow \phi$
--	---

5.0.1. Adequacy

If we were to refer to the problems in exercise -4.12 it is clear that there are only a finite number of functions between finite sets. On the other hand there are an uncountable number of functions on a countable set (see problem 4 in exercise -4.14).

In general, it is inconceivable that all the possible functions from a countable set to a finite set will be expressible (upto equality) using only a finite number of them by composing (see part 3 in definition -4.10 and the problems in exercise -4.3) them in various ways. For instance we cannot conceive of all binary functions on \mathbb{N} or \mathbb{Z} to be expressible using only the operations of addition, subtraction, multiplication and division, since there are an uncountable number of them.

However there are special cases such as the set \mathcal{P} where a small finite set of functions suffice to be able to express all functions of all arities. Such a set of functions will be termed to be adequate for \mathcal{P} . In fact, this property of adequacy is fundamental to the design of digital hardware in 2-valued logic. And it uses the **Boolean identities**. We lift this fact about boolean algebra to the language of propositional logic, using its semantics and the **identities** to show that a very small number of propositional operators suffice to express any propositional operator of any arity whatsoever upto logical equivalence i.e. given any operator of any arity that one might want to include in propositional logic, it is possible to define a logically equivalent operator using only the operators we already have. This property is also called **functional completeness or expressive adequacy**.

Adequacy

Consider the two identities:

$$\phi \leftrightarrow \psi \Leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \quad (21)$$

$$\phi \rightarrow \psi \Leftrightarrow \neg\phi \vee \psi \quad (22)$$

Definition 5.1: Adequacy

A set of operators $O \subseteq \Omega_0$ is said to be **adequate** for \mathcal{P}_0 , if for every formula in \mathcal{P}_0 there is a **logically equivalent** formula using only the operators in O .

Adequacy: Examples

Example 5.1: Adequate sets

1. From the identities (21) and (22) and the two Simplification identities it is clear that $O = \{\neg, \wedge, \vee\}$ is an adequate set of operators for \mathcal{P}_0 .
2. Further given that $O = \{\neg, \wedge, \vee\}$ is adequate and using the De Morgan identity and Negation, we have that

$$\phi \wedge \psi \Leftrightarrow \neg\neg(\phi \wedge \psi) \Leftrightarrow \neg(\neg\phi \vee \neg\psi)$$

and hence $\{\neg, \vee\}$ is an adequate set.

3. We may use the other De Morgan identity

$$\phi \vee \psi \Leftrightarrow \neg\neg(\phi \vee \psi) \Leftrightarrow \neg(\neg\phi \wedge \neg\psi)$$

to conclude that $\{\neg, \wedge\}$ is adequate.

Functional Completeness

It is quite possible that one could extend \mathcal{P}_0 with new operators (perhaps 3-ary or 4-ary or indeed of any arity) and thus make the language more *expressive*.

Definition 5.2: Expressive adequacy

A set O of operators for propositional logic is **functionally complete** (also called **expressively adequate**) if any formula built up using the operators of Ω_0 is logically equivalent to a formula built using only operators from O .

Exercise 5.1

Try to come up with a functionally complete set of operators for the set $= \{0, 1, 2\}$.

Expressively adequate sets

Lemma 5.1: Expressively adequate set

$\{\neg, \wedge, \vee\}$ is a functionally complete set.



Corollary 5.1

Any adequate set of operators for Ω_0 is also functionally complete for \mathcal{P}_0 .

Proof of lemma 5.1.

Proof: By the **semantics** of propositional logic, every operator of propositional logic **corresponds** to an operator of the same arity on the boolean set $\{0, 1\}$. The proof follows from the construction of **truth tables** in the boolean algebra **2**. We show in the sequel that every truth table may be expressed using the boolean operators $\{\neg, \cdot, +\}$.

	a_1	\cdots	a_n	b
0	0	\cdots	0	b_0
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
$2^n - 1$	1	\cdots	1	$b_{2^n - 1}$

Let $o_n : \mathcal{2}^n \longrightarrow \mathcal{2}$ be any n -ary operator on boolean values. Typically there exists a truth table for the function $o_n(a_1, \dots, a_n)$, which defines for each possible set of boolean values of the arguments the boolean value of $o_n(a_1, \dots, a_n) = b$. This truth table consists of 2^n rows and $n + 1$ columns as shown in the table where $b_0, \dots, b_{2^n - 1} \in \mathcal{2}$ (we have numbered the rows by the decimal equivalent of the binary number that the bit-vector (a_1, \dots, a_n) denotes in each case). For $0 \leq i \leq 2^{n-1}$, $1 \leq j \leq n$, let $a_{ij} \in \mathcal{2}$ be the value assigned to variable a_j such that $b_i = o_n(a_{i1}, \dots, a_{in})$. We may express the function as a sum of products. For each boolean value x , let $x^* = \begin{cases} x & \text{if } x = 1 \\ \bar{x} & \text{if } x = 0 \end{cases}$.

The function expressed as a sum of products would be true exactly when the product representing the boolean values of the parameters is true. Letting a_{jr} denote the value of the boolean variable a_j ($1 \leq j \leq n$) in row r ($0 \leq r < 2^n$) and b_r the value of the function in row r , the sum of products which represents this function is then given by

$$o_n(a_1, \dots, a_n) = \sum_{b_r=1, 0 \leq r < 2^n} \left(\prod_{1 \leq j \leq n} a_{rj}^* \right)$$

QED

Exercise 5.2: Adequacy

1. Prove that the following sets are adequate for \mathcal{P}_0 .
 - (a) $\{\rightarrow, \neg\}$
 - (b) $\{\rightarrow, \perp\}$
2. Prove that $\{\wedge, \vee\}$ is not an adequate set. (*Hint: Prove that \neg is not definable upto \Leftrightarrow .*)
3. Prove that if $O \subseteq \Omega$ then $\neg \in O$ or $\perp \in O$.
4. It is possible to take some of the meta-logical concepts defined earlier (4.1) and convert them into operators of propositional logic. As an example let us take the concept of **logical validity** and define a new unary operator \models with the following meaning

$$\mathcal{T}[\models \phi]_\tau \stackrel{\text{df}}{=} \begin{cases} 1 & \text{if for every truth assignment } \tau', \mathcal{T}[\phi]_{\tau'} = 1 \\ 0 & \text{otherwise} \end{cases}$$

- (a) By **functional completeness** this operator should be expressible (upto logical equivalence) using one of the adequate sets of operators. Express $\models \phi$ using only the operators in $\{\rightarrow, \neg\}$.
- (b) Define another unary operator \Vdash which exactly represents the concept of satisfiability of a formula i.e. $\Vdash \phi$ is true exactly when there is exists a truth assignment which makes ϕ true and false otherwise.
- (c) Express $\Vdash \phi$ using only the operators $\{\neg, \vee\}$.

5.1. Duality

Duality

Definition 5.3: Duality

The dual of a formula ϕ , denoted ϕ^∂ , is defined by induction on the structure of formulae as follows

$$\begin{aligned}\perp^\partial &\stackrel{df}{=} \top & \top^\partial &\stackrel{df}{=} \perp \\ (\phi \wedge \psi)^\partial &\stackrel{df}{=} \phi^\partial \vee \psi^\partial & (\phi \vee \psi)^\partial &\stackrel{df}{=} \phi^\partial \wedge \psi^\partial \\ (\phi \rightarrow \psi)^\partial &\stackrel{df}{=} \phi^\partial \leftarrow \psi^\partial & (\phi \leftarrow \psi)^\partial &\stackrel{df}{=} \phi^\partial \rightarrow \psi^\partial \\ (\phi \leftrightarrow \psi)^\partial &\stackrel{df}{=} \phi^\partial \leftrightarrow \psi^\partial\end{aligned}$$

Dual Formulae

More informally, formulae ϕ and ψ are called **duals** of each other if each can be obtained from the other by *simultaneously* replacing all occurrences of

- \perp by \top and \top by \perp
- \wedge by \vee and \vee by \wedge ,
- \rightarrow by \leftarrow and \leftarrow by \rightarrow ,
- \leftrightarrow by itself (self-dual).

Dual Operators

We may then directly extend this notion to the individual operators themselves.

- \top and \perp are **duals** of each other
- \wedge and \vee are **duals** of each other
- \rightarrow and \leftarrow are duals of each other and
- \leftrightarrow is its own dual (*self-dual*)

Definition 5.4: Dual operators

$$\begin{aligned}\perp^\partial &= \top, \quad \top^\partial = \perp \\ \wedge^\partial &= \vee, \quad \vee^\partial = \wedge \\ \rightarrow^\partial &= \leftarrow, \quad \leftarrow^\partial = \rightarrow \\ \leftrightarrow^\partial &= \leftrightarrow\end{aligned}$$

Principle of Duality

Theorem 5.1: Principle of duality

1. If $\text{atoms}(\phi) = \{p_1, \dots, p_n\}$, then if $\phi \equiv o(p_1, \dots, p_n)$ then

$$\neg\phi \Leftrightarrow o^\partial(\neg p_1, \dots, \neg p_n)$$

2. if $\phi \Rightarrow \psi$ then $\psi^\partial \Rightarrow \phi^\partial$ and

3. if $\phi \Leftrightarrow \psi$ then $\psi^\partial \Leftrightarrow \phi^\partial$ and

Proof: By structural induction and the use of the De Morgan and other laws.

QED

By extension for any compound operator o , o^∂ denotes its dual.

5.2. Normal Forms

Literals: Positive and Negative

The adequacy and functional completeness of the set $\{\neg, \wedge, \vee\}$ may be used to build normal forms (or standard forms) by using the logical equivalences as rewrite rules.

Definition 5.5: Literals: Positive and Negative

- A literal is an atom ($p \in A$) or its negation ($\neg p$ for $p \in A$).
- Atoms are called **positive** literals and their negations are called **negative** literals.

Negation Normal Forms: 1

A formula is in *negation normal form* if it is built up from *literals* using only the operators \vee and \wedge . More precisely,

Definition 5.6: Negation normal form

If \mathcal{L}_0 is the set of literals then the set \mathcal{N}_0 of negation normal forms is defined by the BNF

$$\mu, \nu ::= \lambda \in \mathcal{L}_0 \mid (\mu \wedge \nu) \mid (\mu \vee \nu) \quad (23)$$

Negation Normal Forms: 2

Lemma 5.2: NNF

Every formula in \mathcal{P}_0 is logically equivalent to one in negation normal form.

Proof: It suffices to consider only formulas containing the operators \neg , \wedge and \vee , and for every occurrence of negation to push it inward using the De Morgan identities and remove double negations whenever they occur.

QED

Conjunctive Normal Forms

Definition 5.7: CNF

- A **disjunction of literals** is a formula δ of the form

$$\delta \equiv \lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_m \equiv \bigvee_{1 \leq i \leq m} \lambda_i$$

where $m \geq 0$.

- A **conjunctive normal form** is a formula γ of the form $\delta_1 \wedge \delta_2 \wedge \cdots \wedge \delta_n$ where δ_j for each $1 \leq j \leq n$ is a disjunction of literals.

We may analogously define a *conjunction of literals* and a *disjunctive normal form (DNF)*.

CNF and DNF

Theorem 5.2: CNF

Every formula in \mathcal{P}_0 is logically equivalent to a conjunctive normal form.

Proof: It suffices to consider only negation normal forms. In each case use the **distributive law** to distribute \vee over \wedge and use the **negation law** to remove multiple contiguous occurrences of negations. It may also be necessary to use other identities e.g. **Simplification, Inversion, Identity and Zero**. QED

Theorem 5.3: DNF

Every formula in \mathcal{P}_0 is logically equivalent to a disjunctive normal form.

Proof: Analogous proof except use the distribution of \wedge over \vee instead. QED

Exercise 5.3: Duality and Normal forms

1. Let o be a ternary operator defined by $o(\phi, \psi, \chi) \equiv (\phi \wedge \psi) \vee (\neg\phi \wedge \chi)$. What is o^∂ ?
2. Let $\phi \equiv o_1(p_1, \dots, p_n)$ and $\psi \equiv o_2(p_1, \dots, p_n)$ be formulas such that $\phi \Leftrightarrow \psi$. Then prove that

$$\begin{aligned} o_1(\neg p_1, \dots, \neg p_n) &\Leftrightarrow o_2(\neg p_1, \dots, \neg p_n) \\ \neg o_1^\partial(p_1, \dots, p_n) &\Leftrightarrow \neg o_2^\partial(p_1, \dots, p_n) \end{aligned}$$

3. Define BNFs to generate exactly
 - (a) the set \mathcal{C}_0 of conjunctive normal forms
 - (b) the set \mathcal{D}_0 of disjunctive normal forms.
4. Using principles of induction prove that
 - (a) $\mathcal{L}_0 \subset \mathcal{C}_0 \subset \mathcal{N}_0 \subset \mathcal{P}_0$,
 - (b) $\mathcal{L}_0 \subset \mathcal{D}_0 \subset \mathcal{N}_0 \subset \mathcal{P}_0$,
 - (c) Is $\mathcal{L}_0 = \mathcal{C}_0 \cap \mathcal{D}_0$? Justify your answer.

6. Tautology Checking

6: Tautology Checking



1. Arguments
2. Arguments: 2
3. Validity & Falsification
4. Translation into propositional Logic
5. Atoms in Argument
6. The Representation
7. Propositional Rendering
8. The Strategy
9. Checking Tautology
10. Computing the CNF
11. Rewriting Conditionals, Biconditionals
12. Convert to NNF
13. Convert to CNF
14. Falsifying CNF

Arguments

A typical informally stated argument might go as follows:

If prices rise, then the poor and the salaried class will be unhappy.

If taxes are increased then the businessmen will be unhappy.

If the poor and the salaried class or the businessmen are unhappy, the Government will not be re-elected.

Inflation will rise if Government expenditure exceeds its revenue.

Government expenditure will exceed its revenue unless taxes are increased or the Government resorts to deficit financing or takes a loan from the IMF to cover the deficit.

If the Government resorts to deficit financing then inflation will rise.

If inflation rises, the prices will also rise.

The Government will get reelected.

Therefore the Government will take a loan from the IMF.

Arguments: 2

A typical informally stated argument might go as follows:

If prices rise, then the poor and the salaried class will be unhappy.

If taxes are increased then the businessmen will be unhappy.

If the poor and the salaried class are unhappy or the businessmen are unhappy, the Government will not be re-elected.

Inflation will rise if Government expenditure exceeds its revenue.

Government expenditure will exceed its revenue unless taxes are increased or the Government resorts to deficit financing or takes a loan from the IMF to cover the deficit.

If the Government resorts to deficit financing then inflation will rise.

If inflation rises, the prices will also rise.

The Government will get reelected.

Therefore the Government will take a loan from the IMF.

The boxes enclose atomic propositions.

Exercise 6.1: Propositional arguments in natural language

Translate the following arguments into propositional logic (You may have to identify the atoms and rewrite some sentences so as to be more suitable for a direct translation).

1. Therefore if a glass is half-full then it is half-full.
2. Therefore if a glass is half-full then it is half-empty.
3. If a glass is half-full then it is half-empty.
If a glass is full then it is empty.

4. Tame cats are non-violent and vegetarian.
Non-violent cats would not kill mice.
Vegetarian cats are bottle-fed.
Cats eat meat.
Therefore cats are not tame.

5. If we can directly know that God exists, then we can know God exists by experience.
If we can indirectly know that God exists, then we can know God exists by logical inference from experience.
If we can know that God exists, then we can directly know that God exists, or we can indirectly know that God exists.
If we cannot know God empirically, then we cannot know God by experience and we cannot know God by logical inference from experience.
If we can know God empirically, then “*God exists*” is a scientific hypothesis and is empirically justifiable.
“*God exists*” is not empirically justifiable.
Therefore we cannot know that God exists.

Exercise 6.2: More propositional arguments in natural language

1. Consider the following pair of arguments. Can you determine which of them (if any) is logically valid?

- If I am the Prime Minister then I am famous.
I am not the Prime Minister.
Therefore I am not famous.
- If I am the Prime Minister then I am famous.
I am not famous.
Therefore I am not the Prime Minister.

2. Consider the following pair of arguments and point out the differences in the translation into propositional logic. Can you determine which of them (if any) is logically valid?

- If the weather is warm and the sky is clear then I go for a swim or I go for a run.
It is not the case that if I do not go for a swim the sky is not clear.
Therefore the weather is warm or I go for a run.
- If the weather is warm and the sky is clear then either I go for a swim or I go for a run.
It is not the case that if I do not go for a swim the sky is not clear.
Therefore either the weather is warm or I go for a run.

Validity & Falsification

The validity of such arguments involves showing that the conclusion is a **logical consequence** of the hypotheses that precede it. The following alternatives exist:

1. Using theorem **4.1**
2. Using one of the parts of theorem **4.2**.
3. Using a **truth table** after converting the argument into a single propositional logic formula by invoking one of the parts of theorem **4.2** or corollary **4.1**.

If the argument is not valid, then a **falsifying** truth assignment needs to be also given.

Translation into propositional Logic

(see section 2.1 and 2.2)

But even after translating the argument into a suitable propositional logic form, it would be quite tedious to verify the validity of the argument by truth table, since the number of “atomic” propositions could be very large.

Atoms in Argument

The Argument

```
val rise = ATOM "Prices rise";
val pandsun = ATOM "The poor and ... will be unhappy";
val taxes = ATOM "Taxes are increased";
val busun = ATOM "The businessmen will be unhappy";
val reelect = ATOM "The Government will be re-elected";
val inflation = ATOM "Inflation will rise";
val exceeds = ATOM "Government expenditure ... revenue";
val deffin = ATOM "The Government resorts ...";
val imf = ATOM "The Govt. takes a loan ... deficit";
```

In this case the the truth table would have $2^9 = 512$ rows. Further, the truth table will use *all* the atoms, even the irrelevant ones.

The Representation

```
datatype Prop = ATOM of string
              | NOT of Prop
              | AND of Prop * Prop
              | OR of Prop * Prop
              | COND of Prop * Prop
              | BIC of Prop * Prop
```

Propositional Rendering

The Argument

```
val hyp1 = COND (rise , pandsun);  
val hyp2 = COND (taxes , busun);  
val hyp3 = COND (OR (pandsun , busun) , NOT(reelect));  
val hyp4 = COND (exceeds , inflation);  
val hyp5 = COND (exceeds , NOT(OR(taxes , OR(deffin , imf))));  
val hyp6 = COND (deffin , inflation);  
val hyp7 = COND (inflation , rise);  
val hyp8 = reelect;  
val concl = imf;  
val H = [hyp1 , hyp2 , ... , hyp8];  
val Arg1 = (H, concl);
```

Atoms in Argument

The Strategy

We need to either show that

$$Arg1 = \text{COND} (\text{bigAND } H, \text{concl})$$

is a tautology or can be falsified. Using theorem 4.1 for validity

```
val bigAND = leftReduce (AND);
fun Valid ((H, P):Argument) =
  if null (H) then tautology (P)
  else tautology (COND (bigAND (H), P))
```

and for falsification we use

```
fun falsifyArg ((H, P): Argument) =
  if null (H) then falsify (cnf(P))
  else falsify (cnf (COND (bigAND (H), P))))
```

[Skip to falsifying CNF](#)

Checking Tautology

Checking for tautology crucially involves finding falsifying truth assignments for at last one of the conjuncts in the CNF of the argument.

```
fun tautology2 (P) =  
  let val Q = cnf (P);  
    val LL = falsify (Q)  
  in    if null (LL) then (true , [])  
        else (false , LL)  
  end
```

[Next Computing CNF](#)

[Falsifying CNF](#)

Computing the CNF

1. Rewrite conditionals and biconditionals
2. Convert to NNF by pushing negation inward
3. Convert to CNF by distributing *OR* over *AND*

Rewriting Conditionals, Biconditionals

```
fun rewrite (ATOM a)      = ATOM a
|   rewrite (NOT (P))    = NOT ( rewrite (P))
|   rewrite (AND (P, Q)) = AND ( rewrite(P), rewrite(Q))
|   rewrite (OR (P, Q))  = OR ( rewrite(P), rewrite(Q))
|   rewrite (COND (P, Q)) = OR (NOT ( rewrite(P)), rewrite(Q))
|   rewrite (BIC (P, Q)) = rewrite (AND (COND(P, Q), COND (Q, P)))
```

Convert to NNF

```
(* Assume all conditionals and biconditionals have been rewritten *)
fun nnf (ATOM a) = ATOM a
| nnf (NOT (ATOM a)) = NOT (ATOM a)
| nnf (NOT (NOT (P))) = nnf (P)
| nnf (AND (P, Q)) = AND (nnf(P), nnf(Q))
| nnf (NOT (AND (P, Q))) = nnf (OR (NOT (P), NOT (Q)))
| nnf (OR (P, Q)) = OR (nnf(P), nnf(Q))
| nnf (NOT (OR (P, Q))) = nnf (AND (NOT (P), NOT (Q)))
```

Convert to CNF

(* Assume formula is in NNF *)

(* Distribute OR over AND to get a NNF into CNF *)

```
fun distOR (P, AND (Q, R)) = AND (distOR (P, Q), distOR (P, R))
| distOR (AND (Q, R), P) = AND (distOR (Q, P), distOR (R, P))
| distOR (P, Q)           = OR (P, Q)
```

```
fun C_of_D (AND (P, Q)) = AND (C_of_D (P), C_of_D (Q))
| C_of_D (OR (P, Q))   = distOR (C_of_D (P), C_of_D (Q))
| C_of_D (P)            = P
```

```
fun cnf (P) = C_of_D (nnf (rewrite (P)))
```

[Back to the strategy](#)

Falsifying CNF

Assume the CNF is $Q \equiv \bigwedge_{i=1}^m D_i$ where each $D_i \equiv \bigvee_{j=1}^{n_i} L_{ij}$ where the literals of $D_i = P_i \cup N_i$ where P_i is the set of positive literals (atoms) and N_i consists of the atoms appearing as negative literals.

Then D_i can be falsified iff $P_i \cap N_i = \emptyset$.

Arguments in natural language

It turns out that the correctness or otherwise of most arguments depends entirely on the “shapes” of the formulae concerned rather than their intrinsic meaning. Take the previous argument. Suppose we uniformly replace the various atoms by say sentences from nursery rhymes as the following table shows:

Prices rise	Mary has a little lamb
The poor and ...	Little Bo-Peep loses her sheep
Taxes are increased	Jack and Jill go up the hill
The businessmen will be unhappy	Humpty-Dumpty sits on the wall
The Government will get re-elected	Little Miss Muffet sits on a tuffet
Inflation will rise	Little Jack Horner sits in a corner
Government expenditure ... revenue	The boy stands on the burning deck
The Government resorts ...	Wee Willie Winkie runs through the town
The Government takes a loan ... deficit	Eensy Weensy spider climbs up the water spout

Then we get the following ridiculous sounding argument which however is logically valid.

If Mary has a little lamb, then Little Bo-Peep loses her sheep.

If Jack and Jill go up the hill then Humpty-Dumpty sits on the wall.

If Little Bo-Peep loses her sheep or Humpty-Dumpty sits on the wall,

Little Miss Muffet does not sit on a tuffet.

Little Jack Horner sits in a corner if the boy stands on the burning deck.

The boy stands on the burning deck unless Jack and Jill go up the hill

or Wee Willie Winkie runs through the town or

Eensy Weensy spider climbs up the water spout.

If Wee Willie Winkie runs through the town then Little Jack Horner sits in a corner.

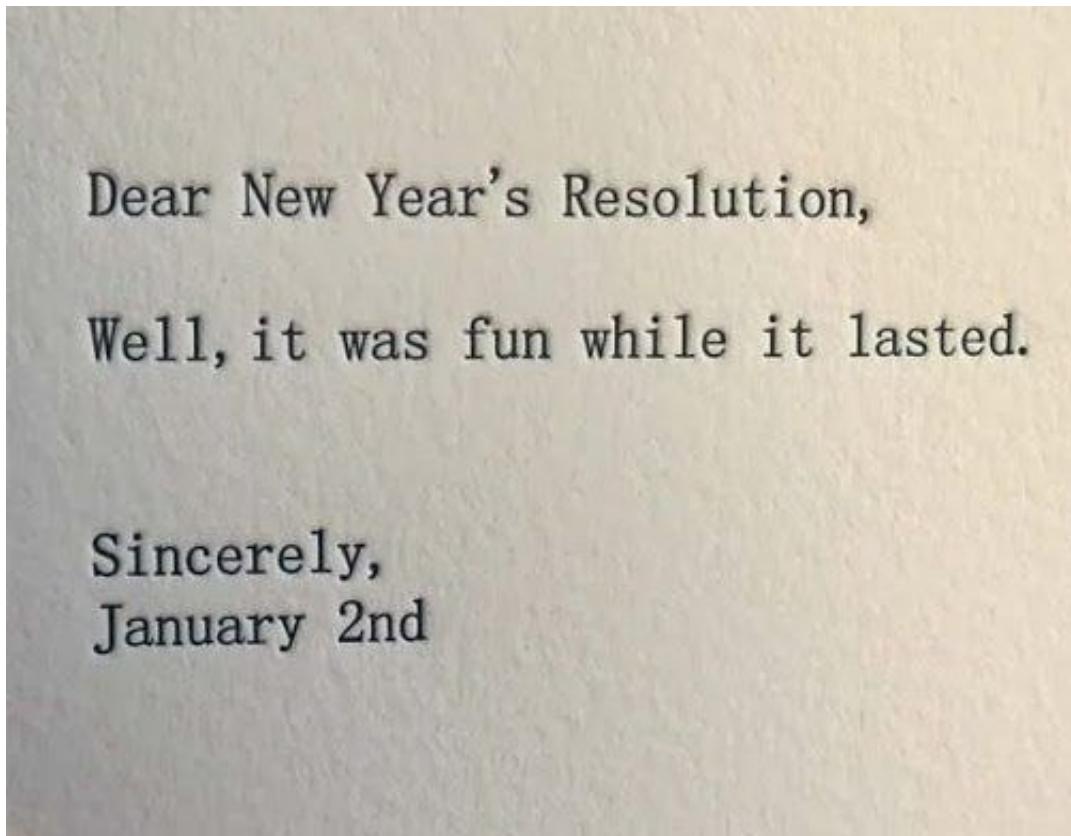
If Little Jack Horner sits in a corner then Mary has a little lamb.

Little Miss Muffet sits on a tuffet.

Therefore Eensy Weensy spider climbs up the water spout.

7. Propositional Unsatisfiability & Resolution

7: Propositional Unsatisfiability & Resolution



1. Tautology Checking
2. CNFs: Set of Sets of Literals
3. Propositional Resolution
4. Clean-up
5. The Resolution Method
6. The Resolution Algorithm
7. Resolution Examples: Biconditional
8. Resolution Examples: Exclusive-Or
9. Resolution Refutation: 1
10. Resolution Refutation: 2
11. Resolvent as Logical Consequence
12. Logical Consequence by Refutation

Tautology Checking

1. Involves conversion of IMP (bigAND H, conc1) into CNF which increases the size of the formula.
2. Involves **checking falsifiability** of the argument.
3. CNF can be obtained more easily for the formula $(\dots ((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \wedge \neg\psi$
 - Convert each individual ϕ_i and $\neg\psi$ to CNF
 - and then append all the lists^a to obtain the required list.
4. More efficient to use theorem 4.2.2 if the technique involves CNF.

^ataking care to remove duplicate copies of clauses and duplicate copies of literals within clauses

CNFs: Set of Sets of Literals

Given a formula ϕ whose CNF is $\gamma \equiv \bigwedge_{1 \leq i \leq m} \delta_i$ where for each i , $1 \leq i \leq m$,

$\delta_i \equiv \bigvee_{1 \leq j \leq n_i} \lambda_{i,j}$ is a disjunction of literals, we will often write γ as a set of sets of literals as follows:

$$\gamma = \{\{\lambda_{1,1}, \dots, \lambda_{1,n_1}\}, \dots, \{\lambda_{m,1}, \dots, \lambda_{m,n_m}\}\}$$

Definition 7.1: Literals and clauses

1. p and $\neg p$ are a **complementary pair** for any atom p .
2. A finite set of literals $\{\lambda_1, \dots, \lambda_k\}$, $k \geq 0$, is a **clause**. A formula ϕ in CNF is (logically equivalent to) a finite set of clauses $\{C_1, \dots, C_m\}$, $m \geq 0$.

Notation 7.1

Note: For the sake of brevity we will abuse terminology by identifying a clause (set of literals) with the formula denoting their disjunction and a set of clauses with the formula denoting their conjunction.

Notes on clauses.

1. A set of clauses represents a conjunction of disjuncts. Hence the **empty set of clauses** represents the **identity element of conjunction** and is always **true**. That is consider a finite set of disjuncts $\{\delta_1, \dots, \delta_m\}$. It is logically equivalent to the set $\{\delta_1, \dots, \delta_m, \top\}$. Hence when $m = 0$ the set of disjuncts is logically equivalent to the \top .
2. A clause containing a **complementary pair** is logically equivalent to \top (since it represents a **disjunction** of the form $p \vee \neg p$).
3. On the other hand a single clause $\{\lambda_1, \dots, \lambda_k\}$ is a set of literals and represents their disjunction and is hence logically equivalent to the set $\{\lambda_1, \dots, \lambda_k, \perp\}$. When $k = 0$ we get the empty clause. By a similar argument as above therefore we have that the **empty clause** $\{\}$ represents the **identity element of disjunction** and is hence always **false**.
4. If $C_1 \subseteq C_2$ for two clauses C_1 and C_2 it is clear that $\{C_1, C_2\}$ is logically equivalent to $\{C_1\}$ by the **absorption** rules.
5. Since $\{\} \subseteq C$ for every clause C it follows that any set $\{C_1, \dots, C_m, \{\}\}$ is logically equivalent to $\{\{\}\}$ which is a **contradiction**.

Propositional Resolution

To show $\Gamma \models \psi$ we show that $\bigwedge \Gamma \wedge \neg\psi$ is a **contradiction** (theorem 4.2.2) by first transforming $\bigwedge \Gamma \wedge \neg\psi$ to a formula in CNF.

This CNF is represented as a *set of sets of literals*^a. Let Δ be the set of sets of literals.

1. Each set $C \in \Delta$ is called a **clause**.
2. Each clause in Δ represents a disjunction of literals.
3. The **empty clause** $\{\}$ represents a **contradiction**.
4. The **unsatisfiability** of the set Δ is shown by deriving the empty clause.

^aSince we are considering a set of sets of literals, rather than a *list of lists of literals* (see the footnote there), there will be duplicate copies of clauses and neither will there be multiple occurrences of literals within a clause

Clean-up

Let Δ be a finite set of clauses representing a formula ϕ . From the identities and the notes above we get

1. If $C \subseteq C'$ in Δ , then C' may be *deleted* from Δ (4),
2. Any clause containing complementary pairs of literals is logically equivalent to \top and so may be *deleted* from Δ ,

while preserving logical equivalence.

Definition 7.2: Clean sets

The resulting clause set Δ' is said to be **clean**.

$$\bigwedge_{C \in \Delta} \bigvee_{L \in C} L \Leftrightarrow \bigwedge_{C' \in \Delta'} \bigvee_{L' \in C'} L'$$

The Resolution Method

For any clean set Δ and an atom p let $\Lambda = \{C \in \Delta \mid p \in C\}$ and $\bar{\Lambda} = \{\bar{C} \in \Delta \mid \neg p \in \bar{C}\}$. Since Δ is a clean set

1. $\Lambda \cap \bar{\Lambda} = \emptyset$.
2. **However C and \bar{C} may not be disjoint.**
3. For each $C \in \Lambda$ and $\bar{C} \in \bar{\Lambda}$, $p \notin \bar{C}$ and $\neg p \notin C$.

The new set of clauses obtained after resolution is

Definition 7.3: The resolvent

$$\begin{aligned} \text{resolve}(\Delta, p) &\stackrel{df}{=} (\Delta - (\Lambda \cup \bar{\Lambda})) \cup \\ &\{D \mid D = (C - \{p\}) \cup (\bar{C} - \{\neg p\}), C \in \Lambda, \bar{C} \in \bar{\Lambda}\} \end{aligned}$$

is called the resolvent.

Lemma 7.1: Satisfiability preservation

If $C_1 = \{\lambda_{1,i} \mid 1 \leq i \leq m\}$ and $C_2 = \{\lambda_{2,j} \mid 1 \leq j \leq n\}$ are clauses such that $p \in C_1 - C_2$ and $\neg p \in C_2 - C_1$ and $D = (C_1 - \{p\}) \cup (C_2 - \{\neg p\})$. Then

1. $C_1 \wedge C_2 \Rightarrow D$ and
2. *resolve* preserves satisfiability, i.e. every truth assignment that satisfies both C_1 and C_2 also satisfies D .

Proof: From the semantics of propositional logic it follows that for any truth assignment τ , $\tau \Vdash C_1 \wedge C_2$ if and only if $\tau \Vdash C_1$ and $\tau \Vdash C_2$. Hence we may prove both parts of the lemma by considering an arbitrary truth assignment τ such that $\tau \Vdash C_1 \wedge C_2$. It suffices to show (for both parts) that $\tau \Vdash D$. $\tau \Vdash C_1$ implies that for some i_0 , $1 \leq i_0 \leq m$, $\mathcal{T}[\lambda_{1,i_0}]_\tau = 1$ and for some j_0 , $1 \leq j_0 \leq n$, $\mathcal{T}[\lambda_{2,j_0}]_\tau = 1$. Further since p and $\neg p$ are complementary, it is impossible that both $p \equiv \lambda_{1,i_0}$ and $\neg p \equiv \lambda_{2,j_0}$ hold simultaneously. Hence at least one of the two literals $\lambda_{1,i_0}, \lambda_{2,j_0}$ is in D . It follows therefore that $\mathcal{T}[D]_\tau = (\sum_{1 \leq i \leq m} \mathcal{T}[\lambda_{1,i}]_\tau) + (\sum_{1 \leq j \leq n, j \neq j_0} \mathcal{T}[\lambda_{2,j}]_\tau) = 1$ if $p \not\equiv \lambda_{1,i_0}$ and

$$\mathcal{T}[D]_\tau = (\sum_{1 \leq i \leq m, i \neq i_0} \mathcal{T}[\lambda_{1,i}]_\tau) + (\sum_{1 \leq j \leq n} \mathcal{T}[\lambda_{2,j}]_\tau) = 1 \text{ if } \neg p \not\equiv \lambda_{2,j_0} \text{ and hence } \tau \Vdash D.$$

QED

Algorithm

Algorithm 7.1

$\text{RESOLUTION } (\Delta) \stackrel{df}{=}$

$$\left\{ \begin{array}{l} \text{requires: } \Delta \text{ a clean set of clauses} \\ \text{while } (\{\}) \notin \Delta \wedge \exists \text{complementary pair } (p, \neg p) \in \Delta \\ \quad \text{do } \left\{ \begin{array}{l} \Delta' := \text{resolve}(\Delta, p) \\ \Delta := \text{Clean-up}(\Delta') \end{array} \right. \end{array} \right.$$

Note:

1. $\{\}$ is the empty clause which represents the proposition \perp (it represents the disjunction of an empty set of literals).
2. The presence of the empty clause in a set of clauses also indicates the unsatisfiability of the set of clauses.

Lemma 7.2: Resolution implication

Let $\Delta_1 = \text{Clean-up}(\text{resolve}(\Delta_0))$. Then

1. $\Delta_0 \Rightarrow \Delta_1$ and
2. if Δ_0 is satisfiable then Δ_1 is satisfiable.

Proof: The proof follows directly from the proof of lemma 7.1 and from problem 4 of exercise 4.1, where it should have been shown that both \wedge and \vee preserve logical implication. QED

Corollary 7.1: Propositional unsatisfiability

If Δ_1 and Δ_0 are as in lemma 7.2 then if Δ_1 is unsatisfiable then so is Δ_0 .

Theorem 7.1: Total correctness of algorithm 7.1

Given a finite set of clean non-empty set Δ_0 of clauses, the propositional resolution algorithm terminates in at most $|\text{atoms}(\Delta_0)|$ iterations, deriving either an empty clause or a set of non-empty clauses which are satisfied by every model of the original set Δ_0 .

Proof: Since in each iteration one atom and its negation are completely eliminated, $|\text{atoms}(\Delta_0)|$ is the number of iterations possible. Further by applying lemma 7.2 to the result of each iteration we get that satisfiability and logical implication are preserved. QED

Resolution Examples: Biconditional

Example 7.1

Sometimes there may be more than one complementary pair of literals in the same pair of clauses. Consider the biconditional operator (\leftrightarrow) on atomic propositions p and q . We have

$$p \leftrightarrow q \Leftrightarrow (p \vee \neg q) \wedge (\neg p \vee q) \equiv \{\{p, \neg q\}, \{\neg p, q\}\}$$

Applying resolution on the pair of literals p and $\neg p$ we obtain the clause set which after clean-up yields the empty set of clauses.

$$\{\{q, \neg q\}\} \equiv \{\} \equiv \top$$

There is really nothing more to this resolvent than that \top is a logical consequence of any proposition (including \perp).

Resolution Examples: Exclusive-Or

Example 7.2

The exclusive-or operation \oplus is simply the negation of the biconditional operator \leftrightarrow . Hence we have

$$p \oplus q \Leftrightarrow (p \vee q) \wedge (\neg p \vee \neg q) \equiv \{\{p, q\}, \{\neg p, \neg q\}\}$$

which on resolution on the pair $(p, \neg p)$ and subsequent clean-up again yields the empty set of clauses.

$$\{\{q, \neg q\}\} \equiv \{\} \equiv \top$$

Resolution Refutation: 1

Example 7.3

Consider the simple logical consequence

$$p \wedge q \models p$$

which we prove by *resolution refutation*. The set of clauses representing the hypothesis and the negation of the conclusion is $\{\{p\}, \{q\}, \{\neg p\}\}$. Resolving on the pair $(p, \neg p)$ yields

$$\{\{\}, \{q\}\}$$

Notice that $\{\} \equiv \perp \equiv \{\{\}, \{q\}\}$.

Since for any clause $C \neq \emptyset$, $C \supseteq \{\}$, the clean-up always reduces every set of clauses Δ to $\{\{\}\}$ whenever $\{\} \in \Delta$.

Resolution Refutation: 2

Example 7.4

Consider the simple logical consequence

$$p \wedge q \models p \leftrightarrow q$$

which we prove by *resolution refutation*. Negating the conclusion yields $p \oplus q \equiv \{\{p, q\}, \{\neg p, \neg q\}\}$. The set of clauses representing the hypothesis and the negation of the conclusion is $\{\{p\}, \{q\}, \{p, q\}, \{\neg p, \neg q\}\}$ which after clean-up yields

$$\{\{p\}, \{q\}, \{\neg p, \neg q\}\}$$

Resolving on the pair $(p, \neg p)$ produces

$$\{\{q\}, \{\neg q\}\}$$

and then on the pair $(q, \neg q)$ produces the empty clause $\{\{\}\}$.

Resolvent as Logical Consequence

The set of clauses resulting from any application of **resolution** is a set of clauses that represents a logical consequence of the original set of clauses

Example 7.5

We use resolution to prove

$$(p \vee q) \wedge (p \vee r) \wedge \neg p \models q \wedge r$$

The set of clauses $\{\{p, q\}, \{p, r\}, \{\neg p\}\}$ may be resolved on the pair $(p, \neg p)$ to yield the set

$$\{\{q\}, \{r\}\} \equiv q \wedge r$$

Note that we have resolved all occurrences of the complementary pair $(p, \neg p)$.

Logical Consequence by Refutation

Example 7.6

Consider the following variation of example 7.5.

$$(p \vee q) \wedge (p \vee r) \wedge \neg p \models \neg p \wedge q \wedge r$$

which also holds. But the set of clauses still remains $\{\{p, q\}, \{p, r\}, \{\neg p\}\}$. Since we resolve *all* occurrences of the complementary pair $(p, \neg p)$, we could not have proved that $\neg p \wedge q \wedge r$ is also a logical consequence of $(p \vee q) \wedge (p \vee r) \wedge \neg p$ which it indeed is.

We may instead use refutation for this purpose by noting that $\neg(\neg p \wedge q \wedge r) \equiv \{p, \neg q, \neg r\}$. We then resolve the set $\{\{p, q\}, \{p, r\}, \{\neg p\}, \{p, \neg q, \neg r\}\}$ on the pair $(p, \neg p)$ to obtain the set $\{\{q\}, \{r\}, \{\neg q, \neg r\}\}$ which may be resolved on the pairs $(q, \neg q)$ and $(r, \neg r)$ subsequently to yield the empty clause.

7.1. Space Complexity of Propositional Resolution.

Assume n is the number of atoms of which Δ is made up. After some iterations of **resolution** and **cleanup**, assume there are k distinct atoms in the set of clauses on which **resolution** is applied. After performing the **cleanup** procedure there could be *at most* 2^k clauses with each clause containing at most k literals. Assuming each literal occupies a unit of memory, the space requirement is given by $\text{size}(\Delta) = 2^k k$ literals.

For any complementary pair $(p, \neg p)$, it is possible that at most half of the 2^k (i.e. 2^{k-1}) clauses contain p and the other half contain $\neg p$. This would be the worst-case scenario, as it yields the maximum number of new clauses. Therefore in performing a single step of resolution over all possible pairs of clauses to yield a new set Δ' of clauses, a maximum of $2^{k-1} \times 2^{k-1}$ unions of distinct pairs of clauses needs to be performed. Before applying the **clean-up** procedure the space required could be as high as $2^{k-1} \times 2^{k-1} = 2^{2(k-1)} > 2^k k$ for $k > 4$. But since Δ' is made up of at most $(k-1)$ atoms, $\text{size}(\Delta') \leq 2^{k-1}(k-1)$, after **clean-up** the space requirement reduces to $2^{k-1}(k-1)$. Since $k \leq n$ the maximum space required after the first application of resolution and before cleaning up exceeds the space required for all other iterations and is bounded by $2^{2(n-1)} = O(2^{2n})$.

7.2. Time Complexity of Propositional Resolution

Given a space of $2^k k$ to represent the clauses containing at most k atoms, we require a time proportional to this amount of space in order to identify which clauses have to be resolved against a particular complementary pair. After **resolution** we

create a space of $2^{2(k-1)}$ which has to be scanned for the **cleanup** operations. Hence the amount of time required to perform a step of resolution and the amount of time required to perform the cleanup are both proportional to $2^{2(k-1)}$. Hence the total time required for performing **resolution** followed by **cleanup** in n iterations (which is the maximum possible) is given by

$$T(n) \geq \sum_{k=1}^n 2^{2k-2}$$

which is clearly exponential.

Hence both the *worst case* time and space complexities are exponential in the number of atoms.

7.3. Unit Resolution

A clause is called a **unit clause** if it contains a single literal. In the **resolution method** it is worthwhile mentioning that given an unit clause $C = \{\lambda\}$ and another clause \bar{C} containing the literal $\bar{\lambda}$, the resolvent produced is $D = \bar{C} - \{\bar{\lambda}\}$. In this case $|D| = |\bar{C}| - 1$. Given m ($m > 0$) clauses $\bar{C}_1, \dots, \bar{C}_m$ in Δ all containing the literal $\bar{\lambda}$, the result is to produce a resolvent Δ' with a reduced number of clauses (since the clause C “disappears” and each clause \bar{C}_i is replaced by \bar{D}_i) and whose size (in terms of the number of literal occurrences) actually reduces by $m + 1$ literals. This process is called **unit resolution** and it has a number of consequences.

1. The reduction in size implies that it is a good heuristic to reduce the space (and consequently the time) complexity by first employing unit resolution whenever possible.

2. Unit resolution may also be used in conjunction with other methods to reduce the complexity of various theorem-proving procedures.
3. If the refutation of a clause set Δ can be performed entirely using unit resolution then the resolution procedure would take time that is *linear* in the number of occurrences of literals and can therefore be extremely fast.

In the sequel we have listed several examples which employ unit resolution (see examples 7.1, 7.2, 7.3, 7.4, 7.5 7.6).

8. Binary Decision Diagrams

8: Binary Decision Diagrams

There are
10 types
of people
in the world:

Those who
understand binary,
and those
who don't. |

1. Binary Decision Diagrams
2. The if-then-else Operator 1
3. The if-then-else truth table
4. The if-then-else Operator 2
5. Boolean Expressions: Variables
6. Boolean Expressions: terms
7. if-then-else: test, high and low
8. Examples: Boolean expressions
9. More on if-then-else
10. Functional Completeness with the new operator
11. Another Normal Form: INF
12. The Shannon Expansion
13. Example: Truth table to Decision tree
14. Example:1
15. Example:2
16. Example:3
17. Example:4,5
18. BDDs

- 19. OBDDs
- 20. ROBDDs
- 21. Order of Variables
- 22. ROBDDs: Representing Boolean functions
- 23. The Canonicity Lemma
- 24. Consequences of Canonicity

8.1. Remarks on Tautology checking and Resolution

In **tautology checking** we had to convert a **humongous** formula representing the **whole argument** into a formula in **CNF**. In the case of **resolution** the problem was **ameliorated** by the fact that rather than construct one huge formula we could convert the individual propositions separately and convert each into CNF (see item 3) and then take the unions of the CNFs of the individual formulae. Further for resolution we also had to compute the negation of the conclusion and then take the union of all the CNFs.

But the conversion of a formula into CNF in general, increases the size of the resultant formula primarily because of use of the identities **Biconditional and Distributivity**. If we could get a more efficient representation which can reduce the size of formulae using identities other than **Biconditional and Distributivity** it might help. One way to do this is to use the formula directly and get a representation using the **truth table**. Of course, the truth table by itself is already **exponential in the number of atoms** and hence very inefficient in its raw form.

In the following slides we illustrate another method based on **truth tables** which does not have the overheads of using identities which may blow-up the size of formulae and yields more **compact representations** which will allow for faster processing. In addition to a compact representation its property of **canonicity** allows for unique representations upto logical equivalence. This is particularly useful for checking **tautologies, contradictions, logical implications and logical equivalences**.

Binary Decision Diagrams

- The **1-1 correspondence** between the operators of the model of truth viz. $2'$ and that of Ω_0 implies that any expression in $2'$ may be directly translated into a propositional formula in which the variables of the expression attain the status of propositional atoms.
- It also means that propositional reasoning could be effectively handled in the model itself provided there are efficient ways of representing the models of propositions.
- We already know that any **adequate** set of operators for Ω_0 is also **functionally complete** for \mathcal{P}_0 (corollary 5.1).
- We define a new ternary operator inspired by the if-then-else (that most programming languages possess) to get a more efficient data structure for propositional verification.

The if-then-else Operator 1

We begin by extending the algebra $\mathbf{2}'$ to $\mathbf{2}'' = \langle \mathbf{2}, \{\bar{\cdot}, ., +, \leq\cdot, \dot{\div}, ? : \}, \{\leq, =\} \rangle$ by adding a new ternary (3-ary) operator called the *if-then-else* denoted $a?b : c$ and defined by the following identity

$$a?b : c = a.b + \bar{a}.c \quad (24)$$

from which the following identities follow easily

$$1?b : c = b$$

$$0?b : c = c$$

The if-then-else truth table

a	b	c	$a?b : c$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The if-then-else Operator 2

Correspondingly we may define in propositional logic a new ternary operator $\text{?} :$ whose meaning is defined for each truth assignment τ as follows:

$$\mathcal{T}[\![\phi?\psi : \chi]\!]_{\tau} \stackrel{df}{=} \mathcal{T}[\![\phi]\!]_{\tau} ? \mathcal{T}[\![\psi]\!]_{\tau} : \mathcal{T}[\![\chi]\!]_{\tau}$$

Boolean Expressions: Variables

Definition 8.1: Boolean variables

Let Var be an infinite collection of variable symbols called **boolean variables**.

We may build up boolean expressions using the variables of Var (typically lower case symbols such as $a, b, c \dots$ suitably decorated by sub-scripts or super-scripts).

Boolean Expressions: terms

The set of boolean expressions may be generated from the following grammar:

Definition 8.2: Boolean expressions

$$s, t, u ::= 0 \mid 1 \mid a \in Var \mid (\bar{t}) \mid (s.t) \mid (s + t) \mid (s \leq t) \mid (s \doteq t) \mid s?t : u \quad (25)$$

Any term generated by the above grammar is called a **boolean expression**.

if-then-else: test, high and low

Definition 8.3: Test, high and low

In any boolean (sub-)expression of the form $(s?t : u)$,

- s is called the **test**,
- t is the **high arm** and
- u is the **low arm**

Examples: Boolean expressions

Example 8.1

We could have various examples of boolean expressions (involving the if-then-else operator)

- $(a \neq b)?(c?(a + d) : (b.c)) : a.d$
- $(a.b)?(c + d) : ((c.b) \leq d)$
- $(a?b : c)?(a.c) : (b?c : (c + a))$

More on if-then-else

Lemma 8.1

The set of operators $\{0, 1, ?: :\}$ is sufficient to express (upto equality) all the expressions of $2'$.

Proof: It is easy to prove the following identities from which the claim of the lemma follows:

$$\begin{aligned}\bar{a} &= a?0 : 1 \\ a.b &= a?b : 0 = b?a : 0 \\ a + b &= a?1 : b = b?1 : a \\ a \leq b &= a?b : 1 \\ a \doteq b &= a?b : \bar{b} = a?b : (b?0 : 1)\end{aligned}$$

QED

Distributive properties of if-then-else

It is interesting that the if-then-else operator satisfies various distributive laws which may be used in optimizing compilers to reduce the number of evaluations of conditionals in programs.

Lemma 8.2: Distributive laws

Let $s = a?s^1 : s^0$ and $t = a?t^1 : t^0$ where s^0 , s^1 , t^0 and t^1 are boolean expressions not involving the boolean variable a . Then

$$\bar{s} = a?\overline{s^1} : \overline{s^0} \quad (26)$$

and for any binary operator $\odot \in \{\cdot, +, \leq\cdot, \doteq\}$,

$$s \odot t = a?(s^1 \odot t^1) : (s^0 \odot t^0) \quad (27)$$



Proof of Lemma 8.2

Proof: We use the definition (24) and the boolean identities to obtain the required results. The reader is encouraged to provide the justification for each step and fill in the gaps where multiple steps have been skipped. In particular, we have proved these equations in such an order that some of the steps from the later proofs use the previously proved identitites.

Equation (26). We have

$$\begin{aligned}
 &= \frac{\bar{s}}{a.s^1 + \bar{a}.s^0} \\
 &= \frac{(a.s^1).(\bar{a}.s^0)}{(a.s^1).(\bar{a}.s^0)} \\
 &= (\bar{a} + \bar{s}^1).(\bar{a} + \bar{s}^0) \\
 &= \bar{a}.\bar{s}^0 + a.\bar{s}^1 + \bar{s}^1.s^0 \\
 &= \bar{a}.\bar{s}^0 + \bar{a}.s^0.s^1 + a.s^1 + a.s^1.s^0 \\
 &= \bar{a}.\bar{s}^0.(1 + \bar{s}^1) + a.s^1.(1 + \bar{s}^0) \\
 &= \bar{a}.\bar{s}^0 + a.s^1 \\
 &= a?\bar{s}^1 : \bar{s}^0
 \end{aligned}$$

Equation (27-+).

$$\begin{aligned}
 &= \frac{s+t}{(a.s^1 + \bar{a}.s^0) + (a.t^1 + \bar{a}.t^0)} \\
 &= a.(s^1 + t^1) + \bar{a}.(s^0 + t^0) \\
 &= a?(s^1 + t^1) : (s^0 + t^0)
 \end{aligned}$$

Equation (27-·).

$$\begin{aligned}
 &= \frac{s.t}{(a.s^1 + \bar{a}.s^0).(a.t^1 + \bar{a}.t^0)} \\
 &= a.s^1.t^1 + \bar{a}.s^0.t^0 \\
 &= a?(s^1.t^1) : (s^0.t^0)
 \end{aligned}$$

Equation (27- \leq^*).

$$\begin{aligned}
 & s \leq^* t \\
 &= \bar{s} + t \\
 &= (a? \bar{s}^1 : \bar{s}^0) + (a? t^1 : t^0) \\
 &= a?(s^1 \leq^* t^1) : (s^0 \leq^* t^0)
 \end{aligned}$$

Equation (27- \doteq).

$$\begin{aligned}
 & s \doteq t \\
 &= (s \leq^* t).(t \leq^* s) \\
 &= (a?(s^1 \leq^* t^1) : (s^0 \leq^* t^0)).(a?(t^1 \leq^* s^1) : (t^0 \leq^* s^0)) \\
 &= a?((s^1 \leq^* t^1).(t^1 \leq^* s^1)) : ((s^0 \leq^* t^0).(t^0 \leq^* s^0)) \\
 &= a?(s^1 \doteq t^1) : (s^0 \doteq t^0)
 \end{aligned}$$

QED

[skip to Functional Completeness with if-then-else](#)

Exercise 8.1: Factoring and distribution

Lemma 8.2 may be generalised in the following ways. Prove each of them.

1. For boolean expressions r, s, t, u, v and any (infix) binary operator \odot ,

$$(r?s : t) \odot (r?u : v) = r?(s \odot u) : (t \odot v) \quad (28)$$

2. Let $o_n : \mathcal{D}^n \rightarrow \mathcal{D}$ be any n -ary boolean operator for any $n > 0$. If $t_i = a?t_i^1 : t_i^0$ for all i , $1 \leq i \leq n$ and boolean variable a , then

$$o_n(t_1, \dots, t_n) = a?o_n(t_1^1, \dots, t_n^1) : o_n(t_1^0, \dots, t_n^0) \quad (29)$$

3. Let $o_n : \mathcal{D}^n \rightarrow \mathcal{D}$ be any n -ary boolean operator for any $n > 0$. If $t_i = s?t_i^1 : t_i^0$ for all i , $1 \leq i \leq n$ and any boolean expression s , then

$$o_n(t_1, \dots, t_n) = s?o_n(t_1^1, \dots, t_n^1) : o_n(t_1^0, \dots, t_n^0) \quad (30)$$

4. **Composition.** Let f and g be boolean expressions and let a be any boolean variable (which may or may not occur in f or g). Prove that

$$\{g/a\}f = g?\{1/a\}f : \{0/a\}f \quad (31)$$

where $\{g/a\}f$ denotes the syntactic substitution of g for all occurrences of a in the expression f .

5. **Generalised Composition.** Let $f(a_{m-1}, \dots, a_0)$ be a boolean function of m variables and let g_{m-1}, \dots, g_0 be boolean expressions. Generalise equation (31) to complete the following equation.

$$\{g_{m-1}/a_{m-1}, \dots, g_0/a_0\}f = \dots \dots \dots \quad (32)$$

Functional Completeness with the new operator

Theorem 8.1: Functional completeness

The set $\{\perp, \top, ?: \}$ is adequate for Ω_0 . Hence it is also functionally complete for \mathcal{P}_0 .

Proof: Follows from lemma 8.1 and the 1-1 correspondence between the operators of \mathcal{D}' and Ω_0 , which may be further extended with the new if-then-else operator. QED

Another Normal Form: INF

Definition 8.4: The if-then-else normal form

An expression in 2^{∞} is said to be in **if-then-else normal form (INF)** if it is built up from variables using only the operators in $\{0, 1, ?, :\}$ such that every test is performed only on a boolean variable.

Notice that none of the boolean expressions of example 8.1 is in INF.

The Shannon Expansion

For convenience we use the notation $s(a_{n-1}, \dots, a_0)$ to denote that s is a boolean expression built up from n variables a_{n-1}, \dots, a_0 .

Theorem 8.2: The Shannon Expansion

Every boolean expression $s(a_{n-1}, \dots, a_0)$ may be expressed in INF.



Proof of Theorem 8.2

Proof: By induction on n the number of boolean variables in the expression.

Basis. For $n = 1$, $s \equiv s(a_0)$. Regardless of how the expression has been formed s has only the following four possibilities — $a_0?0 : 0$, $a_0?1 : 0$, $a_0?0 : 1$ and $a_0?1 : 1$ — all of which are in INF.

Induction Hypothesis (IH).

For each k , $1 \leq k < n$, $n > 1$, every boolean expression $s(a_{k-1}, \dots, a_0)$ may be expressed in INF $t(a_{k-1}, \dots, a_0)$ such that $s = t$.

Induction Step. Let $s \equiv s(a_{n-1}, \dots, a_0)$ be a boolean expression. Now consider the expressions $s_i \equiv \{i/a_{n-1}\}s$, $i \in \{0, 1\}$ obtained by replacing all occurrences of a_{n-1} in s by i . Clearly both $s_0 \equiv s_0(a_{n-2}, \dots, a_0)$ and $s_1 \equiv s_1(a_{n-2}, \dots, a_0)$ are boolean expressions in $n - 1$ variables. By the induction hypothesis $s_0 = t_0$ and $s_1 = t_1$ where both t_0 and t_1 are in INF. It then follows that $s = a_{n-1}?t_1 : t_0$, which is also in INF.

QED

Example: Truth table to Decision tree

Consider the following boolean expression

$$s \equiv (a \dot{=} b)?(c?(a + d) : (b.c)) : a.d$$

whose truth table may be constructed as shown below

a	b	c	d	s
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

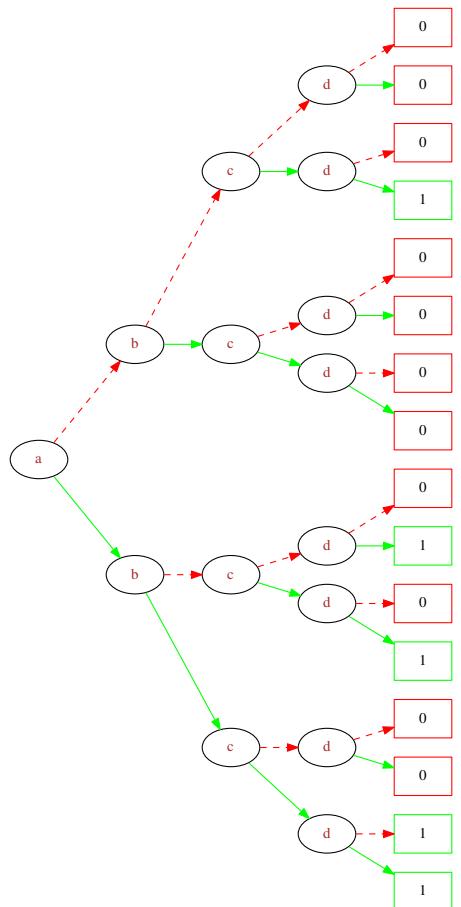
a	b	c	d	s
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

a	b	c	d	s
0	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	1
	1	0	0	0
	1	0	1	0
	1	1	0	0
	1	1	1	0
1	0	0	0	0
	0	0	1	1
	0	1	0	0
	0	1	1	1
	1	1	0	0
	1	0	1	0
	1	1	0	1
	1	1	1	1

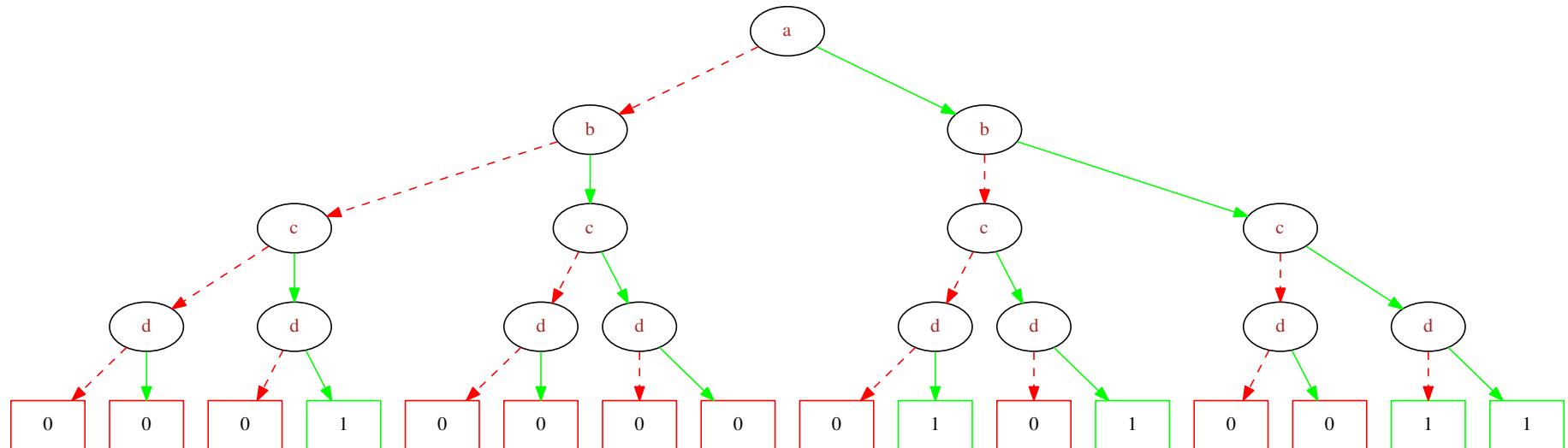
a	b	c	d	s
0		0	0	0
		0	1	0
	0	1	0	0
		1	1	1
		0	0	0
		0	1	0
	1	1	0	0
		1	1	0
1		0	0	0
		0	1	1
	0	0	1	1
		1	0	0
		1	1	1
		0	0	0
		0	1	0
	1	1	0	1
		1	1	1

a	b	c	d	s
0			0	0
		0	1	0
	0		0	0
			1	1
		1	1	1
			0	0
1			0	0
		0	1	1
			0	0
	0		1	1
			1	1
		1	1	0
1			0	0
		0	1	1
			0	0
	0		1	1
			1	1
	1		0	0
1			0	1
		1	1	1
			0	0
	1		0	0
			1	1
	1		1	1

a	b	c	d	s
0	0	0	0	0
		1	0	0
		1	1	1
	1	0	0	0
		0	1	0
		1	0	0
1	0	0	0	0
		1	1	1
		0	0	0
	1	1	1	1
		0	1	0
		1	0	1
		1	1	1



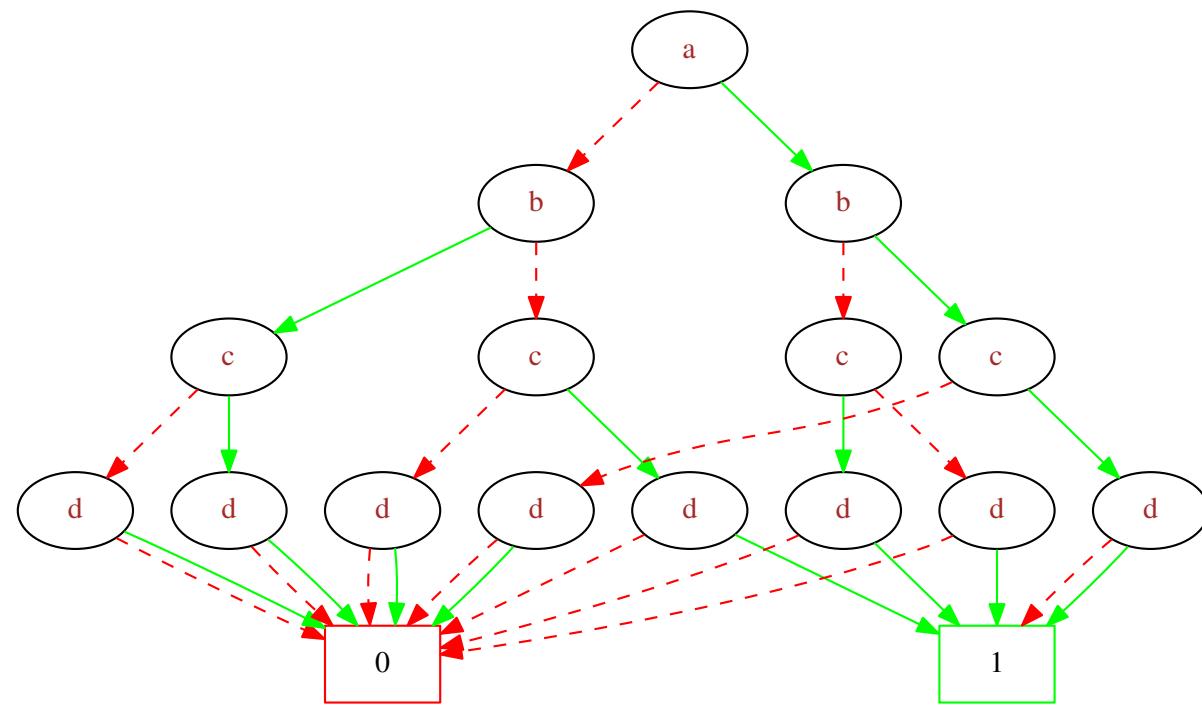
Example:1

$$(a \doteq b)?(c?(a + d) : (b.c)) : a.d$$


Example:2

$$(a \doteq b)?(c?(a + d) : (b.c)) : a.d$$

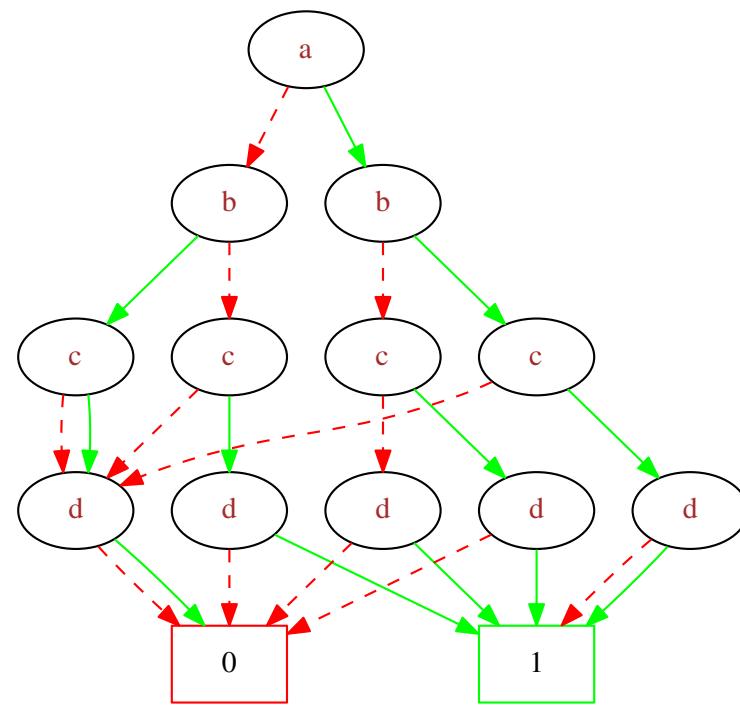
Coalesce all copies of 0s and all copies of 1s since they are isomorphic.



Example:3

$$(a \doteq b)?(c?(a + d) : (b.c)) : a.d$$

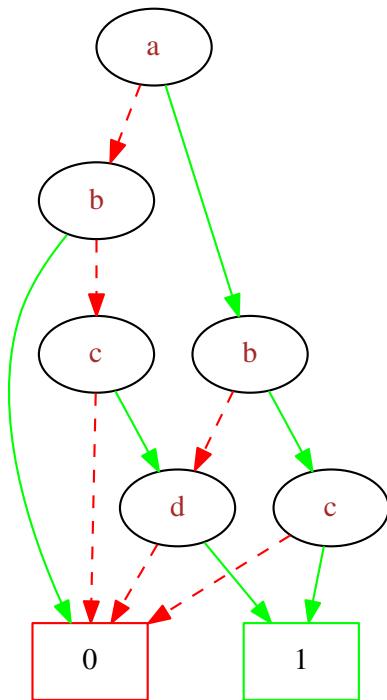
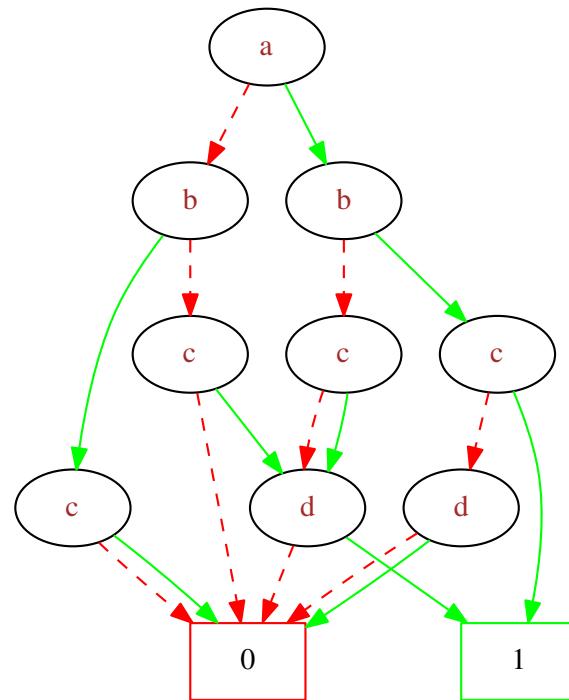
Coalesce isomorphic test nodes.



Example:4,5

$$(a \doteq b)?(c?(a + d) : (b.c)) : a.d$$

Delete nodes with redundant tests by “short-circuiting”.



Definition 8.5: BDD

A **binary decision diagram** (BDD) is a rooted directed acyclic graph (rooted DAG (see definition -2.1) with

terminal nodes: there is at most one terminal (out-degree zero) node labelled **0** and at most one terminal node labelled **1**,

non-terminal nodes: a set of variable nodes labelled with a boolean variable name; each variable node has out-degree two called the **low**-edge and the **high**-edge respectively.

With each non-terminal node n labelled by a variable a with low-edge going to node n_0 and high-edge to node n_1 respectively we associate the following functions:

$$\text{var}(n) = a, \text{lo}(n) = n_0, \text{hi}(n) = n_1$$

OBDDs

Definition 8.6: OBDD

A BDD is an ordered BDD (OBDD) if on all paths, the variables respect a given linear order.

For example we will write our boolean expressions s as $s(a, b, c, d)$ to denote that s is built up (at most) from the variables a , b , c and d and the variables follow the linear order $a < b < c < d$.

ROBDDs

Definition 8.7: ROBDD

A BDD is called a **reduced OBDD (ROBDD)** if it satisfies the following conditions for all nonterminal nodes m, n ,

Linear ordering of variables : There is a linear irreflexive ordering $<$ on the variables such that if a is tested before b then $a < b$.

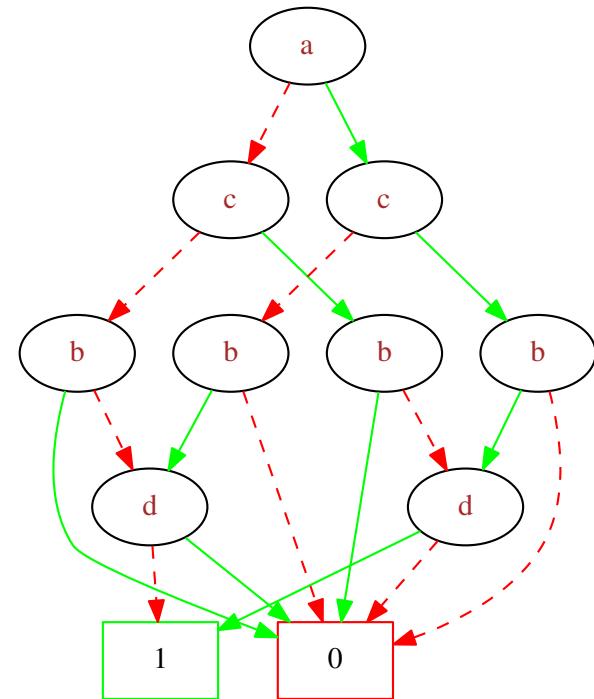
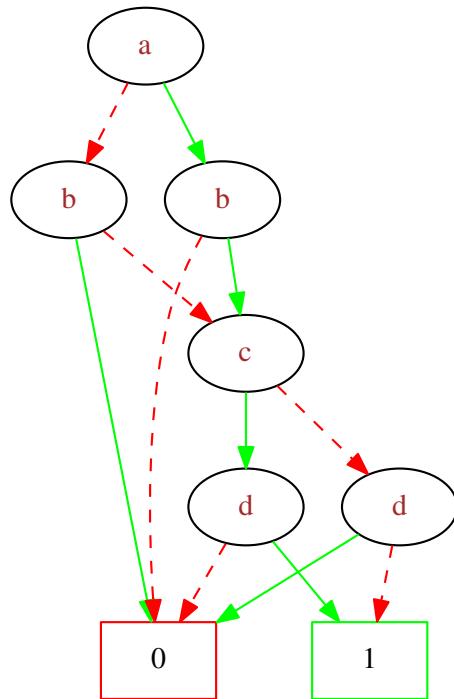
Test Irredundancy : $lo(n) \neq hi(n)$, i.e. both out-edges from a variable node cannot have the same target node.

Node non-isomorphism : $var(m) = var(n)$, $lo(m) = lo(n)$ and $hi(m) = hi(n)$ implies $m = n$, i.e. no two distinct nodes can have isomorphic sub-DAGs rooted at them.

Order of Variables

Consider the same expression with different variable orderings i.e.

$$s(a, b, c, d) \equiv (a \doteq b).(c \doteq d) \equiv t(a, c, b, d)$$



ROBDDs: Representing Boolean functions

Lemma 8.3: Boolean function representation

For a given ordering of variables, every ROBDD represents a unique boolean function.

Proof: We prove it by induction on the structure of nodes in a ROBDD. We may think of each node n as representing a boolean function in INF as follows expressed as a term t^n .

- $t^0 = 0$
- $t^1 = 1$
- $t^n = \text{var}(n) ? t^{hi(n)} : t^{lo(n)}$

QED

The Canonicity Lemma

We have seen right from the [The Shannon Expansion theorem](#) and its [proof](#) that the structure of the BDD does not depend upon the expression, but only on the [boolean function](#) that the expression denotes. Given a boolean function $f : \mathbb{2}^n \rightarrow \mathbb{2}$ of n variables we show that for a given total ordering of the variables, there is a unique ROBDD which represents that function.

Lemma 8.4: ROBDD canonicity

Given any m -ary ($m \geq 0$) boolean function $f : \mathbb{2}^m \rightarrow \mathbb{2}$ and a linear ordering $a_{m-1} < \dots < a_0$ of its variables, there is a unique ROBDD representing f .



We have already seen that for [different orderings of variables](#) the resulting ROBDDs could be different but for each ordering it would be unique.

Proof of Canonicity lemma 8.4.

Proof: By induction on the number of variables.

Basis. For $m = 0$, there exist exactly the two constants 0 with ROBDD 0 and 1 with ROBDD 1 .

Induction Hypothesis (IH).

For each k , $0 \leq k < m$, every k -ary boolean function with variable ordering $a_{k-1} < \dots < a_0$ has unique ROBDD.

Induction Step. Let $f : \mathbb{2}^m \rightarrow \mathbb{2}$ with $m > 0$ and ordering $a_{m-1} < \dots < a_0$. The two functions $f_0(a_{m-2}, \dots, a_0) \stackrel{\text{df}}{=} f(0, a_{m-2}, \dots, a_0)$ and $f_1(a_{m-2}, \dots, a_0) \stackrel{\text{df}}{=} f(1, a_{m-2}, \dots, a_0)$ satisfy the equation

$$f(a_{m-1}, \dots, a_0) = a_{m-1}?f_1(a_{m-2}, \dots, a_0):f_0(a_{m-2}, \dots, a_0) \quad (33)$$

By the induction hypothesis there exist *unique* ROBDDs representing f_1 and f_0 each rooted at nodes n_1 and n_0 respectively such that $t^{n_0} = f_0$ and $t^{n_1} = f_1$.

Case $n_0 = n_1$. Then $f_0 = t^{n_0} = t^{n_1} = f_1$ and hence $t = t^{n_0} = t^{n_1}$. Notice that in this case there is no occurrence of any node n such that $\text{var}(n) = a_{m-1}$. If such a node were introduced, the resulting BDD would not be reduced.

Case $n_0 \neq n_1$. Again by the induction hypothesis, $t^{n_0} = f_0 \neq f_1 = t^{n_1}$. We define a new node n with $\text{var}(n) = a_{m-1}$, $\text{lo}(n) = n_0$ and $\text{hi}(n) = n_1$ which is a reduced OBDD representing equation (33). The uniqueness of the resulting ROBDD again follows from the construction and the induction hypothesis.

QED

Consequences of Canonicity

- Checking equality of two boolean expressions with a given variable ordering is a constant time operation: because if ROBDDs are represented using a global hash table, it is just a matter of checking whether the root nodes of the two ROBDDs are the same node.
- For any propositional formula, checking whether it is a **tautology** amounts to checking whether it is identical to **1**, **contradiction** amounts to checking whether it is identical to **0** and **contingent** amounts to checking whether it has both nodes **1** and **0**.

9. Analytic Tableaux

9: Analytic Tableaux

Indeed, history is nothing more than a tableau of crimes and misfortunes.

Voltaire

1. Against Resolution
2. Against BDDs
3. The Analytic Tableau Method
4. Basic Tableaux Facts
5. Example: AST
6. Example: AST with Complements
7. Tableaux Rules
8. Structure of the Rules
9. Tableaux
10. An Example Tableau Proof
11. Tableaux Heuristics
12. Slim & Short Tableaux

Against Resolution

1. For any argument, it was still necessary to transform the argument into a mammoth formula in CNF in order to be able to perform *resolution*.
2. The numbers of clauses and sizes of clauses could temporarily increase as a result of resolution.
3. Termination relied on the reduction of the number of atoms at each step of resolution.
4. Resolution also requires a clean-up of the initial set of clauses to work correctly.

Against BDDs

1. BDDs completely lose the structure of the original argument because of the various reductions and optimizations that are performed.
2. BDDs cannot be dealt with incrementally to observe the changes in the argument induced by addition/deletion of fresh conditions. Small additions/deletions can completely change the structure of the BDD.
3. BDDs are good only for propositional arguments. They will not work with First-order formulae.

The Analytic Tableau Method

1. Like resolution, the tableau method also checks for the unsatisfiability of a set of formulae.
2. Like resolution, each step of the method *preserves* satisfiability.
3. Unlike resolution or BDDs
 - (a) There are no transformations of mammoth formulae into a normal form (esp. the use of distributivity which can increase sizes of formulae).
 - (b) It works with the list of formulae $[\phi_1, \dots, \phi_n, \neg\psi]$ *directly* by **breaking** up the formulae and building a tree called the **tableau**
 - (c) It relies on the **symmetry between truth and falsehood** at a semantic level to check for satisfiability or unsatisfiability.

Fact 9.1: Basic tableau facts

For any truth assignment τ , and formulae ϕ, ψ ,

$$\mathcal{T}[\neg\phi]_{\tau} = 1 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 0$$

$$\mathcal{T}[\neg\phi]_{\tau} = 0 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 1$$

$$\mathcal{T}[\phi \wedge \psi]_{\tau} = 1 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 1 \quad \& \quad \mathcal{T}[\psi]_{\tau} = 1$$

$$\mathcal{T}[\phi \wedge \psi]_{\tau} = 0 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 0 \quad \mid \quad \mathcal{T}[\psi]_{\tau} = 0$$

$$\mathcal{T}[\phi \vee \psi]_{\tau} = 1 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 1 \quad \mid \quad \mathcal{T}[\psi]_{\tau} = 1$$

$$\mathcal{T}[\phi \vee \psi]_{\tau} = 0 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 0 \quad \& \quad \mathcal{T}[\psi]_{\tau} = 0$$

$$\mathcal{T}[\phi \rightarrow \psi]_{\tau} = 1 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 0 \quad \mid \quad \mathcal{T}[\psi]_{\tau} = 1$$

$$\mathcal{T}[\phi \rightarrow \psi]_{\tau} = 0 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 1 \quad \& \quad \mathcal{T}[\psi]_{\tau} = 0$$

$$\mathcal{T}[\phi \leftrightarrow \psi]_{\tau} = 1 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 1 = \mathcal{T}[\psi]_{\tau}$$

$$\mathcal{T}[\phi]_{\tau} = 0 = \mathcal{T}[\psi]_{\tau}$$

$$\mathcal{T}[\phi \leftrightarrow \psi]_{\tau} = 0 \quad \text{implies} \quad \mathcal{T}[\phi]_{\tau} = 0 = \overline{\mathcal{T}[\psi]_{\tau}}$$

$$\mathcal{T}[\phi]_{\tau} = 1 = \overline{\mathcal{T}[\psi]_{\tau}}$$

The **basic tableau facts** reveal the symmetry between **truth and falsehood** in the boolean algebra. This was also revealed by the **principle of duality** which we have seen earlier. For instance, we could construct for each operator in the boolean algebra its complement operation. The following tables list the truth table each of the complement operations, where the complement of a boolean operation o is denoted \bar{o} .

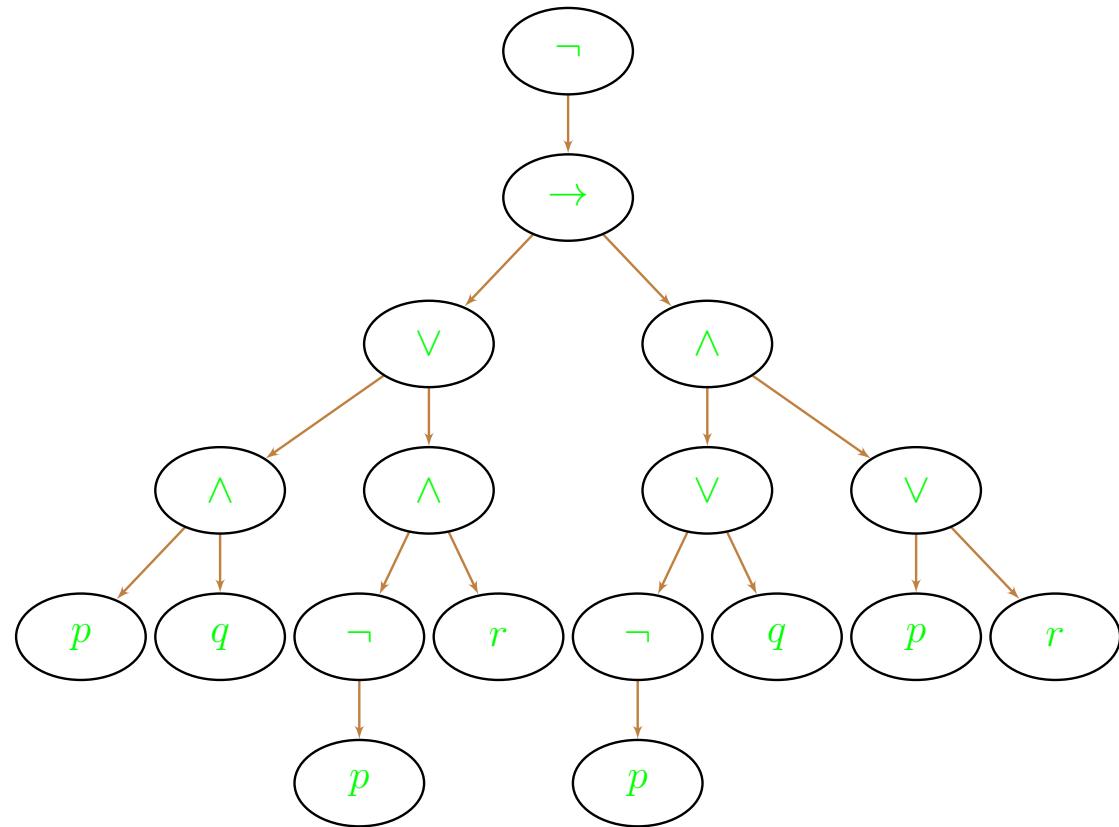
a	\bar{a}
0	0
1	1

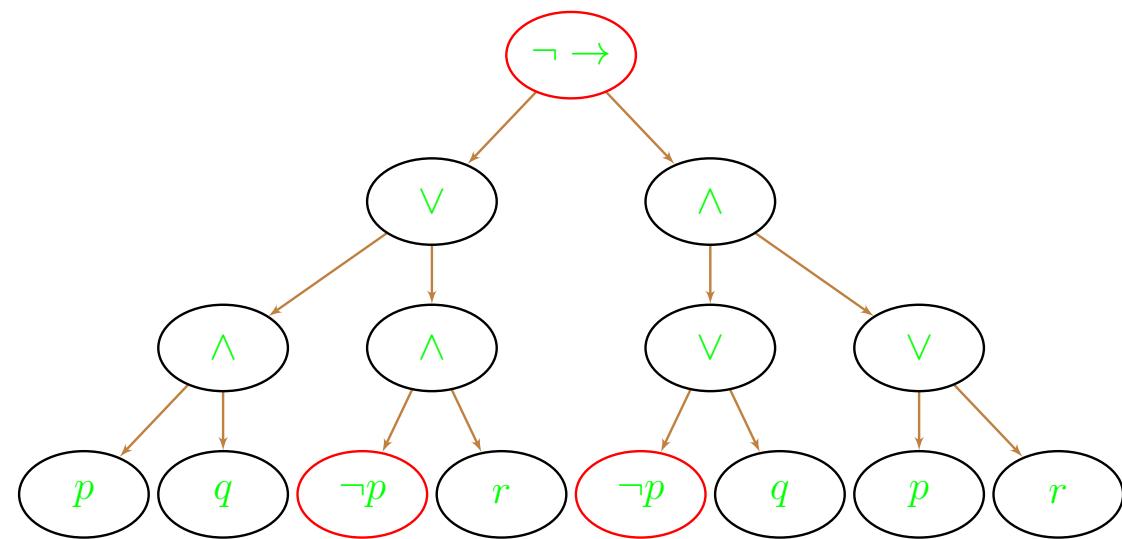
a	b	$a \cdot b$	$a + b$	$a \leq b$	$a = b$
0	0	1	1	0	0
0	1	1	0	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Correspondingly we may think of *literals* rather than *atoms* as the basic building blocks of formulae and the complementary operators in propositional logic as comprising of $\neg\neg$, $\neg\wedge$, $\neg\vee$, $\neg\rightarrow$ and $\neg\Leftrightarrow$, respectively. Hence the abstract syntax tree of the formula

$$\neg((p \wedge q) \vee (\neg p \wedge r) \rightarrow (\neg p \vee q) \wedge (p \vee r)) \quad (34)$$

which would normally be represented as shown in the **first figure** may now be represented with literals and complementary operators as shown in the **second figure**.





Tableaux Rules

	$\neg\neg . \frac{\neg\neg\phi}{\phi}$
$\wedge . \frac{\phi \wedge \psi}{\phi}$ ψ	$\neg \wedge . \frac{\neg(\phi \wedge \psi)}{\neg\phi \mid \neg\psi}$
$\vee . \frac{\phi \vee \psi}{\phi \mid \psi}$	$\neg \vee . \frac{\neg(\phi \vee \psi)}{\neg\phi}$ $\neg\psi$
$\rightarrow . \frac{\phi \rightarrow \psi}{\neg\phi \mid \psi}$	$\neg \rightarrow . \frac{\neg(\phi \rightarrow \psi)}{\phi}$ $\neg\psi$
$\leftrightarrow . \frac{\phi \leftrightarrow \psi}{\phi \wedge \psi \mid \neg\phi \wedge \neg\psi}$	$\neg \leftrightarrow . \frac{\neg(\phi \leftrightarrow \psi)}{\phi \wedge \neg\psi \mid \neg\phi \wedge \psi}$

Structure of the Rules

The **tableaux rules** are of two kinds:

Elongation rules (&) Each of the rules $\neg\neg$, \wedge , $\neg\vee$ and $\neg \rightarrow$ elongates the path without increasing the size of the tableau (since the original formula to which a rule was applied may be discarded^a)

Branching rules (|) These are the rules $\neg\wedge$, \vee , $\neg \rightarrow$, \leftrightarrow and $\neg \leftrightarrow$ which lead to branching of the tableau.

^aThis can happen only in Propositional logic; it would not be possible in First-order logic as we shall see later.

Tableaux

1. A *tableau* is a rooted tree (definition -2.2) where each path of the tableau represents a conjunction of “unbroken” formulae.
2. Each application of the **tableaux rules** preserves satisfiability of the conjunction of “unbroken” formulae in each path.
3. A path of the tableau is **closed** if it contains a complementary pair (the conjunction of the formulae in the path is clearly **unsatisfiable**).
4. The result of applying a tableau rule to an ancestor node has to be distributed in all branches of its descendants.
5. A tableau is **closed** if every path in the tableau is closed signifying that the original set of formulae is unsatisfiable.

An Example Tableau Proof

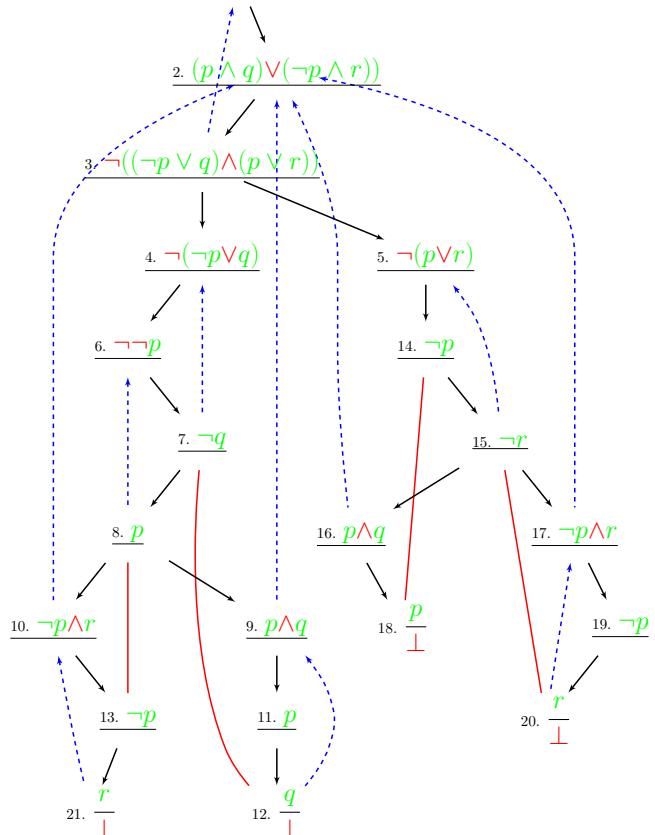
Example 9.1

We provide a tableau proof of the *unsatisfiability* of the formula

$$\neg((p \wedge q) \vee (\neg p \wedge r) \rightarrow (\neg p \vee q) \wedge (p \vee r))$$

The black arrows denote the paths in the tableau. The **blue dashed arrows** provide the justification for each step of the tableau unless it is derived from its parent in the path. The last node in a closed path is marked by a \perp . The **undirected lines** join the complementary pairs occurring in the path and provide the justification for closing the path.

1. $\neg(((p \wedge q) \vee (\neg p \wedge r)) \rightarrow ((\neg p \vee q) \wedge (p \vee r)))$



Tableaux Heuristics

1. Any formula which has been broken up by a tableau rule may be discarded^a.
2. Any branch which has been closed may be discarded.
3. Any formula which dominates several branches of the tableau creates multiple copies (one in each branch of its descendants) when it is broken up.

^aonly in Propositional tableaux; not in first-order tableaux

Slim & Short Tableaux

By applying the elongation rules first (whenever possible) the number of branches over which elongation rules have to be replicated can be reduced.

Once a propositional sentence is broken up and its components distributed on all branches, the original sentence may be discarded.

Exercise 9.1

1. Use the tableau method to prove the following tautologies.
 - (a) $p \rightarrow (q \rightarrow p)$.
 - (b) Peirce's law: $((p \rightarrow q) \rightarrow p) \rightarrow p$.
 - (c) $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$
 - (d) $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$.
2. Prove the **example argument** (given earlier) by the tableau method.

10. Consistency & Completeness

10: Consistency & Completeness



"WE BELIEVE WE WILL BE ABLE TO TEAR DOWN THE ENTIRE ECO SYSTEM IN AN ENVIRONMENTALLY FRIENDLY MANNER."

1. Tableaux Rules: Restructuring
2. Tableaux Rules: 2
3. Tableau Proofs
4. Consistency
5. Satisfiability Preservation
6. Unsatisfiability
7. Derived Formulae
8. Hintikka Sets
9. Degree of a Formula
10. Degree of a Derived Formula
11. Hintikka's Lemma
12. Tableaux and Hintikka sets
13. Soundness of the Tableau Method
14. Completeness of the Tableau Method

10.1. Consistency and Completeness

We use the tableau method and restructure it for the purpose of proving consistency and completeness. We will show that an open tableau i.e. that is one in which not all paths are closed, is actually satisfiable. In fact, one may take all the literals in an open path and use them to assign a satisfying truth assignment. There could be many open paths and therefore many satisfying truth assignments.

Our method of restructuring the tableau is to keep the candidate formulae which have not been “discarded”. Hence each node of the tableau contains formulae derived from its ancestors along with those which still are candidates for extending the tableau. Once the tableau is completed, the set of formulae at the leaf node of an open path would be satisfiable.

Tableaux Rules: Restructuring

In general the **elongation and branching rules** of the tableau look like this

Elongation. $\frac{\phi}{\psi}$ χ	Branching. $\frac{\phi}{\psi \quad \quad \chi}$
--	--

where ψ and χ are **subformulae** of ϕ .

Let $\Gamma = \Delta \cup \{\phi\}$ where $\phi \notin \Delta$ be a set of formulae. It will be convenient to use sets of formulae in the tableau rules. The elongation and branching rules are rendered as follows respectively

Elongation. $\frac{\Delta \cup \{\phi\}}{\Delta \cup \{\psi, \chi\}}$	Branching. $\frac{\Delta \cup \{\phi\}}{\Delta \cup \{\psi\} \quad \quad \Delta \cup \{\chi\}}$
--	--

Tableaux Rules: 2

$\perp.$	$\frac{\Delta \cup \{\phi, \neg\phi\}}{\{\perp\}}$	$\neg\neg.$	$\frac{\Delta \cup \{\neg\neg\phi\}}{\Delta \cup \{\phi\}}$
\wedge	$\frac{\Delta \cup \{\phi \wedge \psi\}}{\Delta \cup \{\phi, \psi\}}$	$\neg\wedge.$	$\frac{\Delta \cup \{\neg(\phi \wedge \psi)\}}{\Delta \cup \{\neg\phi\} \mid \Delta \cup \{\neg\psi\}}$
\vee	$\frac{\Delta \cup \{\phi \vee \psi\}}{\Delta \cup \{\phi\} \mid \Delta \cup \{\psi\}}$	$\neg\vee.$	$\frac{\Delta \cup \{\neg(\phi \vee \psi)\}}{\Delta \cup \{\neg\phi, \neg\psi\}}$
$\rightarrow.$	$\frac{\Delta \cup \{\phi \rightarrow \psi\}}{\Delta \cup \{\neg\phi\} \mid \Delta \cup \{\psi\}}$	$\neg\rightarrow.$	$\frac{\Delta \cup \{\neg(\phi \rightarrow \psi)\}}{\Delta \cup \{\phi, \neg\psi\}}$

$\leftrightarrow.$	$\frac{\Delta \cup \{\phi \leftrightarrow \psi\}}{\Delta \cup \{\phi \wedge \psi\} \mid \Delta \cup \{\neg\phi \wedge \neg\psi\}}$
$\neg\leftrightarrow.$	$\frac{\Delta \cup \{\neg(\phi \leftrightarrow \psi)\}}{\Delta \cup \{\phi \wedge \neg\psi\} \mid \Delta \cup \{\neg\phi \wedge \psi\}}$

Tableau Proofs

1. A tableau is a tree rooted at a node containing a set Γ of formulas
2. Each application of
 - an **elongation** rule $\frac{\Gamma}{\Gamma'}$ to a leaf Γ of the tableau extends the path to Γ' ,
 - a **branching** rule $\frac{\Gamma}{\Gamma' \quad | \quad \Gamma''}$ to a leaf Γ of the tableau extends the tableau to two leaves Γ' and Γ'' .
3. A path of the tableau is **closed** if its leaf is $\{\perp\}$.
4. The tableau is **closed** if every path is closed, otherwise the the tableau is **open**.

Consistency

Definition 10.1: Consistency of sets of propositions

A set Γ of formulas is **consistent** if it is satisfiable i.e. there is a truth assignment under which every formula of Γ is true. Otherwise, it is **inconsistent**.

Fact 10.1: Subset consistency

Every non-empty subset of a consistent set is also consistent

Lemma 10.1: Satisfiability preservation

Each **tableau rule** preserves satisfiability in the following sense.

Elongation Rules ($\frac{\Gamma}{\Gamma'}$) If the numerator Γ is satisfiable then so is the denominator Γ' .

Branching Rules ($\frac{\Gamma}{\Gamma' \quad | \quad \Gamma''}$) If the numerator Γ is satisfiable then at least one of the denominators Γ' or Γ'' is satisfiable.

Proof: It may be shown that for any truth assignment τ ,

Elongation Rules $\frac{\Gamma}{\Gamma'}$. if every formula in Γ is true then every formula in Γ' is also true under τ .

Branching Rules $\frac{\Gamma}{\Gamma'|\Gamma''}$. if every formula in Γ is true under τ then every formula in Γ' or every formula in Γ'' is true under τ .

Unsatisfiability

Definition 10.2: Completed tableau

A **tableau** is **completed** if no leaf in any path may be extended.

Corollary 10.1: Open tableau

If Γ is satisfiable then there exists a completed tableau rooted at Γ which has a satisfiable **leaf**.

Corollary 10.2: Tableau unsatisfiability

A set Γ is unsatisfiable if there exists a closed tableau rooted at Γ .

Question. If a completed tableau rooted at Γ is closed could there be other completed tableaux rooted at Γ which might be open?

Derived Formulae

Definition 10.3: Derived formulae

The derived formulae of any *complex formula* (i.e one which is not a literal or a constant) ϕ (called the *parent*) is defined as follows.

- If $\phi \equiv \neg\neg\psi$ then ψ is the derived formula.
- If $\phi \equiv \psi \odot \chi$ or $\phi \equiv \psi \oplus \chi$ then its derived formulae are ψ' and χ' as specified in the following table.

$\psi \odot \chi$	ψ'	χ'	$\psi \oplus \chi$	ψ'	χ'
$\psi \wedge \chi$	ψ	χ	$\neg(\psi \wedge \chi)$	$\neg\psi$	$\neg\chi$
$\neg(\psi \vee \chi)$	$\neg\psi$	$\neg\chi$	$\psi \vee \chi$	ψ	χ
$\neg(\psi \rightarrow \chi)$	ψ	$\neg\chi$	$\psi \rightarrow \chi$	$\neg\psi$	χ
			$\psi \leftrightarrow \chi$	$\psi \wedge \chi$	$\neg\psi \wedge \neg\chi$
			$\neg(\psi \leftrightarrow \chi)$	$\neg\psi \wedge \chi$	$\psi \wedge \neg\chi$

Hintikka Sets

Definition 10.4: Hintikka set

A finite or infinite set Γ is a **Hintikka set** if

0. $\perp \notin \Gamma$ and for any $p \in A$, $\{p, \neg p\} \not\subseteq \Gamma$,
1. If $\neg\neg\phi \in \Gamma$ then $\phi \in \Gamma$,
2. If $\psi \odot \chi \in \Gamma$ for $\odot \in \{\wedge, \neg\vee, \neg\rightarrow\}$ then $\{\psi', \chi'\} \subseteq \Gamma$,
3. If $\psi \oplus \chi \in \Gamma$ for $\oplus \in \{\vee, \neg\wedge, \rightarrow, \leftrightarrow, \neg\leftrightarrow\}$ then $\{\psi', \chi'\} \cap \Gamma \neq \emptyset$

where ψ' and χ' are defined by the **table of derived formulae**.

Definition 10.5: Degree of a formula

The degree of a formula is defined by induction on the structure of formulae.

$$\text{degree}(\perp) = 0$$

$$\text{degree}(\top) = 0$$

$$\text{degree}(p) = 1,$$

for each atom p

$$\text{degree}(\neg p) = 1$$

$$\text{degree}(\neg\neg\phi) = 1 + \text{degree}(\phi)$$

$$\text{degree}(\psi \odot \chi) = 1 + \text{degree}(\psi) + \text{degree}(\chi),$$

$\odot \in \{\wedge, \vee, \rightarrow, \neg\wedge, \neg\vee, \neg\rightarrow\}$

$$\text{degree}(\psi \oplus \chi) = 1 + 2(\text{degree}(\psi) + \text{degree}(\chi)),$$

$\oplus \in \{\leftrightarrow, \neg\leftrightarrow\}$

Degree of a Derived Formula

Fact 10.2: Degree of derived formula

1. Every **derived formula** has a smaller **degree** than its parent.
2. Each application of a **tableau rule** extends it by formulae that have lower degrees than their parents.

Exercise 10.1: Hintikka sets

1. Prove that $\text{degree}(\psi) < \text{degree}(\phi)$ if ψ is a derived formula of a (complex) formula ϕ .
2. Which of the following are Hintikka sets?
 - (a) The empty set of formulae.
 - (b) The set \mathcal{P}_0 of all propositional formulae.
 - (c) The set A of propositional atoms.
 - (d) The set of all literals.
3. Prove by structural induction that for any proposition ϕ and a Hintikka set Γ , $\{\phi, \neg\phi\} \notin \Gamma$.

Hintikka's Lemma

Lemma 10.2: Hintikka satisfiability

Every Hintikka set is satisfiable.

Proof: Let Γ be a Hintikka set. For any atom p , since $\{p, \neg p\} \not\subseteq \Gamma$, consider the following truth assignment τ .

1. $\tau(p) = 1$ if $p \in \Gamma$,
2. $\tau(p) = 0$ if $\neg p \in \Gamma$ and
3. if $\{p, \neg p\} \cap \Gamma = \emptyset$ then choose any value (say 1 for definiteness).

We may then show by induction on the degree of formulae in Γ that each formula in Γ is satisfiable. QED

Tableaux and Hintikka sets

Theorem 10.1: Tableaux and Hintikka sets

Let $\Gamma_0, \Gamma_1 \dots, \Gamma_n$ be an open path of a completed tableau. Then $\Gamma = \bigcup_{0 \leq m \leq n} \Gamma_m$ is a Hintikka set.

Proof: We may prove that each rule in [Tableaux Rules: 2](#) creates a path for the construction of Hintikka sets. For any open path of the completed tableau, choose a truth assignment τ such that for each atom p , if $p \in \Gamma$ then $\tau(p) = 1$ and if $\neg p \in \Gamma$ then $\tau(p) = 0$. QED

Soundness of the Tableau Method

Theorem 10.2: Soundness of the Tableau Method

If ϕ is a tautology then every completed tableau rooted at $\{\neg\phi\}$ is closed.

Proof: Suppose \mathcal{T} is a completed tableau rooted at $\{\neg\phi\}$ which is open. Then by corollary 10.1 $\neg\phi$ must be satisfiable and therefore ϕ cannot be a tautology. Hence \mathcal{T} must be closed. QED

Completeness of the Tableau Method

Theorem 10.3: Completeness of the Tableau Method

Every tautology is provable by the tableau method.

Proof: If ϕ is a tautology that cannot be proved by the tableau method, there must exist a completed tableau rooted at $\{\neg\phi\}$ which has an open path. But that implies $\{\neg\phi\}$ is satisfiable which implies that ϕ is not a tautology. QED

Exercise 10.2: Soundness and completeness

1. Answer the question that was posed [earlier](#). Let Γ be a non-empty finite set of formulae. Prove that if a completed tableau rooted at Γ is closed then so is every completed tableau rooted at Γ .
2. Prove that [propositional resolution](#) is sound and complete.

11. The Compactness Theorem

11: The Compactness Theorem

1. Satisfiability of Infinite Sets
2. The Compactness Theorem
3. Inconsistency
4. Consequences of Compactness

Satisfiability of Infinite Sets

From corollaries 10.1 and 10.2 we have

Corollary 11.1: Unsatisfiability characterisation

A finite set Γ is unsatisfiable iff there is a closed tableau rooted at Γ .

Corollary 11.2: Subset satisfiability

If a finite set Γ is satisfiable then every nonempty subset of Γ is satisfiable too.

Question 1. Suppose Γ were a denumerable (countably infinite) set. Under what conditions is Γ satisfiable?

Question 2. Suppose every subset of a denumerable set Γ is satisfiable. Then is Γ necessarily satisfiable?

Question 3. Suppose that only all finite subsets of a denumerable set Γ are satisfiable. Then is Γ satisfiable?

The Compactness Theorem

Theorem 11.1: The Compactness Theorem

A countably infinite set is satisfiable if all its nonempty finite subsets are satisfiable.



Corollary 11.3: Satisfiability characterisation

Any (finite or infinite) set of formulae is satisfiable iff all its non-empty finite subsets are satisfiable.

Note:

- If Γ is a countably infinite set then it can be placed in 1-1 correspondence with the set \mathbb{N} of naturals and hence there is some enumeration of its formulae and each formula carries an unique index from \mathbb{N} .

Proof of the Compactness Theorem

Proof: Let Γ be a countably infinite set of propositions such that every finite subset of Γ is consistent. We need to prove that Γ is consistent.

Since Γ is countable, the formulae in Γ may be enumerated in some order, say

$$\{\phi_0, \phi_1, \phi_2, \dots\} \quad (35)$$

where each ϕ_j has the unique index $j \geq 0$. For each $m \geq 0$, let

$$\Gamma_m = \{\phi_0, \phi_1, \phi_2, \dots, \phi_m\} \quad (36)$$

Further, by the hypothesis, each Γ_m is satisfiable. In particular, we may assume that for each $m \geq 0$, there exists a truth assignment τ_m such that every formula in Γ_m under τ_m is true (however the τ_m may all be different).

Claim. Every nonempty finite subset of Γ is satisfiable iff for each $m \geq 0$, Γ_m is satisfiable.

$\vdash (\Rightarrow)$ clearly holds since each Γ_m is a finite subset.

(\Leftarrow) Let $\emptyset \neq \Delta \subseteq_f \Gamma$. Let $k \geq 0$ be the index of the formula with the highest index in Δ . Clearly $\Delta \subseteq_f \Gamma_k$. Since the set Γ_k is satisfiable under some truth assignment τ_k , by corollary 11.2, Δ is also satisfiable under τ_k . \dashv

Hence it suffices to prove the following claim.

Claim. If each of the Γ_i , $i \geq 0$, in the enumeration (36) is satisfiable then Γ is satisfiable.

\vdash Consider a tableau \mathcal{T}_0 rooted at Γ_0 constructed using the **tableau rules**. Since Γ_0 is satisfiable, \mathcal{T}_0 has one or more open paths. Extend each of the open paths with the formula ϕ_1 and continue the tableau. The resulting tableau \mathcal{T}_1 is for the set Γ_1 and it does not close because Γ_1 is satisfiable. Proceeding in this fashion for each value of $k \geq 0$ we get tableaux \mathcal{T}_k for each Γ_k contains at least one open path and all open paths may be extended to yield an open tableau \mathcal{T}_{k+1} for Γ_{k+1} .

Consider the final tableau \mathcal{T} obtained by this process of extension. \mathcal{T} is a *finitely branching infinite tree* with at least one path that does not close. By König's Lemma -2.2 there is an infinite path. Let Φ be the set of all formulae in this path. Since this path contains each of the formulae $\phi_i \in \Gamma$, we have $\Gamma \subseteq \Phi$ and further Φ is a Hintikka set by theorem 10.1. By Hintikka's lemma (lemma 10.2) this set must be satisfiable. In fact the truth assignment that satisfies Γ is any truth assignment τ which ensures that $\tau(p) = 1$ if $p \in \Phi$ and $\tau(p) = 0$ if $\neg p \in \Phi$. For any atom q such that $\{q, \neg q\} \cap \Phi = \emptyset$, $\tau(q)$ could be any truth value. \dashv

QED

Inconsistency

Corollary 11.4: Inconsistency

A set Γ is **inconsistent** if some nonempty finite subset of Γ is unsatisfiable.

Proof: Follows from the **compactness theorem** and its corollary **11.3**.

QED

Fact 11.1: More facts-inconsistency

1. Any superset of an inconsistent set is also inconsistent.
2. Any set containing a complementary pair is inconsistent.
3. (see **table**) If $\Delta \cup \{\psi', \chi'\}$ is inconsistent then so is $\Delta \cup \{\phi\}$ where $\phi \equiv \psi \odot \chi$
4. (see **table**) If both $\Delta \cup \{\psi'\}$ and $\Delta \cup \{\chi'\}$ are inconsistent then so is $\Delta \cup \{\phi\}$ where $\phi \equiv \psi \oplus \chi$.

Consequences of Compactness

Corollary 11.5: Finite deducibility

Given a finite or infinite set Γ , and a formula ψ

1. $\Gamma \cup \{\neg\psi\}$ is inconsistent iff there exists $\Delta = \{\phi_i \mid 1 \leq i \leq n\} \subseteq_f \Gamma$, $n \geq 0$, such that $\Delta \cup \{\neg\psi\}$ is inconsistent.
2. $\Gamma \models \psi$ iff $\Delta \models \psi$ iff $(\bigwedge \Delta) \rightarrow \psi$ is a tautology, for some $\Delta = \{\phi_i \mid 1 \leq i \leq n\} \subseteq_f \Gamma$.

Hence

1. to show that an argument is valid it suffices to prove that the conclusion follows from a finite subset of the hypotheses.
2. to show invalidity of an argument it suffices to find a finite subset of the hypotheses which are inconsistent with the conclusion.

Exercise 11.1: Compactness

1. Now answer the questions posed earlier.
2. The proof of the compactness theorem is general enough that it may be used in other areas of mathematics too. Use the same ideas to prove that every (finite or countable) partially ordered set may be totally ordered.
3. Let Γ be a finite or infinite set of propositions and let ψ be any formula. Then prove that either $\Gamma \cup \{\psi\}$ or $\Gamma \cup \{\neg\psi\}$ is satisfiable.

12. Maximally Consistent Sets

12: Maximally Consistent Sets

1. Consistent Sets
2. Maximally Consistent Sets
3. Properties of Finite Character: 1
4. Properties of Finite Character: Examples:1
5. Properties of Finite Character: Examples:2
6. Properties of Finite Character: Compactness
7. Properties of Finite Character: Tukey's Lemma
8. Lindenbaum's Theorem

Consistent Sets

Lemma 12.1: Consistent sets extension

If Γ is a consistent set then for any formula ϕ at least one of the two sets $\Gamma_1 = \Gamma \cup \{\phi\}$ or $\Gamma_0 = \Gamma \cup \{\neg\phi\}$ is consistent.

Proof: Assume Γ is consistent. Then there exists a truth assignment τ under which every formula in Γ is true. Depending upon whether ϕ is true or false under τ it is clear that one of Γ_0 or Γ_1 has to be consistent.
QED

Maximally Consistent Sets

Definition 12.1: Maximally consistent sets

A set Δ is a **maximally consistent set** if it is satisfiable and no proper superset of Δ is consistent.

Corollary 12.1: Maximally consistent sets partition \mathcal{P}_0

For any maximally consistent set Δ , and any formula ϕ , either $\phi \in \Delta$ or $\neg\phi \in \Delta$ but not both.

Hence if Δ is maximally consistent then the inclusion of any proposition in $\mathcal{P}_0 - \Delta$ would lead to inconsistency.

Our proof of the compactness theorem (theorem 11.1) relied on the use of the Tableau Method. However since the notions of satisfiability, consistency and maximal consistency are semantical notions, it is intuitively clear that the property of compactness should not depend on any particular proof system or method. Hence it must be possible to prove the compactness theorem without invoking any proof method. So we shall try to prove the compactness theorem for Propositional Logic only from semantical notions. Of course if we have not yet proved the compactness theorem, we cannot also assume that consistency/satisfiability is a property of finite character. However we may assume lemma 12.1 which refers to how consistent sets may be extended, definition 12.1 which defines maximally consistent sets and corollary 12.1 which denies the inclusion of both a formula and its negation in a maximally consistent set.

Finite Satisfiability

Definition 12.2: Finite satisfiability

A set Γ is called **finitely satisfiable** if every non-empty finite subset of Γ is satisfiable.

If Γ is satisfiable, then clearly it is also finitely satisfiable. So we concentrate on only one part of the compactness theorem.

Maximally Consistent Sets and Compactness

Theorem 12.1: The Compactness Theorem (\Leftarrow).

Γ is satisfiable if it is finitely satisfiable.

Proof: (\Leftarrow) Assume Γ is finitely satisfiable. The theorem follows from the following two claims.

Claim 1. There exists a maximal set $\Delta \supseteq \Gamma$ such that Δ is finitely satisfiable^a.

Claim 2. There exists a single truth assignment τ such that every formula in Δ is true under τ . QED

Claim 2 shows that the set Δ constructed in the proof of *Claim 1* is indeed a **maximally consistent set**.

^aNote that we are not claiming here that Δ is a maximal consistent set

Claim. There exists a maximal set $\Delta \supseteq \Gamma$ such that Δ is **finitely satisfiable**.

⊤ Since \mathcal{P}_0 is countably infinite (see problem 1 in exercise 2.1) there exists an enumeration

$$\phi_1, \phi_2, \phi_3, \dots \tag{37}$$

Starting from $\Delta_0 = \Gamma$ we now construct the sets Δ_{n+1} by induction as follows¹²:

$$\Delta_{n+1} = \begin{cases} \Delta_n \cup \{\phi_{n+1}\} & \text{if } \Delta_n \cup \{\phi_{n+1}\} \text{ is finitely satisfiable} \\ \Delta_n \cup \{\neg\phi_{n+1}\} & \text{otherwise} \end{cases}$$

This yields the following chain of sets of formulae

$$\Gamma = \Delta_0 \subseteq \Delta_1 \subseteq \Delta_2 \subseteq \dots \tag{38}$$

Let $\Delta = \bigcup_{n \geq 0} \Delta_n$. It is clear that $\Gamma \subseteq \Delta$ and for every ϕ either ϕ or $\neg\phi$ is in Δ . Hence Δ is maximal. Further,

since Γ is finitely satisfiable, by lemma 12.1 and induction on $n \geq 0$ each Δ_n can be shown to be **finitely satisfiable**. Therefore Δ is **finitely satisfiable** because every finite subset of Δ is contained in some Δ_n for some $n \geq 0$ and each Δ_n is **finitely satisfiable**. Hence Δ is a maximal finitely satisfiable superset of Γ . \dashv

Claim. There exists a single truth assignment τ such that every formula in Δ is true under τ .

⊤ Since Δ is maximal, for every atom p , either p or $\neg p$ belongs to Δ . Define the truth assignment τ such that

¹²Of course, if $\phi_{n+1} \in \Delta_n$ or $\neg\phi_{n+1} \in \Delta_n$ then $\Delta_n = \Delta_{n+1}$.

$\tau(p) = 1$ if $p \in \Delta$ and $\tau(p) = 0$ otherwise. We may now prove by induction on the structure of ϕ that

$$\phi \in \Delta \text{ iff } \mathcal{T}[\![\phi]\!]_\tau = 1$$

and

$$\neg\phi \in \Delta \text{ iff } \mathcal{T}[\![\phi]\!]_\tau = 0$$

This structural induction proof is quite easy and left to the reader. \dashv

Properties of Finite Character: 1

Definition 12.3: Property of finite character

A property \mathfrak{p} of sets is called a **property of finite character** if for any set S , S has the property \mathfrak{p} iff every finite subset of S has the property \mathfrak{p} .

Notation: $S \Vdash \mathfrak{p}$ denotes the statement “ S has property \mathfrak{p} ”.

Properties of Finite Character: Examples:1

Example 12.1

The property of a partially ordered set being totally ordered (definition -4.12) is a property of finite character. That is,

If $\langle P, \leq \rangle$ is a partially ordered set, then P is *totally ordered* (i.e. for every $a, b \in P$, $a \leq b$ or $b \leq a$) iff every finite subset of P is totally ordered.

See problem 4 of exercise -4.9.

Properties of Finite Character: Examples:2

Example 12.2

However the property of a totally ordered set $\langle T, \leq \rangle$ being well-ordered (definition -4.17) is not a property of finite character since even though every finite subset of T may be well-ordered, T itself may not be well-ordered. A concrete instance is the set \mathbb{Z} under the usual \leq relation – every finite subset of \mathbb{Z} is well-ordered but \mathbb{Z} itself is not.

Properties of Finite Character: Compactness

By the corollary 11.3 to the compactness theorem, consistency/satisfiability is a property of finite character.

Properties of Finite Character: Tukey's Lemma

Theorem 12.2: Tukey's Lemma

For any denumerable universe U and any property \mathfrak{p} of finite character of subsets of U , any set $S \subseteq U$ such that $S \Vdash \mathfrak{p}$ can be extended to a maximal set S_∞ such that $S \subseteq S_\infty \subseteq U$ with $S_\infty \Vdash \mathfrak{p}$.



Proof of Tukey's Lemma (theorem 12.2)

Proof: Let $S \subseteq U$ be a set with $S \Vdash \mathfrak{p}$. Since U is denumerable its elements can be enumerated in some order

$$a_1, a_2, a_3, \dots \quad (39)$$

Starting with $S = S_0$ consider the sets S_{i+1} , $i \geq 0$

$$S_{i+1} = \begin{cases} S_i \cup \{a_{i+1}\} & \text{if } S_i \cup \{a_{i+1}\} \Vdash \mathfrak{p} \\ S_i & \text{otherwise} \end{cases}$$

Of course, if $a_{i+1} \in S$ then $S_{i+1} = S_i$. Clearly we have the infinite chain

$$S = S_0 \subseteq S_1 \subseteq S_2 \subseteq \dots$$

such that for each S_i , $S_i \Vdash \mathfrak{p}$. Let $S_\infty = \bigcup_{i \geq 0} S_i$.

Claim. $S_\infty \Vdash \mathfrak{p}$.

Let $T \subseteq_f S_\infty$ be any non-empty finite subset of S_∞ . Then $T \subseteq_f S_i$ for some $i \geq 0$. Since $S_i \Vdash \mathfrak{p}$, \mathfrak{p} is a property of finite character and $T \subseteq_f S_i$, $T \Vdash \mathfrak{p}$. Hence (not just every $S_i \subseteq S_\infty$ but) every finite subset of S_∞ has property \mathfrak{p} . Therefore since \mathfrak{p} is a property of finite character, $S_\infty \Vdash \mathfrak{p}$. \dashv

Claim. S_∞ is maximal.

Suppose there exists an element $a \in U$ such that $S_\infty \cup \{a\} \Vdash \mathfrak{p}$. We know from the claim above that $S_\infty \Vdash \mathfrak{p}$ and from the construction of each S_i , that $S_i \Vdash \mathfrak{p}$ for each $i \geq 0$. Clearly $a = a_{j+1}$ for some $j \geq 0$ in the enumeration (39). Hence $S_j \cup \{a_{j+1}\} \Vdash \mathfrak{p}$. But $S_j \cup \{a_{j+1}\} = S_{j+1} \subseteq S_\infty$. Hence $S_\infty = S_\infty \cup \{a\}$ and S_∞ is maximal. \dashv

QED

Lindenbaum's Theorem

Theorem 12.3: Lindenbaum's Theorem

Every consistent set can be extended to a maximally consistent set. More precisely for every consistent set of formulae Γ there exists a maximally consistent set $\Gamma_\infty \supseteq \Gamma$.

Proof: By definition 12.3 and corollary 11.3 consistency of sets of formulae is a property of finite character in the universe \mathcal{P}_0 . From Tukey's lemma (theorem 12.2) it follows that any set consistent set $\Gamma \subseteq \mathcal{P}_0$ may be extended to a maximally consistent set Γ_∞ . QED

An alternative proof

Alternative Proof of Lindenbaum's Theorem *ab initio*

Proof: Let Γ be a consistent set. Since \mathcal{P}_0 is generated from a countably infinite set of atoms and a finite set of operators, \mathcal{P}_0 is a countably infinite set (see problem 1 of exercise 2.1). Hence the formulae of \mathcal{P}_0 can be enumerated in some order

$$\phi_1, \phi_2, \phi_3, \dots \quad (40)$$

Starting with $\Gamma = \Gamma_0$ consider the sets

$$\Gamma_{i+1} = \begin{cases} \Gamma_i \cup \{\phi_{i+1}\} & \text{if } \Gamma_i \cup \{\phi_{i+1}\} \text{ is consistent} \\ \Gamma_i & \text{otherwise} \end{cases}$$

Clearly we have the infinite chain

$$\Gamma = \Gamma_0 \subseteq \Gamma_1 \subseteq \Gamma_2 \subseteq \dots$$

such that each Γ_i is consistent. Let $\Gamma_\infty = \bigcup_{i \geq 0} \Gamma_i$.

Claim. Γ_∞ is consistent.

Let $\Delta \subseteq_f \Gamma_\infty$, then since Δ is finite, it must be the subset of some Γ_i . Since Γ_i is consistent, so is Δ . Hence every finite subset of Γ_∞ is consistent. Therefore by the compactness theorem Γ_∞ is consistent. \dashv

Claim. Γ_∞ is maximal.

Suppose there exists a formula ϕ such that $\Gamma_\infty \cup \{\phi\}$ is consistent. Clearly $\phi \equiv \phi_{i+1}$ for some $i \geq 0$ in the enumeration (40). Since $\Gamma_\infty \cup \{\phi_{i+1}\}$ is consistent, by fact 10.1 $\Gamma_i \cup \{\phi_{i+1}\} \subseteq \Gamma_\infty \cup \{\phi_{i+1}\}$ is also consistent. But then $\Gamma_{i+1} = \Gamma_i \cup \{\phi_{i+1}\} \subseteq \Gamma_\infty$ and hence $\Gamma_\infty = \Gamma_\infty \cup \{\phi\}$. Hence Γ_∞ is maximal. \dashv

QED

Exercise 12.1: Properties of finite character and compactness

1. Let $\langle P, \leq \rangle$ be a finite partial order. Prove using König's lemma that every element of P lies between a maximal element and a minimal element i.e. for each $a \in P$ there exist a minimal element $l \in P$ and a maximal element $u \in P$ such that $l \leq a \leq u$.
2. Prove that every maximally consistent set is a **Hintikka set**.
3. For any given consistent set Γ of formulae, there may exist more than one maximally consistent extension. Give examples of Γ and ψ such that there are two maximally consistent extensions, Γ_∞ and Γ'_∞ with $\psi \in \Gamma_\infty$ and $\neg\psi \in \Gamma'_\infty$.
4. (**Tarski's theorem**). For any set Γ , of formulae, the set Γ^{\models} called the **closure under logical consequence** is defined as $\Gamma^{\models} = \{\psi \in \mathcal{P}_0 \mid \Delta \models \psi, \text{ for some } \Delta \subseteq_f \Gamma\}$.

Let $\mathcal{MC}(\Gamma) = \{\Gamma_\infty \mid \Gamma_\infty \text{ is a maximally consistent extension of } \Gamma\}$ be the set of all maximally consistent extensions of Γ . Prove that $\Gamma^{\models} = \bigcap_{\Gamma_\infty \in \mathcal{MC}(\Gamma)} \Gamma_\infty$ for every consistent set Γ .

5. (**Interpolation**) For any finite set $V \subseteq_f A$, define $T_V = \{\tau_V \mid \tau_V : V \rightarrow \{\perp, \top\}\}$ and for any formula χ such that $V \subseteq \text{atoms}(\chi)$ and any $\tau_V \in T_V$, let $\tau_V(\chi)$ denote the formula obtained from χ by replacing all occurrences of each atom $p \in V$ by the atom $\tau_V(p)$. Further let $T_V(\chi) = \{\tau_V(\chi) \mid \tau_V \in T_V\}$.

Let $X, Y, Z \subseteq_f A$ be pairwise disjoint (finite) sets of atoms and let ϕ and ψ be formulae such that $\models \phi \rightarrow \psi$, $\text{atoms}(\phi) \subseteq X \cup Y$ and $\text{atoms}(\psi) \subseteq Z \cup Y$.

- (a) Let $\lambda \stackrel{df}{=} \bigvee T_X(\phi)$ and $\rho \stackrel{df}{=} \bigwedge T_Z(\psi)$. Then prove that

- i. $\models \phi \rightarrow \lambda$
- ii. $\models \lambda \rightarrow \rho$
- iii. $\models \rho \rightarrow \psi$

- (b) Prove that for any formula θ with $\text{atoms}(\theta) \subseteq Y$, if $\models \phi \rightarrow \theta$ and $\models \theta \rightarrow \psi$ then $\models \lambda \rightarrow \theta$ and $\models \theta \rightarrow \rho$. θ is called an **interpolant** of ϕ and ψ .

13. Formal Theories

13: Formal Theories

Pure mathematics is the subject in which we do not know what we are talking about and whether what we are saying is true.

Bertrand Russell

1. Introduction to Reasoning
2. Proof Systems: 1
3. Requirements of Proof Systems
4. Proof Systems: Desiderata
5. Formal Theories
6. Formal Language
7. Axioms and Inference Rules
8. Axiomatic Theories
9. Syntax and Decidability
10. A Hilbert-style Proof System
11. Axiom Schemas and Rules
12. Rule Patterns

Introduction to Reasoning

1. The methods discussed so far viz. **truth-table**, **tautology checking**, **resolution** and **tableau** – are useful for automated deduction, but
2. they do not reflect the process of reasoning employed by humans and used most often in mathematical proofs called **deduction**.
3. Deduction enables the proof of **validity** of arguments but seldom their **invalidity**.

Proof Systems: 1

A proof system for deduction

1. prohibits the use of meaning in drawing conclusions. (also see the meaning-less but valid argument)
2. has a number of axioms (or axiom schemas) and a small number of (finitary) inference rules.
3. Each proof is a finite tree called a proof tree.
4. each node of the proof tree is either an assumption or an axiom or is an instantiation (see definition 0.4) of an axiom schema or an inference rule.
5. Each proof can be “checked” manually or verified by machine implementable algorithms.

Requirements of Proof Systems

Syntactic. Proof systems are purely syntactic and no use is made of semantics in any proof.

Finitary. All axioms, axiom-schemas and rules of inference must be expressible in a finitary manner.

Decidability. The correctness of any application of a rule of inference must be machine-verifiable.

Soundness. The system must allow the deduction of only valid conclusions from the assumptions.

Completeness. The system must allow all valid truths to be deduced.

The semantics may be used to prove only the soundness and completeness of the proof system.

Proof Systems: Desiderata

There are two **conflicting** desirable properties of proof systems.

Minimality. Inspired by Euclid and the controversy over the parallel postulate. *Is there a minimal set of axioms and inference rules from which all truths and only truths may be deduced?* (Hilbert's system)

Naturalness. *Is there a natural intuitive set of axioms and rules from which all truths and only truths may be deduced?*. (Gentzen's natural deduction)

Formal Theories

Definition 13.1: Formal theories and proof systems

A formal theory $\mathbb{T} = \langle \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ consists of

Formal Language a formal language \mathcal{L} .

Axioms a subset \mathcal{A} of the language \mathcal{L} .

Inference Rules a set \mathcal{R} of inference rules.

The Axioms and Inference rules together constitute a **proof system** for the theory.

Formal Language

1. An alphabet $\Sigma = X \cup \Omega \cup \{(,)\}$ consisting of a set X of *variables* a set Ω of *connectives* each with a pre-defined arity and grouping symbols.
2. The **well-formed formulas** or wffs of \mathcal{L} are defined inductively on the alphabet.
3. Membership of strings (from the alphabet) in \mathcal{L} is **decidable** i.e. there exists an algorithm to decide whether a given string is a well-formed formula

Formal Theories

Axioms and Inference Rules

Axioms A **decidable** subset of \mathcal{L} .

Inference Rules A finite set of rules.

1. Each rule R of arity $m \geq 0$ is a **decidable relation** $R \subseteq \mathcal{L}^m \times \mathcal{L}$ i.e. there exists an algorithm which for any $\phi_1, \dots, \phi_m, \psi$ can determine whether $((\phi_1, \dots, \phi_m), \psi) \in R$
2. For each $((\phi_1, \dots, \phi_m), \psi) \in R$, ϕ_1, \dots, ϕ_m are called the **premises** and ψ a **direct consequence** by virtue of R .
3. Each such rule is presented in the form $R. \frac{X_1 \cdots X_m}{Y}$ where the variables X_1, \dots, X_m, Y are the “shapes” of the formulae allowed by the rule.
4. If $m = 0$, R is called an **axiom schema**

Axiomatic Theories

Definition 13.2: Axiomatic theories

- The axioms and rules of inference of a formal theory together constitute a **proof system** for the set of wffs in the theory.
- A formal theory is said to be **axiomatic** if there exists an algorithm to decide whether a given wff is an axiom.

Syntax and Decidability

1. The purely syntactic nature of a formal theory and all the decidability constraints usually means that each axiom and rule of inference is expressed in terms of syntactic patterns obeying certain “shape” constraints.
2. Each application of an axiom (schema) or inference rule requires pattern-matching and syntactic substitution (see Section 0).
3. The notion of a deduction is not only syntactic but is verifiable by an algorithm given the nature of the formal theory.
4. Further the deductions of a formal theory can be generated by an algorithm (the set of “theorems” is *recursively enumerable*).

A Hilbert-style Proof System

Definition 13.3: Hilbert-style proof system for \mathcal{L}_0

\mathcal{H}_0 , the Hilbert-style proof system for Propositional logic consists of

- The set \mathcal{L}_0 generated from A and $\{\neg, \rightarrow\}$
- The following three axiom schemas

S.

$$\frac{}{(X \rightarrow (Y \rightarrow Z)) \rightarrow ((X \rightarrow Y) \rightarrow (X \rightarrow Z))}$$

K.

$$\frac{}{X \rightarrow (Y \rightarrow X)}$$

N.

$$\frac{}{(\neg Y \rightarrow \neg X) \rightarrow ((\neg Y \rightarrow X) \rightarrow Y)}$$

- A single rule of inference *modus ponens*

MP.

$$\frac{X \rightarrow Y, X}{Y}$$

Axiom Schemas and Rules

Each axiom schema represents an infinite decidable subset of \mathcal{L}_0 while the rule represents an infinite decidable relation on \mathcal{L}_0 .

K. $\mathcal{K}_0 = \{\phi \rightarrow (\psi \rightarrow \phi) \mid \phi, \psi \in \mathcal{L}_0\}$

S. $\mathcal{S}_0 = \{(\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi)) \mid \phi, \psi, \chi \in \mathcal{L}_0\}$

N. $\mathcal{N}_0 = \{(\neg\psi \rightarrow \neg\phi) \rightarrow ((\neg\psi \rightarrow \phi) \rightarrow \psi) \mid \phi, \psi \in \mathcal{L}_0\}$

MP. $\mathcal{MP}_0 = \{\langle\langle\phi \rightarrow \psi, \phi\rangle, \psi\rangle \mid \phi, \psi \in \mathcal{L}_0\}$

Rule Patterns

1. The axiom schema **K** states that for all (simultaneous) substitutions (see definitions **0.1** and **0.4**) $\{\phi/X, \psi/Y\}$ the formulae $\phi \rightarrow (\psi \rightarrow \phi)$ are all axioms of the system.
2. Similarly with **S** and **N**.
3. The rules specify patterns and shapes of formulae. Thus *modus ponens* specifies the relation

$$\text{MP} = \{((\phi \rightarrow \psi, \phi), \psi) \mid \phi, \psi \in \mathcal{L}_0\}$$

and thus asserts that ψ is a direct consequence of $\phi \rightarrow \psi$ and ϕ for all formulae ϕ and ψ .

4. An application of the rule consists of identifying appropriate substitutions (section **0**) of the variables **X** and **Y** by formulae in \mathcal{L}_0 to yield a direct consequence by the same substitution.

14. Proof Theory: Hilbert-style

14: Proof Theory: Hilbert-style

1. More About Formal Theories
2. The Law of The Excluded Middle
3. An Example Proof:1
4. An Example Proof:2
5. An Example Proof:3
6. An Example Proof:4
7. An Example Proof:5
8. Formal Proofs: 1
9. Formal Proofs: 2
10. Provability and Formal Proofs
11. The Deduction Theorem
12. About Formal Proofs

More About Formal Theories

We use the symbol “ $\vdash_{\mathcal{H}_0}$ ” to denote provability using the proof system \mathcal{H}_0 . $\Gamma \vdash_{\mathcal{H}_0} \psi$ denotes that ψ is formally provable from the set Γ in the proof system \mathcal{H}_0 . The subscript is omitted whenever the proof system is clear from the context.

The following properties follow easily from the definition of a **formal theory**.

Theorem 14.1: Formal theories: basic properties

Let Γ and Δ be finite sets of wffs in a theory \mathbb{T} .

Monotonicity If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \psi$, then $\Delta \vdash \psi$.

Compactness $\Delta \vdash \psi$ if and only if there is a finite (possibly empty) subset $\Gamma \subseteq \Delta$ such that $\Gamma \vdash \psi$.

Substitututivity If $\Delta \vdash \psi$ and for each $\phi \in \Delta$, $\Gamma \vdash \phi$, then $\Gamma \vdash \psi$.

The Law of The Excluded Middle

We formally deduce $\phi \rightarrow \phi$ for any formula ϕ using the proof system \mathcal{H}_0 . This formula is also called “the law of the excluded middle” and is an important one which distinguishes “classical logic” (which is what this course is all about) and what are known as “intuitionistic logics”. The formula is **logically equivalent** to the formula $\neg\phi \vee \phi$. It is also a reflexivity property of the conditional.

An Example Proof:1

As is normal practice in mathematics the proof is presented as a sequence of steps.

$$1. \phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi) \quad \{\phi/X, \phi \rightarrow \phi/Y\}K$$

where each step is justified as an instance of an axiom schema or a rule.

An Example Proof:2

As is normal practice in mathematics the proof is presented as a sequence of steps.

1. $\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)$ $\{\phi/X, \phi \rightarrow \phi/Y\}K$
2. $(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$
 $\{\phi/X, \phi \rightarrow \phi/Y, \phi/Z\}S$

where each step is justified as an instance of an axiom schema or a rule.

An Example Proof:3

As is normal practice in mathematics the proof is presented as a sequence of steps.

1. $\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)$ $\{\phi/X, \phi \rightarrow \phi/Y\}K$
2. $(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$
 $\{\phi/X, \phi \rightarrow \phi/Y, \phi/Z\}S$
3. $((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$ $\{2, 1\}MP$

where each step is justified as an instance of an axiom schema or a rule.

An Example Proof:4

As is normal practice in mathematics the proof is presented as a sequence of steps.

1. $\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)$ $\{\phi/X, \phi \rightarrow \phi/Y\}K$
2. $(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$
 $\{\phi/X, \phi \rightarrow \phi/Y, \phi/Z\}S$
3. $((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$ $\{2, 1\}MP$
4. $(\phi \rightarrow (\phi \rightarrow \phi))$ $\{\phi/X, \phi/Y\}K$

where each step is justified as an instance of an axiom schema or a rule.

An Example Proof:5

As is normal practice in mathematics the proof is presented as a sequence of steps.

1. $\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)$ $\{\phi/X, \phi \rightarrow \phi/Y\}K$
2. $(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$
 $\{\phi/X, \phi \rightarrow \phi/Y, \phi/Z\}S$
3. $((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$ $\{2, 1\}MP$
4. $(\phi \rightarrow (\phi \rightarrow \phi))$ $\{\phi/X, \phi/Y\}K$
5. $\phi \rightarrow \phi$ $\{3, 4\}MP$

where each step is justified as an instance of an axiom schema or a rule.

However the proof is better written out as an “upside-down” *proof tree* where

1. each node is a formula. The leaves are axioms (or empty in the case of axiom schemas).
2. each internal node is an application of a rule of appropriate arity applied to the appropriate target (definition -2.2) nodes in the tree.
3. The line-segments between the various levels on the tree show how a node depends on the nodes in the immediately “succeeding” level.
4. the root node is the final formula to be proven.
5. The labels on each line-segment separating a direct consequence from its premise(s) also provide the justification.

The **same proof** may then be rendered as follows:

$$\begin{array}{c}
 1 \frac{}{\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)} K \\
 2 \frac{}{(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))} S \\
 3 \frac{1 \quad 2}{((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))} MP \\
 4 \frac{}{(\phi \rightarrow (\phi \rightarrow \phi))} K \\
 5 \frac{3 \quad 4}{\phi \rightarrow \phi} MP
 \end{array}$$

where the justifications of each step are stated on the right side of each application.

1. $\{\phi/X, \phi \rightarrow \phi/Y\}K$
2. $\{\phi/X, \phi \rightarrow \phi/Y, \phi/Z\}S$
3. $\{2, 1\}MP$
4. $\{\phi/X, \phi/Y\}K$
5. $\{3, 4\}MP$

Each node in this proof tree is said to be an **instance** of a rule or an axiom-schema. The horizontal line separates the premise(s) from the conclusion in each application of a rule or axiom (schema).

Formal Proofs: 1

Definition 14.1: Formal proof

A **formal proof** of a formula ϕ from a finite set Γ of formulae is a *finite* tree of formulae

- rooted at the formula ϕ ,
- the leaves are axioms or instances of axiom schemas or members from Γ .
- each non-leaf node is a direct consequence of one or more nodes at the “succeeding” level by virtue of application of a rule of inference of the appropriate arity.

Formal Proofs: 2

Definition 14.2: Formal provability and formal theorems

- ϕ is said to be (formally) provable (or (formally) deducible) from Γ in the proof system \mathcal{H}_0 if there exists a formal proof of ϕ in the system \mathcal{H}_0 .
- ϕ is a (formal) theorem if $\Gamma = \emptyset$ and is denoted $\vdash_{\mathcal{H}_0} \phi$

Notation 14.1: Deducibility

- $\Gamma \vdash_{\mathcal{H}_0} \phi$ (or simply $\Gamma \vdash \phi$ when \mathcal{H}_0 is understood) denotes that ϕ is provable in the system \mathcal{H}_0 from the assumptions Γ .
- $\vdash_{\mathcal{H}_0} \phi$ (or simply $\vdash \phi$ when \mathcal{H}_0 is understood) denotes that ϕ is a formal theorem of \mathcal{H}_0 .

Provability and Formal Proofs

Fact 14.1: Provability

Given any theory $\mathbb{T} = \langle \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ and a wff $\psi \in \mathcal{L}$,

1. If ψ is an axiom or instance of an axiom-schema then $\vdash \psi$ and hence ψ is a formal theorem.
2. If $\vdash \psi$ then all the leaf nodes in any proof tree of ψ are either axioms or instances of axiom-schemas.
3. If ψ is an axiom or an instance of an axiom-schema then $\Gamma \vdash \psi$ for any $\Gamma \subseteq \mathcal{L}$.
4. For any $\phi \in \Gamma$, $\Gamma \vdash \phi$.

The Deduction Theorem

Notation 14.2

Given a set Γ and a formula ϕ , " $\Gamma, \phi \vdash \psi$ " denotes " $\Gamma \cup \{\phi\} \vdash \psi$ ".

Theorem 14.2: The Deduction Theorem

For all $\Gamma \subseteq_f \mathcal{L}_0$ and formulae ϕ and ψ , $\Gamma, \phi \vdash \psi$ if and only if $\Gamma \vdash \phi \rightarrow \psi$.



The Deduction theorem justifies our usual notion of a direct proof from the hypotheses of a conditional conclusion – the *antecedent* of the conditional is added to the assumptions and the *consequent* is proven.

Proof of The Deduction Theorem (theorem 14.2)

Proof: (\Leftarrow). Assume $\Gamma \vdash \phi \rightarrow \psi$. Let \mathcal{T} be a formal proof tree rooted at $\phi \rightarrow \psi$ with m nodes for some $m > 0$. By monotonicity (theorem 14.1) $\Gamma, \phi \vdash \phi \rightarrow \psi$ is proven by the same tree (with the extra assumption ϕ). We may extend \mathcal{T} to the tree \mathcal{T}' by adding a new $(m + 1)$ -st leaf node ϕ and creating the $(m + 2)$ -nd root node ψ .

$$\frac{m \frac{\nwarrow \mathcal{T} \nearrow}{\phi \rightarrow \psi} \quad m+1 \frac{\phi}{\psi}}{m+2 \frac{\psi}{\psi}} \text{MP}$$

\mathcal{T}' is a proof of $\Gamma, \phi \vdash \psi$.

(\Rightarrow). Assume $\Gamma, \phi \vdash \psi$. Then there exists a proof tree \mathcal{T} rooted at ψ with nodes $\psi_1, \dots, \psi_m \equiv \psi$ such that every leaf node has a smaller index than a non-leaf node and every non-leaf node has a larger index than any of its children. It is clear that ψ_1 will be a leaf node and ψ_m (having the highest index) is the root of the proof tree.

Then the following stronger claim proves the required result.

Claim. $\Gamma \vdash \phi \rightarrow \psi_i$ for all i , $1 \leq i \leq m$.

\vdash By induction on the structure of the proof tree \mathcal{T}

Basis. ψ_i is a leaf node. Then ψ_i is either a premise or an axiom. We have the following cases to consider.

Case $\psi_i \equiv \phi$. Then the claim follows from the law of the Excluded Middle and monotonicity (theorem 14.1).

Case $\psi_i \in \Gamma$ or ψ_i is an axiom. In either case there exists a subtree \mathcal{T}_i (of \mathcal{T}) rooted at ψ_i which may be used to construct the tree \mathcal{T}'_i as follows. Assume there are i' steps in the proof of ψ_i .

$$\frac{i' \frac{\nwarrow \mathcal{T}_i \nearrow}{\psi_i} \quad i'+1 \frac{\psi_i \rightarrow (\phi \rightarrow \psi_i)}{\phi \rightarrow \psi_i}}{i'+2 \frac{\phi \rightarrow \psi_i}{\phi \rightarrow \psi_i}} \text{K MP}$$

which proves the claim.

Induction Hypothesis (IH).

$$\Gamma \vdash \phi \rightarrow \psi_i \text{ for all } i \text{ such that } 1 \leq i < l \text{ for some index } l \leq m.$$

Induction Step. If ψ_l is a leaf node then the proof is just as in the basis. Hence assume it is not a leaf node. Since ψ_l is a non-leaf node it is neither an axiom nor a premise and must have been obtained by virtue of the rule **MP** applied to its immediate children say ψ_i and ψ_j with $i \neq j$ such that $i, j < l$. Without loss of generality we may assume $\psi_j \equiv \psi_i \rightarrow \psi_l$.

By the induction hypothesis, we know $\Gamma \vdash \phi \rightarrow \psi_i$ and $\Gamma \vdash \phi \rightarrow \psi_j$. Hence for some $i', j' > 0$, there exist proof trees \mathcal{T}'_i of i' nodes rooted at $\phi \rightarrow \psi_i$ and \mathcal{T}'_j of j' nodes rooted at $\phi \rightarrow \psi_j \equiv \phi \rightarrow (\psi_i \rightarrow \psi_l)$ respectively. We construct the tree \mathcal{T}'_l rooted at $\phi \rightarrow \psi_l$ from \mathcal{T}'_i and \mathcal{T}'_j as follows.

$$\frac{j' + 2 \frac{j' \frac{\nwarrow \mathcal{T}'_j \nearrow}{\phi \rightarrow (\psi_i \rightarrow \psi_l)}}{(\phi \rightarrow (\psi_i \rightarrow \psi_l)) \rightarrow ((\phi \rightarrow \psi_i) \rightarrow (\phi \rightarrow \psi_l))} \text{S}}{j' + i' + 3 \frac{(\phi \rightarrow \psi_i) \rightarrow (\phi \rightarrow \psi_l)}{\phi \rightarrow \psi_l}} \text{MP}$$

where $j' + 1$ is an instance of **S**, and $j' + 2$ and $j' + i' + 3$ are both applications of **MP** to their respective children in the tree.

⊣

QED

About Formal Proofs

- Since a formal proof is a tree, it is acyclic (i.e. there is no circularity in the proof).
- Every formal proof is a *finite* tree i.e. a proof is a finitary object.

Questions.

1. *What if the set of assumptions is infinite?*
2. *Are there statements which have only infinite proofs and no finite proof?*

15. Derived Rules

15: Derived Rules

1. Simplifying Proofs
2. Derived Rules
3. The Sequent Form
4. Proof trees in sequent form
5. Transitivity of Conditional
6. Derived Double Negation Rules
7. Derived Operators
8. Rules for Derived Operators
9. Gentzen's System
10. Natural Deduction: 1
11. Natural Deduction: 2
12. Natural Deduction: 3
13. Natural Deduction: 4
14. Natural Deduction: 5

Simplifying Proofs

- The deduction theorem allows “*movement*” of formulae between the set (sequence) of assumptions and the formula to be proven.
- Hence the set (sequence) of formulae which form the assumptions is an important part of the proof.
- We use the notion of a *sequent* to formalize this *movement* which may take place at any stage.

Definition 15.1: Sequent

A **sequent** is a meta-formula of the form $\Gamma \vdash \phi^a$.

^aIn the literature the term sequent refers to “ $\Gamma \Rightarrow \Delta$ ” (intuitively representing $\wedge \Gamma \rightarrow \vee \Delta$) where Γ and Δ are finite sets of of formulae. However we will not pursue this approach here.

Derived Rules

- By **substitutivity** (theorem 14.1) we may simplify our proofs by incorporating theorems and meta-theorems as *derived rules* of our proof system.
- These **rules** may be presented in sequent (definition 15.1) form to specify the exact assumptions under which a formulae is claimed to have been deduced.
- The proof of the **reflexivity of the conditional** may be rendered in sequent form by simply pre-pending each node in the tree with “ \vdash ”.
- **Reflexivity** may be expressed in sequent form as a **derived rule**.
- The **Deduction Theorem** and its converse may be rendered in sequent form as derived rules $DT \Leftarrow$ and $DT \Rightarrow$ respectively.
- These derived rules may be directly invoked in later proofs.

The Sequent Form

Let Γ be a set (sequence) of formulae.

$$\text{K. } \frac{}{\Gamma \vdash X \rightarrow (Y \rightarrow X)}$$

$$\text{N. } \frac{}{\Gamma \vdash (\neg Y \rightarrow \neg X) \rightarrow ((\neg Y \rightarrow X) \rightarrow Y)}$$

$$\text{S. } \frac{}{\Gamma \vdash (X \rightarrow (Y \rightarrow Z)) \rightarrow ((X \rightarrow Y) \rightarrow (X \rightarrow Z))}$$

$$\text{MP. } \frac{\begin{array}{c} \Gamma \vdash X \rightarrow Y \\ \Gamma \vdash X \end{array}}{\Gamma \vdash Y}$$

$$\text{R} \rightarrow . \frac{}{\Gamma \vdash X \rightarrow X}$$

$$\text{DT} \Leftarrow . \frac{\Gamma \vdash X \rightarrow Y}{\Gamma, X \vdash Y}$$

$$\text{DT} \Rightarrow . \frac{\Gamma, X \vdash Y}{\Gamma \vdash X \rightarrow Y}$$

Notation 15.1

Since we will be presenting most formal proofs in the form of proof trees (see 14), it is no longer necessary to number the steps as was done in the [previous proofs](#). Instead only justifications in the form of references to rules will be presented (wherever necessary).

Proof trees in sequent form

Theorem 15.1: Transitivity-Conditional

For all ϕ , ψ and χ ,

$$\vdash (\phi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi))$$

Proof: Let $\Gamma_1 = \phi \rightarrow \psi$, $\Gamma_2 = \Gamma_1, \psi \rightarrow \chi$ and $\Gamma_3 = \Gamma_2, \phi$

$$\begin{array}{c}
 \text{MP} \frac{\Gamma_3 \vdash \phi \rightarrow \psi \quad \Gamma_3 \vdash \phi}{\Gamma_3 \vdash \psi} \quad \Gamma_3 \vdash \psi \rightarrow \chi \\
 \text{MP} \frac{}{\Gamma_3 \vdash \psi} \quad \Gamma_3 \vdash \psi \rightarrow \chi \\
 \text{DT} \Rightarrow \frac{\Gamma_2, \phi \vdash \chi}{\Gamma_2 \vdash \phi \rightarrow \chi} \\
 \text{DT} \Rightarrow \frac{\Gamma_2 \vdash \phi \rightarrow \chi}{\Gamma_1 \vdash (\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi)} \\
 \text{DT} \Rightarrow \frac{\Gamma_1 \vdash (\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi)}{\vdash (\phi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi))}
 \end{array}$$

QED

Transitivity of Conditional

From theorem 15.1 we get a derived axiom schema

$$T \rightarrow . \frac{}{\Gamma \vdash (X \rightarrow Y) \rightarrow ((Y \rightarrow Z) \rightarrow (X \rightarrow Z))}$$

But equivalently by applying the derived rule $DT \Leftarrow$ to $T \rightarrow$ above we also get a derived rule of inference which is often more convenient to use.

$$T \Rightarrow . \frac{\begin{array}{c} \Gamma \vdash X \rightarrow Y \\ \Gamma \vdash Y \rightarrow Z \end{array}}{\Gamma \vdash X \rightarrow Z}$$

Exercise 15.1: Hilbert-style proofs

1. Prove that each of the axiom schemas in \mathcal{H}_0 represents a collection of tautologies.
2. Prove that *Modus Ponens in sequent form* preserves logical consequence i.e. if $\Gamma \models \phi \rightarrow \psi$ and $\Gamma \models \phi$ then $\Gamma \models \psi$.
3. Using the above prove that the proof system \mathcal{H}_0 is *sound* i.e. If $\Gamma \vdash_{\mathcal{H}_0} \psi$ then $\Gamma \models \psi$.
4. Find the fallacy in the following proof of theorem 15.1. Assume Γ_1 , Γ_2 and Γ_3 are as in the proof of theorem 15.1.

$$\begin{array}{c}
 \text{DT} \Rightarrow \frac{\Gamma_3 \vdash \psi \rightarrow \chi}{\Gamma_2 \vdash \phi \rightarrow (\psi \rightarrow \chi)} \quad \text{MP} \frac{\text{S} \quad \frac{\Gamma_2 \vdash (\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi))}{\Gamma_2 \vdash (\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow (\phi \rightarrow \chi)} \quad \Gamma_2 \vdash \phi \rightarrow \psi}{\Gamma_2 \vdash (\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow (\phi \rightarrow \chi)} \\
 \text{MP} \frac{\text{DT} \Rightarrow \frac{\Gamma_2 \vdash \phi \rightarrow \chi}{\Gamma_1 \vdash (\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi)}}{\vdash (\phi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\phi \rightarrow \chi))}
 \end{array}$$

5. Prove the following derived rule of inference (You may use any of the rules of inference derived earlier in addition to the usual proof rules).

$$\boxed{\text{R2} \Rightarrow . \quad \frac{\begin{array}{c} \Gamma \vdash X \rightarrow (Y \rightarrow Z) \\ \Gamma \vdash Y \end{array}}{\Gamma \vdash X \rightarrow Z}}$$

6. Could we have consequently reordered our theorems by first proving $\text{R2} \Rightarrow$ and then proving $\text{T} \rightarrow ?$ Discuss whether there is anything fallacious in this approach.

Derived Double Negation Rules

$$\text{DNE. } \frac{\Gamma \vdash \neg\neg X}{\Gamma \vdash X}$$

$$\text{DNI. } \frac{\Gamma \vdash X}{\Gamma \vdash \neg\neg X}$$

The following proof trees yield proofs of the derived double negation **elimination** and **introduction** rules respectively.

Alternatively we may regard them as derived axiom schemas

$$\text{DNE} \rightarrow . \frac{}{\Gamma \vdash \neg\neg X \rightarrow X}$$

$$\text{DNI} \rightarrow . \frac{}{\Gamma \vdash X \rightarrow \neg\neg X}$$

Proof of derived rule DNE and axiom schema DNE \rightarrow

Proof:

$$\begin{array}{c}
 \frac{\text{K} \quad \frac{\neg\neg\phi \rightarrow (\neg\phi \rightarrow \neg\neg\phi)}{\text{T} \Rightarrow \neg\neg\phi \rightarrow (\neg\phi \rightarrow \neg\neg\phi)}}{\text{R2} \Rightarrow \frac{\frac{\text{N} \quad \frac{(\neg\phi \rightarrow \neg\neg\phi) \rightarrow ((\neg\phi \rightarrow \neg\phi) \rightarrow \phi)}{(\neg\phi \rightarrow \neg\neg\phi) \rightarrow \phi}}{\text{R} \Rightarrow \frac{\neg\phi \rightarrow \neg\phi}{\neg\phi \rightarrow \neg\neg\phi \rightarrow \phi}}}{\frac{\neg\neg\phi \rightarrow \phi}{\neg\neg\phi \vdash \phi}}}
 \end{array}$$

QED

Proof of derived rule DNI and axiom schema DNI \rightarrow

Proof:

$$\begin{array}{c}
 \frac{\text{K} \quad \frac{\phi \rightarrow (\neg\neg\phi \rightarrow \phi)}{\text{T} \Rightarrow \phi \rightarrow (\neg\neg\phi \rightarrow \phi)}}{\text{MP} \quad \frac{\text{N} \quad \frac{(\neg\neg\neg\phi \rightarrow \neg\phi) \rightarrow ((\neg\neg\neg\phi \rightarrow \phi) \rightarrow \neg\phi)}{(\neg\neg\neg\phi \rightarrow \phi) \rightarrow \neg\phi}}{\text{DNE} \quad \frac{\neg\neg\neg\phi \rightarrow \neg\phi}{\neg\neg\neg\phi \rightarrow \neg\neg\phi}}}
 \end{array}$$

QED

Exercise 15.2: Hilbert's system with derived rules

1. Give a formal proof of Peirce's law $((\phi \rightarrow \psi) \rightarrow \phi) \rightarrow \phi$.
2. Prove the axiom schema $\boxed{N'. \quad \frac{(\neg Y \rightarrow \neg X) \rightarrow (X \rightarrow Y)}{}}$ A deduction theorem variant of this schema is also called the *modus tollens* rule or the contrapositive rule.
3. Prove the following axiom schemas in \mathcal{H}_0 . In each case you are allowed to use any version of the theorems previously proven.

(a) $\boxed{\perp. \quad \frac{\neg X \rightarrow (X \rightarrow Y)}{}}$ What can you conclude about the system \mathcal{H}_0 from your proof?

(b) $\boxed{N''. \quad \frac{(X \rightarrow Y) \rightarrow (\neg Y \rightarrow \neg X)}{}}$

(c) $\boxed{N2. \quad \frac{X \rightarrow (\neg Y \rightarrow \neg(X \rightarrow Y))}{}}$

(d) $\boxed{C. \quad \frac{(X \rightarrow Y) \rightarrow ((\neg X \rightarrow Y) \rightarrow Y)}{}}$ Derive the proof by cases rule $\boxed{\text{Cases. } \frac{\Gamma, X \vdash Y \quad \Gamma, \neg X \vdash Y}{\Gamma \vdash Y}}$

(e) Derive the *proof by contradiction* also called the *indirect proof method* rule I in the system \mathcal{H}_0 .

$$\boxed{I. \quad \frac{\Gamma, X \vdash \neg Y \quad \Gamma, X \vdash Y}{\Gamma \vdash \neg X}}$$

4. Prove the derived axiom $\boxed{C2. \quad \frac{\Gamma \vdash (\neg X \rightarrow X) \rightarrow X}{}}$

Exercise 15.3: Variant of \mathcal{H}_0

1. A variant of the system \mathcal{H}_0 is the system \mathcal{H}'_0 obtained by replacing the schema **N** by **N'**.
 - (a) Prove the axiom schema **N** in the system \mathcal{H}'_0 .
 - (b) Prove the double negation rules **DNE** and **DNI** in \mathcal{H}'_0 .

Derived Operators

$$\top \stackrel{df}{=} X \rightarrow X$$

$$\perp \stackrel{df}{=} \neg(X \rightarrow X)$$

$$X \vee Y \stackrel{df}{=} \neg X \rightarrow Y$$

$$X \wedge Y \stackrel{df}{=} \neg(X \rightarrow \neg Y)$$

$$X \leftrightarrow Y \stackrel{df}{=} \neg((X \rightarrow Y) \rightarrow \neg(Y \rightarrow X))$$

Several other binary and other operators of varying arities may be defined.

Rules for Derived Operators

Corresponding to each derived operator defined as $O(X_1, \dots, X_n) \stackrel{df}{=} \omega(X_1, \dots, X_n)$ where ω is constructed only from the set $\{\neg, \rightarrow\}$ we have the introduction and elimination rules.

$$\text{OE. } \frac{\Gamma \vdash O(X_1, \dots, X_n)}{\Gamma \vdash \omega(X_1, \dots, X_n)}$$

$$\text{OI. } \frac{\Gamma \vdash \omega(X_1, \dots, X_n)}{\Gamma \vdash O(X_1, \dots, X_n)}$$

Gentzen's System

Minimal vs. Natural

- Gentzen's Natural Deduction system is not a minimal system, instead it is somewhat more natural in the sense that it has explicit *introduction* and *elimination* rules for each operator.
- Further there is some *redundancy* in the rules. So different proofs are possible using different rules.
- Not all the rules may actually be useful, but they possess a pleasing symmetry.
- However, it is necessary to prove both the soundness and the completeness of the system.
- We present a **sequent** version of the system which we refer to as \mathcal{G}_0 .

Natural Deduction: 1

	Introduction	Elimination
\perp	$\perp I.$ $\frac{\Gamma \vdash X \wedge \neg X}{\Gamma \vdash \perp}$	$\perp E.$ $\frac{\Gamma \vdash \perp}{\Gamma \vdash X}$
\top	$\top I.$ $\frac{}{\Gamma \vdash \top}$	$\top E.$ $\frac{\Gamma \vdash \top}{\Gamma \vdash X \vee \neg X}$

Natural Deduction: 2

	Introduction	Elimination
\neg	$\neg I.$ $\frac{\Gamma, X \vdash \perp}{\Gamma \vdash \neg X}$	$\neg E.$ $\frac{\Gamma, \neg X \vdash \perp}{\Gamma \vdash X}$
$\neg\neg$	$\neg\neg I.$ $\frac{\Gamma \vdash X}{\Gamma \vdash \neg\neg X}$	$\neg\neg E.$ $\frac{\Gamma \vdash \neg\neg X}{\Gamma \vdash X}$

Natural Deduction: 3

	Introduction	Elimination
\vee	$\vee I 1. \frac{\Gamma \vdash X}{\Gamma \vdash X \vee Y}$ $\vee I 2. \frac{\Gamma \vdash Y}{\Gamma \vdash X \vee Y}$	$\vee E. \frac{\Gamma \vdash X \vee Y}{\Gamma \vdash Z}$

Natural Deduction: 4

	Introduction	Elimination	
\wedge	$\wedge I. \frac{\Gamma \vdash X \quad \Gamma \vdash Y}{\Gamma \vdash X \wedge Y}$	$\wedge E1. \frac{\Gamma \vdash X \wedge Y}{\Gamma \vdash X}$	$\wedge E2. \frac{\Gamma \vdash X \wedge Y}{\Gamma \vdash Y}$

Natural Deduction: 5

	Introduction	Elimination	
→	→ I. $\frac{\Gamma, X \vdash Y}{\Gamma \vdash X \rightarrow Y}$	→ E. $\frac{\Gamma \vdash X}{\Gamma \vdash Y}$	
↔	↔ I. $\frac{\Gamma \vdash X \rightarrow Y \quad \Gamma \vdash Y \rightarrow X}{\Gamma \vdash X \leftrightarrow Y}$	↔ E1. $\frac{\Gamma \vdash X \leftrightarrow Y}{\Gamma \vdash X \rightarrow Y}$	↔ E2. $\frac{\Gamma \vdash X \leftrightarrow Y}{\Gamma \vdash Y \rightarrow X}$

Exercise 15.4: Hilbert and Gentzen

1. Prove the logical equivalences of \mathcal{P}_0 using the system \mathcal{H}_0 .
2. Prove the non-obvious logical equivalences of \mathcal{P}_0 in the system \mathcal{G}_0 .
3. Derive each of the rules of \mathcal{G}_0 from the system \mathcal{H}_0 . You may use the rules OE and OI as and when needed for each operator.
4. Derive the axiom schemas K, S and N in \mathcal{G}_0 .

16. The Hilbert System: Soundness

16: The Hilbert System: Soundness

1. Formal Theory: Issues
2. Soundness of Formal Theories
3. Soundness of the Hilbert System
4. Soundness of the Hilbert System

Formal Theory: Issues

The major questions concerning any formal theory are two-fold:

Soundness. Is the theory sound? Especially considering that we may not have well-defined models in which to test the truth or falsehood of statements.

Completeness. Is the theory complete? That is, is every fact provable from the axioms and inference rules of the theory?

16.1. Soundness

In exercise 15.1 problem 3 we gave a definition of soundness of a proof system which more or less agrees wth our intuition that any proof system should only derive as true those formulae which are semantically true and consistent with the assumptions, i.e. proof systems should somehow be totally compatible with our semantics and should not allow for inconsistency in any form *unless the assumptions themselves are inconsistent*.

Question 1. *Is it possible that every formula in the language \mathcal{L}_0 has a formal proof in the system \mathcal{H}_0 ?*

If the answer to the above question is “yes” then it means that for every formula ψ there exist formal proofs of both ψ and $\neg\psi$. In particular this means that even for each atom p , both p and $\neg p$ are provable. In the particular case of \mathcal{H}_0 (see problems 1 and 2 in exercise 15.1) it would mean that both ψ and $\neg\psi$ are tautologies. If this were the case then clearly our proof theory is unsound in the sense that it does not reflect the semantics of the language \mathcal{L}_0 . We do know that there exist contingent and unsatisfiable formulae in the language. Hence any proof system which allows every formula to be provable is necessarily unsound.

Question 2. *Let \mathcal{S} be any proof system for \mathcal{P}_0 and consider the set $T = \{\psi \mid \vdash_{\mathcal{S}} \psi\}$ of all theorems provable in the proof system \mathcal{S} and suppose $T \neq \mathcal{P}_0$. Is it possible that there exists at least one formula ψ such that both $\vdash_{\mathcal{S}} \psi$ and $\vdash_{\mathcal{S}} \neg\psi$?*

Clearly if the answer to the above quesiton is “yes” in the system \mathcal{H}_0 then it follows by two applications of MP from the derived rule \perp or from rule I (see exercise 15.2) that for every formula χ , both χ and $\neg\chi$ are also theorems of \mathcal{H}_0 . This implies that $T = \mathcal{P}_0$ contradicting the assumption $T \neq \mathcal{P}_0$. In fact any proof system \mathcal{S} that allows (derived) rules of the form \perp or I would not be sound unless there is at least one formula that is not provable. Hence if a proof system is sound

then not every formula would be provable in the system.

Question 3. *What do we need to show in order to be convinced that we have a sound proof system?*

We need to show that each axiom is true (that is no false formulae can be proven from the each axiom). Further we need to show that each proof rule preserves truth i.e. proof rules do not allow the derivation of contradictory formulae unless there is a contradiction in the assumptions.

In fact we will see that the only formal theorems of \mathcal{H}_0 are the tautologies of \mathcal{P}_0 .

Soundness of Formal Theories

Given that the proof theory may be the only finitary tool available to us in reasoning about some domain we need to define the notion of consistency of the theory in terms of the proof-theoretic notions.

Definition 16.1: Soundness

A **formal theory** with a proof system \mathcal{S} is **sound** if every formal theorem is logically valid (i.e. for every formula ϕ , $\vdash_{\mathcal{S}} \phi$ implies $\models \phi$). Otherwise it is said to be **unsound**.

Points to ponder:

- If Γ is an infinite set then any proof of $\Gamma \vdash \psi$ would be a **finite tree** requiring only a finite subset $\Gamma_f \subseteq_f \Gamma$ of assumptions $(\Gamma_f \vdash_{\mathcal{H}_0} \psi)$
- By **corollary 11.5** every logical consequence is deducible from a finite subset of assumptions.
- Our proof system \mathcal{H}_0 (or even \mathcal{G}_0) is sound if it allows us to derive only conclusions which *preserve truth* under all truth assignments i.e. $\Gamma \vdash_{\mathcal{H}_0} \psi$ must imply that ψ is a **logical consequence** of the assumptions Γ .
- By theorem **4.1** every valid argument in propositional logic can be represented by a tautology.

Soundness of the Hilbert System

Lemma 16.1: Soundness of axiom schemas

1. Every instance of every axiom schema in \mathcal{H}_0 is a tautology.
2. The *Modus Ponens* rule MP preserves tautologousness i.e if ϕ and $\phi \rightarrow \psi$ are tautologies then so is ψ .

QED

A truth table technique would serve the purpose for \mathcal{H}_0 alone but would not be possible when \mathcal{H}_0 is extended to \mathcal{H}_1 . We therefore we give a proof more in the tradition of mathematical proofs.

Proof of lemma 16.1

Proof:

1. We prove the case of any instance of the axiom schema K. We need to show that for all ϕ and ψ , $\phi \rightarrow (\psi \rightarrow \phi)$ is a tautology. Suppose it is not a tautology. Then there exists a truth assignment τ such that $\mathcal{T}[\phi \rightarrow (\psi \rightarrow \phi)]_{\tau} = 0$ which is possible only if $\mathcal{T}[\phi]_{\tau} = 1$ and $\mathcal{T}[\psi \rightarrow \phi]_{\tau} = 0$ which in turn is possible only if $\mathcal{T}[\psi]_{\tau} = 1$ and $\mathcal{T}[\phi]_{\tau} = 0$ which is impossible. Hence there is no such truth assignment. So $\phi \rightarrow (\psi \rightarrow \phi)$ must be a tautology.

A similar reasoning may be applied to the axiom schemas S and N.

2. Assume for some ϕ and ψ that ϕ and $\phi \rightarrow \psi$ are tautologies but ψ is not. It is easy to see that for any truth assignment τ , $\mathcal{T}[\psi]_{\tau} = 0$ implies $\mathcal{T}[\phi]_{\tau} = 0$ contradicting the assumption that ϕ is a tautology.

QED

Soundness of the Hilbert System

Theorem 16.1: Theorems are tautologies

Every formal theorem of \mathcal{H}_0 is a tautology.

The theorem follows by induction on the heights of proof trees, since every leaf would be a tautology and every internal node preserves tautologousness.

Corollary 16.1: Soundness of \mathcal{H}_0

The system \mathcal{H}_0 (with all its derived rules and derived operators with introduction and elimination rules) is sound.

By problem 3 of exercise 15.4 it follows that

Corollary 16.2: Soundness of \mathcal{G}_0

The system \mathcal{G}_0 is sound.

17. The Hilbert System: Completeness

17: The Hilbert System: Completeness

1. Formal Theory: Incompleteness
2. Towards Completeness
3. Towards Proofs of Truth-tables
4. The Truth-table Lemma
5. The Completeness Theorem

17.1. Completeness of a proof system

Assume that we have proof system \mathcal{S} . It is said to be **(deductively) complete** if every fact of the system is provable. For the purposes of this section, let us assume that we have a sound proof system. The notion of completeness is the converse of the notion of soundness. Whereas soundness referred to showing that all deductions conform to the semantics of the formal language i.e. $\Gamma \vdash_{\mathcal{S}} \psi$ implies $\Gamma \models \psi$, completeness is the converse property i.e. $\Gamma \models \psi$ implies $\Gamma \vdash_{\mathcal{S}} \psi$.

In the case of propositional logic it suffices to show that for every *finite* set of assumptions Γ , $\Gamma \models \psi$ implies $\Gamma \vdash_{\mathcal{S}} \psi$. Even if Γ is infinite, $\Gamma \models \psi$ if and only if $\Gamma \cup \{\neg\psi\}$ is **unsatisfiable** and by the **compactness** theorem and its **consequences** it follows that $\Gamma \cup \{\neg\psi\}$ would be unsatisfiable if and only if some non-empty finite subset $\Delta \subseteq_f \Gamma \cup \{\neg\psi\}$ is unsatisfiable. If $\neg\psi \notin \Delta$ then $\Delta \subseteq_f \Gamma$ and hence Γ itself is inconsistent. On the other hand if Γ is consistent then every non-empty finite subset of Γ is consistent and hence $\Delta - \{\neg\psi\}$ would be consistent whereas Δ itself would be **inconsistent**. It follows then that $\Delta \models \psi$. Hence for completeness it suffices that $\Delta \vdash_{\mathcal{S}} \psi$. Since Δ is non-empty and finite we further have that $\Delta \models \psi$ if and only if the formula $\bigwedge \Delta \rightarrow \psi$ is **valid** (i.e. $\models \bigwedge \Delta \rightarrow \psi$). Hence for the system \mathcal{S} to be complete it suffices that $\bigwedge \Delta \rightarrow \psi$ be a formal theorem of \mathcal{S} (i.e. $\vdash_{\mathcal{S}} \bigwedge \Delta \rightarrow \psi$) for each finite set Δ and sentence ψ such that $\Delta \models \psi$.

Formal Theory: Incompleteness

Suppose a given formal theory is incomplete. There are several possibilities.

Question 1. *Can the theory be made complete by adding or changing some axioms and inference rules (without making the theory inconsistent)?*

Question 2. *Is the theory inherently incomplete? That is, is there no way of achieving completeness by adding only a finite number of new axioms and inference rules?*

Towards Completeness

1. By theorem 16.1 the only theorems of the system \mathcal{H}_0 are tautologies.
2. By theorems 4.1 and 4.2 the question of completeness of \mathcal{H}_0 reduces to that of whether *every tautology of \mathcal{P}_0 is provable in \mathcal{H}_0 .*
3. If \mathcal{H}_0 is complete then by exercise 15.4.4, \mathcal{G}_0 is also complete.

Towards Proofs of Truth-tables

1. Restricting ourselves to showing that every tautology is provable in \mathcal{H}_0 is sufficient.
2. But we proceed to show that every truth table of a formula ϕ can be *simulated* as a proof in \mathcal{H}_0 , thereby capturing all of the semantic features of the language \mathcal{P}_0 in its proof theory. We show this in two steps.
 - (a) We first show that each row of a truth table of ϕ can be simulated by a proof.
 - (b) We next show that the proofs of the individual rows of a truth table can then be combined to form one mammoth proof of the formula ϕ if it is a tautology.

The Truth-table Lemma

Lemma 17.1: The Truth-table Lemma

Let ϕ be a formula with $atoms(\phi) \subseteq \{p_1, \dots, p_k\}$. For each truth assignment τ ,

$$p_1^*, \dots, p_k^* \vdash_{\mathcal{H}_0} \phi^*$$

where for each i , $1 \leq i \leq k$,

$$p_i^* \equiv \begin{cases} p_i & \text{if } \tau(p_i) = 1 \\ \neg p_i & \text{otherwise} \end{cases}$$

and

$$\phi^* \equiv \begin{cases} \phi & \text{if } \mathcal{T}[\phi]_\tau = 1 \\ \neg \phi & \text{otherwise} \end{cases}$$



Proof of lemma 17.1

Proof: By induction on the number n of operators in ϕ . Let $\Gamma = \{p_1^*, \dots, p_k^*\}$.

Basis. $n = 0$. Then ϕ is an atom say, $\phi \equiv p_1$. The claim then trivially follows since $p_1^* \equiv \phi^*$.

Induction Hypothesis (IH).

The claim holds for all wffs with less than $n \geq 0$ occurrences of the operators.

Induction Step. Suppose ϕ is a wff with n operators. Then there are two cases to consider.

Case $\phi \equiv \neg\psi$, where ψ has less than n operators. Then by the induction hypothesis we have a proof tree $\frac{\Gamma \vdash \psi^*}{\Gamma \vdash \psi^*}$.

Subcase $\mathcal{T}[\psi]_\tau = 1$. Then $\mathcal{T}[\phi]_\tau = 0$ and $\psi^* \equiv \psi$ and $\phi^* \equiv \neg\phi \equiv \neg\neg\psi$. Then we have the following deduction.

$$\frac{\text{DNI} \rightarrow \frac{\Gamma \vdash \psi \rightarrow \neg\neg\psi}{\Gamma \vdash \psi}}{\text{MP} \quad \frac{\Gamma \vdash \neg\neg\psi \equiv \phi^*}{\Gamma \vdash \neg\neg\psi \equiv \phi^*}}$$

Subcase $\mathcal{T}[\psi]_\tau = 0$. Then $\mathcal{T}[\phi]_\tau = 1$ and $\psi^* \equiv \neg\psi$ and $\phi^* \equiv \phi \equiv \neg\psi$. By the induction hypothesis we have $\Gamma \vdash \neg\psi \equiv \phi^*$.

Case $\phi \equiv \psi \rightarrow \chi$, where each of ψ and χ has less than n operators. By the induction hypothesis there exist proof trees $\frac{\Gamma \vdash \psi^*}{\Gamma \vdash \psi^*}$ and $\frac{\Gamma \vdash \chi^*}{\Gamma \vdash \chi^*}$.

Here again we have three subcases.

Subcase $\mathcal{T}[\psi]_\tau = 0$. We have $\psi^* \equiv \neg\psi$ so tree \mathcal{T}_1 is $\frac{\Gamma \vdash \neg\psi}{\Gamma \vdash \neg\psi}$, $\mathcal{T}[\phi]_\tau = 1$ and $\phi^* \equiv \phi \equiv \psi \rightarrow \chi$. We then have the proof tree

$$\text{MP} \frac{\perp \frac{}{\Gamma \vdash \neg\psi \rightarrow (\psi \rightarrow \chi)} \quad \frac{\nwarrow \mathcal{T}_1 \nearrow}{\Gamma \vdash \neg\psi}}{\Gamma \vdash \psi \rightarrow \chi \equiv \phi^*}$$

which proves the claim.

Subcase $\mathcal{T}[\chi]_\tau = 1$. Then $\chi^* \equiv \chi$ and $\mathcal{T}[\phi]_\tau = 1$ and $\phi^* \equiv \phi \equiv \psi \rightarrow \chi$. Hence tree \mathcal{T}_2 is $\frac{\nwarrow \mathcal{T}_2 \nearrow}{\Gamma \vdash \chi}$. We may then construct the following proof tree to prove the claim.

$$\text{MP} \frac{\text{K} \frac{}{\Gamma \vdash \chi \rightarrow (\psi \rightarrow \chi)} \quad \frac{\nwarrow \mathcal{T}_2 \nearrow}{\Gamma \vdash \chi}}{\Gamma \vdash \psi \rightarrow \chi \equiv \phi^*}$$

Subcase $\mathcal{T}[\psi]_\tau = 1$ and $\mathcal{T}[\chi]_\tau = 0$. Then $\psi^* \equiv \psi$ and $\chi^* \equiv \neg\chi$ and $\mathcal{T}[\phi]_\tau = 0$ from which we get $\phi^* \equiv \neg(\psi \rightarrow \chi)$. By induction hypotheses we therefore have the trees $\frac{\nwarrow \mathcal{T}_1 \nearrow}{\Gamma \vdash \psi}$ and $\frac{\nwarrow \mathcal{T}_2 \nearrow}{\Gamma \vdash \neg\chi}$ using which we construct the following proof tree to prove our claim.

$$\text{N2} \frac{}{\Gamma \vdash \psi \rightarrow (\neg\chi \rightarrow \neg(\psi \rightarrow \chi))} \quad \text{MP} \frac{\frac{\nwarrow \mathcal{T}_1 \nearrow}{\Gamma \vdash \psi} \quad \frac{\nwarrow \mathcal{T}_2 \nearrow}{\Gamma \vdash \neg\chi}}{\Gamma \vdash \neg\chi \rightarrow \neg(\psi \rightarrow \chi)} \quad \text{MP} \frac{}{\Gamma \vdash \neg(\psi \rightarrow \chi) \equiv \phi^*}$$

QED

The Completeness Theorem

We are now ready to prove the completeness theorem by restricting it to all the tautologies of propositional logic.

Theorem 17.1: The Completeness of \mathcal{H}_0

Every tautology is a formal theorem of \mathcal{H}_0 .



Proof of the Hilbert Completeness Theorem 17.1

Proof: Let ϕ be a tautology expressed in the language \mathcal{L}_0 and let $atoms(\phi) = \{p_1, \dots, p_k\}$. Since every row of the truth-table for ϕ assigns it a truth value 1 we have $\phi^* \equiv \phi$. Each bit-string $s_k \in \{0, 1\}^k$ indexes a row of the truth table containing 2^k distinct rows. Let $\Gamma_{s_k} = \{p_i^* \mid 1 \leq i \leq k\}$ denote the set of assumptions for the s_k -th row of the truth table. By the truth table lemma 17.1, there exists a distinct proof tree,

for each such s_k . For each bit-string $s_j \in \{0, 1\}^j$, $0 < j \leq k$, we construct the proof tree from the proof trees and , where $s_{j-1}0$ and $s_{j-1}1$ are the two bit-strings of length j , which differ only in the right-most bit.

Note that $s_0 = \epsilon$ is the empty string, $\Gamma_\epsilon = \emptyset$ and we need to construct the proof tree from the 2^k distinct proof trees .

Now consider any two proof trees whose indexes differ only in the rightmost bit. That is, for any $s_{j-1} \in \{0, 1\}^{j-1}$, we have the proof trees and . We construct the proof tree as follows.

$$\frac{\text{DT} \Rightarrow \frac{\text{DT} \Rightarrow \frac{\Gamma_{s_{j-1}0} \vdash \phi}{\Gamma_{s_{j-1}} \vdash \neg p_j \rightarrow \phi}}{\text{MP} \frac{\Gamma_{s_{j-1}} \vdash p_j \rightarrow \phi}{\Gamma_{s_{j-1}} \vdash (\neg p_j \rightarrow \phi) \rightarrow \phi}} \quad \text{DT} \Rightarrow \frac{\Gamma_{s_{j-1}1} \vdash \phi}{\Gamma_{s_{j-1}} \vdash p_j \rightarrow \phi}}{\text{MP} \frac{\Gamma_{s_{j-1}} \vdash (\neg p_j \rightarrow \phi) \rightarrow \phi}{\Gamma_{s_{j-1}} \vdash \phi}}$$

We can thus eliminate the atom p_j from the assumptions by applying the above proof procedure to all pairs of proof trees whose assumptions differ only in the value of p_j^* .

Thus the 2^k proof trees are combined pairwise to produce 2^{k-1} proof trees that are independent of the atom p_k . Proceeding in a like manner we may eliminate all the atoms one by one by using similar proof constructions so that finally we obtain a single monolithic proof tree $\Gamma_\epsilon \vdash \phi$ where $\Gamma_\epsilon = \emptyset$, thus concluding the proof that the tautology ϕ is a formal theorem of \mathcal{H}_0 . QED

Exercise 17.1: Hilbert Completeness

1. Consider the truth-table of the tautology $\psi \equiv (p \wedge q) \rightarrow (p \leftrightarrow q)$.
 - (a) Give a proof of ψ in \mathcal{H}_0 (you may use whatever derived rules have been proved so far).
 - (b) Now construct the truth table of ψ .
 - Give a proof of each row of the truth-table as per lemma 17.1.
 - Combine the proofs of the 4 rows of the truth-table as described in the [proof of completeness](#).
 - (c) What difference do you see in the proofs of the above two parts?

18. Introduction to Predicate Logic

18: Introduction to Predicate Logic

1. Predicate Logic: Introduction-1
2. Predicate Logic: Introduction-2
3. Internal Structure of Sentences
4. Internal Structure of Sentences
5. Parameterisation-1
6. Parameterisation-2
7. Predicate Logic: Introduction-3
8. Predicate Logic: Symbols
9. Predicate Logic: Signatures
10. Predicate Logic: Syntax of Terms
11. Predicate Logic: Syntax of Formulae
12. Precedence Conventions
13. Predicates: Abstract Syntax Trees
14. Subterms: Depth and Size
15. Variables in a Term
16. Bound Variables And Scope
17. Bound Variables And Scope: Example
18. Scope Trees

- 19. Free Variables
- 20. Bound Variables
- 21. Closure

Predicate Logic: Introduction-1

There are many kinds of arguments which cannot be proven in propositional logic and which require the notion of quantifiers. The most famous one perhaps containing only very basic and simple declarative statements is

Example 18.1

All humans are mortal.

Socrates is a human.

Therefore Socrates is mortal.

Predicate Logic: Introduction-2

The validity of this argument does not depend on any *propositional connectives* since there are none. Hence it is not provable in propositional logic. However it is valid and its validity depends upon

- the *internal structure* of the sentences,
- the meaning of certain operative words and phrases such as “All”
- certain properties of objects e.g. “mortal”
- the description of certain classes by their properties and membership or other relations on these classes e.g. “is a”.

Internal Structure of Sentences

There are a large number of propositions which could be parameterized.

Aristotle is a human

Example 18.2

All humans are mortal.

Aristotle is a human.

Therefore Aristotle is mortal.

Internal Structure of Sentences

There are a large number of propositions which could be parameterized.

All humans are mammalian

Example 18.3

All humans are mammalian.

Aristotle is a human.

Therefore Aristotle is mammalian.

Parameterisation-1

x is a human

All humans are y

Example 18.4

All humans are y .

x is a human.

Therefore x is y .

Parameterisation-2

x is a z

All z are y

Example 18.5

All z are y .

x is a z .

Therefore x is y .

- x is a placeholder for an element from some set (*the universe of discourse*).
- y is a property of a subset of elements from the *universe of discourse*.
- z is either a property or a set of elements from the *universe of discourse*.

Predicate Logic: Introduction-3

1. The deeper relationships that exist between otherwise simple propositions require parameterisation of propositions so that the inter-relationships become clear.
2. Parameterised propositions are called *predicates*.
3. Each predicate specifies either a property (1-ary predicates) or a relationship between objects (n -ary predicates).
4. The propositions of propositional logic are 0-ary predicates.
5. Mathematical theories are often about collections of infinite objects whose relationships and inter-relationships are finite expressions involving *functions*, *relations* and *expressions* involving them.

Predicate Logic: Symbols

- V : a countably infinite collection of **variable symbols**; $x, y, z, \dots \in V$.
- F : a countably infinite collection of **function symbols**; $f, g, h, \dots \in F$.
- A : a countably infinite collection of **atomic predicate symbols**; $p, q, r, \dots \in A$.
- Grouping symbols: (and) for arguments of terms, (and) for parameters of predicates, [and] for delimiting the scope of the bound variable of a quantifier.
- All of the above sets are pairwise disjoint.

see also Section -1.2

Definition 18.1: Signature

A **signature** (or more accurately **1-sorted signature**) Σ is a denumerable (finite or countably infinite) collection of strings of the form

$$f : s^m \rightarrow s, m \geq 0$$

or

$$p : s^n, n \geq 0$$

such that there is at most one string for each $f \in F$ and each $p \in A$. m and n are respectively the **arity** of f and p .

Here s is merely a symbol signifying a **sort** of elements. A generalization to many-sorted algebras would require the use of as many such symbols s_1, \dots, s_m as there are sorts.

Predicate Logic: Syntax of Terms

(see also section -1.2)

Definition 18.2: Σ -Terms

Given a signature Σ , the set $\mathbb{T}_\Sigma(V)$ of Σ -terms is defined inductively by the following grammar

$$s, t, u ::= x \in V \mid f(t_1, \dots, t_m)$$

where $f : s^m \rightarrow s \in \Sigma$ and $t_1, \dots, t_m \in \mathbb{T}_\Sigma(V)$. If $m = 0$ then $f()$ is called a **constant** and simply written f . We usually use the symbols a, b, c, \dots to denote constants.

Predicate Logic: Syntax of Formulae

Definition 18.3: Σ -formulae

Given a signature Σ and the set $\mathbb{T}_\Sigma(V)$ of Σ -terms.

- A **Σ -atomic formula** or **Σ -atom** is a string of the form $p(t_1, \dots, t_n)$ where $p : s^n \in \Sigma$. $A(\Sigma)$ is the set of Σ -atoms.
- $\Omega_1 = \Omega_0 \cup \{\forall x, \exists x \mid x \in V\}$
- The set of $\mathcal{P}_1(\Sigma)$ of Σ -formulae is defined inductively by the following grammar.

$\phi, \psi ::= \perp$	$ $	T
$ p(t_1, \dots, t_n) \in A(\Sigma)$	$ $	$(\neg\phi)$
$ (\phi \wedge \psi)$	$ $	$(\phi \vee \psi)$
$ (\phi \rightarrow \psi)$	$ $	$(\phi \leftrightarrow \psi)$
$ \forall x[\phi]$	$ $	$\exists x[\phi]$

Precedence Conventions

- The operator precedence conventions are as **before**.
- The two new operators are called the *universal quantifier* (\forall) and *existential quantifier* (\exists) respectively and are parameterised by variables.
- The scope of the (variable in a) quantified formula is delimited by the matching pair of brackets ([and]).
- If a formula ϕ is preceded by several quantifiers (e.g. $\forall x[\exists y[\forall z[\phi]]]$) we collapse the scoping brackets where there is no ambiguity (e.g $\forall x\exists y\forall z[\phi]$).
- We will think of both Σ -terms and Σ -formulae as **abstract syntax trees**. The brackets delimiting the scope of a quantified variable then become redundant.

18.1. Predicates in Natural Languages

The translation of propositional sentences expressed in natural language into propositions (as defined in section 2.1) was actually pretty straight-forward (with a few exceptions). In general, one could easily identify the propositional connectives between the component sentences by a more or less direct transliteration. The translation was more or less “syntax-directed” and the structure of compound sentences was reflected in the structure of the translated propositions. Even then we had to take several exceptions into account in the use of connectives such as *and* and *or*.

The issue of translating natural language sentences with quantifiers is somewhat more complicated. One major complication is the fact that properties of objects, qualifications and subclasses of objects sharing common properties all have the same structure in natural language. The so-called *is-a* relationship is overloaded in a natural language. Another complication relates to the fact that predicate logic is about relationships and a property of an object is merely taken as a unary relation.

We begin first with identifying quantifiers and then go deeper into the issue of translation. Necessarily, most of what we state here with respect to translation is not to be taken as “gospel truth” but merely as a guideline and even those may be applicable only to the English language.

Operation	English rendering
$\forall x[\phi]$	for all x, ϕ for each x, ϕ for every x, ϕ for any x, ϕ

Operation	English rendering
$\exists x[\phi]$	for some x, ϕ there exists x , such that ϕ for at least one x, ϕ

Operation	English rendering
$\neg\forall x[\phi]$	not all x, ϕ not the case that for each x, ϕ not for every x, ϕ

Operation	English rendering
$\neg\exists x[\phi]$	for no x, ϕ there does not exist x , such that ϕ there is no x , such that ϕ

is-a statements

Consider the following statements which are universal statements with some implicit containment or implicational relationship between classes of objects.

1. All humans are mortal.
2. All Greeks are human.

3. All non-Greeks are also human.

Here the class of *humans* is a sub-class of the class of *mortals*, the class of *Greeks* is a sub-class of the class of *humans*, and the class of *non-Greeks* is a sub-class of the class of *humans*. We may think of unary predicates $m(x)$, $h(x)$, $g(x)$ as representing the characteristic property of the three classes respectively. In particular, they represent the properties of “mortality”, “human-ness” and “greek-hood”. The translation of the above sentences into predicate logic is then in terms of the properties of the sub-classes.

1. $\forall x[h(x) \rightarrow m(x)]$
2. $\forall x[g(x) \rightarrow h(x)]$
3. $\forall x[\neg g(x) \rightarrow h(x)]$

Once a sub-class is characterised by a property, other properties could also be used as qualifiers in conjunction with it. For example,

1. *All rich humans are mortal.*
2. *All rich humans are obese mortals.*

may be translated as follows (letting r stand for the quality of being *rich* and o for *obesity*).

1. $\forall x[(h(x) \wedge r(x)) \rightarrow m(x)]$

$$2. \forall x[(h(x) \wedge r(x)) \rightarrow (m(x) \wedge o(x))]$$

Action predicates

Besides the **is-a** relationships, predicates may also be used to represent actions (which are actually verbs and not adjectives or qualifiers). Consider the following statements.

1. *Ram wedded Sita.*
2. *The Pandavas wedded Draupadi.*
3. *Elizabeth Taylor was married.*

“Reasonable” translations which capture the meanings of these statements in terms of the binary action predicate *w* which we use to denote both *wedded* and *married* yield the following (with *r* for *Ram*, *s* for *Sita*, *d* for *Draupadi*, *p* for the class *Pandavas*, and *e* for *Elizabeth Taylor*).

1. $w(r, s)$
2. $\forall x[p(x) \rightarrow w(x, d)]$
3. $\exists x[w(x, e)]$

Then there are statements like

1. Nobody loves Charlie Brown.
2. Everybody loves Calvin.
3. There is nobody who does not love Calvin.

whose “reasonable” translations using $l(x, y)$ as a binary predicate symbol for x loves y and constant symbols b for *Charlie Brown* and c for *Calvin* would be

1. $\neg\exists x[l(x, b)]$
2. $\forall x[l(x, c)]$
3. $\neg\exists x[\neg l(x, c)]$

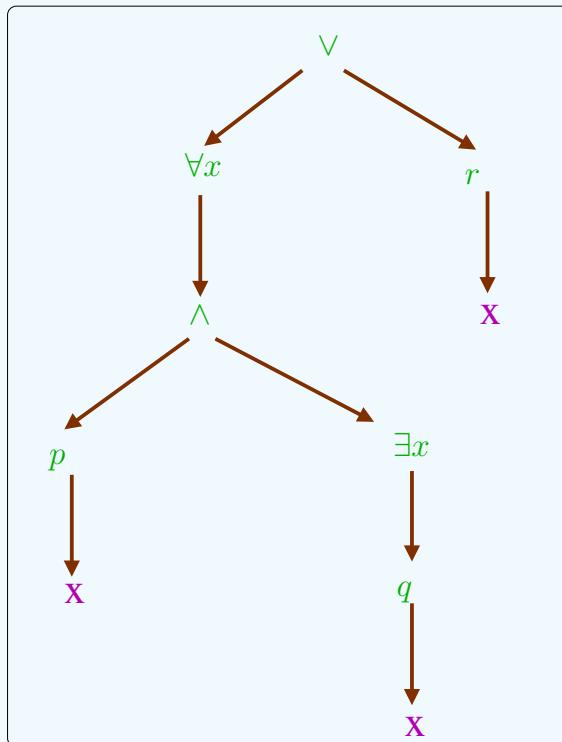
Notice that the last two statements mean the same thing, however they have syntactically different renderings. We will also see that they are actually logically equivalent statements.

One could go on and on with several examples. But since this is not meant to be exhaustive (and could never be) we will not take it any further, but instead rely on “common-sense” to capture the intended meaning of a predicate or a sentence with quantifiers.

Predicates: Abstract Syntax Trees

Example 18.6

Let p , q and r be unary predicates. The first-order logic formula $\forall x[p(x) \wedge \exists x[q(x)]] \vee r(x)$ is represented by the following abstract syntax tree (see also example -1.13).



Subterms: Depth and Size

Definition 18.4: Subterms, variables, depth and size

For each term t , $ST(t)$ denotes the set of subterms of t (including t itself). The set of *proper subterms* of t is the set $ST(t) - \{t\}$.

t	$Var(t)$	$depth(t)$	$size(t)$	$ST(t)$
$c()$	\emptyset	1	1	$\{t\}$
x	$\{x\}$	1	1	$\{t\}$
$f(t_1, \dots, t_n)$	$\bigcup_{i=1}^n Var(t_i)$	$\max_{i=1}^n depth(t_i)$ $+1$	$\sum_{i=1}^n size(t_i)$ $+1$	$\bigcup_{i=1}^n ST(t_i)$ $\cup \{t\}$

Bound Variables And Scope

Definition 18.5: Bound variables

In a formula of the form $\mathcal{O}x[\psi]$ the variable x is said to be **bound** by the quantifier \mathcal{O} . The brackets $[\dots]$ delimit the **scope** of the binding.

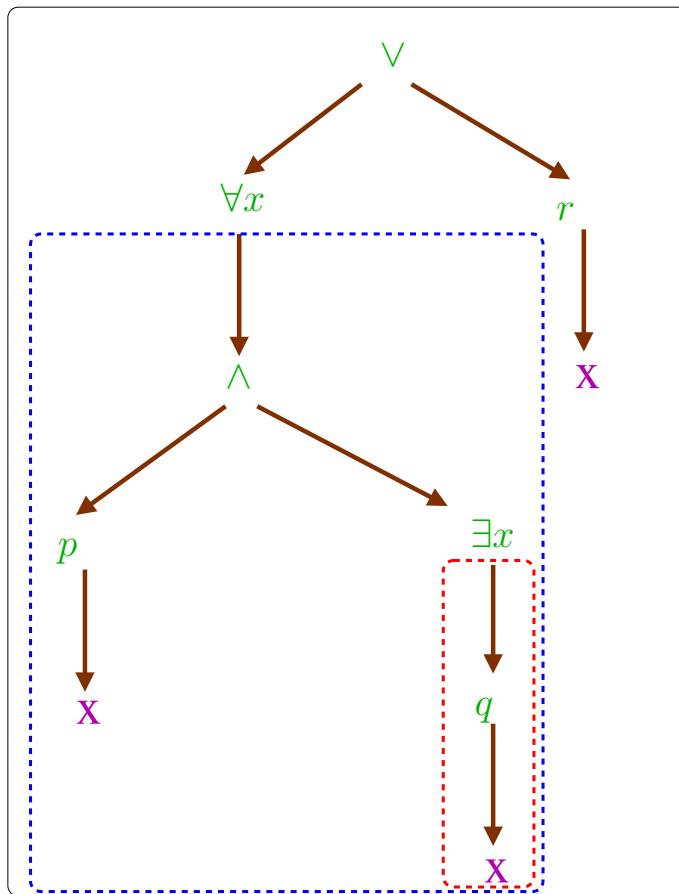
Example 18.7: Bound variables and scopes

In the **abstract syntax tree** of the predicate

$$\forall x[p(x) \wedge \exists x[q(x)]] \vee r(x)$$

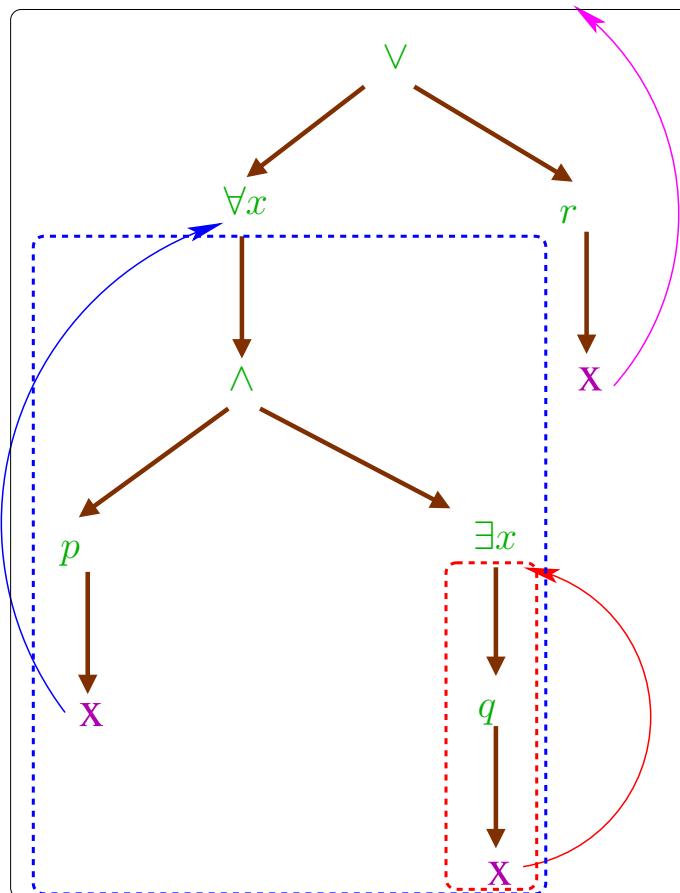
the scopes of the various bindings are indicated by dashed boxes. Notice that the **red** scope is a “hole” in the **blue** scope.

Bound Variables And Scope: Example



Scope Trees

The abstract syntax tree also determines the scope of the individual bound variables.



Free Variables

Definition 18.6: Free variables

For any predicate ϕ the set of free variables occurring in it (denoted $FV(\phi)$) and the set of sub-formulae (denoted $SF(\phi)$) are defined by induction on the structure of predicates.

ϕ	$FV(\phi)$	$SF(\phi)$	Condition
$p(t_1, \dots, t_n)$	$\bigcup_{1 \leq i \leq n} Var(t_i)$	$\{p(t_1, \dots, t_n)\}$	
$\neg\psi$	$FV(\psi)$	$\{\neg\psi\} \cup SF(\psi)$	
$o(\psi, \chi)$	$FV(\psi) \cup FV(\chi)$	$\{o(\psi, \chi)\} \cup SF(\psi) \cup SF(\chi)$	$o \in \Omega_0 - \{\neg\}$
$\mathcal{O}x[\psi]$	$FV(\psi) - \{x\}$	$\{\mathcal{O}x[\psi]\} \cup SF(\psi)$	$\mathcal{O} \in \{\forall, \exists\}$

Bound Variables

Definition 18.7: Bound variables

If $\mathcal{O}x[\psi]$ is a sub-formula of some formula ϕ then ψ is said to be the scope of the quantifier $\mathcal{O}x$ and every free occurrence of the variable x in the formula ψ is said to be bound in the scope of the quantifier $\mathcal{O}x$ in which it occurs.

Notice that if $\mathcal{O}x[\chi]$ is a sub-formula of ψ in the definition 18.7 then any $x \in FV(\chi)$ is not a free variable of ψ .

Notation 18.1: Free variables of a predicate

A predicate ϕ may be written $\phi(x_1, \dots, x_m)$ to indicate that $FV(\phi) \subseteq \{x_1, \dots, x_m\}$.

Closure

Definition 18.8: Closed formulae and closures

1. A formula ϕ is called **closed** if $FV(\phi) = \emptyset$. A closed formula is also called a **sentence** (c.f. definition 2.4).
2. $\forall \vec{x}_1, \dots, \vec{x}_m[\phi]$ is the **universal closure** of $\phi(\vec{x}_1, \dots, \vec{x}_m)$.
3. $\exists \vec{x}_1, \dots, \vec{x}_m[\phi]$ is the **existential closure** of $\phi(\vec{x}_1, \dots, \vec{x}_m)$.

Notation 18.2: Closures

- $\vec{\forall}[\phi]$: Universal closure of ϕ ,
- $\vec{\exists}[\phi]$: Existential closure of ϕ and
- $\mathcal{P}_1^\circ(\Sigma) \subset \mathcal{P}_1(\Sigma)$ is the set of first-order sentences over Σ .

Notice that all propositional formulae are also first order sentences i.e. $\mathcal{P}_0 \subset \mathcal{P}_1^\circ(\Sigma)$.

Exercise 18.1: Natural language to Predicate logic

1. Translate the following statements into first-order logic statements. (You may use the function symbols that are normally used in mathematics, e.g. “0” for zero, “=” for equality, “+” for addition etc.). The names x , y etc. stand for variables.

- Every number has a unique successor.
- Not every number has a predecessor.
- The sum of any two odd numbers is even.
- x is a prime.
- There is no largest prime.
- x is a divisor of y .
- x and y are relatively prime.
- Define the notion of greatest common divisor of two numbers as as a ternary predicate $gcd(x, y, z)$ in terms of the previous parts. In other words, $gcd(x, y, z)$ stands for the statement

z is the greatest common divisor of x and y

2. Symbolize the following arguments in first order logic You may assume in each case that the universe of discourse contains the various relations mentioned as predicates (and nothing more!).

- There is a man whom all men despise. Therefore at least one man despises himself.
- All hotels are expensive and depressing. Some hotels are shabby. Therefore some expensive hotels are shabby.
- Anyone who is loved loves everyone. No one loves Charlie Brown. Therefore no one loves anyone.

Exercise 18.2: More verbal arguments

Symbolize the following arguments in first order logic You may assume in each case that the universe of discourse contains the various relations mentioned as predicates (and nothing more!).

1. *Whoever visited the building was observed. Anyone who had observed Ajay, would have remembered him. Nobody remembered Ajay. Therefore Ajay did not visit the building.*
2. *If all drugs are contaminated then all negligent technicians are scoundrels. If there are any drugs that are contaminated then all drugs are contaminated and unsafe. All pesticides are drugs. Only the negligent are absent-minded. Therefore if any technician is absent-minded, then if some pesticides are contaminated, then he is a scoundrel.*
3. *Some criminal robbed the mansion. Whoever robbed the mansion broke in or had an accomplice among the servants. To break in one would have to smash the door or pick the lock. Only an expert locksmith could have picked the lock. Had anyone smashed the door, he would have been heard. Nobody was heard. If the criminal who robbed the mansion managed to fool the guard, he must have been a convincing actor. Nobody could rob the mansion unless he fooled the guard. No criminal could be both an expert locksmith and a convincing actor. Therefore some criminal had an accomplice among the servants.*

19. The Semantics of Predicate Logic

19: The Semantics of Predicate Logic

"There's glory for you!"

"I don't know what you mean by 'glory,'" Alice said. Humpty Dumpty smiled contemptuously. "Of course you don't – till I tell you. I meant 'there's a nice knock-down argument for you!'"

"But 'glory' doesn't mean 'a nice knock-down argument,'" Alice objected.

"When I use a word," Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean – neither more nor less."

"The question is," said Alice, "whether you can make words mean so many different things."

"The question is," said Humpty Dumpty, "which is to be master – that's all."

Lewis Carroll, *Through the Looking-Glass*

1. Structures
2. Notes on Structures
3. Infix Convention
4. Expansions and Reducts
5. Valuations and Interpretations
6. Evaluating Terms
7. Coincidence Lemma for Terms
8. Valuation Variants
9. Variant Notation
10. Semantics of Formulae
11. Notes on the Semantics
12. Coincidence Lemma for Formulae

Structures

Given a signature Σ , a Σ -structure or Σ -algebra \mathbf{A} consists of

- a non-empty set $\mathbb{A} = |\mathbf{A}|$ called the **domain** (or **carrier** or **universe**) of \mathbf{A} ,
- a function $f_{\mathbf{A}} : \mathbb{A}^m \rightarrow \mathbb{A}$ for each m -ary function symbol $f \in \Sigma$ (including symbols for each constant)
- a relation $p_{\mathbf{A}} \subseteq \mathbb{A}^n$ for each n -ary atomic predicate symbol $p \in \Sigma$ and
- (for completeness) a truth value $p_{\mathbf{A}} \in \mathfrak{2} = \{0, 1\}$ for each (0-ary) atomic proposition $p \in \Sigma$.

When the Σ -algebra is understood or is the only structure under consideration, we omit the subscript \mathbf{A} from the functions and relations.

Notes on Structures

1. The domain has to be non-empty.
2. All functions are *total*.
3. One way to deal with *partial functions* (e.g. division on natural numbers) of arity $m > 1$ is to treat them as $(m + 1)$ -ary relations and define predicate symbols to represent them.
4. Another way to deal with *partial functions* is to “*guard*” the predicates in which they occur in such a way as to bound the domain of allowable values only to those for which the function is defined. The guard is itself another predicate. This is a technique that is useful where certain functions or predicates are defined only for some subset of the allowable values. Also see equation (85) and the predicate ϕ_{decomp} in section ??.

Infix Convention

If in particular structures, functions or relations are normally written in infix form, then we use the infix form in the logical language too.

Example 19.1

If $\mathbf{N} = \langle \mathbb{N}; +; < \rangle$ the set of natural numbers under the binary operation of addition (+) and the binary relation less-than (<) is the structure, then we write predicate formulae using the corresponding symbols in the language in infix form. For example, the formula

$$\forall x [\exists y [x < x + y]]$$

with the operation in infix form is more easily understood in place of the more pedantic

$$\forall x [\exists y [< (x, +(x, y))]]$$

Expansions and Reducts

Definition 19.1: Reduct and Expansion

Let $\Sigma \subseteq \Omega$ be signatures. A Σ -structure \mathbf{A} is called a **reduct** to an Ω -structure \mathbf{B} iff

- $|\mathbf{A}| = |\mathbf{B}|$,
- $f_{\mathbf{A}} = f_{\mathbf{B}}$ for each $f \in \Sigma$ and
- $p_{\mathbf{A}} = p_{\mathbf{B}}$ for each $p \in \Sigma$

\mathbf{B} is also called an **expansion** of \mathbf{A} and is denoted $\mathbf{A} \triangleleft \mathbf{B}$.

Valuations and Interpretations

To be able to define the truth or falsehood of a predicate with free variables, it is necessary to be able to first define a valuation for the variables.

Definition 19.2: Valuation

Given a Σ -structure \mathbf{A} , a **valuation** (also called state) is a function $v_{\mathbf{A}} : V \rightarrow |\mathbf{A}|$ which assigns to each variable a unique value in $|\mathbf{A}|$.

Definition 19.3: Σ -interpretation

Given a Σ -structure \mathbf{A} and a valuation $v_{\mathbf{A}} : V \rightarrow |\mathbf{A}|$, $(\mathbf{A}, v_{\mathbf{A}})$ is called a Σ -interpretation.

Evaluating Terms

Definition 19.4: Values of terms

Given Σ -interpretation (\mathbf{A}, v) the value of a term t in the interpretation is defined by induction on the structure of the term.

$$\mathcal{V}_{\mathbf{A}}[\underline{x}]_v \stackrel{df}{=} v(\underline{x}), \quad x \in V$$

$$\mathcal{V}_{\mathbf{A}}[f(t_1, \dots, t_m)]_v \stackrel{df}{=} f_{\mathbf{A}}(\mathcal{V}_{\mathbf{A}}[\underline{t}_1]_v, \dots, \mathcal{V}_{\mathbf{A}}[\underline{t}_m]_v), \quad f : s^m \rightarrow s \in \Sigma$$

Notation 19.1

1. If $Var(t) \subseteq \{\underline{x}_1, \dots, \underline{x}_m\}$, we sometimes write $t(\underline{x}_1, \dots, \underline{x}_m)$ to denote this fact.
2. The subscript “ \mathbf{A} ” is often omitted when the context makes clear the structure that is being referred to.

Coincidence Lemma for Terms

The following lemma may be easily proved by induction on the structure of terms.

Lemma 19.1: Coincidence lemma for terms

Given two Σ -interpretations (\mathbf{A}, v) and (\mathbf{A}, v') , and a term t , if $v(x) = v'(x)$ for each $x \in \text{Var}(t)$, then $\mathcal{V}[\![t]\!]_v = \mathcal{V}[\![t]\!]_{v'}$.

Notation 19.2

If $t(x_1, \dots, x_m)$, $v(x_1) = a_1, \dots, v(x_m) = a_m$ for $a_1, \dots, a_m \in |\mathbf{A}|$ in some Σ -interpretation (\mathbf{A}, v) , then we write $t_{\mathbf{A}}(a_1, \dots, a_m)$ instead of $\mathcal{V}_{\mathbf{A}}[\![t(x_1, \dots, x_m)]\!]_v$, since only the values of the variables occurring in t are needed to evaluate it.

Valuation Variants

Definition 19.5: Valuation-variant

Two valuations $v, v' : V \rightarrow |A|$ are said to be **X-variants** of each other (denoted $v =_{\setminus X} v'$ for any $X \subseteq V$ if for all $y \in V - X$, $v(y) = v'(y)$, i.e. they differ from each other *at most* in the values for variables from X . If X is finite then v' is a **finite variant** of v .

Fact 19.1: Equivalence of variants

1. For any $X \subseteq V$ and valuation v , v is an X -variant of itself i.e. $v =_{\setminus X} v$.
2. $=_{\setminus X}$ is an equivalence relation on valuations.

Variant Notation

Notation 19.3

1. When $X = \{\underline{x}\}$ is a singleton we refer to v and v' as \underline{x} -variants and denote it by $v =_{\underline{x}} v'$.
2. If $v_{\underline{x}} =_{\underline{x}} v$ and $v_{\underline{x}}(\underline{x}) = a \in |\mathbf{A}|$ we write $v_{\underline{x}} = v[\underline{x} := a]$.
3. If $X = \{\underline{x}_1, \dots, \underline{x}_n\}$, $v_X =_{\underline{X}} v$ and $v_X(\underline{x}_i) = a_i \in |\mathbf{A}|$, for each i , $1 \leq i \leq n$, then we write $v_X = v[\underline{x}_1 := a_1, \dots, \underline{x}_n := a_n]$ or $v_X = v[\underline{x}_1, \dots, \underline{x}_n := a_1, \dots, a_n]$.

Semantics of Formulae

Definition 19.6: Semantics of Predicates

Let (A, v_A) be a Σ -interpretation. Then $\mathcal{T}[\phi]_v$ is defined by induction on the structure of ϕ . We omit the **propositional connectives** as being obvious and concentrate only on the other constructs.

$$\mathcal{T}_A[p(t_1, \dots, t_n)]_{v_A} \stackrel{df}{=} \begin{cases} 1 & \text{if } (\mathcal{V}_A[t_1]_{v_A}, \dots, \mathcal{V}_A[t_n]_{v_A}) \in p_A \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{T}_A[\forall x[\phi]]_{v_A} \stackrel{df}{=} \prod \{\mathcal{T}_A[\phi]_{v'_A} \mid v'_A =_{\setminus x} v_A\}$$

$$\mathcal{T}_A[\exists x[\phi]]_{v_A} \stackrel{df}{=} \sum \{\mathcal{T}_A[\phi]_{v'_A} \mid v'_A =_{\setminus x} v_A\}$$

Definitions of \sum, \prod

The subscript “ A ” is often omitted when the context makes clear the structure that is being referred to.

Notes on the Semantics

1. Truth value is now evaluated under a *valuation* instead of a *truth assignment*.
2. A *valuation* defines a *truth assignment to parameterised propositions* depending upon the values of the parameters.
3. The set $\{\mathcal{T}_A[\phi]_{v'_A} \mid v'_A =_{\setminus x} v_A\}$ is nonempty since v_A is also an x -variant of itself.
4. The number of x -variants of v equals the cardinality of $|A|$.
5. $1 \leq |\{\mathcal{T}_A[\phi]_{v'_A} \mid v'_A =_{\setminus x} v_A\}| \leq 2$ since there are only two possible truth values (even though there may be infinitely many x -variants of a valuation v_A). Hence the **sum and product** in the semantics of quantifiers are both *finitary*.

Coincidence Lemma for Formulae

The following lemma analogous to lemma 19.1 may be proved by induction on the structure of formulae.

Lemma 19.2: Coincidence lemma for formulae

Given two Σ -interpretations (\mathbf{A}, v) and (\mathbf{A}, v') , and a formula ϕ , if $v(\textcolor{violet}{x}) = v'(\textcolor{violet}{x})$ for each $x \in FV(\phi)$, then $\mathcal{T}[\![\phi]\!]_v = \mathcal{T}[\![\phi]\!]_{v'}$.



Proof of The Coincidence Lemma for formulae (lemma 19.2)

Variant Notation

Semantics of Formulae

Proof: By induction on the structure of ϕ . However the interesting cases are those of atomic predicates and quantified formulae.

Case $\phi \equiv p(t_1, \dots, t_n)$ where p is an n -ary predicate symbol. For each $x \in FV(p(t_1, \dots, t_n)) = \bigcup_{1 \leq i \leq n} Var(t_i)$ we have $v(x) = v'(x)$. Hence for each t_i we have $\mathcal{V}[t_i]_v = \mathcal{V}[t_i]_{v'}$ from which we get $\mathcal{T}[p(t_1, \dots, t_n)]_v = \mathcal{T}[p(t_1, \dots, t_n)]_{v'}$.

Case $\phi \equiv \bigcirc x[\psi]$, $\bigcirc \in \{\forall, \exists\}$. Assume $\phi \equiv \phi(x_1, \dots, x_n)$. We consider two cases.

Sub-case $x \notin FV(\psi)$. Then $x \notin \{x_1, \dots, x_n\}$ and hence for every v_x and v'_x which are x -variants of v and v' respectively we have $\mathcal{T}[\psi(x_1, \dots, x_n)]_{v_x} = \mathcal{T}[\psi(x_1, \dots, x_n)]_{v'_x}$ from which we obtain

$$\begin{aligned} \prod \{\mathcal{T}[\psi]_{v_x} \mid v_x =_{\setminus x} v\} &= \prod \{\mathcal{T}[\psi]_{v'_x} \mid v'_x =_{\setminus x} v'\} \\ \sum \{\mathcal{T}[\psi]_{v_x} \mid v_x =_{\setminus x} v\} &= \sum \{\mathcal{T}[\psi]_{v'_x} \mid v'_x =_{\setminus x} v'\} \end{aligned}$$

which implies $\mathcal{T}[\phi]_v = \mathcal{T}[\phi]_{v'_x}$.

Sub-case $x \in FV(\psi)$. Then $FV(\psi) = \{x, x_1, \dots, x_n\}$. Let $T(x) = \{\mathcal{T}[\psi]_{v_x} \mid v_x =_{\setminus x} v\}$ and $T'(x) = \{\mathcal{T}[\psi]_{v'_x} \mid v'_x =_{\setminus x} v'\}$. Note that $T(x), T'(x) \in \{\{0\}, \{1\}, \{0, 1\}\}$. Assume for some $\bigcirc \in \{\prod, \sum\}$, $\mathcal{T}[\phi]_v \neq \mathcal{T}[\phi]_{v'_x}$. Then $T(x) \neq T'(x)$ which implies there exists $a \in A = |\mathbf{A}|$ such that for $v_x = v[x := a]$ and $v'_x = v'[x := a]$, $\mathcal{T}[\psi]_{v_x} \neq \mathcal{T}[\psi]_{v'_x}$. But this is impossible since $FV(\psi) = \{x, x_1, \dots, x_n\}$, $v_x(x) = a = v'_x(x)$ and for each $x_i \in \{x_1, \dots, x_n\}$, we have $v_x(x_i) = v'_x(x_i)$. Hence $\mathcal{T}[\phi]_v = \mathcal{T}[\phi]_{v'_x}$. QED

Exercise 19.1: Interpretations

1. Define interpretations on universes of discourse containing at least 3 elements and give a valuation in which the following hold. Assume p and q are atomic predicate symbols of any positive arity of your choice.
 - (a) $\neg(p \wedge \exists x[q] \rightarrow \exists x[p \wedge q])$
 - (b) $\neg(\exists x[p \rightarrow q] \rightarrow (\forall x[p] \rightarrow q))$
2. Prove that the following predicates have no models at all where p and q are atomic predicates (of any positive arity).
 - (a) $\neg(p \rightarrow \forall x[q] \rightarrow \forall x[p \rightarrow q])$
 - (b) $\neg((\forall x[p] \rightarrow q) \rightarrow \exists x[p \rightarrow q])$
3. Consider the universe of discourse to be the set of all nodes of graphs and let the atomic binary predicate symbol e stand for the *edge* relation on nodes, i.e. $e(x, y)$ stands for *there is an edge from x to y* . Further let “=” stand for the usual identity relation on nodes. What properties on graphs do the following first-order predicates define?
 - (a) $\forall x[\exists y[\neg(x = y) \wedge e(x, y)]]$
 - (b) $\forall x[\forall y[e(x, y) \rightarrow \neg(x = y)]]$
 - (c) $\forall x[\forall y[\neg(x = y) \rightarrow (e(x, y) \rightarrow e(y, x))]]$
 - (d) $\forall x[\forall y[\forall z[e(x, y) \wedge e(y, z) \rightarrow e(x, z)]]]$

20. Substitutions

20: Substitutions

1. Substitutions
2. Instantiation of Terms
3. The Substitution Lemma for Terms
4. Admissibility
5. Instantiations of Formulae
6. The Substitution Lemma for Formulae

Definition 20.1: Substitutions

- A substitution θ is a (total) function $\theta : V \rightarrow T_\Sigma(V)$ which is almost everywhere the identity.
- $S_\Sigma(V)$ is the set of all substitutions.
- The domain of a substitution θ is the finite set $dom(\theta) = \{x \mid x \not\equiv \theta(x)\}$. θ acts on the variables in $dom(\theta)$.
- θ is called a ground substitution if $FV(\theta(x)) = \emptyset$ for each $x \in dom(\theta)$.

Notation 20.1

Equivalently a substitution θ may be represented as a *finite* (possibly empty) set $\theta = \{s/x \mid \theta(x) = s \not\equiv x\}$ containing only the non-identical elements and their images under θ .

Instantiation of Terms

Definition 20.2: Application of a substitution

Let θ be a substitution.

- The **application** of θ to a term $t \in \mathbb{T}_\Sigma(V)$ is denoted θt and defined inductively as follows.

$$\theta y = y, \quad y \notin \text{dom}(\theta)$$

$$\theta x = \theta(x), \quad x \in \text{dom}(\theta)$$

$$\theta f(t_1, \dots, t_m) = f(\theta t_1, \dots, \theta t_m), \quad f : s^m \rightarrow s \in \Sigma$$

- θt is called a (substitution) instance of t .

The Substitution Lemma for Terms

Lemma 20.1: Substitution lemma for terms

Given a Σ -interpretation (\mathbf{A}, v) , a term t and a substitution $\{s/x\}$, let $\mathcal{V}[s]_v = a \in |\mathbf{A}|$. Then

$$\mathcal{V}[\{s/x\}t]_v = \mathcal{V}[t]_{v[x:=a]}$$

□

This lemma establishes the connection between **finite variants** of a valuation and shows that they may be represented by syntactic substitutions. In particular if $X = \{x_i \mid 1 \leq i \leq n\}$ and $\mathcal{V}[s_i]_v = a_i$ for $1 \leq i \leq n$, then

$$\mathcal{V}[\{s_1/x_1, \dots, s_n/x_n\}t]_v = \mathcal{V}[t]_{v[x_1:=a_1, \dots, x_n:=a_n]}$$

Proof of The Substitution Lemma for Terms 20.1

Proof: By induction on the structure of t .

Case $t \equiv x$. Then $\{s/x\}x \equiv s$ and we have $\mathcal{V}[\{s/x\}t]_v = \mathcal{V}[s]_v = a = \mathcal{V}[t]_{v[x:=a]}$.

Case $t \equiv y \not\equiv x$. Then $\{s/x\}y \equiv y$ and $\mathcal{V}[\{s/x\}t]_v = \mathcal{V}[y]_v = \mathcal{V}[t]_{v[x:=a]}$ since $v(y) = v[x := a](y)$.

Case $t \equiv f(t_1, \dots, t_m)$. Then

$$\begin{aligned}
 & \mathcal{V}[\{s/x\}t]_v \\
 &= \mathcal{V}[\{s/x\}f(t_1, \dots, t_m)]_v \\
 &= \mathcal{V}[\mathbf{f}(\{s/x\}t_1, \dots, \{s/x\}t_m)]_v \\
 &= \mathbf{f}_{\mathbf{A}}(\mathcal{V}[\{s/x\}t_1]_v, \dots, \mathcal{V}[\{s/x\}t_m]_v) \\
 &= \mathbf{f}_{\mathbf{A}}(\mathcal{V}[t_1]_{v[x:=a]}, \dots, \mathcal{V}[t_m]_{v[x:=a]}) \quad \text{By the induction hypothesis} \\
 &= \mathcal{V}[\mathbf{f}(t_1, \dots, t_m)]_{v[x:=a]}
 \end{aligned}$$

QED

Free variable capture. Much care is required to actually prevent the so-called “capture” of free variables during the application of a substitution, because “free variable capture” by a binding of the same variable name (within the scope of the binding) can alter the meaning of expressions to some wholly unintended one¹³. The following example illustrates the meaning of this term

Example 20.1: Free variable capture

Consider the expression on numbers $\sum_{z=1}^n y$ which may have been obtained as a result of some derivation or some calculation. Assuming all the symbols here have their usual meanings in the integers, we may safely assume that for any value of the free variables n and y , the value of this expression should be ny (i.e the product of n and y). So if we were to substitute any integer-valued function such as $y = x^2$, we expect that on substituting x^2 for y would yield the result nx^2 . However, if it so happened that y was defined to be $y = z^2$ instead, then the corresponding result obtained is unfortunately not nz^2 . Instead it becomes the sum of the first n squares of positive integers which is an unintended result of the substitution because of the confusion between two different occurrences of the variable z . This happens because the variable z which is free in the definition $y = z^2$ has got “captured” by the binding of z in $\sum_{z=1}^n y$.

So whereas $\{x^2/y\}$ is a substitution which yields $\sum_{z=1}^n x^2$, the substitution $\{z^2/y\}$ does not yield $\sum_{z=1}^n z^2$ since the free variable z would be “captured” by the binding $\sum_{z=1}^n$ in the host term. This is expressed by saying that the substitution $\{z^2/y\}$ is admissible in $\sum_{z=1}^n y$ though it leaves the expression $\sum_{z=1}^n y$ unchanged. The free variable z (unlike the bound variable z in the expression $\sum_{z=1}^n y$) has a meaning and significance outside the scope of the expression $\sum_{z=1}^n y$. The bound variable z , on the other hand, has its meaning and significance limited to the expression $\sum_{z=1}^n y$ and has no other significance outside it.

The only way to perform the substitution $\{z^2/y\}$ in the sub-term y is to first ensure that all bound occurrences of z in the expression $\sum_{z=1}^n y$ are renamed to some variable which preferably does not clash with any other variable in that occurs in

¹³Loosely put, you might get wrong or wholly unexpected and unintended answers.

$\Sigma_{z=1}^n y$. However $\Sigma_{z=1}^n y$ may equally well be written $\Sigma_{u=1}^n y$ without changing the meaning of the expression (see definition 0.2). It would then be safe to apply the substitution $\{z^2/y\}$ to $\Sigma_{u=1}^n y$ to yield the expression $\Sigma_{u=1}^n z^2$

We therefore require to define when a substitution of a term for a variable is “admissible” in a given context and what the result of that substitution would be.

Admissibility

Definition 20.3: Admissibility

Let θ be a substitution and ϕ a Σ -formula.

1. $s/x \in \theta$ is admissible for a formula ϕ if the (free) variables of s remain free after instantiating the formula.
2. θ is admissible for ϕ if every element of θ is admissible for ϕ .

Let θ be a substitution. An element $s/x \in \theta$ is admissible for

- $\partial y[\phi]$ if $x \not\equiv y$, $y \notin Var(s)$ and s/x is admissible for ϕ .
- $p(t_1, \dots, t_n)$,
- $\neg\phi$ if it is admissible for ϕ ,
- $(\phi \odot \psi)$ if it is admissible for both ϕ and ψ , $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
- $\partial x[\phi]$ (but the substitution leaves the formula unchanged since $x \notin \partial x[\phi]$),

Note.

1. Substitution $\{s/x\}$ is not admissible for a formula of the form $\mathcal{O}y[\phi]$ if $y \in Var(s)$ because of the danger of free variable capture. In such a situation the substitution is effected by uniformly renaming all *bound* variables in $\mathcal{O}y[\phi]$ that may clash with the variables of s . For simplicity assume we need to perform the substitution $\{s/x\}\mathcal{O}y[\phi]$ where $y \in Var(s)$. Choose a variable z such that $z \notin Var(s)$ and $z \notin FV(\mathcal{O}y[\phi])$ ¹⁴. Now perform the substitution(s) $\{s/x\}\mathcal{O}z[\{z/y\}\phi]$. This will be effective even if z occurs bound in ϕ and y occurs free in the scope of the binding of z . This is because the substitution $\{z/y\}\phi$ would not be admissible since $z \in Var(z)$ and can be potentially captured by the binding of z , in which case another renaming of the bound variable z in ϕ would be necessitated in order to perform the substitution successfully. However this process cannot go on indefinitely (even theoretically) since all formulae are finite.
2. It is possible that $x \in Var(s)$ in $\{s/x\}$. Then s takes the place of all free occurrences of x in ϕ .
3. Substitution $\{s/x\}$ is admissible for a formula $\mathcal{O}x[\phi]$, but since there are no free occurrences of x in $\mathcal{O}x[\phi]$, the substitution has no effect.
4. We have defined our notion of admissibility assuming that the signature Σ contains no operator schemas (see definitions -1.8 and -1.9) which create their own bindings of variables in terms. In the presence of operator schemas our definition of admissibility of a susbtitution would also have to ensure that no free variable in the host term is captured because of the substitution of a free variable in the predicate by a term with an operator schema which occurs in the range of the substitution. That is, if $s/x \in \theta$ and z occurs bound in s and $z \in FV(\phi)$ then none of the free occurrences of z in ϕ should be captured by the binding of z in s as a result of the substitution $\{s/x\}\phi$. To ensure this, we would then need

¹⁴This is necessary since one does not want to inadvertently bind variables that are free in the predicate, by renaming some other bound variable with the same name!

to rename all the bound occurrences of z in s by a new variable.

5. The above rules and conditions are quite confusing and often the best policy when choosing a new variable name for renaming, is to choose one which has no occurrences at all in the domain, range and the host of the substitution. In any automated system, this is often done by maintaining a *symbol table* and generating new names (which do not already occur in the symbol table) and entering them as and when they are generated.

Instantiations of Formulae

Definition 20.4: Substitution instantiation

Let θ be a substitution which is admissible for ϕ .

- The **application** of θ to a formula $\phi \in \mathcal{P}_1(\Sigma)$ is denoted $\theta\phi$ and defined inductively as follows.

$$\theta p(t_1, \dots, t_n) = p(\theta t_1, \dots, \theta t_n), \quad p : s^n \in \Sigma$$

$$\theta \neg \psi = \neg(\theta \psi),$$

$$\theta(\psi \odot \chi) = (\theta \psi \odot \theta \chi), \quad \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

$$\theta \mathcal{O}x[\psi] = \mathcal{O}x[\theta' \psi], \quad \theta' = \theta - \{\theta(x)/x\}, \quad \mathcal{O} \in \{\forall, \exists\}$$

Fact 20.1

If θ is a substitution which is admissible for ϕ then every $\theta' \subseteq \theta$ is also admissible for ϕ .

The Substitution Lemma for Formulae

Lemma 20.2: Substitution lemma for formulae

Given a Σ -interpretation (\mathbf{A}, v) , a formula ϕ and an admissible substitution $\{s/x\}$, let $\mathcal{V}[\![s]\!]_v = a \in |\mathbf{A}|$. Then $\mathcal{T}[\!\{s/x\}\phi]\!]_v = \mathcal{T}[\!\phi]\!]_{v[x:=a]}$.



Proof of The Substitution Lemma for Formulae 20.2

Proof: Case $x \notin FV(\phi)$. Then it trivially follows that $\{s/x\}\phi \equiv \phi$ and $\mathcal{T}[\{s/x\}\phi]_v = \mathcal{T}[\phi]_{v_x}$ for all x -variants of v .

Case $x \in FV(\phi)$. We proceed by induction on the structure of ϕ .

Sub-case $\phi \equiv p(t_1, \dots, t_n)$. We have

$$\begin{aligned} &= \mathcal{T}[\{s/x\}\phi]_v \\ &= \mathcal{T}[\{s/x\}p(t_1, \dots, t_n)]_v \\ &= \mathcal{T}[p(\{s/x\}t_1, \dots, \{s/x\}t_n)]_v \\ &= \mathcal{T}[p(t_1, \dots, t_n)]_{v[x:=a]} \quad \text{By lemma 20.1} \end{aligned}$$

The sub-cases involving the propositional connectives are trivial and the only interesting cases left are those of quantified formulae.

Sub-case $\phi \equiv \exists y[\psi]$. Since $x \in FV(\phi)$ clearly $x \not\equiv y$ and hence $x \in FV(\psi)$. Further since $\{s/x\}$ is admissible for ϕ , $y \notin Var(s)$. This implies that

1. $\{s/x\}$ is admissible for ψ ,
2. $\{s/x\}\phi \equiv \exists y[\{s/x\}\psi]$ and
3. $a = \mathcal{V}[s]_v = \mathcal{V}[s]_{v[y:=b]}$ for arbitrary $b \in |\mathbf{A}|$ since $y \notin Var(s)$.

By the induction hypothesis for any $b \in |\mathbf{A}|$ we have

$$\mathcal{T}[\{s/x\}\psi]_{v[y:=b]} = \mathcal{T}[\psi]_{v[y:=b][x:=a]} = \mathcal{T}[\psi]_{v[x:=a][y:=b]}$$

from which we get

$$\{\mathcal{T}[\{s/x\}\psi]_{v_y} \mid v_y =_{\backslash y} v\} = \{\mathcal{T}[\psi]_{v[x:=a]_y} \mid v[x := a]_y =_{\backslash y} v[x := a]\}$$

which implies for any quantifier $\mathcal{T}[\{s/x\}\phi]_v = \mathcal{T}[\phi]_{v[x:=a]}$ regardless of the quantifier involved.

QED

Exercise 20.1: Substitutions and coincidence

1. Let (\mathbf{A}_1, v_1) be a Σ_1 -interpretation and (\mathbf{A}_2, v_2) a Σ_2 -interpretation such that $|\mathbf{A}_1| = |\mathbf{A}_2|$. Let $\Sigma = \Sigma_1 \cap \Sigma_2$.
 - (a) Let t be a Σ -term. Prove that if (\mathbf{A}_1, v_1) and (\mathbf{A}_2, v_2) agree on the symbols occurring in t , then $\mathcal{V}_{\mathbf{A}_1}[\![t]\!]_{v_1} = \mathcal{V}_{\mathbf{A}_2}[\![t]\!]_{v_2}$.
 - (b) Let ϕ be a Σ -formula. Prove that if (\mathbf{A}_1, v_1) and (\mathbf{A}_2, v_2) agree on the symbols occurring in ϕ , then $\mathcal{T}_{\mathbf{A}_1}[\![\phi]\!]_{v_1} = \mathcal{T}_{\mathbf{A}_2}[\![\phi]\!]_{v_2}$.

21. Models, Satisfiability and Validity

21: Models, Satisfiability and Validity

Everything is vague to a degree you do not realise till you have tried to make it precise.

Bertrand Russell, *Philosophy of Logical Atomism*, 1918

1. Satisfiability
2. Models and Consistency
3. Examples of Models:1
4. Other Models
5. Examples of Models:2
6. Examples of Models:3
7. Logical Consequence
8. Validity
9. Validity of Sets of Formulae
10. Negations of Semantical Concepts

Satisfiability

Definition 21.1: Satisfaction

- A Σ -interpretation (\mathbf{A}, v) satisfies a Σ -formula ϕ denoted $(\mathbf{A}, v) \Vdash \phi$ if and only if $\mathcal{T}[\![\phi]\!]_v = 1$.
- ϕ is said to be **satisfiable** in \mathbf{A} if there is a valuation v such that $(\mathbf{A}, v) \Vdash \phi$.
- A Σ -formula ϕ is **satisfiable** if there exists a Σ -interpretation that satisfies it.

Models and Consistency

Definition 21.2: Models

- A Σ -structure \mathbf{A} is a **model** (or more accurately a Σ -**model**) of a Σ -formula $\phi \in \mathcal{P}_1(\Sigma)$ (denoted $\mathbf{A} \Vdash \phi$) if and only if for all valuations v , $(\mathbf{A}, v) \Vdash \phi$.
- For a nonempty set Φ of Σ -formulas,
 - a Σ -structure \mathbf{A} is a Σ -**model** of Φ , (denoted $\mathbf{A} \Vdash \Phi$) if and only if it is a model of every formula in Φ .
 - Φ is said to be an **axiom system** for all models of Φ .
 - Φ is said to be **consistent** iff it has a Σ -**model**.

Examples of Models:1

Example 21.1

Let $\Sigma = \{0 : \longrightarrow s, +1 : s \longrightarrow s, = : s^2, < : s^2\}$, let

$$\mathbf{N} = \langle \mathbb{N}; 0, +1, =, < \rangle$$

be the Σ -structure where \mathbb{N} is the set of naturals, $+1$ is the unary successor function, $=$ is the atomic binary equality predicate and $<$ is the binary “less-than” predicate. Let

$$\begin{array}{ll} \phi_1 \stackrel{df}{=} \neg(+1(x) = 0) & \phi_2 \stackrel{df}{=} (+1(x) = +1(y)) \rightarrow (x = y) \\ \phi_3 \stackrel{df}{=} \forall x \exists y [y = +1(x)] & \phi_4 \stackrel{df}{=} \forall x [\neg(x = 0) \rightarrow \exists y [x = +1(y)]] \end{array}$$

We have $\mathbf{N} \Vdash \Phi$ where $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$

Other Models

A consistent set of axioms may describe not just one model but a class of models (some of them wholly unintended by the propounder of the axioms!).

Example 21.2

Consider an alphabet consisting of a single symbol $|$. Consider the carrier set $|^*$ consisting of finite sequences of $|$ (including the empty sequence denoted by ϵ)

$$\text{Bars} = \langle |^*; \epsilon, |.; =, < \rangle$$

where for any $x \in |^*$, $|.x = |x$, $=$ is the usual equality relation on sequences of symbols and $<$ is the “proper prefix” ordering on sequences.

It is easy to verify that **Bars** $\Vdash \Phi$ and is hence a model of the axioms of example 21.1.

21.1. An excursion into Ordinals

For a given set of non-logical axioms it may be possible to construct several interesting models. For one we show that an alternative model satisfying the axioms that we have given for the naturals (example 21.1) may be easily constructed inductively using only sets starting from the empty set. Just to make clear the similarities and the differences of our construction with what we usually associate as the naturals we use underscored versions of symbols for the naturals. Let

$$\underline{0} \stackrel{\text{df}}{=} \emptyset$$

This is called the “zeroth limit ordinal”. It is also the “smallest finite ordinal”. This is the basis of our inductive construction. The inductive step for the construction of the successor ordinal $\underline{n+1}$ from \underline{n} for each natural number $n > 0$ is defined as

$$\underline{n+1} \stackrel{\text{df}}{=} \underline{n} \cup \{\underline{n}\}$$

That is, each successor ordinal (i.e. the image of the function $+1$ applied to each ordinal) is obtained from the previous ordinal by taking its union with the singleton set containing the previous ordinal. Notice that this inductive construction

yields

$$\begin{aligned}
 \underline{0} &= \emptyset \\
 \underline{1} &= \{\underline{0}\} \\
 \underline{2} &= \{\underline{0}, \underline{1}\} \\
 \underline{3} &= \{\underline{0}, \underline{1}, \underline{2}\} \\
 \vdots &\quad \vdots \quad \vdots \\
 \underline{n+1} &= \{\underline{0}, \underline{1}, \underline{2}, \dots, \underline{n}\} \\
 \vdots &\quad \vdots \quad \vdots
 \end{aligned}$$

and so on *ad infinitum*.

The ordinals satisfy the following properties for each natural m, n , which leads us to the definition of $<$ on the ordinals. To put it succinctly we have

$$\underline{m} < \underline{n} \stackrel{df}{=} \underline{m} \in \underline{n} \text{ iff } \underline{m} \subset \underline{n} \text{ iff } m < n \quad (41)$$

and by obvious extensions to the definitions of \leq , $>$ and \geq respectively.

Further notice that for each natural n we have

$$\underline{n} = \{\underline{m} \mid \underline{m} < \underline{n}\} = \bigcup_{\underline{m} < \underline{n}} \underline{m} = \{\underline{m} \mid m < n\} = \bigcup_{m < n} \underline{m} \quad (42)$$

Each ordinal is a set consisting of all the ordinals that are less than it. By extension there is absolutely nothing that prevents

us from considering the set consisting of all the finite ordinals,

$$\underline{\omega} \stackrel{df}{=} \bigcup_{m \in \mathbb{N}} \underline{m} \quad (43)$$

and claiming it to be an ordinal too. $\underline{\omega}$ consists of all the finite ordinals and is greater than any of them. We see that for all $n \in \mathbb{N}$, by (41) we have that $\underline{n} < \underline{\omega}$ since $\underline{n} \in \underline{\omega}$ (equivalently $\underline{n} \subset \underline{\omega}$). Hence $\underline{\omega}$ is also an ordinal and consists of all the ordinals that are less than itself. $\underline{\omega}$ is the “first limit” ordinal that is “transfinite”. But our ordinal construction (unlike in the case \mathbb{N}) need not stop here. We could apply the same definition and come up with successor transfinite ordinals too.

$$\begin{aligned}
 \underline{\omega + 1} &\stackrel{df}{=} \underline{\omega} \cup \{\underline{\omega}\} \\
 &= \{\underline{0}, \underline{1}, \dots, \underline{\omega}\} \\
 \underline{\omega + 2} &\stackrel{df}{=} \underline{\omega + 1} \cup \{\underline{\omega + 1}\} \\
 &= \{\underline{0}, \underline{1}, \dots, \underline{\omega}, \underline{\omega + 1}\} \\
 \underline{\omega + 3} &\stackrel{df}{=} \underline{\omega + 2} \cup \{\underline{\omega + 2}\} \\
 &= \{\underline{0}, \underline{1}, \dots, \underline{\omega}, \underline{\omega + 1}, \underline{\omega + 2}\} \\
 &\vdots && \vdots \\
 \underline{\omega + n + 1} &\stackrel{df}{=} \underline{\omega + n} \cup \{\underline{\omega + n}\} \\
 &= \{\underline{0}, \underline{1}, \dots, \underline{\omega}, \underline{\omega + 1}, \underline{\omega + 2}, \dots, \underline{\omega + n}\} \\
 &\vdots && \vdots
 \end{aligned}$$

which brings us to the next limit ordinal viz.

$$\underline{\omega + \omega} \stackrel{df}{=} \{\underline{0}, \underline{1}, \dots, \underline{\omega}, \underline{\omega + 1}, \underline{\omega + 2}, \dots, \underline{\omega + n}, \dots\} \quad (44)$$

which we refer to as $\underline{\omega.2}$ (and not $\underline{2\omega}$) (we could also identify $\underline{\omega}$ with $\underline{\omega.1}$ and $\underline{0}$ with $\underline{\omega.0}$). Proceeding further in a similar manner we get the sequence of ordinals

$$\underline{0}, \dots, \underline{\omega} = \underline{\omega.1}, \dots, \underline{\omega.2}, \dots, \underline{\omega.3}, \dots, \underline{\omega.n}, \dots, \underline{\omega.\omega} \quad (45)$$

We refer to $\omega \cdot \omega$ as ω^2 . We could continue this even further obtaining ordinals,

$$\underline{0}, \dots, \underline{\omega} = \underline{\omega^1}, \dots, \underline{\omega^2}, \dots, \underline{\omega^3}, \dots, \underline{\omega^n}, \dots, \underline{\omega^\omega} \quad (46)$$

There really is no reason to stop here. Proceeding further from ω^ω we obtain

$$\underline{0}, \dots, \underline{\omega^\omega} = \underline{\omega^{(\omega^1)}}, \dots, \underline{\omega^{(\omega^2)}}, \dots, \underline{\omega^{(\omega^3)}}, \dots, \underline{\omega^{(\omega^n)}}, \dots, \underline{\omega^{(\omega^\omega)}} \quad (47)$$

Following the normal precedence convention for exponentials and assuming that exponentiation is right associative (see subsection 2.3) we may write the above sequence without the messy parentheses as follows.

$$\underline{0}, \dots, \underline{\omega^\omega} = \underline{\omega^{\omega^1}}, \dots, \underline{\omega^{\omega^2}}, \dots, \underline{\omega^{\omega^3}}, \dots, \underline{\omega^{\omega^n}}, \dots, \underline{\omega^{\omega^\omega}} \quad (48)$$

We really don't need to stop here and we could actually take it further and create a "staircase of ω s with an ω number of steps" as shown below

$$\underline{\omega}, \dots, \underline{\omega^\omega}, \dots, \underline{\omega^{\omega^\omega}}, \dots, \underline{\omega^{\omega^{\omega^\omega}}}, \dots, \underline{\omega^{\omega^{\omega^\omega}}}, \dots \quad (49)$$

Let's stop here. Even though this construction is quite mind-boggling, it turns out that all these ordinals are only countable – i.e. we have not even scratched the surface of uncountability!

Exercise 21.1: Ordinals

1. Prove that $\omega + 1$ is a countable set.
2. Prove that ω^2 is a countable set.
3. Prove that each of the transfinite ordinals described in (45), (46), (47), (48) and (49) is countable.
4. Assuming that an ordinal α is countable, prove that its successor $\alpha + 1$ is also countable.
5. Assuming that $\beta \stackrel{df}{=} \bigcup_{\alpha < \beta} \alpha$ and each α is a countable ordinal, prove that β is also countable.
6. Assuming that the class of ordinals can be further extended “*ad infinitum*”, which of the axioms in example 21.1 does this class fail to satisfy? Justify your answer with an example from the transfinite ordinals.
7. If we limit ourselves to the class of ordinals ending with the last ordinal in (49) which other axioms of example 21.1 does it fail to satisfy?

Examples of Models:2

Example 21.3: Groups

Let $\Sigma = \{0 : \longrightarrow s, + : s^2 \longrightarrow s, = : s^2\}$ and let \mathbf{Z} be the Σ -structure

$$\mathbf{Z} = \langle \mathbb{Z}; 0, +; = \rangle$$

where \mathbb{Z} is the set of integers, 0 and $+$ represent the integer zero and the binary addition operation respectively, and $=$ is the atomic binary equality predicate. \mathbf{Z} is a model of the following set Φ of formulae

$$\phi_{associative} \stackrel{df}{=} \forall x, y, z[(x + y) + z = x + (y + z)] \quad (50)$$

$$\phi_{identity} \stackrel{df}{=} \forall x[x + 0 = x] \quad (51)$$

$$\phi_{right-inverse} \stackrel{df}{=} \forall x \exists y [x + y = 0] \quad (52)$$

Examples of Models:3

Example 21.4: Abelian groups

The set Φ defined in example 21.3 is the set of axioms which defines the notion of a *group* in algebra. The addition of an extra axiom

$$\phi_{commutative} \stackrel{df}{=} \forall x, y [x + y = y + x] \quad (53)$$

i.e. $\Phi' = \Phi \cup \{\phi_{commutative}\}$ excludes all non-commutative groups from the models of the set Φ' .

Logical Consequence

Definition 21.3: Logical consequence

A Σ -formula ψ is a logical consequence of a set Φ of Σ -formulae, denoted $\Phi \models \psi$, if and only if every model \mathbf{A} of Φ is also a model of ψ . If Φ is empty then ψ is said to be logically valid (denoted $\models \psi$).

Example 21.5

Let Σ be the signature in example 21.3 and let the formula

$$\phi_{\text{left-inverse}} \stackrel{\text{df}}{=} \forall x \exists y [y + x = 0] \quad (54)$$

A typical proof in a mathematics text that $\phi_{\text{left-inverse}}$ is a logical consequence of the axioms of group theory (example 21.3) might go as follows.

Proof: Let x be any element. Then by axiom $\phi_{right-inverse}$ there exists y such that

$$x + y = 0 \quad (55)$$

Again by $\phi_{right-inverse}$ for some z we get

$$y + z = 0 \quad (56)$$

We then have

$$\begin{aligned} y + x &= (y + x) + 0 && (\phi_{identity}) \\ &= (y + x) + (y + z) && (56) \\ &= y + (x + (y + z)) && (\phi_{associative}) \\ &= y + ((x + y) + z) && (\phi_{associative}) \\ &= y + (0 + z) && (55) \\ &= (y + 0) + z && (\phi_{associative}) \\ &= y + z && (\phi_{identity}) \\ &= 0 && (56) \end{aligned}$$

QED

Effectively from the group axioms we have extracted fresh knowledge about groups in general namely that, **the existence of a right inverse for each element of the group implies the existence of a left-inverse.**

Example 21.6

Let Σ be the signature in example 21.3. The formula

$$\phi_{left-right-inverse} \stackrel{df}{=} \forall x, y [(x + y = 0) \rightarrow (y + x = 0)] \quad (57)$$

is logical consequence of the **group axioms**.

Proof:

Replace the opening line

“Let x be any element. Then by axiom $\phi_{right-inverse}$ there exists y such that”

of the proof of formula (54) by the line

“Let x and y be elements such that”

and copy the rest of the proof verbatim.

QED

The formula (57) actually makes a “stronger” statement about groups in general viz. that **the right inverse of an element not only exists, but is its left inverse as well.**

Validity

Definition 21.4: Validity

Let \mathbf{A} be a Σ -structure, ϕ and ψ be Σ -formulae.

- $(\mathbf{A} \Vdash \phi)$. ϕ is valid in \mathbf{A} if and only if \mathbf{A} is a model of ϕ .
- $(\mathfrak{A} \Vdash \phi)$. ϕ is valid in a class \mathfrak{A} of Σ -structures if it is valid in each structure $\mathbf{A} \in \mathfrak{A}$.
- $(\Vdash \phi)$. ϕ is (logically) valid if and only if every Σ -structure is a model of ϕ .
- $(\phi \Leftrightarrow \psi)$. ϕ is (logically) equivalent to ψ if $\Vdash \phi \leftrightarrow \psi$.

Validity of Sets of Formulae

Definition 21.5: Validity of a set of formulae

Let Φ be a set of Σ -formulae.

- ($\mathbf{A} \Vdash \Phi$). Φ is valid in \mathbf{A} if and only if \mathbf{A} is a model of each $\phi \in \Phi$.
- ($\mathfrak{A} \Vdash \Phi$). Φ is valid in a class \mathfrak{A} of Σ -structures if $\mathbf{A} \Vdash \Phi$ for each $\mathbf{A} \in \mathfrak{A}$.
- ($\Vdash \Phi$). Φ is valid if and only if every Σ -structure is a model of each $\phi \in \Phi$.

21.2. Meta-operators and overloading

We have deviated in our meta-logical notation from standard textbooks. Notice from the definitions of **logical consequence**, **logical validity** and **validity of sets of formulae** that $\Phi \models \psi$ if and only if for every structure Σ -structure \mathbf{A} , $\mathbf{A} \Vdash \Phi$ implies $\mathbf{A} \Vdash \psi$. Most textbooks on first-order logic actually use only one overloaded symbol \models for both concepts. This is due to the fact that a set of axioms or axiom schemas often define the class of models. As an example we have seen that the class \mathfrak{G} of **groups** is exactly the class of models that satisfy the set Φ of group axioms. If we further consider the properties $\phi_{\text{left-inverse}}$ (example 21.5) and $\phi_{\text{left-right-inverse}}$ (example 21.6) we see that $\mathfrak{G} \Vdash \Phi$ and since $\Phi \models \phi_{\text{left-inverse}}$ and $\Phi \models \phi_{\text{left-right-inverse}}$, we also have $\mathfrak{G} \Vdash \phi_{\text{left-inverse}}$ and $\mathfrak{G} \Vdash \phi_{\text{left-right-inverse}}$.

However we have decided to keep the two symbols \Vdash and \models separate since the concept of logical consequence involves sets of **formulae** of a formal language whereas validity (as opposed to logical validity) of a formula refers to **models**.

Notice also that for any formula ϕ , both $\Vdash \phi$ and $\models \phi$ denote that ϕ is **(logically) valid**. By extension, therefore **logical equivalence** defined as $\models \phi \leftrightarrow \psi$ may equally well be defined as $\Vdash \phi \leftrightarrow \psi$ for all sentences (i.e. closed formulae) of first-order logic. But an important distinction exists when we refer to the notion of “equivalence” of formulae. For example, $\phi_{\text{left-inverse}} \not\leftrightarrow \phi_{\text{left-right-inverse}}$, i.e. $\not\models \phi_{\text{left-inverse}} \leftrightarrow \phi_{\text{left-right-inverse}}$. However the two formulae $\phi_{\text{left-inverse}}$ and $\phi_{\text{left-right-inverse}}$ are equivalent for every model in \mathfrak{G} since they are true for exactly the same class of models. Equivalently, it follows from example 21.5 that $\Phi \models \phi_{\text{left-inverse}}$ and from example 21.6 that $\Phi \models \phi_{\text{left-right-inverse}}$ and hence $\Phi \models \phi_{\text{left-inverse}} \wedge \phi_{\text{left-right-inverse}}$ and since $\phi_{\text{left-inverse}} \wedge \phi_{\text{left-right-inverse}} \models \phi_{\text{left-inverse}} \leftrightarrow \phi_{\text{left-right-inverse}}$ we also have $\Phi \models$

$\phi_{left-inverse} \leftrightarrow \phi_{left-right-inverse}$. But that still does not make $\phi_{left-inverse}$ logically equivalent to $\phi_{left-right-inverse}$.

Further, $\Vdash \phi \leftrightarrow \psi$ and $\models \phi \leftrightarrow \psi$ are not equivalent as meta-logical concepts for formulae with free variables (see problem 10 in exercises 21.3)

Exercise 21.2: First-order validity and consequence

1. Consider the axioms ϕ_1 and ϕ_2 of example 21.1 replaced by

$$\phi'_1 \stackrel{df}{=} \forall x[\neg(+1(x) = 0)] \quad \phi'_2 \stackrel{df}{=} \forall x[(+1(x) = +1(y)) \rightarrow (x = y)] \ n$$

- (a) Does **N** $\Vdash \Phi' = \{\phi'_1, \phi'_2, \phi_3, \phi_4\}$ hold? Why or why not?
 - (b) Does **Bars** $\Vdash \Phi' = \{\phi'_1, \phi'_2, \phi_3, \phi_4\}$ hold? Why or why not?
 - (c) Let $\phi''_2 \stackrel{df}{=} \forall x, y[(+1(x) = +1(y)) \rightarrow (x = y)]$ and let $\Phi'' = \{\phi'_1, \phi''_2, \phi_3, \phi_4\}$. Then do the following hold? Why or why not?
 - i. **Bars** $\Vdash \Phi''$
 - ii. **N** $\Vdash \Phi''$
2. Based on your answers to the above question what can you say is the relation between free variables in the axioms of a model, validity and logical consequence?
3. If instead of universally quantifying the statements, we had existentially quantified them, would your answers to the questions in the problems change? Justify.

Negations of Semantical Concepts

- We use the symbols \nVdash , $\not\models$, $\not\leftrightarrow$ to denote the negations of the corresponding relations.

Exercise 21.3: Satisfiability, models and validity

1. Prove that \mathbf{A} is a model of ϕ iff \mathbf{A} is a model of $\vec{\forall}[\phi]$.
2. Prove that $\Phi \models \psi$ iff $\vec{\forall}[\Phi] \models \vec{\forall}[\psi]$, where $\vec{\forall}[\Phi]$ denotes the universal closure of each formula in Φ .
3. Prove that $\Vdash \phi$ if and only if $\models \phi$ (i.e. $\emptyset \models \phi$). Hence in most books on logic, the same symbol \models is used for both logical consequence and for validity in models).
4. Prove that $\Vdash \phi$ if and only if $\Vdash \vec{\forall}[\phi]$.
5. Prove that ϕ is satisfiable in \mathbf{A} if and only if $\vec{\exists}[\phi]$ is satisfiable in \mathbf{A} .
6. Show that $\phi \vee \neg\phi$ is valid for any formula ϕ .
7. In general every first-order logic formula which has a tautological “shape” in propositional logic is a valid formula. Formalize this notion and prove it.
8. Let $\mathcal{O}x[\psi] \in SF(\phi)$ and $y \notin FV(\psi)$. Let ϕ' be obtained from ϕ by replacing $\mathcal{O}x[\psi]$ by $\mathcal{O}y[\{y/x\}\psi]$. Then ϕ and ϕ' are said to be α -equivalent and denoted by $\phi \equiv_\alpha \phi'$. Prove that two α -equivalent formulae are equivalent.
9. If $x \notin FV(\psi)$ then $\mathcal{O}x[\psi]$ is equivalent to ψ .
10. A student claimed that the following definition is a stronger definition of logical consequence than definition 21.3. Justify or refute his claim.

Definition 21.6

A Σ -formula ψ is a **logical consequence** of a set Φ of Σ -formulae, denoted $\Phi \models' \psi$, if and only if for every interpretation (\mathbf{A}, v) , $(\mathbf{A}, v) \Vdash \phi$ for each $\phi \in \Phi$ implies $(\mathbf{A}, v) \Vdash \psi$.

Exercise 21.4: Validity and Invalidity

1. Prove that \forall and \exists are “duals” i.e. show that

- (a) $\Vdash \forall x[\phi] \leftrightarrow \neg \exists x[\neg\phi]$
- (b) $\Vdash \exists x[\phi] \leftrightarrow \neg \forall x[\neg\phi]$

2. Prove that for any binary predicate ψ ,

$$\forall x, y[\psi(x, y) \rightarrow \psi(y, x)] \Leftrightarrow \forall x, y[\psi(x, y) \leftrightarrow \psi(y, x)]$$

3. Give examples of interpretations (A, v) to show that the following formulae are not valid.

- (a) $\exists x[\psi] \rightarrow \forall x[\psi]$
- (b) $\forall x \exists y[\psi(x, y)] \rightarrow \exists y \forall x[\psi(x, y)]$

4. Prove that the following formulae are valid.

- (a) $\exists x[\psi(x) \rightarrow \forall x[\psi(x)]]$.
- (b) $\exists x \forall y[\psi(x, y)] \rightarrow \forall y \exists x[\psi(x, y)]$

5. Prove that $\phi \Leftrightarrow \psi$ iff $\phi \models \psi$ and $\psi \models \phi$.

21.3. Some more Model Theory

When we consider the notion of a model, we come across various notions which express properties of the models. For example, if $<$ denotes an irreflexive ordering relation, then a sentence such as $\forall x \exists y [x < y]$ specifies that there is no greatest element in the ordering. Such a sentence therefore has no finite models. Its negation however has only finite models. Another sentence such as $\forall x \forall y [x = y]$ in the language of first-order logic with equality has only singleton models.

If all models of a set Φ of sentences are finite then there must be a *finite bound on the size of the models* (i.e. the carrier set of the structure must be of a fixed finite cardinality). Otherwise, as the following theorem shows there would also be infinite models. For instance as Enderton [4] states:

It is a priori conceivable that there might be some very subtle equation of group theory that was true in every finite group but false in every infinite group.

But theorem 21.1 assures us that such a possibility does not exist.

Theorem 21.1

If a set Φ of Σ -formulae has arbitrarily large finite models, then it has an infinite model.

Proof: Assume Φ has arbitrarily large finite models. Now consider the following formulae ψ_k for each $k \geq 2$,

$$\begin{aligned}\psi_2 &\stackrel{df}{=} \exists \textcolor{violet}{x}_1, \textcolor{violet}{x}_2 [\neg(\textcolor{violet}{x}_1 = \textcolor{violet}{x}_2)] \\ \psi_3 &\stackrel{df}{=} \exists \textcolor{violet}{x}_1, \textcolor{violet}{x}_2, \textcolor{violet}{x}_3 [\neg(\textcolor{violet}{x}_1 = \textcolor{violet}{x}_2) \wedge \neg(\textcolor{violet}{x}_1 = \textcolor{violet}{x}_3) \wedge \neg(\textcolor{violet}{x}_2 = \textcolor{violet}{x}_3)] \\ &\vdots \quad \vdots \quad \vdots\end{aligned}$$

where each ψ_k states that there are at least k distinct elements in the model. Now consider the set $\Psi = \Phi \cup \{\psi_k \mid k \geq 2\}$. By hypothesis, every finite subset of Ψ has a model. By the compactness theorem therefore Ψ also has a model. However clearly no model of Ψ can be finite. Hence Ψ must have an infinite model. This infinite model is also a model of Φ since $\Phi \subset \Psi$. QED

22. Structures and Substructures

22: Structures and Substructures

1. Satisfiability and Expansions
2. Distinguishability
3. Evaluations under Different Structures
4. Isomorphic Structures
5. The Isomorphism Lemma
6. Substructures
7. Substructure Facts
8. Substructure Examples
9. Quantifier-free Formulae
10. Lemma on Quantifier-free Formulae
11. Universal and Existential Formulae
12. The Substructure Lemma

Satisfiability and Expansions

Our notions of satisfiability, consequence and validity are all with respect to a specific signature.

Lemma 22.1

Let $\Sigma_1 \subseteq \Sigma_2$. For any set Φ of Σ_1 -formulae, Φ is satisfiable with respect to Σ_1 iff Φ is satisfiable with respect to Σ_2 .

Proof: Essentially the interpretations of the common symbols should coincide, whereas the interpretations of the symbols in $\Sigma_2 - \Sigma_1$ may be arbitrary. It then follows from problem 1 of exercise 20.1. QED

Distinguishability

Example 22.1

For $\Sigma = \{= : s^2, < : s^2\}$ consider the Σ -structures $\mathbf{Z} = \langle \mathbb{Z}; ; =, < \rangle$ and $\mathbf{Q} = \langle \mathbb{Q}; ; =, < \rangle$ where \mathbb{Z} is the set of integers, \mathbb{Q} is the set of rational numbers, $<$ are respectively the “less-than” relations on the two sets respectively. Now consider the formula

$$\phi_{density} \stackrel{df}{=} \forall x, z [x < z \rightarrow \exists y [(x < y) \wedge (y < z)]]$$

Clearly $\mathbf{Q} \Vdash \phi_{density}$ whereas $\mathbf{Z} \nvDash \phi_{density}$.

$\phi_{density}$ distinguishes the two structures.

Evaluations under Different Structures

When evaluating terms and formulae under different structures say \mathbf{A} and \mathbf{B} using valuations $v_{\mathbf{A}} : V \rightarrow A$ and $v_{\mathbf{B}} : V \rightarrow B$ where $\mathbb{A} = |\mathbf{A}|$ and $\mathbb{B} = |\mathbf{B}|$ we use $\mathcal{V}_{\mathbf{A}}$, $\mathcal{T}_{\mathbf{A}}$ and $\mathcal{V}_{\mathbf{B}}$, $\mathcal{T}_{\mathbf{B}}$ respectively to distinguish the possibly different values and truth values.

Isomorphic Structures

Example 22.2: Structure isomorphism

Let $\Sigma = \{0 : \rightarrow s, +1 : s \rightarrow s, = : s^2, < : s^2\}$. Now consider the structures

$$\mathbb{N} = \langle \mathbb{N}; 0, +1, =, < \rangle$$

$$2\mathbb{N} = \langle 2\mathbb{N}; 0, +2, =, < \rangle$$

where $2\mathbb{N}$ is the set of even natural numbers, $+1, +2$ denote the respective “successor” functions, $<$ is the usual “less-than” relation on both structures. The two structures are clearly isomorphic and there is an isomorphism $\pi : \mathbb{N} \rightarrow 2\mathbb{N}$ such that $\pi(n) = 2n$ which along with the inverse map $\pi^{-1}(2n) = n$ maps one structure exactly onto the other.

Isomorphic Σ -structures cannot be distinguished by $\mathcal{P}_1(\Sigma)$.

The Isomorphism Lemma

Lemma 22.2: The Isomorphism Lemma

If \mathbf{A} and \mathbf{B} are isomorphic Σ -structures then for all formulae ϕ , $\mathbf{A} \Vdash \phi$ if and only if $\mathbf{B} \Vdash \phi$.



Corollary 22.1

If $\pi : \mathbf{A} \cong \mathbf{B}$ then for any formula $\phi(x_1, \dots, x_n)$ and any valuation $v_{\mathbf{A}}$, $v_{\mathbf{B}}$ and values $a_1, \dots, a_n \in |\mathbf{A}|$,

$$(\mathbf{A}, v_{\mathbf{A}}[x_1 := a_1, \dots, x_n := a_n]) \Vdash \phi \text{ iff }$$

$$(\mathbf{B}, v_{\mathbf{B}}[x_1 := \pi(a_1), \dots, x_n := \pi(a_n)]) \Vdash \phi$$

Proof of the Isomorphism Lemma 22.2

Proof: Assume there is an isomorphism $\pi : \mathbf{A} \cong \mathbf{B}$. Since π is an isomorphism, $\pi : \mathbb{A} \xrightarrow[\text{onto}]{} \mathbb{B}$ is a bijection that preserves structure, where $\mathbb{A} = |\mathbf{A}|$ and $\mathbb{B} = |\mathbf{B}|$. Hence $\pi^{-1} : \mathbb{B} \xrightarrow[\text{onto}]{} \mathbb{A}$ also exists and preserves structure. Further we have

1. $\pi(f_{\mathbf{A}}(a_1, \dots, a_m)) = f_{\mathbf{B}}(\pi(a_1), \dots, \pi(a_m))$ for every m -ary function $f_{\mathbf{A}}$,
2. $\pi^{-1}(f_{\mathbf{B}}(b_1, \dots, b_m)) = f_{\mathbf{A}}(\pi^{-1}(b_1), \dots, \pi^{-1}(b_m))$ for every m -ary function $f_{\mathbf{B}}$,
3. $(a_1, \dots, a_n) \in p_{\mathbf{A}}$ iff $(\pi(a_1), \dots, \pi(a_n)) \in p_{\mathbf{B}}$ for each n -ary relation $p_{\mathbf{A}}$.
4. $(b_1, \dots, b_n) \in p_{\mathbf{B}}$ iff $(\pi^{-1}(b_1), \dots, \pi^{-1}(b_n)) \in p_{\mathbf{A}}$ for each n -ary relation $p_{\mathbf{B}}$.

Further for each $v_{\mathbf{A}} : V \rightarrow \mathbb{A}$, $\pi \circ v_{\mathbf{A}} : V \rightarrow \mathbb{B}$ and for each $v_{\mathbf{B}} : V \rightarrow \mathbb{B}$, $\pi^{-1} \circ v_{\mathbf{B}} : V \rightarrow \mathbb{A}$.

For every formula ϕ , we may prove the following stronger claims by induction on the structure of ϕ .

$$\begin{aligned}\mathcal{T}_{\mathbf{A}}[\![\phi]\!]_{v_{\mathbf{A}}} &= \mathcal{T}_{\mathbf{B}}[\![\phi]\!]_{\pi \circ v_{\mathbf{A}}} \\ \mathcal{T}_{\mathbf{B}}[\![\phi]\!]_{v_{\mathbf{B}}} &= \mathcal{T}_{\mathbf{A}}[\![\phi]\!]_{\pi^{-1} \circ v_{\mathbf{B}}}\end{aligned}$$

The proof is left as an exercise to the interested reader.

QED

The Isomorphism Lemma then raises the following interesting question which we will answer later.

Question. Are Σ -structures which satisfy the same Σ -formulae isomorphic?

But regardless of the answer to the above question the isomorphism lemma also brings the realization that for any given set of Σ -formulae there could be more than one model, in fact a class of models, for a given nonempty set Φ of Σ -formulae.

Substructures

Definition 22.1: Substructures

A Σ -structure \mathbf{A} is a **substructure** of another Σ -structure \mathbf{B} (denoted $\mathbf{A} \subseteq \mathbf{B}$) if

- $\emptyset \neq |\mathbf{A}| \subseteq |\mathbf{B}|$,
- for each $f : s^m \rightarrow s \in \Sigma$, $f_{\mathbf{A}} = f_{\mathbf{B}} \upharpoonright |\mathbf{A}|^m$,
- for each $p : s^n \in \Sigma$, $p_{\mathbf{A}} = p_{\mathbf{B}} \cap |\mathbf{A}|^n$.

where $f_{\mathbf{B}} \upharpoonright |\mathbf{A}|^m$ denotes the restriction of $f_{\mathbf{B}}$ to elements of $|\mathbf{A}|^m$.

Since \mathbf{A} is a Σ -structure, it must be Σ -closed, i.e. for each $f : s^m \rightarrow s \in \Sigma$ and each $(a_1, \dots, a_m) \in |\mathbf{A}|^m$, $f_{\mathbf{A}}(a_1, \dots, a_m) \in |\mathbf{A}|$.

Substructure Facts

Fact 22.1

If $\mathbf{A} \subseteq \mathbf{B}$, then

1. for each $f : s^m \rightarrow s \in \Sigma$ and each $(a_1, \dots, a_m) \in |\mathbf{A}|^m$,
 $f_{\mathbf{B}}(a_1, \dots, a_m) \in |\mathbf{A}|$.
2. Conversely for each $\mathbb{X} \subseteq |\mathbf{B}|$, such that \mathbb{X} is Σ -closed there exists a unique Σ -substructure $\mathbf{X} \subseteq \mathbf{B}$.

Substructure Examples

Example 22.3: Substructures

1. $\mathbf{N} = \langle \mathbb{N}; 0, +; = \rangle$ is a substructure of $\mathbf{Z} = \langle \mathbb{Z}; 0, +; = \rangle$ which in turn is a substructure of $\mathbf{Q} = \langle \mathbb{Q}; 0, +; = \rangle$ which in turn is a substructure of $\mathbf{R} = \langle \mathbb{R}; 0, +; = \rangle$.
2. $2\mathbf{N} = \langle 2\mathbb{N}; 0, +; = \rangle$ is a substructure of $\mathbf{N} = \langle \mathbb{N}; 0, +; = \rangle$.
3. However the odd numbers are not closed under addition and hence they do not form a substructure of \mathbf{N} under addition.

Quantifier-free Formulae

Definition 22.2: Quantifier-free formulae

For any signature Σ , the set $Q\mathcal{F}_1(\Sigma)$ of quantifier-free formulae is given by the following grammar

$$\begin{array}{lcl} \chi, \xi ::= & \perp & | \top \\ & | p(t_1, \dots, t_n) \in A(\Sigma) & | (\neg \chi) \\ & | (\chi \wedge \xi) & | (\chi \vee \xi) \\ & | (\chi \rightarrow \xi) & | (\chi \leftrightarrow \xi) \end{array}$$

Lemma on Quantifier-free Formulae

Lemma 22.3

Let $\mathbf{A} \subseteq \mathbf{B}$ be Σ -structures and let $v_{\mathbf{A}}$ be any valuation in \mathbf{A} . Then for every Σ -term t and every quantifier-free formula χ

$$\begin{aligned}\mathcal{V}_{\mathbf{A}}[t]_{v_{\mathbf{A}}} &= \mathcal{V}_{\mathbf{B}}[t]_{v_{\mathbf{A}}} \\ \mathcal{T}_{\mathbf{A}}[\chi]_{v_{\mathbf{A}}} &= \mathcal{T}_{\mathbf{B}}[\chi]_{v_{\mathbf{A}}}\end{aligned}$$

Proof: Use the Isomorphism lemma 22.2 on the common subset of the two carrier sets using the identity isomorphism. QED

Universal and Existential Formulae

Definition 22.3

For any signature Σ , the set $\mathcal{O}_1(\Sigma)$ of \mathcal{O} -formulae for each $\mathcal{O} \in \{\forall, \exists\}$ is defined inductively by the following grammar

$$\begin{aligned}\phi, \psi ::= & \chi \in \mathcal{QF}_1(\Sigma) \\ & | \ (\phi \wedge \psi) \quad | \ (\phi \vee \psi) \\ & | \ \mathcal{O}x[\phi]\end{aligned}$$

$\forall_1(\Sigma)$ is the set of universal Σ -formulae and $\exists_1(\Sigma)$ is the set of existential Σ -formulae.

The Substructure Lemma

Lemma 22.4: The Substructure Lemma

Let $\mathbf{A} \subseteq \mathbf{B}$ be Σ -structures and let $\phi(x_1, \dots, x_n) \in \forall_1(\Sigma)$. Then for any valuations $v_{\mathbf{A}}$, $v_{\mathbf{B}}$ and $a_1, \dots, a_n \in A$,

if $(\mathbf{B}, v_{\mathbf{B}}[x_1 := a_1, \dots, x_n := a_n]) \Vdash \phi$
then $(\mathbf{A}, v_{\mathbf{A}}[x_1 := a_1, \dots, x_n := a_n]) \Vdash \phi$

Proof: By induction on the structure of universal formulae.

QED

Exercise 22.1: Structures

Theorem 22.1: The Homomorphism Theorem

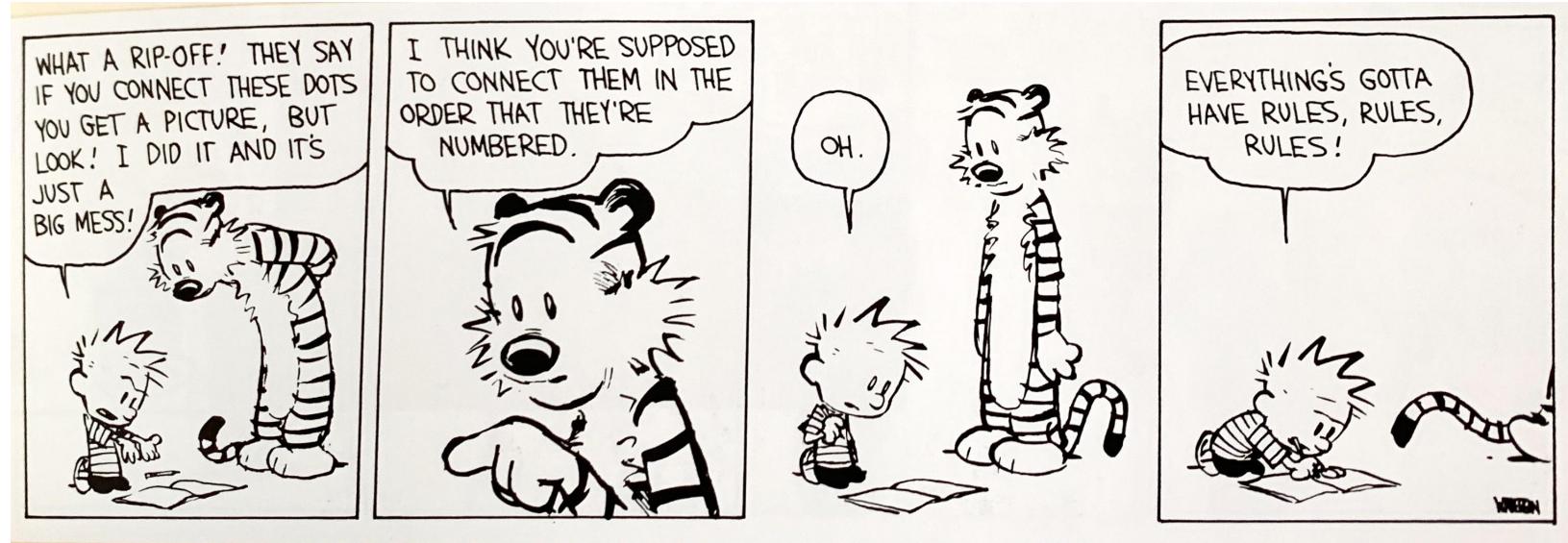
Let $\varpi : A \longrightarrow B$ be a homomorphism from a structure A *into* a structure B and let v_A be a valuation.

1. For any term t , $\mathcal{V}_B[t]_{\varpi \circ v_A} = \varpi(\mathcal{V}_A[t]_{v_A})$
2. If ϖ is *injective* then for any quantifier-free formula χ , $\mathcal{T}_B[\chi]_{\varpi \circ v_A} = \varpi(\mathcal{T}_A[\chi]_{v_A})$

1. Prove theorem 22.1.
2. Prove that if ϖ is not surjective, the two structures may be distinguished by a formula.
3. In part 2 of theorem 22.1 why is the condition of injectivity necessary? (*Hint.* Let $=$ be the atomic binary equality predicate whose semantics is defined in an obvious fashion. Now show that if ϖ is not injective then the two structures can be distinguished using equality.)
4. Prove that if ϖ is *surjective* but not necessarily *injective*, then in the absence of any predicate from which equality or inequality of terms may be expressed or derived, part 2 of theorem 22.1 may be extended to quantified formulae.

23. Predicate Logic: Proof Theory

23: Predicate Logic: Proof Theory



1. Proof Theory: First-Order Logic
2. Proof Rules: Hilbert-Style
3. The Mortality of Socrates
4. The Mortality of the Greeks
5. Faulty Proof:2
6. A Correct Proof
7. The Sequent Forms
8. The Case of Equality
9. Semantics of Equality
10. Theories of Predicate Calculus
11. Axioms for Equality
12. Symmetry and Transitivity
13. Symmetry of Equality
14. Transitivity of Equality

Proof Theory: First-Order Logic

1. A first-order proof system for a mathematical theory with signature Σ consists of two parts

(Logical) Axioms and Rules These are extensions of existing propositional proof systems to deal with quantification. However, they are still parameterised on Σ since they involve the use of terms and substitutions of terms for free variables.

Postulates or (Non-logical axioms) These are the axioms which define the models to which they are applicable.

2. **(First-order) Predicate Calculus:** The first-order theory with no non-logical axioms.

Proof Rules: Hilbert-Style

Definition 23.1: The Proof system $\mathcal{H}_1(\Sigma)$

- The language $\mathcal{L}_1(\Sigma)$ generated from A and $\{\neg, \rightarrow, \forall\}$
- The three *logical* axiom schemas K, S and N,
- The two axiom schemas

$$\forall E. \frac{}{\forall x[X] \rightarrow \{t/x\}X}, t \equiv x \text{ or } \{t/x\} \text{ admissible for } X$$

$$\forall D. \frac{}{\forall x[X \rightarrow Y] \rightarrow (X \rightarrow \forall x[Y])}, x \notin FV(X)$$

- The *modus ponens* (MP) rule and

$$\forall I. \frac{\{y/x\}X}{\forall x[X]}, y \equiv x \text{ or } y \notin FV(X)$$

23.1. The System \mathcal{H}_1 explained

Note the following about the formal theory of the proof system \mathcal{H}_1 .

The language $\mathcal{L}_1(\Sigma)$ expands on the language \mathcal{L}_0 by including predicates and the quantifier \forall and is parameterized by the signature Σ .

The axioms K , S , N represent **larger** sets of formulae viz. $\mathcal{K}_1 \supset \mathcal{K}_0$, $\mathcal{S}_1 \supset \mathcal{S}_0$ and $\mathcal{N}_1 \supset \mathcal{N}_0$ respectively.

The rule MP is also a **larger** relation ($\mathcal{MP}_1 \supset \mathcal{MP}_0$) that includes predicates from $\mathcal{L}_1(\Sigma)$

Side conditions Some explanations are perhaps in order regarding the “side conditions” attached to the quantifier axiom schemas and rules.

VE. The side condition “ $t \equiv x$ ” allows the substituted term to be the variable x , which is not surprising. But this is explicitly mentioned only because the side condition “ $\{t/x\}$ admissible for X ” does not include the possibility of replacing x by itself.

But more important is the side condition “ $\{t/x\}$ admissible for X ”. If this condition were not present then we could have the following situation which would be clearly unsound.

Let $p(x, y)$ be a binary predicate and let $\phi(x) \stackrel{df}{=} \neg \forall y [p(x, y)]$.

Now clearly $\{y/x\}$ is not admissible in $\phi(x)$. If the condition of admissibility were absent then we could have replaced the free variable x with y yielding the following instance of axiom schema $\forall E$

$$\forall x[\neg \forall y[p(x, y)]] \rightarrow \neg \forall y[p(y, y)] \quad (58)$$

Now consider any model consisting of at least two distinct elements and let $p(x, y)$ denote the identity relation on the carrier set. Clearly in such a model while the antecedent $\forall x[\neg \forall y[p(x, y)]]$ is clearly valid (true for each valuation), the consequent $\neg \forall y[p(y, y)]$ would be invalid (false for certain valuations).

- $\forall D$. In this axiom schema if we were to relax the side-condition and allow the quantifier to be moved in even if $x \in FV(X)$ it would result in the following situation when $X = \phi(x) = Y$.

$$\forall x[\phi(x) \rightarrow \phi(x)] \rightarrow (\phi(x) \rightarrow \forall x[\phi(x)]) \quad (59)$$

Whereas the antecedent of the formula in (59) is logically valid, the truth of the consequent is not guaranteed in any interpretation in which $\phi(x)$ is true only for some elements of the domain and false for others.

- $\forall I$. The hypothesis of the rule asserts that a formula $\phi(\dots, x, \dots)$ holds when the free variable x is replaced uniformly throughout the formula by any variable that does not occur free in ϕ . Then clearly the formula holds for any “arbitrary” value that x may take, and hence the variable x may be universally quantified.

The Mortality of Socrates

The Mortality of Socrates

A translation into predicate logic involves

- including two unary atomic predicates h and m denoting properties of objects (*human* and *mortal* respectively),
- including a constant s (for *Socrates*).

$$\forall x [h(x) \rightarrow m(x)], h(s) \vdash m(s)$$

$$\frac{\forall E \frac{\forall x [h(x) \rightarrow m(x)]}{h(s) \rightarrow m(s)} \quad h(s)}{m(s)}$$

The Mortality of the Greeks

All humans are mortal.

All Greeks are human.

Therefore all Greeks are mortal.

$$\forall x[h(x) \rightarrow m(x)], \forall x[g(x) \rightarrow h(x)] \vdash \forall x[g(x) \rightarrow m(x)]$$

A faulty proof:

$$\text{T} \Rightarrow \frac{\forall x[g(x) \rightarrow h(x)] \quad \forall x[h(x) \rightarrow m(x)]}{\forall x[g(x) \rightarrow m(x)]}$$

Faulty Proof:2

$$\begin{array}{c} \forall E \frac{\forall x[g(x) \rightarrow h(x)]}{g(y) \rightarrow h(y)} \quad \forall E \frac{\forall x[h(x) \rightarrow m(x)]}{h(z) \rightarrow m(z)} \\ \hline ?? \frac{g(y) \rightarrow h(y) \quad h(z) \rightarrow m(z)}{\forall I \frac{g(y) \rightarrow m(y)}{\forall x[g(x) \rightarrow m(x)]}} \end{array}$$

A Correct Proof

$$\begin{array}{c} \forall E \frac{\forall x[g(x) \rightarrow h(x)]}{g(y) \rightarrow h(y)} \quad \forall E \frac{\forall x[h(x) \rightarrow m(x)]}{h(y) \rightarrow m(y)} \\ T \Rightarrow \frac{g(y) \rightarrow m(y)}{\forall x[g(x) \rightarrow m(x)]} \end{array}$$

The Sequent Forms

The sequent forms of K, S, N, MP are as before. The sequent forms of the quantification rules are as follows:

$$\forall E. \frac{}{\Gamma \vdash \forall x[X] \rightarrow \{t/x\}X}, t \equiv x \text{ or } \{t/x\} \text{ admissible in } X$$

$$\forall D. \frac{}{\Gamma \vdash \forall x[X \rightarrow Y] \rightarrow (X \rightarrow \forall x[Y])}, x \notin FV(X)$$

$$\forall I. \frac{\Gamma \vdash \{y/x\}X}{\Gamma \vdash \forall x[X]}, y \notin FV(X) \cup FV(\Gamma)$$

Note that the variable y being quantified should not occur free in any of the assumptions Γ .

The Case of Equality

1. In most algebraic formulations equality is a necessary binary predicate of the signature.
2. In other cases where equality may not otherwise play a prominent role, it becomes necessary because of one or more of the following reasons.
 - (a) *Syntactically distinct* terms may represent the same value in a structure either because of some axioms or because of some identifications made in a valuation i.e. it is possible that even though $s \not\equiv t$, $\mathcal{V}_A[s]_{v_A} = \mathcal{V}_A[t]_{v_A}$. (see also exercise 22.1).
 - (b) Two *differently named* entities are proven to be the same entity (generally in proofs of uniqueness or in proofs by contradiction). See example 23.2.

Generally in mathematics, if there are two named entities x and y and nothing is stated about the relationship between them, we can neither assume they stand for distinct entities nor can we exclude the possibility that they both stand for the same entity.

For instance, it becomes essential when speaking of models which contain at least two elements, to make a first-order statement like $\exists x \exists y [\neg(x = y)]$ which states that there exist at least two distinct objects.

Example 23.1

Consider any first-order theory of boolean algebra. If we allow for the possibility that $1 = 0$ every equation of boolean algebra would still be satisfied and the boolean identities would still continue to hold. But in addition we would also have $0 \leq 1 \leq 0$. If we were to adopt this as a model for truth, then every formula would be implied by every other formula and the whole edifice of mathematical logic as we know it would collapse to a triviality without the assumption that the value 1 is distinct from the value 0 .

On the other hand, it may so happen that one may define properties of objects and prove a theorem stating in effect that if the two objects x and y both satisfy a certain property ϕ , then they are both the same object. This is usually expressed as $(\phi(x) \wedge \phi(y)) \rightarrow (x = y)$. As an example of this phenomenon consider the following example.

Example 23.2

Let $\langle \mathbb{S}, + \rangle$ be any semi-group i.e. \mathbb{S} is closed under the binary product operation $+$ and $+$ is associative. That is, $\langle \mathbb{S}, + \rangle \Vdash \phi_{associative}$ (see equation (50)). Now consider the structure $\langle \mathbb{S}', + \rangle$ obtained by adding two distinguished elements i_L and i_R to \mathbb{S} i.e. $\mathbb{S}' = \mathbb{S} \cup \{i_L, i_R\}$ along with the axioms

$$\phi_{left-identity} \stackrel{\text{df}}{=} \forall x[i_L + x = x] \quad (60)$$

$$\phi_{right-identity} \stackrel{\text{df}}{=} \forall x[x + i_R = x] \quad (61)$$

It follows easily that $i_L = i_L + i_R = i_R$ and hence $i_L = i_R$.

Notice how equality creeps in even without our wanting to bring it in!

Example 23.3

We have consciously avoided using equality in definitions -4.4 and in the definition of antisymmetry -4.11 but most books do not.

- The definition of injective functions in many standard mathematics texts uses equality or inequality in its definition. A function $f : A \rightarrow B$ is injective if for any two elements a, a' such that $a \neq a'$, $f(a) \neq f(a')$. Alternatively, f is injective if $f(a) = f(a')$ implies $a = a'$.
- See footnote in definition -4.11.

The use of equality or inequality in such situations is inescapable. Hence equality has a special place in mathematics and logic and is usually required as a basic relation between named entities, even when it is not primarily a relation of interest in the models that are being studied.

Semantics of Equality

The semantics of the binary infix atomic predicate $=$ is defined as follows.

Definition 23.2: Semantics of Equality

$$\mathcal{T}_A[s = t]_{v_A} \stackrel{df}{=} \begin{cases} 1 & \text{if } \mathcal{V}_A[s]_{v_A} = \mathcal{V}_A[t]_{v_A} \\ 0 & \text{otherwise} \end{cases}$$

In the sequel we do not explicitly include equality in the signature, but assume that it is present as part of the language of **First-order Predicate Logic with Equality**.

Theories of Predicate Calculus

Notation 23.1: First-order theories

The First-order theory of the Predicate Calculus will be denoted by $\text{Th}(\mathbb{PC}_1)$.

The First-order theory of Predicate Calculus with Equality with no non-logical axioms except the axioms for equality will be denoted by $\text{Th}(\mathbb{PC}_1 =)$.

Axioms for Equality

Equality usually is a reflexive, symmetric, transitive and substitutive relation on structures. However the following axioms are sufficient.

$$= R. \frac{}{t = t}$$

$$= C. \frac{}{(s = t) \rightarrow (\{s/x\}u = \{t/x\}u)}$$

$$= S. \frac{}{(s = t) \rightarrow (\{s/x\}X \rightarrow \{t/x\}X)}, \{s/x\}, \{t/x\} \text{ admissible in } X$$

Explanations.

=R This is the **reflexivity** axiom schema, which asserts that any term equals itself.

=C This is the **congruence** axiom schema and it asserts that equals may be substituted for equals in any term context.

=S This is the **substitutivity** axiom schema which again asserts that the replacement of equals by equals does not alter the truth value of formulae. However due to the presence of quantifiers and bound variables one must ensure that the substitution of equals by equals does not result in the capture of free variables of either *s* or *t*.

Symmetry and Transitivity

The rule of substitutivity ($=S$) is sufficiently powerful to force the properties of symmetry and transitivity with the help of reflexivity and modus ponens.

$$\text{= Sym. } \frac{\Gamma \vdash s=t}{\Gamma \vdash t=s}$$

$$\text{= T. } \frac{\Gamma \vdash s=t \quad \Gamma \vdash t=u}{\Gamma \vdash s=u}$$

Symmetry of Equality

Proof of derived rule $=\text{Sym}$

Let $\Gamma = \{s = t\}$ and Let $\phi \stackrel{df}{=} x = s$. Then we have

$$\begin{aligned}\{s/x\}\phi &\equiv s = s \\ \{t/x\}\phi &\equiv t = s\end{aligned}$$

$$\text{MP} \frac{\Gamma \vdash s = t \quad =S \frac{}{\Gamma \vdash s = t \rightarrow (s = s \rightarrow t = s)}}{\text{MP} \frac{\Gamma \vdash s = s \rightarrow t = s}{\Gamma \vdash t = s}} =R \frac{}{\Gamma \vdash s = s}$$

Transitivity of Equality

Proof of derived rule $=T$

Let $\Delta = \{s = t, t = u\}$ and $\psi \equiv s = x$. Then

$$\frac{\text{MP} \quad \frac{\text{MP} \quad \frac{\Delta \vdash t = u \rightarrow (s = t \rightarrow s = u)}{\Delta \vdash t = u}}{\Delta \vdash s = t \rightarrow s = u}}{\Delta \vdash s = u} \Delta \vdash s = t$$

24. Predicate Logic: Proof Theory (Contd.)

24: Predicate Logic: Proof Theory (Contd.)

1. Alpha Conversion
2. The Deduction Theorem for Predicate Calculus
3. Useful Corollaries
4. Soundness of Predicate Calculus
5. Soundness of The Hilbert System

Alpha Conversion

Notation 24.1

We use " $\phi \dashv\vdash \psi$ " as an abbreviation for the two statements " $\phi \vdash \psi$ " and " $\psi \vdash \phi$ ".

Lemma 24.1: Alpha conversion

For every formula ϕ for which $\{y/x\}$ is admissible $\forall x[\phi] \dashv\vdash \forall y[\{y/x\}\phi]$

Proof: If $\{y/x\}$ is admissible in ϕ then we may readily see that $\{x/y\}$ is also admissible in $\{y/x\}\phi$ since $x \notin FV(\{y/x\}\phi)$ and

$$\{x/y\}\{y/x\}\phi \equiv \phi$$

Further since $x \notin FV(\forall x[\phi])$ we have by axiom schema $\forall E \ \forall x[\phi] \rightarrow \phi$ i.e.

$$\forall x[\phi] \rightarrow \{x/y\}\{y/x\}\phi$$

We then have the following proofs.

QED

Proof trees of $\forall x[\phi] \dashv\vdash \forall y[\{y/x\}\phi]$

Proof tree of $\forall x[\phi] \vdash \forall y[\{y/x\}\phi]$

$$\text{MP} \frac{\overline{\forall x[\phi] \vdash \forall x[\phi]} \quad \forall E \frac{\overline{\forall x[\phi] \vdash \forall x[\phi] \rightarrow \{x/y\}\{y/x\}\phi}}{\forall x[\phi] \vdash \{x/y\}\{y/x\}\phi}}{\forall x[\phi] \vdash \forall y[\{y/x\}\phi]}$$

Proof tree of $\forall y[\{y/x\}\phi] \vdash \forall x[\phi]$

$$\text{MP} \frac{\overline{\forall y[\{y/x\}\phi] \vdash \forall y[\{y/x\}\phi]} \quad \forall E \frac{\overline{\forall y[\{y/x\}\phi] \vdash \forall y[\{y/x\}\phi] \rightarrow \{y/x\}\phi}}{\forall y[\{y/x\}\phi] \vdash \{y/x\}\phi}}{\forall y[\{y/x\}\phi] \vdash \forall x[\phi]}$$

The Deduction Theorem for Predicate Calculus

(c.f. [Deduction Theorem for Propositional Calculus](#))

Theorem 24.1: The Deduction Theorem for Predicate Calculus

Let $\Gamma \subseteq_f \mathcal{L}_1(\Sigma)$ and $\phi, \psi \in \mathcal{L}_1(\Sigma)$.

1. (DT \Leftarrow). If $\Gamma \vdash \phi \rightarrow \psi$ then $\Gamma, \phi \vdash \psi$.
2. (DT \Rightarrow). Let $\Gamma, \phi \vdash \psi$ such that no variable in $FV(\phi)$ is generalized (by an application of $\forall I$) in the proof. Then $\Gamma \vdash \phi \rightarrow \psi$.



Proof of the Deduction Theorem for Predicate Calculus (theorem 24.1)

Proof:

1. ($\text{DT} \Leftarrow$). The proof is identical to that of the corresponding **propositional case**.
2. ($\text{DT} \Rightarrow$). Assume $\Gamma, \phi \vdash \psi$. Then there exists a proof tree \mathcal{T} rooted at ψ with nodes $\psi_1, \dots, \psi_m \equiv \psi$. Then the stronger claim that each step of the proof of ψ_i can be matched by a proof of $\phi \rightarrow \psi_i$ is again proven here by induction. **The proof proceeds in a manner similar to the corresponding propositional case for all applications of the propositional axioms and the inference rule (MP).** We consider only the cases of quantification.
If ψ_j for $0 < j \leq m$ is an axiom (including an instance of **VE** or **VD**) or $\psi_j \in \Gamma$, then the proof tree \mathcal{T}' rooted at $\phi \rightarrow \psi_j$ is constructed from the proof tree \mathcal{T}_j rooted at ψ_j as follows.

$$\frac{j' \frac{\nwarrow \mathcal{T}_j \nearrow}{\psi_j} \quad j' + 1 \frac{\psi_j \rightarrow (\phi \rightarrow \psi_j)}{\phi \rightarrow \psi_j}}{j' + 2}$$

Suppose ψ_j was obtained by the application of the axiom schema **AI** on some ψ_i such that $i < j$. Then $\psi_j \equiv \forall x[\psi_i]$. By the induction hypothesis there exists a proof tree \mathcal{T}'_i rooted at $\phi \rightarrow \psi_i$ and such that no free variable of ϕ has been generalized in the application. Further $x \notin FV(\phi)$. We may now extend \mathcal{T}'_i to a proof tree \mathcal{T}' as follows.

$$\frac{i' \frac{\nwarrow \mathcal{T}'_i \nearrow}{\phi \rightarrow \psi_i} \quad i' + 2 \frac{\forall x[\phi \rightarrow \psi_i] \rightarrow (\phi \rightarrow \forall x[\psi_i])}{\phi \rightarrow \forall x[\psi_i] \equiv \phi \rightarrow \psi_j}}{i' + 3}$$

QED

Useful Corollaries

Corollary 24.1

If the proof of $\Gamma, \phi \vdash \psi$ involves no generalization of any free variable of ϕ then $\Gamma \vdash \phi \rightarrow \psi$.

Corollary 24.2

If ϕ is a closed formula and $\Gamma, \phi \vdash \psi$ then $\Gamma \vdash \phi \rightarrow \psi$.

Corollary 24.3

If no free variable of $\Gamma = \{\phi_1, \dots, \phi_m\}$ is generalized in a proof of $\Gamma \vdash \psi$, then $\vdash \phi_1 \rightarrow \dots \rightarrow \phi_m \rightarrow \psi$.

Soundness of Predicate Calculus

Proposition 24.1

Every wff of $\mathcal{L}_1(\Sigma)$ which is an instance of a tautology of Propositional logic is a theorem of PC and may be obtained using only the axiom schemas K, S, N and the rule MP.

Theorem 24.2: Consistency of $\text{Th}(\text{PC})$

The theory PC is consistent i.e. the set $\text{Th}(\text{PC})$ of theorems of PC form a consistent set.



Proof of theorem 24.2

Proof: Define the *erasure* of a formula $e(\phi)$ as the propositional formula obtained by deleting all terms and all quantifiers and retaining only the atomic predicate symbols and propositional connectives. The function e may be defined by induction on the structure of the formula ϕ .

- *Claim 0.* For each instance of the axioms of \mathcal{H}_1 , the erasure of the formulae yields tautologies (of propositional logic)
- *Claim 1.* The erasure of each application of the rules of \mathcal{H}_1 , preserves tautologous-ness, i.e. if the erasure of the premises of a rule are all propositional tautologies then so is the erasure of the conclusion.
- *Claim 2.* The erasure of every formula in $\mathbf{Th}(\mathbb{PC})$ is a tautology.

If $\mathbf{Th}(\mathbb{PC})$ were inconsistent then $\mathbf{Th}(\mathbb{PC}) = \mathcal{L}_1$. In particular, there exist formulae $\phi, \neg\phi \in \mathbf{Th}(\mathbb{PC})$. But then by the definition of erasure we have $e(\neg\phi) \equiv \neg e(\phi)$ which contradicts *Claim 2*. QED

Soundness of The Hilbert System

Theorem 24.3: Soundness of \mathcal{H}_1

Every theorem of \mathcal{H}_1 is logically valid.



Proof of theorem 24.3

Proof: Here are some easily proven claims which hold for any interpretation of Σ -formulae and hence also hold for the predicate calculus.

- *Claim 1* Every instance of a (propositional) tautology is true for every interpretation.
- *Claim 2* All instances of the axioms $\forall E$ and $\forall D$ are logically valid.
- *Claim 3* The rules MP and $\forall I$ preserve logical validity i.e. if the premises of the rules are logically valid formulae under every Σ -interpretation then so are the conclusions.

It then follows by induction on the structure of the proof tree that every theorem is logically valid.

QED

Exercise 24.1: Predicate Logic: Proof theory

1. Prove the arguments in Problem 2 of exercise 18.1 using the system \mathcal{H}_1 .
2. Prove the arguments in exercise 18.2 using the system \mathcal{H}_1 .
3. Prove the conclusions of Problem 3 in exercise 25.2 using only the proof rules of \mathcal{H}_1 .
4. Let $\psi \in SF(\phi)$. If ϕ' is obtained from ϕ by replacing zero or more occurrences of ψ by ψ' . Then prove the following.

- (a) $\vec{\forall}[\psi \leftrightarrow \psi'] \vdash \phi \leftrightarrow \phi'$
- (b) $\psi \leftrightarrow \psi' \vdash \phi \leftrightarrow \phi'$

(c) We also have the derived rule of inference $\Leftrightarrow .$
$$\boxed{\frac{\Gamma \vdash \psi \leftrightarrow \psi' \quad \Gamma \vdash \phi}{\Gamma \vdash \phi'}}$$
 which is the only rule that allows the use of a rule that applies to proper sub-formulae of a formula.

5. If ϕ and ψ are formulae such that $x \notin FV(\phi)$, then the following are theorems of \mathcal{H}_1 ($\exists x[\phi]$ is an abbreviation of $\neg\forall x[\neg\phi]$).
- (a) $\vdash \phi \rightarrow \forall x[\phi]$ and hence $\vdash \phi \leftrightarrow \forall x[\phi]$
 - (b) $\vdash \exists x[\phi] \rightarrow \phi$ and hence by rule $\exists I \vdash \exists x[\phi] \leftrightarrow \phi$
 - (c) $\vdash \forall x[\phi \rightarrow \psi] \leftrightarrow (\phi \rightarrow \forall x[\psi])$
 - (d) $\vdash \forall x[\psi \rightarrow \phi] \leftrightarrow (\exists x[\psi] \rightarrow \phi)$

25. Existential Quantification

25: Existential Quantification

1. Existential Quantification
2. Existential Elimination
3. Remarks on Existential Elimination
4. Restrictions on Existential Elimination
5. Equivalence of Proofs
6. Natural Deduction: 6

Existential Quantification

Existential quantification is the derived operator defined by

$$\exists x[\phi] \stackrel{df}{=} \neg \forall x[\neg \phi]$$

We then have the rules

$$\exists I. \frac{\Gamma \vdash \{t/x\}\phi, \{t/x\} \text{ admissible in } \phi}{\Gamma \vdash \exists x[\phi]}$$

$$\exists E. \frac{\Gamma \vdash \exists x[\phi]}{\Gamma \vdash \{a/x\}\phi}, a \notin FV(\Gamma) \cup FV(\exists x[\phi]) \text{ is fresh}$$

- $\exists I$ is a derived rule
- However $\exists E$ is not a derived rule in the Hilbert-System.

Proof of Derived rule $\exists I$

Proof: Assuming $\{t/x\}$ is admissible in ϕ suppose $\Gamma \vdash \{t/x\}\phi$.

$$\text{MP} \frac{\forall E \frac{}{\Gamma \vdash \forall x[\neg\phi] \rightarrow \neg\{t/x\}\phi}}{\Gamma \vdash \neg\neg\{t/x\}\phi \rightarrow \neg\forall x[\neg\phi]} \quad \text{N''} \frac{\Gamma \vdash (\forall x[\neg\phi] \rightarrow \neg\{t/x\}\phi) \rightarrow (\neg\neg\{t/x\}\phi \rightarrow \neg\forall x[\neg\phi])}{\Gamma \vdash \neg\neg\{t/x\}\phi \rightarrow \neg\forall x[\neg\phi]}$$

From the root of the tree we have

$$T \rightarrow \frac{\Gamma \vdash \neg\neg\{t/x\}\phi \rightarrow \neg\forall x[\neg\phi]}{\Gamma \vdash \{t/x\}\phi \rightarrow \neg\forall x[\neg\phi]} \quad \text{DNI} \frac{\Gamma \vdash \{t/x\}\phi \rightarrow \neg\neg\{t/x\}\phi}{\Gamma \vdash \{t/x\}\phi \rightarrow \neg\forall x[\neg\phi]}$$

Assuming $\Gamma \vdash \{t/x\}\phi$, the last step gives

$$\text{MP} \frac{\Gamma \vdash \{t/x\}\phi \rightarrow \neg\forall x[\neg\phi] \quad \Gamma \vdash \{t/x\}\phi}{\Gamma \vdash \neg\forall x[\neg\phi] \equiv \exists x[\phi]}$$

Note. In the last step “ $\equiv \exists x[\phi]$ ” is not really part of the proof tree. It is just an abuse of (meta-)notation to indicate that $\neg\forall x[\neg\phi]$ is really its syntactic equivalent $\exists x[\phi]$. QED

Existential Elimination

Existential generalisation from a constant introduced somehow into the proof is very common in mathematics.

Example 25.1: Existential generalization from a constant

Consider a purported **proof** of

$$\exists x[\phi \rightarrow \psi], \forall x[\phi] \vdash \exists x[\psi]$$

where x may be a free variable of both ϕ and ψ . The standard practice is to assume the existence of some “constant symbol” a and proceed with it to eventually generalize. Normally we cannot assume a term of any shape (as we did in the case of **universal instantiation**) since only certain terms may yield a value that might satisfy a predicate. On the other hand symbolizing this value by a constant symbol would be safe.

Formal proof example 25.1

Let $\Gamma = \{\exists x[\phi \rightarrow \psi], \forall x[\phi]\}$.

A proof using $\exists E$ ($\Gamma \vdash_{\exists E} \exists x[\psi]$)

$$\frac{\begin{array}{c} \exists E \frac{\Gamma \vdash \exists x[\phi \rightarrow \psi]}{\Gamma \vdash \{a/x\}(\phi \rightarrow \psi) \equiv \{a/x\}\phi \rightarrow \{a/x\}\psi} \\ MP \end{array}}{\exists I \frac{\Gamma \vdash \{a/x\}\psi}{\Gamma \vdash \exists x[\psi]}} \quad \forall E \frac{\Gamma \vdash \forall x[\phi]}{\Gamma \vdash \{a/x\}\phi}$$

Unless the same “constant symbol” is used both in the application of $\exists E$ and $\forall E$ the proof will not go through. Here is a proof not involving the use of the constant (which is reminiscent of a proof by contradiction) which assumes there is no value for x which will make ψ true.

A proof without using $\exists E$ ($\Gamma \vdash \exists x[\psi]$)

Let $\Delta = \{\forall x[\phi], \forall x[\neg\psi]\}$

$$\begin{array}{c}
 \forall E \frac{\Delta \vdash \forall x[\phi]}{\Delta \vdash \phi} \quad \forall E \frac{\Delta \vdash \forall x[\neg\psi]}{\Delta \vdash \neg\psi} \\
 \wedge I \frac{}{\Delta \vdash \neg(\phi \rightarrow \psi)} (\neg(\phi \rightarrow \psi) \equiv \phi \wedge \neg\psi) \\
 \forall I \frac{}{\Delta \vdash \forall x[\neg(\phi \rightarrow \psi)]} \\
 DT \Rightarrow \frac{}{\forall x[\phi] \vdash \forall x[\neg\psi] \rightarrow \forall x[\neg(\phi \rightarrow \psi)]}
 \end{array}$$

Note that the application of $DT \Rightarrow$ is correct since no free variable of Δ has been generalised in the proof so far. We may now proceed as follows. First from rule N'' we get

$$N'' \frac{}{\forall x[\phi] \vdash (\forall x[\neg\psi] \rightarrow \forall x[\neg(\phi \rightarrow \psi)]) \rightarrow (\neg \forall x[\neg(\phi \rightarrow \psi)] \rightarrow \neg \forall x[\neg\psi])}$$

An application of MP then yields

$$DT \Leftarrow \frac{\forall x[\phi] \vdash \neg \forall x[\neg(\phi \rightarrow \psi)] \rightarrow \neg \forall x[\neg\psi]}{\forall x[\phi], \neg \forall x[\neg(\phi \rightarrow \psi)] \equiv \exists x[\phi \rightarrow \psi] \vdash \neg \forall x[\neg\psi] \equiv \exists x[\psi]}$$

Note. The whole proof could have been written as a single “monolithic” proof tree with full justification. We have however split the tree and written out the various sub-trees in order to explain some of the steps and to be able to fit it into the limited page-size available.

Remarks on Existential Elimination

1. $\exists E$ is not a derived rule.
2. Proofs which utilize $\exists E$ are denoted $\vdash_{\exists E}$.
3. There are some restrictions on the use of rules in $\vdash_{\exists E}$ proofs.
4. Proofs involving existential formulae which utilize $\exists E$ are likely to be more direct and intuitively simpler than proofs which avoid all uses of $\exists E$ even for existential formulae.

Restrictions on Existential Elimination

Definition 25.1: Correct $\vdash_{\exists E}$ proofs

A $\vdash_{\exists E}$ proof is **correct** provided

1. Each application of $\exists E$ invokes a “fresh constant” symbol not used previously in the proof.
2. If constant symbol a (earlier introduced by an application of $\exists E$ to a formula $\exists y[\psi]$) appears in a formula $\{a/x\}\phi$ in a proof, then $\forall z[\{a/x\}\phi]$ cannot be deduced for any variable $z \in FV(\exists y[\psi]) \cap FV(\{a/x\}\phi)$ (by applying $\forall I$ to $\{a/x\}\phi$).
3. A formula $\{a/x\}\phi$, where a is a constant symbol and $x \in FV(\phi)$ can only be generalised to $\exists x[\phi]$.

Equivalence of Proofs

Theorem 25.1: Existential-Elimination Elimination Theorem

If $\Gamma \vdash_{\exists E} \phi$ is a correct proof then $\Gamma \vdash \phi$ (i.e. ϕ is provable from Γ without invoking $\exists E$) provided no constants introduced in the proof $\Gamma \vdash_{\exists E} \phi$ occur in ϕ .



However this theorem is not applicable if ϕ does contain any of the constants introduced by the proof $\Gamma \vdash_{\exists E} \phi$.

Proof of theorem 25.1

Proof: Let \mathcal{T} rooted at $\Gamma \vdash_{\exists E} \phi$ be a correct proof which involves one or more applications of rule $\exists E$. Assume there are k applications of rule $\exists E$ in $\Gamma \vdash_{\exists E} \phi$.

Claim. For each i , $1 \leq i \leq k$ there exist proof trees

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{i-1}/y_{i-1}\}\psi_{i-1} \vdash_{\exists E} \exists y_i[\psi_i]$$

which are completely free from any application of rule $\exists E$ and

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_i/y_i\}\psi_i \vdash_{\exists E} \phi$$

which have $(k - i)$ applications of rule $\exists E$.

↪ Starting from the highest level of \mathcal{T} there exists a subtree

$$\exists E \frac{\Gamma \vdash \exists y_1[\psi_1]}{\Gamma \vdash_{\exists E} \{a_1/y_1\}\psi_1}$$

such that the proof $\Gamma \vdash_{\exists E} \{a_1/y_1\}\psi_1$ does not involve any application of rule $\exists E$. For $i = 1$, this is also the required tree $\Gamma \vdash_{\exists E} \exists y_1[\psi_1]$.

By adding the formula $\{a_1/y_1\}\psi_1$ to the set of assumptions Γ and removing the subtree $\Gamma \vdash_{\exists E} \exists y_1[\psi_1]$ from \mathcal{T} we get a proof tree

$$\Gamma, \{a_1/y_1\}\psi_1 \vdash_{\exists E} \phi$$

in which there are only $k - 1$ applications of rule $\exists E$ and $\Gamma, \{a_1/y_1\}\psi_1 \vdash_{\exists E} \{a_1/y_1\}\psi_1$ is a leaf node of \mathcal{T}^1 .

Starting with \mathcal{T}^1 we again remove the next application of rule $\exists E$ viz.

$$\exists E \frac{\Gamma, \{a_1/y_1\}\psi_1 \vdash \exists y_2[\psi_2]}{\Gamma, \{a_1/y_1\}\psi_1 \vdash_{\exists E} \{a_2/y_2\}\psi_2}$$

to obtain

$$\Gamma, \{a_1/y_1\}\psi_1, \{a_2/y_2\}\psi_2 \vdash_{\exists E} \phi$$

It is again clear that

$$\Gamma, \{a_1/y_1\}\psi_1 \vdash \exists y_2[\psi_2]$$

does not involve any application of rule $\exists E$ and is the required tree

$$\Gamma, \{a_1/y_1\}\psi_1 \vdash \exists y_2[\psi_2]$$

Proceeding in this fashion we get proof trees

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_i/y_i\}\psi_i \vdash_{\exists E} \phi$$

in which there are exactly $k - i$ applications of rule $\exists E$ and for $i = k$ we get

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_k/y_k\}\psi_k \vdash \phi$$

which is completely free of all applications of rule $\exists E$.

Further it is also clear that for each i , $1 \leq i \leq k$ there exist proof trees

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{i-1}/y_{i-1}\}\psi_{i-1} \vdash \exists y_i[\psi_i]$$

which are also completely free of any application of rule $\exists E$. ⊣

If each of these deductions is replaced by the leaf node

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_k/y_k\}\psi_k \vdash \{a_i/y_i\}\psi_i$$

for each i , $1 \leq i \leq k$, and by adding these extra assumptions to each of the other steps of the proof we get a proof of

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_k/y_k\}\psi_k \vdash \phi$$

which does not have any applications of $\exists E$.

By part 2 of definition 25.1 no variable $z \in \bigcup_{1 \leq i \leq k} FV(\exists y_i[\psi_i])$ is generalized anywhere in the proof of $\Gamma \vdash_{\exists E} \phi$. Hence the conditions for the application of $\text{DT} \Rightarrow$ hold.

By $\text{DT} \Rightarrow$ we get

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \{a_k/y_k\}\psi_k \rightarrow \phi \quad (62)$$

Since the original proof is correct by definition 25.1, there is no occurrence of any of the constants a_i , $1 \leq i \leq k$, in ϕ . Further the proof (62) does not utilise the constant a_k as anything more than a symbol. It also does not generalise on a_k anywhere within the proof. Clearly then replacing this constant a_k by a “fresh” variable symbol z_k (which occurs nowhere in any of the proofs including proof (62)) does not affect the correctness of the proof.

Take a fresh variable z_k which does not occur anywhere in the proof (62) and replace all occurrences of a_k by z_k to obtain proof (63).

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \{z_k/y_k\}\psi_k \rightarrow \phi \quad (63)$$

Proof (63) may now be extended as follows.

$$\forall I \frac{\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \{z_k/y_k\}\psi_k \rightarrow \phi}{\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \forall z_k[\{z_k/y_k\}\psi_k \rightarrow \phi]}$$

which is again a correct proof.

Applying exercise 24.1.5d to the last step gives us

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \exists y_k[\psi_k] \rightarrow \phi$$

By claim 1 we know there exists a proof of

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \exists y_k[\psi_k]$$

which is completely free of any application of rule $\exists E$. By applying rule MP we therefore get

$$\Gamma, \{a_1/y_1\}\psi_1, \dots, \{a_{k-1}/y_{k-1}\}\psi_{k-1} \vdash \phi$$

By a similar process we may eliminate each of the constants a_{k-1} down to a_1 to eventually obtain a proof $\Gamma \vdash \phi$.

QED

Step (63) seems like some sleight of hand procedure which could look unsound to some readers, but its soundness cannot be questioned as the following argument shows.

To simplify our argument, let us assume that a proof of $\Gamma \vdash \exists y[\psi]$ exists. We know that the system \mathcal{H}_1 is sound (theorem 24.3). For simplicity assume that y is the only free variable of ψ . Further let $\Gamma \vdash \{a/y\}\psi \rightarrow \phi$ and suppose the proof obtained by replacing all occurrences of a by a fresh variable z were not sound, then there exists an interpretation \mathbf{A} such that $\mathbf{A} \Vdash \Gamma$, and $\mathbf{A} \Vdash \exists y[\psi]$, but that for some valuation v , where $v(y) = a$, $(\mathbf{A}, v) \Vdash \psi$ holds but $\mathbf{A} \Vdash \phi$ does not hold. But then this implies that $(\mathbf{A}, v) \not\Vdash \{a/y\}\psi \rightarrow \phi$ and hence $\mathbf{A} \not\Vdash \{a/y\}\psi \rightarrow \phi$. However $\Gamma \vdash \{a/y\}\psi \rightarrow \phi$ and the soundness of \mathcal{H}_1 imply that $(\mathbf{A}, v) \Vdash \{a/y\}\psi \rightarrow \phi$ which is a contradiction.

Exercise 25.1: Existential quantification

1. Use the method outlined in the proof of theorem 25.1 to transform the proof $\Gamma \vdash_{\exists E} \exists x[\psi]$ of example 25.1 to one without the use of rule $\exists E$.

Natural Deduction: 6

The introduction and elimination rules for the propositional operators along with the rules $\forall I$, $\forall E$, $\exists I$ and $\exists E$ comprise the system \mathcal{G}_1 .

	Introduction	Elimination
\forall	$\forall I.$ $\frac{\Gamma \vdash \{t/x\}X}{\Gamma \vdash \forall x[X]}$ $\{t/x\}$ admissible in X	$\forall E.$ $\frac{\Gamma \vdash \forall x[X]}{\Gamma \vdash \{t/x\}X}$
\exists	$\exists I.$ $\frac{\Gamma \vdash \{t/x\}X}{\Gamma \vdash \exists x[X]}$	$\exists E.$ $\frac{\Gamma \vdash \exists x[\phi]}{\Gamma \vdash \{a/x\}\phi}$ $a \notin FV(\Gamma) \cup FV(\exists x[\phi])$ is fresh

Exercise 25.2

1. Prove the arguments in Problem 2 of exercise 18.1 using Natural Deduction.
2. Prove the arguments in exercise 18.2 using Natural Deduction.
3. There have been frequent complaints that *Logic* (of any order) is cold-blooded of the first order. Let's dispel this notion. Consider the following premises.

All the world loves a lover. Romeo loves Juliet.

Now prove the following conclusions using Natural Deduction.

- (a) *Therefore I love you.*
 - (b) *Therefore Love loves Love.*^a
 - (c) *Therefore if I love you, then you love me.*
 - (d) *Therefore you love yourself.*
 - (e) *Therefore everyone loves everyone.*
4. Refer to the premises in Problem 3 above. Which of the conclusions becomes invalid if the premise *Romeo loves Juliet* is removed? Further, does it follow that love is an equivalence relation?

^aThis is of course a dirty trick!

26. Normal Forms and Skolemization

26: Normal Forms and Skolemization

1. Moving Quantifiers
2. Quantifier Movement
3. More on Quantifier Movement
4. Prenex Normal Forms
5. The Prenex Normal Form Theorem
6. Prenex Conjunctive Normal Form
7. Skolem's Theorem
8. Equisatisfiability
9. Skolem Normal Forms
10. SCNF

Moving Quantifiers

Notation 26.1

1. $\overrightarrow{\mathcal{O}^x}$ denotes a sequence of quantifiers

$$\mathcal{O}_1 x_1 \mathcal{O}_2 x_2 \dots \mathcal{O}_m x_m$$

where $m \geq 0$ and each $\mathcal{O}_i \in \{\forall, \exists\}$, for $1 \leq i \leq m$.

2. For any quantifier $\mathcal{O} \in \{\forall, \exists\}$, $\overline{\mathcal{O}}$ denotes its dual. That is, if $\mathcal{O} = \forall$, then $\overline{\mathcal{O}} = \exists$ and if $\mathcal{O} = \exists$ then $\overline{\mathcal{O}} = \forall$.

Quantifier Movement

Lemma 26.1: Quantifier movement

Let $z \notin FV(\phi) \cup FV(\psi) \cup \{x_1, \dots, x_n\}$. Then the following logical equivalences hold for $\partial' \in \{\forall, \exists\}$.

1. $\overrightarrow{\partial x} \neg \partial' y [\phi] \Leftrightarrow \overrightarrow{\partial x} \overrightarrow{\partial'} y [\neg \phi]$
2. $\overrightarrow{\partial x} [\partial' y [\phi] \vee \psi] \Leftrightarrow \overrightarrow{\partial x} \partial' z [\{z/y\} \phi \vee \psi]$
3. $\overrightarrow{\partial x} [\phi \vee \partial' y [\psi]] \Leftrightarrow \overrightarrow{\partial x} \partial' z [\phi \vee \{z/y\} \psi]$



More on Quantifier Movement

We may use the above lemma to obtain *prenexing* rules for the propositional connectives \wedge and \rightarrow as well, as shown in the following corollary. However, note the change of quantifier that marks the transformation of \rightarrow in the last equivalence.

Corollary 26.1: Quantifier movement

4. $\overrightarrow{\partial^x}[\partial'y[\phi] \wedge \psi] \Leftrightarrow \overrightarrow{\partial^x}\partial'z[\{z/y\}\phi \wedge \psi]$
5. $\overrightarrow{\partial^x}[\phi \wedge \partial'y[\psi]] \Leftrightarrow \overrightarrow{\partial^x}\partial'z[\phi \wedge \{z/y\}\psi]$
6. $\overrightarrow{\partial^x}[\phi \rightarrow \partial'y[\psi]] \Leftrightarrow \overrightarrow{\partial^x}\partial'z[\phi \rightarrow \{z/y\}\psi]$
7. $\overrightarrow{\partial^x}[\partial'y[\phi] \rightarrow \psi] \Leftrightarrow \overrightarrow{\partial^x}\partial'z[\{z/y\}\phi \rightarrow \psi]$

Exercise 26.1: Quantifier movement

1. Prove lemma [26.1](#).
2. Prove corollary [26.1](#).

Prenex Normal Forms

Definition 26.1: Prenex Normal Forms

The set $\mathcal{PNF}_1(\Sigma)$ of prenex normal forms is generated by the grammar

$$\phi, \psi ::= \chi \in \mathcal{QF}_1(\Sigma) \mid \forall x[\psi] \mid \exists x[\psi]$$

A formula is in prenex normal form if all the quantifiers occurring in the formula appear as a prefix $\overrightarrow{\mathcal{Q}x}$ (called the **prenex**) of a body that is “quantifier-free” and consists only of atomic predicates with propositional connectives.

The Prenex Normal Form Theorem

Theorem 26.1: Prenex Normal Forms

For any formula ϕ there exists a logically equivalent formula ψ in prenex normal form (PNF).



Proof of theorem 26.1

Proof: Given a formula ϕ we go through the following steps.

1. Replace all subformulae of the form $\theta \leftrightarrow \chi$ by subformulae

$$(\theta \rightarrow \chi) \wedge (\chi \rightarrow \theta)$$

respectively to yield a new formula ϕ' which is free of all occurrences of the connective \leftrightarrow .

2. Use α -conversion to obtain unique names for all bound and free variables¹⁵.
3. Now proceed by induction on the structure of ϕ' by systematically applying the results obtained from lemma 26.1 and corollary 26.1. This would yield a formula ψ in prenex normal form.

QED

¹⁵That is, ensure that no two quantifiers use the same bound variable and no variable occurs both free and bound in the formula.

Prenex Conjunctive Normal Form

Given a formula in prenex normal form, its body consists entirely of propositional connectives atomic predicates. By theorem 5.2 every propositional form may be converted into CNF.

We may apply the same method to the body of a formula in PNF to obtain a **Prenex Conjunctive Normal Form (PCNF)**. So we have

Corollary 26.2: PCNF

For any formula ϕ there exists a logically equivalent formula ψ in prenex conjunctive normal form (PCNF).



Skolem's Theorem

Theorem 26.2: Skolem Normal Form Theorem

Let $\phi \equiv \vec{\forall} \mathbf{x} \exists \mathbf{y} [\psi]$ where $\mathbf{x} = x_1, \dots, x_n$ and \mathbf{y} are all distinct variables and ψ does not contain any occurrence of any of the quantifiers ∂x_i ^a. Let $\Sigma_g = \Sigma \cup \{g : s^n \rightarrow s\}$ be an **expansion** of the signature Σ . Then

1. every Σ_g -model of $\phi' \equiv \vec{\forall} \mathbf{x} [\{g(x_1, \dots, x_n)/y\} \psi]$ is a Σ_g -model of ϕ .
2. every Σ -model of ϕ can be expanded to a Σ_g -model of ϕ' .

^a ϕ need not be in PNF and ψ need not be quantifier-free



Note that $\phi' \notin \mathcal{P}_1(\Sigma)$, whereas both $\phi, \phi' \in \mathcal{P}_1(\Sigma_g)$.

Proof of theorem 26.2

Proof: First of all note that $\{g(x_1, \dots, x_n)/y\}$ is admissible in ψ .

1. We have $\models \phi' \rightarrow \phi$ and hence for every Σ_g -structure \mathbf{B} , if $\mathbf{B} \Vdash \phi'$, $\mathbf{B} \Vdash \phi$ holds too.
2. Conversely for every Σ -structure \mathbf{A} , such that $\mathbf{A} \Vdash \phi$, for each $(a_1, \dots, a_n) \in |\mathbf{A}|^n$ there exists *at least one* element $a \in |\mathbf{A}|$ such that

$$\mathcal{T}[\![\psi]\!]_{v[x_1:=a_1, \dots, x_n:=a_n][y:=a]} = 1$$

Define a function g which for each n -tuple $(a_1, \dots, a_n) \in |\mathbf{A}|^n$ provides a single element $a \in |\mathbf{A}|$ such that $\mathcal{T}[\![\psi]\!]_{v[x_1:=a_1, \dots, x_n:=a_n][y:=a]} = 1$. Then clearly for every $(a_1, \dots, a_n) \in |\mathbf{A}|^n$ we have

$$\mathcal{T}[\![\psi]\!]_{v[x_1:=a_1, \dots, x_n:=a_n][y:=g_{\mathbf{A}}(a_1, \dots, a_n)]} = 1$$

Now let $\mathbf{A} \triangleleft \mathbf{B}$ where \mathbf{B} is a Σ_g -algebra with $g_{\mathbf{B}} = g$. It is then clear that for every valuation $v_{\mathbf{B}}$,

$$\mathcal{T}[\![\{g(x_1, \dots, x_n)/y\}\psi]\!]_{v_{\mathbf{B}}} = 1$$

and hence $\mathbf{B} \Vdash \phi'$.

QED

Equisatisfiability

Corollary 26.3

Let ϕ and ϕ' be as in theorem 26.2. Then

1. there exists a model of ϕ iff there exists a model of ϕ' .
2. ϕ is unsatisfiable iff ϕ' is unsatisfiable.



Skolem Normal Forms

Definition 26.2

1. The set $\mathcal{SNF}_1(\Sigma)$ of **Skolem normal forms (SNF)** is the set of **universal closures** of **quantifier-free formulae**. The quantifier-free formula is called the **body** of the SNF.
2. A formula is in **Skolem conjunctive normal form (SCNF)** if it is a SNF whose body is in **CNF**. $\mathcal{SCNF}_1(\Sigma)$ is the set of Σ -formulae in SCNF.

SCNF

Theorem 26.3: Skolemization

For every sentence (closed formula) $\phi \in \mathcal{P}_1(\Sigma)$ there is an algorithm *sko* to construct a closed universal formula $\psi \in \mathcal{SCNF}_1(\Sigma')$ such that $\Sigma \triangleleft \Sigma'$ and ϕ has a Σ -model iff ψ has a Σ' -model.

**Definition 26.3**

1. The function g in theorem 26.2 is called a **Skolem function**
2. The process of constructing the function g in theorem 26.2 is called **Skolemization**.
3. ϕ and $sko(\phi) \equiv \psi$ are said to be **equisatisfiable**.

Proof of theorem 26.3

Proof: The procedure may be briefly outlined as follows.

1. Clearly if ϕ is a closed formula, by corollary 26.2 we can construct a logically equivalent formula $\phi_0 \in \mathcal{PCNF}_1(\Sigma)$.
2. If there is no existential quantifier ϕ_0 then the formula ϕ_0 is in SCNF. Otherwise Skolemize the left-most occurrence of an existential quantifier to obtain a formula ϕ_1 .
3. ϕ_1 has one existential quantifier less than ϕ_0 . Perform step 2 on ϕ_1 .

Each execution of step 2 of the above procedure results in a formula which satisfies the conclusions of theorem 26.2.

QED

Exercise 26.2: Skolemization

1. Prove that $\models \phi' \rightarrow \phi$ in theorem 26.2.
2. Prove that $\vdash \phi' \rightarrow \phi$ in theorem 26.2.
3. Skolemization does not produce a SNF that is unique upto logical equivalence. Construct an example of a formula ϕ which has two (or even more) different SNFs, ϕ' , ϕ'' such that

$$\phi \not\equiv \phi' \not\equiv \phi'' \not\equiv \phi$$

27. Herbrand's theorem

27: Herbrand's theorem

1. The Herbrand Algebra
2. Terms in a Herbrand Algebra
3. Herbrand Interpretations
4. Herbrand Models
5. Ground Quantifier-free Formulae
6. Ground Instances
7. Herbrand's Theorem
8. The Herbrand Tree of Interpretations
9. Compactness of Sets of Ground Formulae
10. Compactness of Closed Formulae
11. The Löwenheim-Skolem Theorem

The Herbrand Algebra

Definition 27.1: The Herbrand Universe

Let Σ be a signature containing at least one constant symbol a . $\mathbb{T}_\Sigma \subseteq \mathbb{T}_\Sigma(V)$ the set of ground terms (see definition -1.6) is also called the Herbrand Universe. A literal $p(t_1, \dots, t_n)$ or $\neg p(t_1, \dots, t_n)$ containing no variables is called a ground literal.

Substitution lemma for terms

Substitution lemma for formulae

Definition 27.2: The Herbrand Algebra

A Σ -algebra $H(\Sigma)$ where Σ has at least one constant symbol, is called a **Herbrand algebra** iff $|H(\Sigma)| = \mathbb{T}_\Sigma$.

Terms in a Herbrand Algebra

In a Herbrand algebra $\mathbf{H}(\Sigma)$

- every function symbol represents itself. That is for each $f : s^m \rightarrow s \in \Sigma$,
 $f_{\mathbf{H}(\Sigma)} = f$
- a valuation is simply a function $v_{\mathbf{H}(\Sigma)} : V \rightarrow T_\Sigma$
- However the predicate symbols require to be associated with relations on ground terms in some way.
- When Σ is understood, we will often omit it and simply write \mathbf{H} instead of $\mathbf{H}(\Sigma)$ and $v_{\mathbf{H}}$ instead of $v_{\mathbf{H}(\Sigma)}$.

Herbrand Interpretations

Lemma 27.1: Herbrand interpretations

Given a Herbrand interpretation $(\mathbf{H}, v_{\mathbf{H}})$ where for each variable x , $v_{\mathbf{H}}(x) = s_x \in \mathbb{T}_{\Sigma}$. For any term t with $Var(t) = \{x_1, \dots, x_k\}$

$$\mathcal{V}_{\mathbf{H}}[t]_{v_{\mathbf{H}}} = \{s_{x_1}/x_1, \dots, s_{x_k}/x_k\}t$$

That is every valuation defines a substitution of variables by ground terms.



Here valuations represent merely substitutions on terms. Also see **Substitution lemma for terms**.

Herbrand Models

Definition 27.3: Herbrand model

A **Herbrand model** of a set Φ of Σ -formulae is merely a valuation v_H such that every formula in Φ is true under the substitution defined by $v_H \upharpoonright FV(\Phi)$, where $FV(\Phi) = \bigcup_{\phi \in \Phi} FV(\phi)$.

Theorem 27.1: Ground quantifier-free formulae

Let Σ be a signature containing at least one constant and let $\Lambda = \{\lambda_1, \dots, \lambda_k\}$ be a nonempty finite set of **ground literals**. Then

1. $\bigwedge_{1 \leq i \leq k} \lambda_i$ has a model iff Λ does not contain a **complementary pair**.
2. $\bigwedge_{1 \leq i \leq k} \lambda_i$ is never logically valid.
3. $\bigvee_{1 \leq i \leq k} \lambda_i$ always has a model.
4. $\bigvee_{1 \leq i \leq k} \lambda_i$ is logically valid iff it has a **complementary pair**.

Proof of theorem 27.1

Proof: Notice that every literal in Λ is ground and hence each of them is an instance of an atomic predicate (or negation of one) and has no free variables. Hence each of them is actually a sentence (pure proposition). Further note that for $i \neq j$, λ_i and λ_j could be instances of the same atomic predicate symbol or instances of negations of the same atomic predicate symbol or one could be a positive instance and the other a negative instance of the same atomic predicate.

1. Clearly if Λ contains a complementary pair it does not have a model since $\bigwedge_{1 \leq i \leq k} \lambda_i \Leftrightarrow \perp$. Conversely assume it does not contain a complementary pair. We may define a Herbrand model (algebra) \mathbf{H}_Λ as follows: For each atomic predicate symbol $p : s^n$ define $p_{\mathbf{H}_\Lambda} = \{(t_1, \dots, t_n) \in \mathbb{T}_\Sigma^n \mid p(t_1, \dots, t_n) \in \Lambda\}$. Clearly $\mathbf{H}_\Lambda \Vdash \lambda_i$ for each $\lambda_i \in \Lambda$ since if $\lambda_i \equiv p(t_1, \dots, t_n)$ and then $p(t_1, \dots, t_n) \in \Lambda$ and $(t_1, \dots, t_n) \in p_{\mathbf{H}_\Lambda}$. On the other hand if $\lambda_i \equiv \neg p(t_1, \dots, t_n)$ then $p(t_1, \dots, t_n) \notin \Lambda$ and hence $(t_1, \dots, t_n) \notin p_{\mathbf{H}_\Lambda}$ otherwise it would contradict the assumption that Λ contains no complementary pair. Hence $\mathbf{H}_\Lambda \Vdash \lambda_i$ for each λ_i . Hence $\bigwedge_{1 \leq i \leq k} \lambda_i$ has a model.
2. $\bigwedge_{1 \leq i \leq k} \lambda_i$ cannot be valid unless each λ_i is valid. From the previous part we know that both λ_i and $\overline{\lambda_i}$ have models, where $\overline{\lambda_i}$ is the complement of λ_i .
3. $\bigvee_{1 \leq i \leq k} \lambda_i$ has a model because λ_i has a model.
4. $\bigvee_{1 \leq i \leq k} \lambda_i$ is valid iff $\bigwedge_{1 \leq i \leq k} \overline{\lambda_i}$ has no model iff $\overline{\Lambda} = \{\overline{\lambda_i} \mid 1 \leq i \leq k\}$ contains a complementary pair iff Λ contains a complementary pair.

QED

Ground Instances

Definition 27.4: Ground instances

Let Σ be a signature containing at least one constant and let Φ be a nonempty set of **closed universal** Σ -formulae. For any $\phi \equiv \vec{\forall}[\chi]$ where $\chi \in \mathcal{QF}_1(\Sigma)$, the **ground-instances** of ϕ denoted $\mathfrak{g}(\phi)$ is the set $\{\{t_1/x_1, \dots, t_n/x_n\}\chi \mid FV(\chi) = \{x_1, \dots, x_n\}, t_1, \dots, t_n \in \mathbb{T}_\Sigma\}$ and $\mathfrak{g}(\Phi) = \bigcup_{\phi \in \Phi} \mathfrak{g}(\phi)$.

Herbrand's Theorem

Theorem 27.2: Herbrand's Theorem

Let Σ and Φ be as in definition 27.4. Then the following statements are equivalent.

1. Φ has a model.
2. Φ has a Herbrand model.
3. $\text{g}(\Phi)$ has a model.
4. $\text{g}(\Phi)$ has a Herbrand model.



Proof of theorem 27.2

Proof: Clearly the following implications are trivial.

$$\begin{array}{ccc} \text{Statement 2: } " \Phi \text{ has a Herbrand model}" & \Rightarrow & \text{Statement 1: } " \Phi \text{ has a model}" \\ \downarrow & & \downarrow \\ \text{Statement 4: } " \mathbf{g}(\Phi) \text{ has a Herbrand model}" & \Rightarrow & \text{Statement 3: } " \mathbf{g}(\Phi) \text{ has a model}" \end{array}$$

It suffices therefore to prove only the following claim.

Claim. Statement 3 \Rightarrow Statement 2.

\vdash Let $\mathbf{A} \Vdash \mathbf{g}(\Phi)$. We define a Herbrand interpretation $(\mathbf{H}, v_{\mathbf{H}})$ as follows. For each $p : s^n \in \Sigma$ let

$$p_{\mathbf{H}} = \{(t_1, \dots, t_n) \in \mathbb{T}_{\Sigma} \mid \mathbf{A} \Vdash p(t_1, \dots, t_n)\}$$

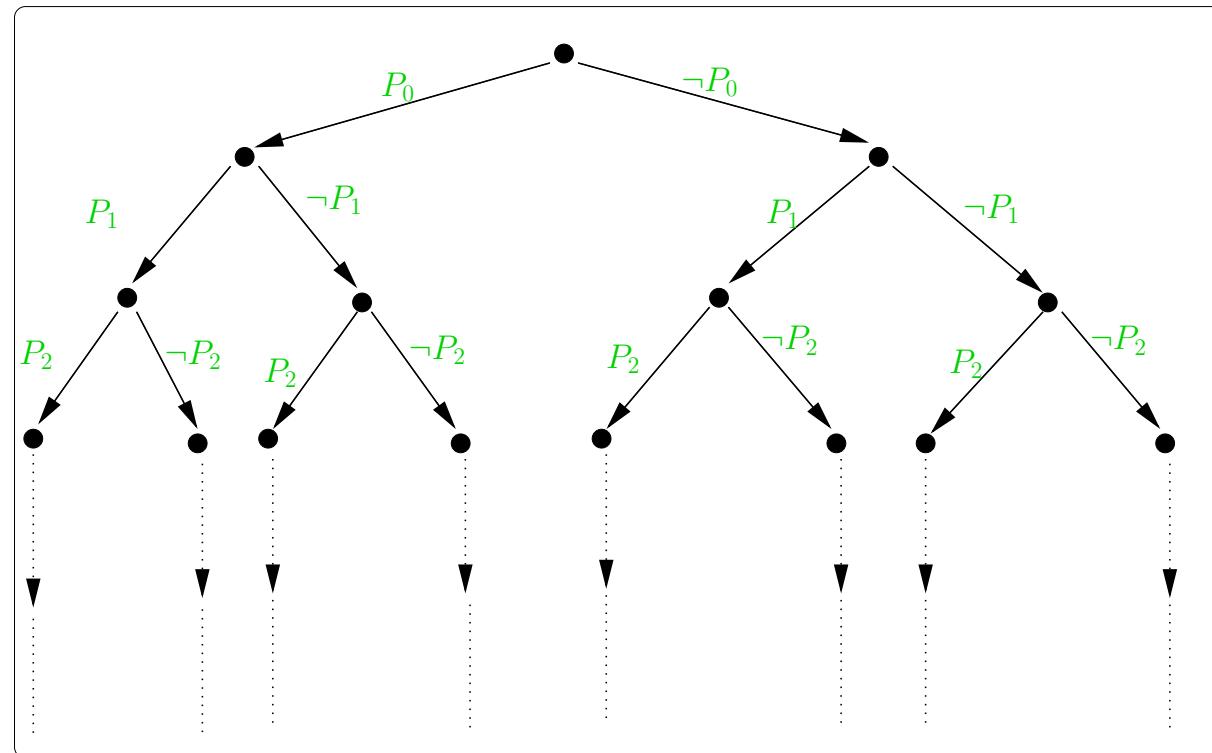
In particular if p is an atomic proposition then $p_{\mathbf{H}} = p_{\mathbf{A}}$. With this construction exactly the same atomic formulae are valid in both \mathbf{A} and \mathbf{H} . This result may be extended by structural induction to arbitrary universally closed quantifier-free formulae. It is then easy to see that if $\mathbf{A} \Vdash \mathbf{g}(\Phi)$ then $\mathbf{H} \Vdash \Phi$. \dashv

QED

The Herbrand Tree of Interpretations

Let Σ be a signature containing at least one constant symbol. Let P_0, P_1, P_2, \dots be an enumeration of all the *ground atomic formulae* of $\mathcal{P}_1(\Sigma)$. The **Herbrand Tree of interpretations** is the infinite tree shown **schematically**. Each infinite path (called a **Herbrand Base**) of the tree represents a Herbrand interpretation.

Herbrand Tree of Interpretations: Schematic



Compactness of Sets of Ground Formulae

Lemma 27.2: Compactness of ground quantifier-free formulae

Let Θ be a (finite or infinite) set of ground quantifier-free formulae. Then Θ has a model iff every finite subset of Θ has a model.



Proof of lemma 27.2

Proof: (\Rightarrow) Clearly if Θ has a model then every finite subset of Θ also has a model.

(\Leftarrow) Assume **every finite subset of Θ has a model but Θ itself does not have a model**. By Herbrand's theorem (theorem 27.2) each finite subset of Θ has a Herbrand model. We identify each path π in the **Herbrand tree** with a valuation v_π . Then since Θ does not have a model, it does not have a Herbrand model. Hence for every path π there exists a formula $\chi_\pi \in \Theta$ such that $(\mathbf{H}, v_\pi) \not\models \chi_\pi$. In fact, there exists a finite point ℓ_{χ_π} in each path π at which $(\mathbf{H}, v_\pi) \not\models \chi_\pi$ since χ_π is made up of only a finite number of ground atoms.

Claim. $\{\chi_\pi \mid (\mathbf{H}, v_\pi) \not\models \chi_\pi\}$ is a finite set.

\vdash Consider the tree \mathcal{T}_H obtained from the **Herbrand tree** such that from each path π the subtree rooted at $\ell_{\chi_\pi} + 1$ has been removed. Hence \mathcal{T}_H is a finitely branching tree with only finite-length paths. Hence by (the contra-positive of) König's lemma (corollary -2.2) any finitely-branching tree with only finite-length paths must be finite) \mathcal{T}_H must be a finite tree where each path π' is an initial segment of an infinite path π from the Herbrand tree of interpretations. The leaf-nodes of each of these paths π' determines a formula χ_π that is not satisfied. Clearly then the set consisting of these formulae viz. $\{\chi_\pi \mid (\mathbf{H}, v_\pi) \not\models \chi_\pi\}$ is then a finite set. \dashv

It is clear that the finite set $\{\chi_\pi \mid (\mathbf{H}, v_\pi) \not\models \chi_\pi\}$ is a finite subset of Θ that does not possess a Herbrand model, contradicting the assumption that all finite subsets of Θ possess a Herbrand model. QED

Compactness of Closed Formulae

Theorem 27.3: Compactness of closed formulae

A set Φ of closed Σ -formulae has a model iff every finite subset of Φ has a model.



Corollary 27.1: Finite Unsatisfiability

A set Φ of closed Σ -formulae is unsatisfiable iff there is a non-empty finite unsatisfiable subset of Φ .



Proof of theorem 27.3

Proof: (\Rightarrow) is trivial.

(\Leftarrow) Assume Φ does not possess a model but every finite subset of Φ has a model. Transform each formula into SNF. Since Φ has no model $sko(\Phi) = \{sko(\phi) \mid \phi \in \Phi\}$ has no model either (by corollary 26.3). By Herbrand's theorem (theorem 27.2) the set $g(sko(\Phi))$ also does not possess a model. By lemma 27.2 we can find a finite subset of $g(sko(\Phi))$ which does not have a Herbrand model. This finite set is a subset of a finite subset of Φ that does not possess a model. Hence there is a finite subset of Φ which does not have a model, contradicting the assumption that every finite subset of Φ has a model. QED

The Löwenheim-Skolem Theorem

Theorem 27.4: The Löwenheim-Skolem Theorem

If a set Φ of closed formulae has a model, then it has a model with a domain which is at most countable.

Proof: Assume Φ has a model. Then $sko(\Phi)$ has a model too. By theorem 27.2 $sko(\Phi)$ has a Herbrand model. Since a Herbrand model has a domain which is at most countable and since every model of $sko(\Phi)$ is also a model of Φ , it follows that Φ has a model with at most a countable domain. QED

28. Substitutions and Unification

28: Substitutions and Unification

1. Ground Substitutions
2. Unifiability
3. Unification Examples:1
4. Unification Examples:2
5. Elementary Facts
6. Generality of Unifiers
7. Generality: Facts
8. Most General Unifiers
9. More on Positions
10. Disagreement Set
11. Example: Disagreement 1
12. Example: Occurs Check
13. Example: Disagreement 3
14. Example: Disagreement 4
15. Disagreement and Unifiability
16. Example: Unify 1
17. Example: Unify 2
18. The Unification Theorem
19. Unification: consequences

Ground Substitutions

From section 0 recall that

- $\theta = \{t_i/x_i \mid t_i \not\equiv x_i, 1 \leq i \leq n\}$ is a **ground substitution** if each t_i , $1 \leq i \leq n$, is a ground term.
- A term u is called an **instance** of a term t if there exists a substitution θ such that $u \equiv \theta t$.
- u is a **ground instance** of t if u is an instance of t and is ground.
- u is a **common instance** of two or more terms t_1, \dots, t_n if there exist substitutions $\theta_1, \dots, \theta_n$ such that

$$u \equiv \theta_1 t_1 \equiv \dots \equiv \theta_n t_n$$

- Terms t and u are called **variants** of each other if there exist substitutions θ and τ such that $\theta t \equiv u$ and $\tau u = t$.

28.1. Syntactic Unification as equation solving

Syntactic unification is the problem of finding substitutions (definition 0.1) θ so as to make two or more terms syntactically identical. It may be thought of as a special form of equation solving where one attempts to find solutions θ to the problem $\theta s \equiv \theta t$ by finding suitable instances of the variables in the two terms in order to make the two terms look syntactically identical. The solution of such an equation on essentially uninterpreted terms is a substitution and the process of finding this solution is called *unification*. As in normal equation solving the substitution is to be applied to all the terms that have to be unified. Moreover as in equation solving, it is possible that no solution exists. A set consisting of two or more terms is said to be *unifiable* if such a substitution exists.

More generally, given a non-empty finite collection of Σ -terms, $T = \{t_i \in \mathbb{T}_\Sigma(V) \mid 1 \leq i \leq n\}$, the problem of unification is to find a substitution θ , if one exists, such that

$$|\theta T| = |\{\theta t_i \in \mathbb{T}_\Sigma(V) \mid 1 \leq i \leq n\}| = 1$$

and otherwise to be able to declare that T cannot be unified.

Unifiability

Definition 28.1: Unifiability

A nonempty finite set of terms $\{t_i \mid 1 \leq i \leq n\}$, $n > 1$ is said to be **unifiable** if there exists a substitution θ such that

$$\theta t_1 \equiv \theta t_2 \equiv \cdots \equiv \theta t_n$$

θ is called a **unifier** of $\{t_i \mid 1 \leq i \leq n\}$

Unification Examples:1

Example 28.1

Let f and g be distinct binary operators and $x, y, v, w \in V$.

1. The terms $f(x, y)$ and $f(v, w)$ may be unified by the substitution $\theta = \{x/v, y/w\}$ since $\theta f(v, w) \equiv f(x, y) \equiv \theta f(x, y)$. They may also be unified by $\theta^{-1} = \{v/x, w/y\}$.
2. Let r, s, t be any three terms. Then $f(x, y)$ and $f(v, w)$ may be unified by the substitution $\tau = \{g(s, t)/v, f(r, r)/w, g(s, t)/x, f(r, r)/y\}$.
3. The terms $f(x, y)$ and $f(y, x)$ may be unified by $\chi = \{x/y\}$ since $\chi f(x, y) \equiv f(x, x) \equiv \chi f(y, x)$.

Unification Examples:2

Example 28.2

Let f and g be distinct binary operators.

1. The terms $f(x, y)$ and $g(x, y)$ cannot be unified by any substitution.
2. The terms $f(x, y)$ and $f(y, x)$ cannot be unified by $\rho = \{x/y, y/x\}$ since $\rho f(x, y) \equiv f(y, x)$ and $\rho f(y, x) \equiv f(x, y)$. Hence $\rho f(x, y) \not\equiv \rho f(y, x)$.

Elementary Facts

Refer definition -1.11, notation -1.3 to fact -1.1.

Fact 28.1

Let s and t be any two terms and let $p \in Pos = Pos(s) \cap Pos(t)$. Then

1. If $s|_p \equiv t|_p$ for any position $p \in Pos$ then for all positions $p, q \in Pos$, $p \preceq q$ implies $s|_q \equiv t|_q$.
2. If $rootsym(s|_p) = o_1 \not\equiv o_2 = rootsym(t|_p)$ then s and t are not unifiable under any substitution.
3. If s and t are unifiable then for every position $p \in pos$, $rootsym(s|_p) \not\equiv rootsym(t|_p)$ implies at least one of the symbols is a variable i.e. $\{rootsym(s|_p), rootsym(t|_p)\} \cap V \neq \emptyset$

Exercise 28.1

1. Generalize the fact 28.1 to nonempty finite sets of terms.
2. Construct an example to show that the converse of fact 28.1.3 does not hold.

Generality of Unifiers

There is a certain sense in which θ may be regarded as being more general than τ in example 28.1.

Definition 28.2

- A substitution θ is *at least as general* as another substitution τ (denoted $\theta \gtrsim \tau$) if there exists a substitution χ such that $\tau = \chi \circ \theta$.
- $\theta \sim \tau$ if $\theta \gtrsim \tau \gtrsim \theta$.
- θ is *strictly more general* than τ (denoted $\theta \not\gtrsim \tau$) if $\theta \gtrsim \tau$ and $\tau \not\gtrsim \theta$.

Generality: Facts

Fact 28.2

1. \gtrsim is a preordering relation on $\mathbf{S}_\Omega(V)$ i.e. it is a reflexive and transitive relation.
2. \gtrless is an irreflexive and transitive relation on $\mathbf{S}_\Omega(V)$.
3. \sim is an equivalence relation on $\mathbf{S}_\Omega(V)$.
4. If $\theta \sim \tau$ and $\rho \circ \theta = \tau$ then ρ is a pure-variable substitution.

Most General Unifiers

Definition 28.3: MGU

Let $T = \{t_i \mid 1 \leq i \leq n\}$ be a unifiable set of terms. A substitution θ is called a **most general unifier (mgu)** of T if for each unifier τ of T , there exists a substitution ρ such that $\tau = \rho \circ \theta$.

Fact 28.3

1. If a set of terms T is unifiable then it has a mgu.
2. If θ and τ are both mgu's of a set T then $\theta \sim \tau$.
3. If θ and τ are both mgu's of a set T then there exist (pure-variable) substitutions $\rho, \rho^{-1} : V \longrightarrow V$ such that $\rho \circ \theta = \tau$ and $\theta = \rho^{-1} \circ \tau$

More on Positions

Definition 28.4: Common set of positions

For any nonempty set of terms T ,

- $Pos(T) = \bigcap_{t \in T} Pos(t)$,
- For any position $p \in Pos(T)$, $T|_p = \{t|_p \mid t \in T\}$
- $rootsym(T|_p) = \{rootsym(t|_p) \mid t \in T\}$

Every term in T has a root and hence

Fact 28.4

If T is a non-empty set of terms then $Pos(T) \neq \emptyset$.

Disagreement Set

Definition 28.5: Disagreement

Given a set T ($|T| > 1$) of terms (also viewed as a set of abstract syntax trees), the **disagreement set** of T is defined as the set $T|_q$ of subterms rooted at some position q such that

1. not all the terms in $T|_q$ have the same root symbol and
2. for every $p \prec q$, $|rootsym(T|_p)| = 1$, where \prec is the proper-prefix relation on strings.

We have seen that $Pos(t)$ for any term t is partially ordered by the relation \prec which inverts the proper sub-term ordering (definition -1.12) on $ST(t)$ (Fact -1.1 part 3b). For the purpose of specifying the unification algorithm, it is useful to define a *total order* $<$ on the positions of terms which is consistent with \prec . Intuitively if $u = o(t_1, t_2 \dots, t_n)$ we would like to specify *recursively* that

- the root position u precedes the root positions of all the subterms t_1, \dots, t_n . (which is taken care of by the prefix ordering \prec on positions) i.e. $\epsilon < i$ for all $1 \leq i \leq n$
- for each i, j such that $1 \leq i < j \leq n$, the position of the root of t_i precedes that of t_j in the total ordering.
- If $i < j$, then the position of the root of any proper subterm of t_i precedes the position of any subterm of t_j (including the root).

Definition 28.6

For any positions $p, q \in pos(t)$, $p < q$ iff one of the following conditions holds.

- $p = \epsilon \neq q$ or
- $(p = i.p', q = i.q' \text{ and } p' < q')$ or
- $(p = i.p', q = j.q' \text{ and } i < j)$.

If all operators in Ω are always used in prefix form then each term may also be regarded as a string in $(\Sigma \cup \{(\,),\})^*$. The ordering $<$ on $pos(t)$ simply becomes the left-to-right ordering of symbols in the well-formed terms of $\mathcal{T}_\Omega(V)$ represented as strings. In the following algorithm we compute the disagreement set. The function Min used in the algorithm 28.1 is the minimum position with respect to the total ordering $<$ on positions (definition 28.6) in a term.

Algorithm

Algorithm 28.1

DISAGREEMENT (T) $\stackrel{df}{=}$

```

    { requires:  $T \subseteq_f \mathbb{T}_\Omega(V) \wedge |T| > 1$ 
      DISAGREE ( $T, \epsilon, Pos(T) - \{\epsilon\}$ )
      where
      rec fun DISAGREE ( $T, p, P$ )  $\stackrel{df}{=}$ 

        let  $p' := Min(P)$ 
        in
        {
          {  $|rootsym(T|_p)| = 1 \rightarrow \begin{cases} P \neq \emptyset \rightarrow \text{DISAGREE } (T, p', P - \{p'\}) \\ \text{else } \rightarrow \text{No disagreement} \end{cases}$ 
            { else  $\rightarrow T|_p$  is the disagreement set
        }
    }
  
```

Example: Disagreement 1

Example 28.3

Consider the set of terms

$$S_1 = \{f(a, x, h(g(z))), f(z, h(y), h(y))\}$$

where a is a constant, f is a ternary operator and g and h are unary operators. In this case, reading the terms from left to right we get a disagreement set $D_1 = \{a, z\}$. On the other hand, reading from right to left we obtain the disagreement set $D'_1 = \{g(z), y\}$ which requires going down one level deeper.

The algorithm 28.1 however will compute the leftmost disagreement D_1 always.

Example: Occurs Check

Example 28.4: Occurs check

Consider the set

$$S_2 = \{f(g(z), x, h(g(z))), f(z, h(y), h(y))\}$$

The disagreement set $\{g(z), z\}$ is such that S_2 is not unifiable, because for any substitution θ , θz can never be syntactically identical with $\theta g(z)$. This is an example of the notorious **occurs check** problem. Hence S_2 is not unifiable.

Example: Disagreement 3

Example 28.5

Consider the set

$$S_3 = \{f(a, x, h(g(z))), f(b, h(y), h(y))\}$$

where a and b are both constant symbols. Here a disagreement set is $D_3 = \{a, b\}$. Again it is clear that S_3 is not unifiable.

Example: Disagreement 4

Example 28.6

Consider the set

$$S_4 = \{f(h(z), x, h(g(z))), f(g(x), h(y), h(y))\}$$

Here we have a disagreement set $D_4 = \{h(z), g(x)\}$. Since $h(z)$ cannot be unified with $g(x)$ under any substitution of free variables, S_4 is not unifiable.

Disagreement and Unifiability

Fact 28.5: Disagreement and unifiability

If S' is the disagreement set of S then

1. S is unifiable implies S' is unifiable.
2. If S is unifiable and θ' is a mgu of S' then there exists a mgu θ of S such that $\theta' \gtrsim \theta$.

The above facts reduce the problem of finding a unifier if it exists, to that of systematically finding disagreement sets and unifying them.

Finding a unifier for a disagreement set is a pre-requisite for finding a unifier for the original set of terms. A disagreement set consists of subterms of the original set of terms at a particular position such that at least two distinct (sub-)terms exist in the set. Further a disagreement set is unifiable only if there is at most one non-variable term in it (the rest are all variables). By choosing a substitution $\{t/x\}$ where both t and x are terms in the disagreement set satisfying the condition $x \notin FV(t)$, there is a possibility of unifying the disagreement set. Each such substitution will unify the (sub-)term with the leaf node x with the (sub-)term t . Each of the other variables in the disagreement set will be found one-by-one in sequence and unified whenever possible.

The unification algorithm 28.2 constructs a sequence of singleton substitutions whose composition yields a most general unifier if it exists. We have used the expression $FV(S)$ instead of the more usual $Var(S)$, to keep it general enough to include operator schemas. However the usual caveats regarding **admissibility** and capture of free variables need to be respected in every substitution in the algorithm.

Algorithm

Algorithm 28.2 $\text{UNIFY } (S) \stackrel{df}{=}$

```

    requires:  $S \subseteq_f \mathbb{T}_\Omega(V) \wedge |S| > 1$ 
    let  $V := FV(S)$  in PARTIALUNIFY ( $\mathbf{1}, S$ )
    where
    rec fun PARTIALUNIFY  $(\theta, T) \stackrel{df}{=}$ 
         $|T| = 1 \rightarrow \theta$ 
        else  $\rightarrow \begin{cases} \text{let } D := \text{DISAGREEMENT } (T) \text{ //algorithm 28.1} \\ \text{in} \\ \quad \exists x \in D \cap V : \forall t \in T [x \notin FV(t)] \rightarrow \text{Choose } t \in T : x \notin FV(t); \text{PARTIALUNIFY } (\{t/x\} \circ \theta, \{t/x\} T) \\ \quad \text{else} \end{cases}$ 
                 $\rightarrow \text{Occurs-check fails. } S \text{ is not unifiable}$ 

```

ensures: If S is not unifiable then fail else $\exists \theta \in \mathbf{S}_\Omega(V) : |\theta S| = 1$ and θ is a mgu of S

Example: Unify 1

Example 28.7

Consider the set $S = S_1$ in example 28.3. Starting with $\theta_0 = \mathbf{1}$ we go through the following steps to obtain a unifier of S .

i	θ_i	$\theta_i S$	D_i
0	$\theta_0 = \mathbf{1}$	$\theta_0 S = \{f(a, x, h(g(z))), f(z, h(y), h(y)\}$	$D_0 = \{a, z\}$
1	$\theta_1 = \{a/z\} \circ \theta_0$	$\theta_1 S = \{f(a, x, h(g(\textcolor{red}{a}))), f(\textcolor{red}{a}, h(y), h(y)\}$	$D_1 = \{x, h(y)\}$
2	$\theta_2 = \{h(y)/x\} \circ \theta_1$	$\theta_2 S = \{f(a, \textcolor{red}{h}(y), h(g(\textcolor{red}{a}))), f(a, h(y), h(y)\}$	$D_2 = \{g(\textcolor{red}{a}), y\}$
3	$\theta_3 = \{g(a)/y\} \circ \theta_2$	$\theta_3 S = \{f(a, h(\textcolor{red}{g}(a)), h(\textcolor{red}{g}(a))\}$	$D_3 = \emptyset$

Hence the required unifier is $\theta_3 = \{g(a)/y\} \circ \{h(y)/x\} \circ \{a/z\} \circ \mathbf{1} = \{a/z, h(g(z))/x, g(z)/y\}$.

Example: Unify 2

Example 28.8

Let $S = \{f(y, z, w), f(g(x, x), g(y, y), g(z, z))\}$ where f is a ternary operator and g is a binary operator. An attempt to apply the **algorithm** yields the following sequence of substitutions: $\theta_1 = \{g(x, x)/y\}$ from which we get $\theta_1 S = \{f(g(x, x), z, w), f(g(x, x), g(g(x, x), g(x, x)), g(z, z))\}$ and then $\theta_2 = \{g(g(x, x), g(x, x))/z\}$ which yields

$\theta_2 S = \{f(g(x, x), g(g(x, x), g(x, x)), w), f(g(x, x), g(g(x, x), g(x, x)), g(g(x, x), g(x, x))), g(g(g(x, x), g(x, x)), g(g(x, x), g(x, x)))\}$
and finally $\theta_3 = \{g(g(g(x, x), g(x, x)), g(g(x, x), g(x, x)))/w\} \circ \theta_2$

Hence in general there are pathological cases which make the algorithm very expensive to run, having a complexity that is exponential in the length of the input i.e. to unify the set

$$\{f(x_1, \dots, x_n), f(g(x_0, x_0), \dots, g(x_{n-1}, x_{n-1}))\}$$

would require a substitution that has $2^k - 1$ occurrences of the symbol g in the substitution of the variable x_k .

The Unification Theorem

Theorem 28.1: The Unification Theorem

The unification algorithm 28.2 terminates satisfying its postcondition (If S is not unifiable then **fail** else $\exists \theta \in S_\Omega(V) : |\theta S| = 1$ and θ is a mgu of S) for any set of terms satisfying its preconditions ($S \subseteq_f T_\Omega(V)$ and $|S| > 1$).



Proof of theorem 28.1

Proof:

Claim. Termination

Let $V_0 = \bigcup_{s \in S} FV(s)$ be the (finite) set of free variables occurring in S . With each evaluation of the condition $\exists x \in D \cap V : \forall t \in T[x \notin FV(t)]$ in unification algorithm 28.2 either it terminates because it fails or a substitution $\{t_{j+1}/x_{j+1}\}$ such that $x_{j+1} \notin FV(t_{j+1})$ is generated with $\theta_{j+1} = \{t_{j+1}/x_{j+1}\} \circ \theta_j$, we have $\theta_{j+1}S$ has one variable less than θ_jS . Since V_0 is finite the algorithm must terminate. \dashv

Claim. If S is not unifiable then it terminates returning `fail`.

Let Trivial. \dashv

Claim. If S is unifiable then it terminates returning a most general unifier θ

Let ρ be any unifier of S , and let $1 = \theta_0, \theta_1, \dots, \theta_k = \theta$ be the sequence of substitutions generated by the algorithm. We prove by induction that for every θ_i , there exists a substitution τ_i such that $\rho = \tau_i \circ \theta_i$.

Basis. For $i = 0$ clearly $\tau_0 = \rho$.

Induction Hypothesis (IH).

Assume for some j , $0 \leq j < k$, there exists τ_j such that $\rho = \tau_j \circ \theta_j$.

Induction Step. Clearly since θ_jS is not a singleton, a disagreement set D_j will be found for θ_jS . Since $\rho = \tau_j \circ \theta_j$ is a unifier of S , clearly τ_j must unify D_j , which means there exists a variable x and a term t with $x \notin FV(t)$ such that τ_j unifies D_j which in effect implies $\tau_jx = \tau_jt$. Without loss of generality we may assume $\{t/x\}$ is the chosen substitution so that $\theta_{j+1} = \{t/x\} \circ \theta_j$. Now define $\tau_{j+1} = \tau_j - \{\tau_jx/x\}$.

Case $x \in \text{dom}(\tau_j)$. Then $\tau_j = \{\tau_j x/x\} \cup \tau_{j+1} = \{\tau_j t/x\} \cup \tau_{j+1}$. Since $x \notin FV(t)$, we have $\tau_j = \{\tau_{j+1} t/x\} \cup \tau_{j+1} = \tau_{j+1} \circ \{t/x\}$ by the definition of composition. Finally from $\rho = \tau_j \circ \theta_j$ we get $\rho = \tau_{j+1} \circ \{t/x\} \circ \theta_j = \tau_{j+1} \circ \theta_{j+1}$.

Case $x \notin \text{dom}(\tau_j)$. Then $\tau_j = \tau_{j+1}$ and each element of D_j is a variable and $\tau_j = \tau_{j+1} \circ \{t/x\}$. Thus $\rho = \tau_j \circ \theta_j = \tau_{j+1} \circ \{t/x\} \circ \theta_j = \tau_{j+1} \circ \theta_{j+1}$ as required.

Since for any unifier ρ of S there exists τ_k such that $\rho = \tau_k \circ \theta_k$, θ_k must be a mgu of S . \dashv

QED

Unification: consequences

Corollary 28.1

Let θ be the mgu of a unifiable set S computed by the unification algorithm. Then for some $k > 0$

1. $\theta = \theta_k = \{t_k/x_k\} \circ \dots \circ \{t_1/x_1\}$.
2. For each j with $0 < j \leq k$, let $\theta_j = \{t_j/x_j\} \circ \dots \circ \{t_1/x_1\}$. Then
 - (a) $\{x_j, \dots, x_1\} \cap FV(t_j) = \emptyset$
 - (b) $\{x_j, \dots, x_1\} \cap FV(\theta_j S) = \emptyset$
 - (c) If ρ is any unifier of S , then for each j , $0 < j \leq k$,
 - i. $\rho = \tau_j \circ \theta_j$ for some τ_j .
 - ii. for each i , $0 < i \leq j$, $(\theta_i \circ \theta_j)S = \theta_j S$

Exercise 28.2: Unification

1. Generalize the facts 28.1 to a finite set S of terms where $|S| \geq 2$.
2. Identify the relationships among the different substitutions θ , θ^{-1} , τ , χ , ρ and ρ^{-1} in examples 28.1 and 28.2
3. Let D be the disagreement set of S .
 - (a) Can $|D|$ be different from $|S|$? Justify your answer.
 - (b) If $S = D$ then under what conditions is S unifiable?
 - (c) If $S \neq D$ then what can you say about the depths of terms in D as compared to the depths of terms in S ?
4. Construct an example of a set S of terms with disagreement set D in which there exist a variable x and a term t such that $x \in FV(t)$ and yet the set S is unifiable.
5. Prove that if S is unifiable then the mgu computed by the unification algorithm 28.2 is idempotent. *Hint: Use corollary 28.1*

29. Resolution in FOL

29: Resolution in FOL

1. Standardizing Variables Apart
2. Factoring
3. Example: 1
4. Example: 2
5. Refutation
6. Refutations: Using the Herbrand Universe

Recapitulation

1. For any set $\Phi \cup \{\psi\}$ (where Φ may or may not be empty) of closed Σ -formulae $\Phi \models \psi$ iff $\Phi \cup \{\neg\psi\}$ is unsatisfiable.
2. A non-empty set Φ of closed Σ -formulae is unsatisfiable iff it contains a non-empty finite unsatisfiable subset.
3. A set Φ of closed Σ -formulae has a model iff it has a Herbrand model
4. A non-empty finite set Φ of closed Σ -formulae is unsatisfiable iff the formula $\psi \equiv \bigwedge_{\phi \in \Phi} \phi$ is unsatisfiable.

SCNFs and Models

1. A closed Σ -formula ψ has a model iff the universally closed Σ -formula $sko(\psi)$ has a model.
2. Every closed Σ -formula ψ may be transformed into an “*equi-satisfiable*” (universally closed) formula in SCNF.
3. A universally closed Σ -formula ψ in SNF has model iff $\mathfrak{g}(\psi)$ the ground-instances (see definition 27.4) of ψ has a model.

SCNFs and Unsatisfiability

1. A closed Σ -formula ψ and the (universally closed) Σ -formula $sko(\psi)$ are “*equi-unsatisfiable*”.
2. A universally closed Σ -formula ψ in SNF is unsatisfiable iff $g(\psi)$ is unsatisfiable.
3. Since $g(\psi)$ consists of only closed formulae, $g(\psi)$ is unsatisfiable iff there is a finite subset of $g(\psi)$ which is unsatisfiable.

Representing SCNFs

Definition 29.1: SCNF representation

Let the SCNF $sko(\psi)$ be represented by a set $sko(\psi) = \{C_i \mid 1 \leq i \leq m\}$ such that $sko(\psi) \equiv \vec{\forall}[\bigwedge_{1 \leq i \leq m} C_i]$ where each (quantifier-free) conjunct C_i , $1 \leq i \leq m$, $C_i \equiv \bigvee_{1 \leq j \leq n_i} \lambda_j$ is called a **clause** and is represented by a set $C_i = \{\lambda_j \mid 1 \leq j \leq n_i\}$ of literals.

Clauses: Terminology

Definition 29.2: Clauses: Terminology

1. A **clause** is a finite set of literals.
2. The **empty clause** is the empty set of literals ($\{\}$).
3. A **ground clause** is a clause with no occurrences of variables.
4. For any substitution θ , and clause $C = \{\lambda_j \mid 1 \leq j \leq n\}$, $\theta C = \{\theta \lambda_j \mid 1 \leq j \leq n\}$.

Compare with clauses in propositional logic

Clauses: Ground Instances

Definition 29.3: Clauses: ground instances

1. The set of **ground instances** of a clause C is the set

$$\mathfrak{g}(C) = \{\theta C \mid \theta \text{ is a ground substitution}\}$$

2. For any set $S = \{C_i \mid 1 \leq i \leq m\}$ of clauses, the set of **ground instances** of S is the set

$$\mathfrak{g}(S) = \bigcup_{C \in S} \mathfrak{g}(C)$$

Facts about Clauses

Lemma 29.1

Let $\{C_i \mid 1 \leq i \leq m\}$ be a set of clauses. Then

$$\vec{\forall}[\bigwedge_{1 \leq i \leq m} C_i] \Leftrightarrow \bigwedge_{1 \leq i \leq m} \vec{\forall}[C_i]$$

Proof: Follows from the semantics of \forall and \wedge or alternatively from corollary 26.1.

QED

Notice that even if there are free variables common between two clauses, this lemma holds, mainly because of the fact that there are no existential quantifiers. For example

$$\forall x[p(x) \wedge q(x)] \Leftrightarrow \forall x[p(x)] \wedge \forall y[q(y)] \Leftrightarrow \forall x, y[p(x) \wedge q(y)]$$

Clauses: Models

Definition 29.4: Models of clauses

1. A structure \mathbf{A} is a model of a

- clause $C = \{\lambda_j \mid 1 \leq j \leq n\}$ (denoted $\mathbf{A} \Vdash C$) iff $n > 0$ and $\mathbf{A} \Vdash \vec{\forall}[\bigvee_{1 \leq j \leq n} \lambda_j]$.

- a set S of clauses (denoted $\mathbf{A} \Vdash S$) if it is a model of every clause in S .

2. $S \models C$ iff every model of S is also a model of C .

Note: An empty clause has no models.

Clauses and Herbrand's Theorem

From **Herbrand's Theorem, compactness and finite unsatisfiability** we have

Proposition 29.1: Herbrand models of clauses

1. A set S of clauses possesses a model iff every finite subset of S possesses a model.
2. A set S of clauses is unsatisfiable iff there is a finite subset $S' \subseteq_f S$ of clauses which is unsatisfiable.
3. $S' \subseteq_f S$ is unsatisfiable iff $\mathfrak{g}(S')$ does not possess a model.



Resolution in FOL

Compare with resolution in propositional logic

Let S be a non-empty finite set of clauses, $C_i, C_j \in S$ with $i \neq j$, $FV(C_i) \cap FV(C_j) = \emptyset$ and p an atomic predicate symbol such that

- $L_i = \{p(\vec{s}_{i'}) \in C_i \mid 1 \leq i' \leq m_i\} \neq \emptyset$
- $L_j = \{\neg p(\vec{t}_{j'}) \in C_j \mid 1 \leq j' \leq m_j\} \neq \emptyset$
- $L = L_i \cup \overline{L_j}$ is a set of **unifiable** literals.
- $C'_i = C_i - L_i$ and $C'_j = C_j - L_j$
- $\mu = \text{UNIFY}(L)$ (algorithm 28.2) is an mgu of L
- $C'_{ij} = \mu(C'_i \cup C'_j) = (\mu C'_i) \cup (\mu C'_j)$ is called the *resolvent* of C_i and C_j .

The Resolution Rule for FOL

Compare with resolution in propositional logic

$$\text{Res1} \quad \frac{S}{(S - \{C_i, C_j\}) \cup \{C'_{ij}\}}$$

Note.

1. L_i and L_j are merely non-empty finite sets of literals such that $L = L_i \cup \overline{L_j}$ is unifiable. They need not exhaust all literals naming p in C_i or C_j .
2. Unlike **propositional resolution**, each application of the resolution rule may not eliminate any atomic predicate symbol.
3. Therefore the resolvent C'_{ij} could contain occurrences of the predicate symbol p (in both positive and negative forms) which may be used for **other** steps in resolution.

30. More on Resolution in FOL

30: More on Resolution in FOL

1. Soundness of FOL Resolution
2. Ground Clauses
3. The Lifting Lemma
4. Lifting Lemma: Figure
5. Completeness of Resolution Refutation: 1
6. Completeness of Resolution Refutation: 2
7. Completeness of Resolution Refutation: 3

Standardizing Variables Apart

1. The free variables of distinct clauses *must* be disjoint and variables should be renamed if necessary.

Example 30.1: Standardizing variables apart

Let $S = \{C_1, C_2\}$ where

$$\begin{aligned}C_1 &= \{p(\textcolor{violet}{x})\} \\C_2 &= \{\neg p(f(x))\}\end{aligned}$$

S represents the set of closed formulae

$$\{\forall \textcolor{violet}{x}[p(\textcolor{violet}{x})], \forall \textcolor{violet}{x}[\neg p(\textcolor{violet}{f}(x))]\}$$

which is clearly unsatisfiable. However the empty clause cannot be derived (because of the *occurs check*) unless $\textcolor{violet}{x}$ is renamed in one of the clauses.

Factoring

2. Unlike in the case of **propositional resolution** it should be possible to eliminate several literals **at once**

Example 30.2

Consider the set $S = \{C_1, C_2\}$ where

$$\begin{aligned}C_1 &= \{p(\textcolor{violet}{x}), p(\textcolor{violet}{y})\} \\C_2 &= \{\neg p(\textcolor{violet}{u}), \neg p(\textcolor{violet}{v})\}\end{aligned}$$

S is clearly unsatisfiable but by removing only one literal at a time with the substitution $\{\textcolor{violet}{x}/\textcolor{violet}{u}\}$ yields the new clause

$$C'_{12} = \{p(\textcolor{violet}{y}), \neg p(\textcolor{violet}{v})\}$$

from which the **empty clause cannot be** derived.

Example: 1

Example 30.3

Consider the set $S = \{C_1, C_2\}$ where

$$\begin{aligned}C_1 &= \{\neg q(\textcolor{violet}{x}, \textcolor{violet}{y}), \neg q(\textcolor{violet}{y}, \textcolor{violet}{z}), q(\textcolor{violet}{x}, \textcolor{violet}{z})\} \\&\equiv \forall \textcolor{violet}{x}, \textcolor{violet}{y}, \textcolor{violet}{z}[(q(\textcolor{violet}{x}, \textcolor{violet}{y}) \wedge q(\textcolor{violet}{y}, \textcolor{violet}{z})) \rightarrow q(\textcolor{violet}{x}, \textcolor{violet}{z})]\end{aligned}$$

which represents *transitivity* and

$$\begin{aligned}C_2 &= \{\neg q(\textcolor{violet}{u}, \textcolor{violet}{v}), q(\textcolor{violet}{v}, \textcolor{violet}{u})\} \\&\equiv \forall \textcolor{violet}{u}, \textcolor{violet}{v}[q(\textcolor{violet}{u}, \textcolor{violet}{v}) \rightarrow q(\textcolor{violet}{v}, \textcolor{violet}{u})]\end{aligned}$$

which represents *symmetry*.

A logical consequence of these two properties is the property derived below.

$$\frac{C_1 = \{\neg q(\textcolor{violet}{x}, \textcolor{violet}{y}), \underline{\neg q(\textcolor{red}{y}, \textcolor{violet}{z})}, q(\textcolor{violet}{x}, \textcolor{violet}{z})\}, \{\neg q(\textcolor{violet}{u}, \textcolor{violet}{v}), \underline{q(\textcolor{red}{v}, \textcolor{violet}{u})}\} = C_2}{\{\neg q(\textcolor{violet}{x}, \textcolor{violet}{y}), q(\textcolor{violet}{x}, \textcolor{violet}{z}), \neg q(\textcolor{violet}{z}, \textcolor{violet}{y})\} = C'_{12}} \mu = \{\textcolor{violet}{z}/\textcolor{violet}{u}, \textcolor{violet}{y}/\textcolor{violet}{v}\}$$

$$\begin{aligned} C'_{12} &= \{\neg q(\textcolor{violet}{x}, \textcolor{violet}{y}), \neg q(\textcolor{violet}{z}, \textcolor{violet}{y}), q(\textcolor{violet}{x}, \textcolor{violet}{z})\} \\ &\equiv \forall \textcolor{violet}{x}, \textcolor{violet}{y}, \textcolor{violet}{z}[(q(\textcolor{violet}{x}, \textcolor{violet}{y}) \wedge q(\textcolor{violet}{z}, \textcolor{violet}{y})) \rightarrow q(\textcolor{violet}{x}, \textcolor{violet}{z})] \end{aligned}$$

Example: 2

Example 30.4: Symmetry from reflexivity and euclideanness

Suppose we need to prove that if a binary relation p is *reflexive*

$$\phi_{p\text{-reflexive}} \stackrel{df}{=} \forall \mathbf{x}[p(\mathbf{x}, \mathbf{x})]$$

and *euclidean*

$$\phi_{p\text{-euclidean}} \stackrel{df}{=} \forall \mathbf{x}, \mathbf{y}, \mathbf{z}[(p(\mathbf{x}, \mathbf{y}) \wedge p(\mathbf{x}, \mathbf{z})) \rightarrow p(\mathbf{y}, \mathbf{z})]$$

then it is also *symmetric*

$$\phi_{p\text{-symmetric}} \stackrel{df}{=} \forall \mathbf{x}, \mathbf{y}[p(\mathbf{x}, \mathbf{y}) \rightarrow p(\mathbf{y}, \mathbf{x})]$$

After renaming bound variables and converting into SCNF to get the clauses

$$\begin{aligned}C_1 &= \{\underline{p(x, x)}\} \\C_2 &= \{\neg p(u, v), \neg p(u, w), p(v, w)\}\end{aligned}$$

The mgu $\mu = \{x/u, x/w\}$ yields the clause

$$C'_{12} = \{\neg p(x, v), p(v, x)\}$$

which represents the sentence $\forall x, v[p(x, v) \rightarrow p(v, x)]$ representing the symmetry of p .

Refutation

Refutation in propositional logic

Example 30.5: A refutation

We could also prove symmetry in example 30.4 by a *refutation* as follows. Taking the negation of the conclusion we get

$$\begin{aligned} & \neg\phi_{p-\text{symmetry}} \\ \Leftrightarrow & \exists u, v [p(u, v) \wedge \neg p(v, u)] \\ (\text{Sko}) \quad & p(a, b) \wedge \neg p(b, a) \\ \equiv & \{\{p(a, b)\}, \{\neg p(b, a)\}\} \\ \stackrel{df}{=} & \{C_3, C_4\} \end{aligned}$$

where a and b are skolem constants.

$$\frac{\{C_1 = \{p(\textcolor{violet}{x}, \textcolor{violet}{x})\}, C_2 = \{\neg p(\textcolor{violet}{u}, \textcolor{violet}{v}), \neg p(\textcolor{violet}{u}, \textcolor{violet}{w}), p(\textcolor{violet}{v}, \textcolor{violet}{w})\}, C_3 = \{p(\textcolor{red}{a}, \textcolor{red}{b})\}, C_4 = \{\neg p(\textcolor{red}{b}, \textcolor{red}{a})\}\}}{C'_{12} = \{\neg p(\textcolor{violet}{x}, \textcolor{violet}{v}), p(\textcolor{violet}{v}, \textcolor{violet}{x})\}, C_3 = \{p(\textcolor{red}{a}, \textcolor{red}{b})\}, C_4 = \{\neg p(\textcolor{red}{b}, \textcolor{red}{a})\}\}} \mu = \{\textcolor{violet}{x}/\textcolor{violet}{u}, \textcolor{violet}{x}/\textcolor{violet}{w}\}$$

$$\frac{C'_{124} = \{\neg p(\textcolor{red}{a}, \textcolor{red}{b}), \}\}, C_3 = \{p(\textcolor{red}{a}, \textcolor{red}{b})\}, \}}{\{\{\}\}} \mu' = \{\textcolor{red}{a}/\textcolor{violet}{x}, \textcolor{red}{b}/\textcolor{violet}{v}\}$$

$$\frac{\{\{\}\}}{\{\{\}\}} \mu'' = \mathbf{1}$$

Alternatively a proof starting with the clauses C_3 and C_4 works too.

$$\frac{\{C_1 = \{p(\textcolor{violet}{x}, \textcolor{violet}{x})\}, C_2 = \{\neg p(\textcolor{violet}{u}, \textcolor{violet}{v}), \neg p(\textcolor{violet}{u}, \textcolor{violet}{w}), p(\textcolor{violet}{v}, \textcolor{violet}{w})\}, C_3 = \{p(\textcolor{red}{a}, \textcolor{red}{b})\}, C_4 = \{\neg p(\textcolor{red}{b}, \textcolor{red}{a})\}\}}{\{C_1 = \{p(\textcolor{violet}{x}, \textcolor{violet}{x})\}, C_3 = \{p(\textcolor{red}{a}, \textcolor{red}{b})\}, C'_{24} = \{\neg p(\textcolor{violet}{u}, \textcolor{red}{b}), \neg p(\textcolor{violet}{u}, \textcolor{red}{a})\}\}} \theta_1 = \{\textcolor{red}{b}/\textcolor{violet}{v}, \textcolor{red}{a}/\textcolor{violet}{w}\}$$

$$\frac{\{C_1 = \{p(\textcolor{violet}{x}, \textcolor{violet}{x})\}, C'_{234} = \{\neg p(\textcolor{red}{a}, \textcolor{red}{a})\}\}}{\{\{\}\}} \theta_2 = \{\textcolor{red}{a}/\textcolor{violet}{u}\}$$

$$\frac{\{\{\}\}}{\{\{\}\}} \theta_3 = \{\textcolor{red}{a}/\textcolor{violet}{x}\}$$

Refutations: Using the Herbrand Universe

Example 30.6

Let a be a constant symbol and f a unary function symbol. We use first-order resolution to prove that

$$\Phi = \{\neg p(a), p(f(f(a))), \forall x[p(\textcolor{violet}{x}) \vee \neg p(\textcolor{violet}{f}(x))]\}$$

is unsatisfiable.

The Herbrand universe consists of at least the following terms

$$\{\textcolor{red}{a}, \textcolor{violet}{f}(a), f(f(a)), \dots, \textcolor{violet}{f}^n(a), \dots\}$$

where $\textcolor{violet}{f}^n(a)$ refers to the n -fold application of f to a .

The set of ground clauses of Φ therefore is a some superset of

$$\{\{\neg p(a)\}, \{p(f^2(a))\}\} \cup \{\{p(f^i(a)), \neg p(f^{i+1}(a))\} \mid i \geq 0\}$$

We could take the clauses in the given order as follows:

$$\begin{array}{l} \{\neg p(a)\}, \{p(f^2(a))\}, \{p(a), \neg p(f(a))\}, \{p(f(a)), \neg p(f^2(a))\}, \\ \{p(f^2(a))\}, \neg p(f^3(a)), \dots \end{array}$$

Resolving on $p(a)$

$$\{p(f^2(a))\}, \{\neg p(f(a))\}, \{p(f(a)), \neg p(f^2(a))\}, \dots$$

Resolving on $p(f(a))$

$$\{p(f^2(a))\}, \{\neg p(f^2(a))\}, \{p(f^2(a)), \neg p(f^3(a))\}, \dots$$

Resolving on $p(f^2(a))$

$$\{\}, \{p(f^2(a)), \neg p(f^3(a))\}, \dots$$

by which the empty clause has been derived through propositional resolution. To use first-order resolution one would proceed as follows

$$\begin{array}{l} \{\neg p(a)\}, \{p(f^2(a))\}, \{p(x), \neg p(f(x))\} \\ \text{substituting } \{a/x\} \quad \{\neg p(a)\}, \{p(f^2(a))\}, \{p(a), \neg p(f(a))\}, \{p(x_1), \neg p(f(x_1))\} \end{array}$$

Resolving on $p(a)$

$$\{p(f^2(a))\}, \{\neg p(f(a))\}, \{p(x_1), \neg p(f(x_1))\}, \dots$$

substituting $\{f(a)/x_1\}$

$$\{p(f^2(a))\}, \{\neg p(f(a))\}, \{\neg p(f^2(a))\}, \{p(x_2), \neg p(x_2)\}, \dots$$

Resolving on $p(f^2(a))$

$$\{\}, \{p(x_2)\}, \neg p(x_2)\}$$

which has derived the empty clause.

Exercise 30.1: First-order resolution and refutation

1. Prove the conclusion of example 30.3 by a refutation.
2. Are there any other unifiers by which symmetry may be proved in example 30.4?
3. Try a refutation proof using $\nu = \{x/u, x/w, x/v\}$ as the first unifier in example 30.5. Why doesn't it work?
4. Try a refutation proof using $\nu = \{x/u, x/w, y/v\}$ as the first unifier in example 30.5. Why does it work?

31. Resolution: Soundness and Completeness

31: Resolution: Soundness and Completeness

1. Soundness of FOL Resolution
2. Ground Clauses
3. The Lifting Lemma
4. Lifting Lemma: Figure
5. Completeness of Resolution Refutation: 1
6. Completeness of Resolution Refutation: 2
7. Completeness of Resolution Refutation: 3

Lemma 31.1

The resolvent C'_{ij} obtained by resolving the clauses C_i and C_j in the **resolution method** is a logical consequence of the set $\{C_i, C_j\}$.



The following theorem then follows.

Theorem 31.1: Resolution and logical consequence

If S' is the set of clauses obtained by a single application of the resolution rule **Res1**, then $S \models S'$.



Corollary 31.1: Refutation as unsatisfiability

If the empty clause is derivable from a set S of clauses, then S is unsatisfiable.

Proof of lemma 31.1

Proof: Assume

- $C_i = C'_i \cup \{p(\vec{s}_{i'}) \mid 1 \leq i' \leq m_i\}$,
- $C_j = C'_j \cup \{\neg p(\vec{t}_{j'}) \mid 1 \leq j' \leq m_j\}$,
- $FV(C_i) \cap FV(C_j) = \emptyset$ and
- $L = \{p(\vec{s}_{i'}) \mid 1 \leq i' \leq m_i\} \cup \{p(\vec{t}_{j'}) \mid 1 \leq j' \leq m_j\}$ is a set of unifiable literals.

Let $\mathbf{A} \Vdash \{C_i, C_j\}$. Therefore

$$\mathbf{A} \Vdash \vec{\forall}[\bigvee_{\lambda_i \in C_i} \lambda_i] \quad (64)$$

$$\mathbf{A} \Vdash \vec{\forall}[\bigvee_{\lambda_i \in C_j} \lambda_j] \quad (65)$$

and for any substitution θ we have

$$\mathbf{A} \Vdash \theta \bigvee C_i \quad (66)$$

$$\mathbf{A} \Vdash \theta \bigvee C_j \quad (67)$$

If θ is a unifier of L and $\theta L = \{\lambda\}$ we get

$$\mathbf{A} \Vdash \bigvee (\{\lambda\} \cup \theta C'_i) \quad (68)$$

$$\mathbf{A} \Vdash \bigvee (\{\bar{\lambda}\} \cup \theta C'_j) \quad (69)$$

Let $\theta C'_i = \{\kappa_{i'} \mid 1 \leq i' \leq k\}$ and $\theta C'_j = \{\lambda_{j'} \mid 1 \leq j' \leq l\}$. Then we have the following table which shows a case analysis for the various values of k and l .

	$\{\lambda\} \cup \theta C'_i \Leftrightarrow$	$\{\bar{\lambda}\} \cup \theta C'_j \Leftrightarrow$	$\theta C'_i \cup \theta C'_j \Leftrightarrow$
$k = 0 = l$	λ	$\bar{\lambda}$	$\{\}$
$k = 0, l > 0$	λ	$\lambda \rightarrow (\lambda_1 \vee \dots \vee \lambda_l)$	$\lambda_1 \vee \dots \vee \lambda_l$
$k > 0, l = 0$	$\bar{\lambda} \rightarrow (\kappa_1 \vee \dots \vee \kappa_k)$	$\bar{\lambda}$	$\kappa_1 \vee \dots \vee \kappa_k$
$k, l > 0$	$\neg(\kappa_1 \vee \dots \vee \kappa_k) \rightarrow \lambda$	$\lambda \rightarrow (\lambda_1 \vee \dots \vee \lambda_l)$	$\neg(\kappa_1 \vee \dots \vee \kappa_k) \rightarrow (\lambda_1 \vee \dots \vee \lambda_l)$

It is easy to see that in each case

$$\{\{\lambda\} \cup \theta C'_i, \{\bar{\lambda}\} \cup \theta C'_j\} \models \theta C'_i \cup \theta C'_j \quad (70)$$

It follows also from (64), (65) and (70) that $\mathbf{A} \Vdash \vec{\forall}[\bigvee C'_{ij}]$ and hence C'_{ij} is a logical consequence of the set $\{C_i, C_j\}$.

QED

Ground Clauses

Theorem 31.2: Completeness of resolution for ground clauses

Let G be a set of ground clauses. If G does not possess a model, the empty clause ($\{\}$) may be derived by **Res0**.



Here **Res0** is the **propositional resolution rule** given by

$$\text{Res0} \quad \frac{S}{(S - \{C_i, C_j\}) \cup \{C'_{ij}\}}$$

where $C'_{ij} = C'_i \cup C'_j$ and for some literal λ , $C_i = C'_i \cup \{\lambda\}$ and $C_j = C'_j \cup \{\bar{\lambda}\}$. Note that there is no substitution involved anywhere since all clauses are ground.

Proof of theorem 31.2.

Proof: Let $G = \{C_i \mid 1 \leq i \leq n, n > 0\}$ be a set of n clauses. If $\{\} \in G$ there is nothing to prove. So assume $\{\} \notin G$. Consider the following measure

$$\#G = (\sum_{1 \leq i \leq n} |C_i|) - n$$

Clearly, $\#G = 0$ iff every clause is made up of a single literal. We proceed to prove the theorem by induction on $\#G$.

Basis. $\#G = 0$. Then each $C_i = \{\lambda_i\}$ and $G \equiv \bigwedge_{1 \leq i \leq k} \lambda_i$ and by theorem 27.1, G is unsatisfiable iff it contains a complementary pair. Clearly by rule

Res0 the resolvent of this complementary pair is the empty clause.

Induction Hypothesis (IH).

For some $m > 0$, for all k , $0 \leq \#G = k < m$, if G does not possess a model, then the empty clause is derivable from G .

Induction Step. Assume $\#G = m > 0$. There must be at least one clause C_i which contains more than one literal. So let $C_i = \{\lambda_i\} \cup D_i$ with $\lambda_i \notin D_i \neq \emptyset$. Let $G_{i_1} = (G - \{C_i\}) \cup \{D_i\}$ and $G_{i_2} = (G - \{C_i\}) \cup \{\lambda_i\}$. Clearly $\#G_{i_1} < \#G$ and $\#G_{i_2} < \#G$. Further if G does not have a model then neither G_{i_1} nor G_{i_2} has a model (if either of them had a model then so would G since $C_i \equiv \lambda_i \vee \bigvee D_i$). By the induction hypothesis,

1. there exists a resolution proof \mathcal{R}_1 from G_{i_1} which derives the empty clause and
2. there is another resolution proof \mathcal{R}_2 from G_{i_2} which also derives the empty clause.

Notice that since we are dealing only with ground literals, all resolvents are obtained by applying the rule **Res0**.

Consider the proof \mathcal{R}'_1 obtained from \mathcal{R}_1 by adding the literal λ_i to D_i resulting in the set G and performing exactly the same sequence of resolutions as in \mathcal{R}_1 .

- Case 1. If the proof \mathcal{R}_1 did not involve the use of any of the literals from D_i and the empty clause was derived, then clearly the same sequence with λ_i included would also derive the empty clause and that completes the proof.
- Case 2 On the other hand if one or more steps in proof \mathcal{R}_1 involved literals from D_i then the resulting proof \mathcal{R}'_1 may derive the clause $\{\lambda_i\}$ in place of the empty clause. However we do know that the empty clause is derived from G_{i_2} in proof \mathcal{R}_2 . This implies there exist resolution steps in \mathcal{R}_2 involving the literal λ_i which derive the empty clause. Therefore there exists at least one clause containing the literal $\bar{\lambda}_i$ in the set of final clauses obtained in \mathcal{R}'_1 . By applying the resolution steps of \mathcal{R}_2 which do not appear anywhere in \mathcal{R}'_1 , the empty clause would again be derived.

QED

The Lifting Lemma

Lemma 31.2: Lifting Lemma

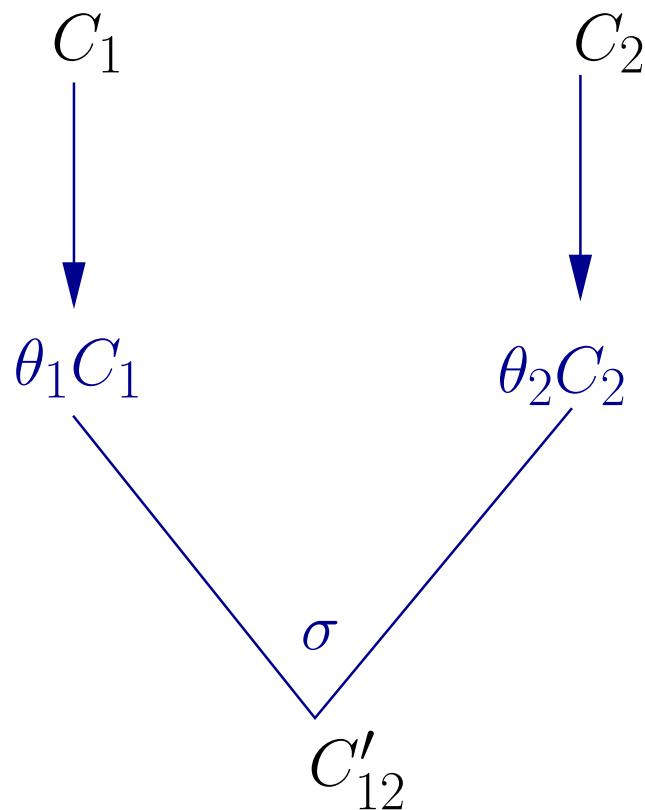
(see **figure**) Let C_1 and C_2 be clauses and let $\theta_1, \theta_2, \sigma$ be substitutions such that

- $FV(C_1) \cap FV(C_2) = \emptyset$,
- $FV(\theta_1 C_1) \cap FV(\theta_2 C_2) = \emptyset$ and
- C'_{12} is the resolvent of $\theta_1 C_1$ and $\theta_2 C_2$ via a substitution σ , by a single application of resolution.

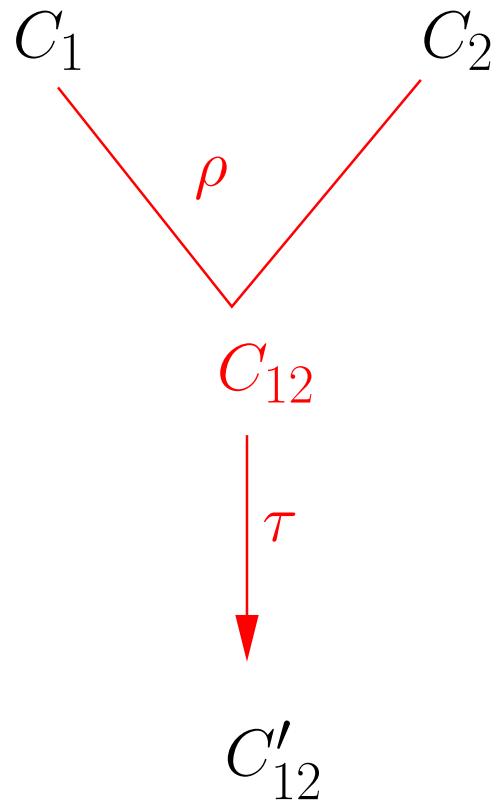
Then there exists a resolvent C_{12} of C_1 and C_2 by a single application of resolution via a substitution ρ and a substitution τ such that $C'_{12} \equiv \tau C_{12}$.



Lifting Lemma: Figure



lifted to



Proof of lemma 31.2

Proof: Let

$$C_1 = C'_1 \cup L_1 \text{ where } L_1 = \{\lambda_i \mid 1 \leq i \leq m, m > 0\}$$

$$C_2 = C'_2 \cup L_2 \text{ where } L_2 = \{\overline{\lambda_j} \mid 1 \leq j \leq n, n > 0\}$$

such that σ is a mgu of $(\theta_1 L_1) \cup (\theta_2 \overline{L_2})$ and $C'_{12} = \sigma((\theta_1 C'_1) \cup (\theta_2 C'_2))$.

Since $FV(C_1) \cap FV(C_2) = \emptyset$, $dom(\theta_1) \cap dom(\theta_2) = \emptyset$ and since $FV(\theta_1 C_1) \cap FV(\theta_2 C_2) = \emptyset$ we have $FV(ran(\theta_1)) \cap FV(ran(\theta_2)) = \emptyset$ and hence $\theta_1 C'_1 = (\theta_1 \cup \theta_2) C'_1$ and $\theta_2 C'_2 = (\theta_1 \cup \theta_2) C'_2$.

Since σ is a mgu of $(\theta_1 L_1) \cup (\theta_2 \overline{L_2})$ we have $\sigma \circ (\theta_1 \cup \theta_2)$ is a unifier of $L_1 \cup \overline{L_2}$. If $L_1 \cup \overline{L_2}$ is unifiable, then it has a most general unifier $\rho \gtrsim \sigma \circ (\theta_1 \cup \theta_2)$ such that $C'_{12} = \rho(C'_1 \cup C'_2)$ is the resolvent of C_1 and C_2 .

$\rho \gtrsim \sigma \circ (\theta_1 \cup \theta_2)$ implies there exists a substitution τ such that

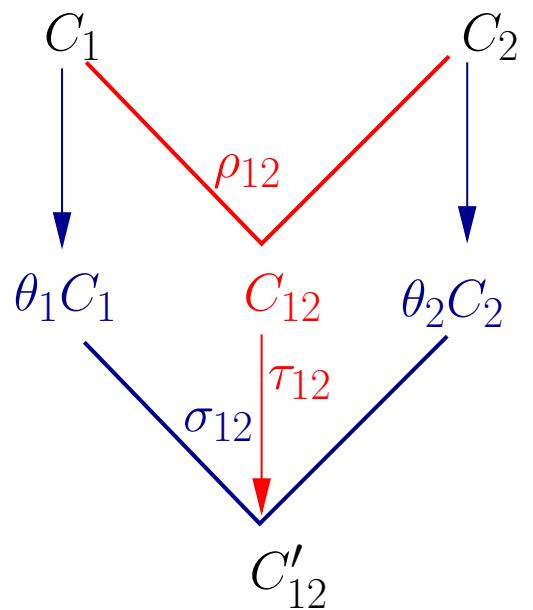
$$\tau \circ \rho = \sigma \circ (\theta_1 \cup \theta_2)$$

and

$$C'_{12} = \tau(\rho(C'_1 \cup C'_2)) = (\sigma \circ (\theta_1 \cup \theta_2))(C'_1 \cup C'_2)$$

QED

A superposed version of the figure is shown below.



Completeness of Resolution Refutation: 1

1. The lifting lemma helps us to use the completeness of resolution refutation for ground clauses and “lift” it to clauses with variables.
2. By standardizing variables apart we may guarantee that the conditions of disjointness of free variables between different clauses (lemma 31.2) may be enforced.
3. Any set of clauses $S = \{C_i \mid 1 \leq i \leq m\}$ represents the conjunction of the universal closure of each clause.

Completeness of Resolution Refutation: 2

1. The **lifting lemma 31.2** guarantees that if the substitutions θ_1 and θ_2 are *ground*, then there exists a corresponding *ground substitution* τ which produces the same effect after resolution.
2. By **Herbrand's theorem** a set Φ is unsatisfiable iff a finite subset of ground instances of Φ is unsatisfiable.
3. To prove the completeness of resolution refutation it is sufficient to consider only the *finite set of ground clauses* from which the empty clause $\{\}$ may be derived.

Completeness of Resolution Refutation: 3

Theorem 31.3: Completeness of Resolution Refutation

If a set Φ of clauses is unsatisfiable then the empty clause is derivable from Φ .

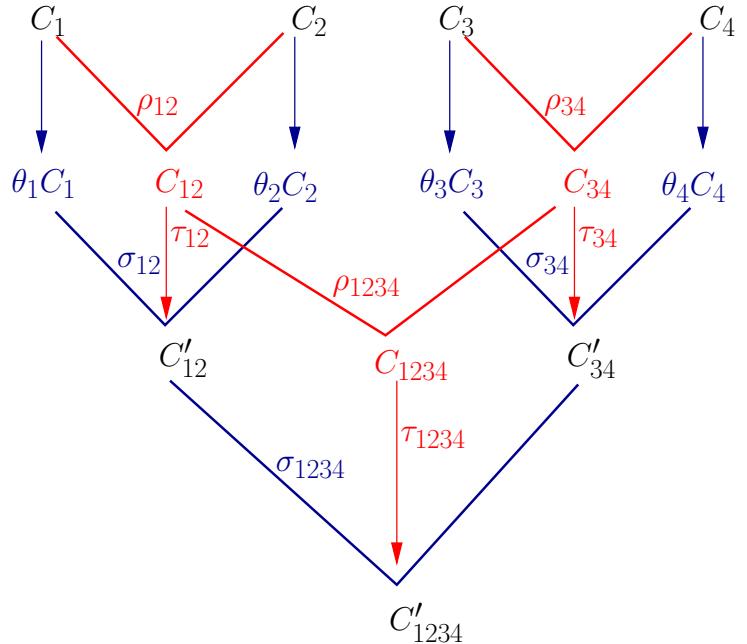


Proof of theorem 31.3

Proof: Without loss of generality we may assume that the variables in every clause are disjoint from the variables occurring in any other clause.

By Herbrand's theorem there exists a finite set of ground clauses $G = \{gC_i \mid 1 \leq i \leq m\} \subseteq_f g(\Phi)$ such that G is unsatisfiable. Each $gC_i \in G$ is obtained by a substitution on some clause i.e. $gC_i = \theta_i C_i$ for some substitution θ_i and some clause $C_i \in \Phi$. Further for $i \neq j$, $dom(\theta_i) \cap dom(\theta_j) = \emptyset$ and since all the clauses in G are ground the disjointness conditions of the lifting lemma are trivially satisfied.

Each application of rule Res0 in G may be lifted to finding a mgu of the appropriate clauses. This fact may be proved by induction on the height of the resolution proof tree and we leave it as an exercise to the interested reader. However the following diagram illustrates it for two steps of a resolution proof tree.



QED

32. Resolution and Tableaux

32: Resolution and Tableaux

1. FOL: Tableaux
2. FOL: Tableaux Rules
3. FOL Tableaux: Example 1
4. FOL Tableaux: Example 1 Contd
5. First-Order Tableaux
6. First-Order Tableaux: Heuristics
7. FOL Tableaux: Example 2
8. FOL Tableaux: Example 2 Contd

FOL: Tableaux

1. The tableau method in principle is similar to
 - **natural deduction** in its use of “syntactical decomposition”,
 - **resolution** in using **unsatisfiability** to prove validity.
2. The tableau method for both propositional and predicate logic has some **advantages over resolution**.
3. FOL resolution requires formulae to be converted into **PCNF** and then **SCNF** before resolution may be applied.
4. As in the case of **Natural Deduction** the tableau method uses the rules $\exists E$ and $\forall E$ to decompose quantified formulae along with the same restrictions.

FOL: Tableaux Rules

Besides the usual **tableau rules** for the propositional connectives we have the following.

$\forall.$	$\frac{\forall x[\phi]}{\{t/x\}\phi}$	$\neg\forall.$	$\frac{\neg\forall x[\phi]}{\neg\{a/x\}\phi}$
$\exists.$	$\frac{\exists x[\phi]}{\{a/x\}\phi}$	$\neg\exists.$	$\frac{\neg\exists x[\phi]}{\neg\{t/x\}\phi}$

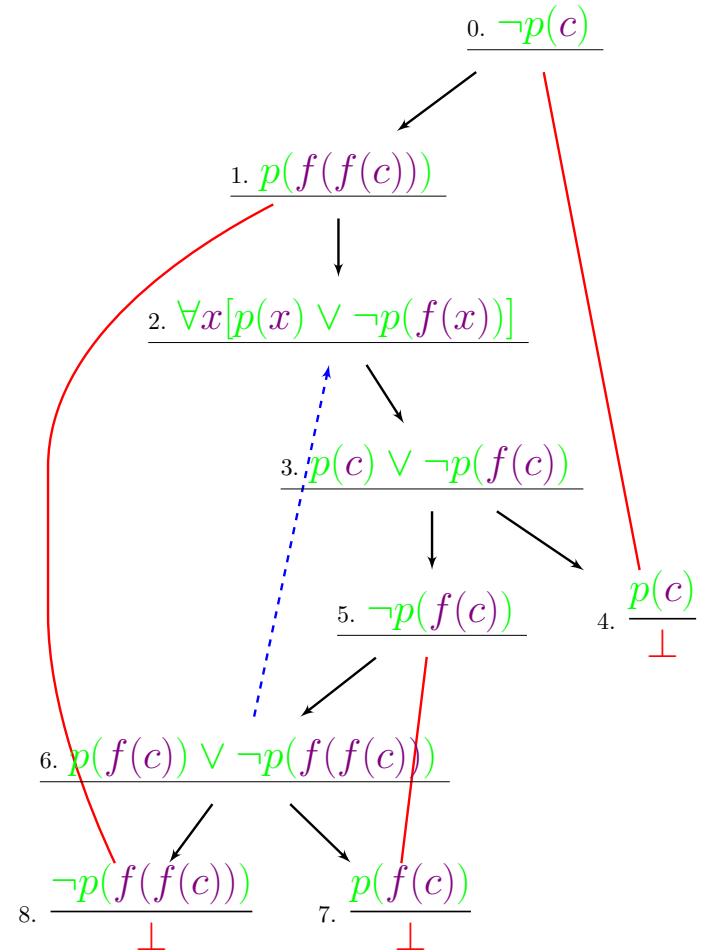
1. The restrictions on the use of a constant symbol a in both rules $\neg\forall.$ and $\exists.$ are the same as those for $\exists E$ in both \mathcal{H}_1 and \mathcal{G}_1 .
2. Since $\forall E$ holds for all terms t , the rules $\forall.$ and $\neg\exists.$ may have to be applied several times before unsatisfiability can be proven.

FOL Tableaux: Example 1

Example 32.1

Let c be a constant symbol and f a unary function symbol. Then $\Phi = \{\neg p(c), p(f(f(c))), \forall x[p(x) \vee \neg p(f(x))]\}$ is unsatisfiable.

Notice the **two applications** of rule $\forall E$. “ \perp ” indicates a closed path in the tableau.



First-Order Tableaux

1. Unlike **propositional tableaux**, any satisfiable finite set Φ of quantified formulae can potentially yield an *infinite* tableau, since a formula of the form $\forall x[\phi] \in \Phi$ can have an infinite number of instances.
2. Unlike the propositional case, especially when a quantifier elimination rule is applied, quantified formulae cannot be **discarded** since other instances of the formula may be required to close a path of the tableau.

First-Order Tableaux: Heuristics

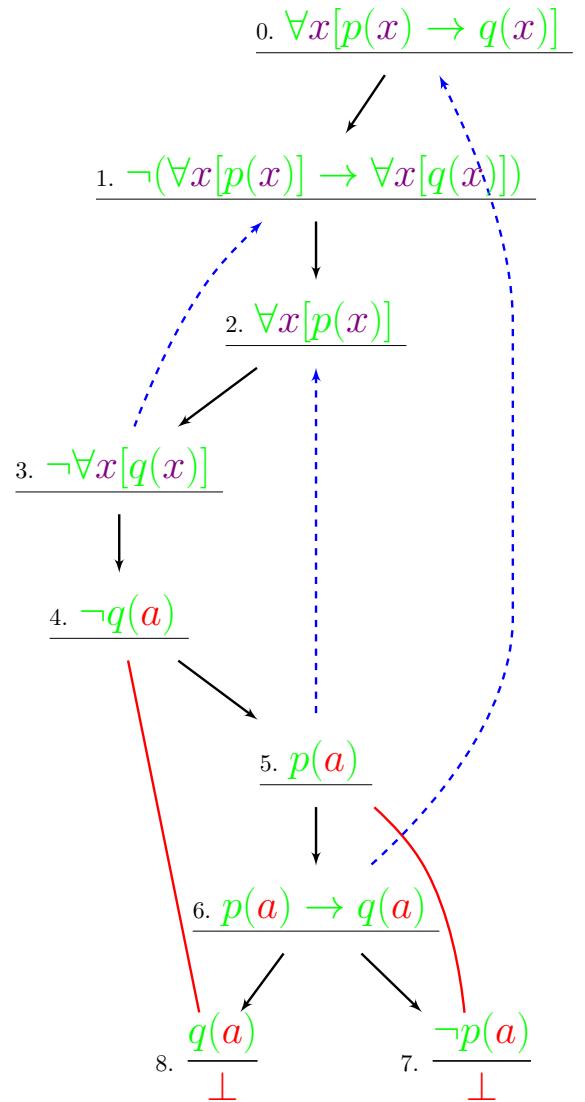
For unsatisfiable sets, closed finite tableaux may be constructed by applying the following heuristics

- Whenever possible apply propositional rules before applying quantifier rules
- Apply rules $\exists.$ and $\neg\forall.$ before applying $\forall.$ and $\neg\exists.$ in order to direct the proof towards a propositional contradiction.
- As in the case of **first-order resolution**, **unification** may be required (and mgus may have to be calculated) in order to guide the construction towards closing paths containing quantified formulae.

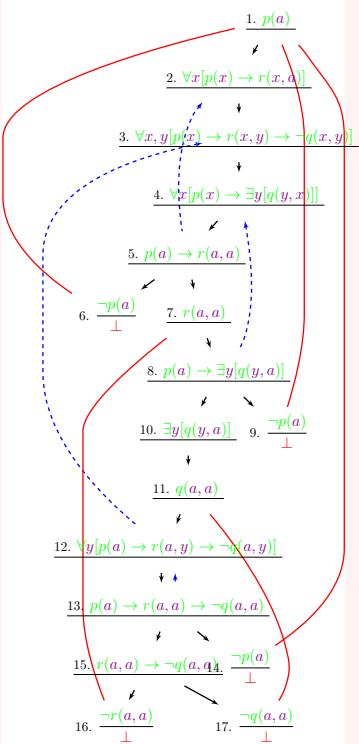
FOL Tableaux: Example 2

Example 32.2

We prove that $\forall x[p(x) \rightarrow q(x)] \models \forall x[p(x)] \rightarrow \forall x[q(x)]$ by showing the unsatisfiability of the set $\{\forall x[p(x) \rightarrow q(x)], \neg(\forall x[p(x)] \rightarrow \forall x[q(x)])\}$.



Exercise 32.1



1. The above is an attempt at proving that the set of formulae $\Phi = \{p(a), \forall x[p(x) \rightarrow r(x, a)], \forall x, y[p(x) \rightarrow r(x, y) \rightarrow \neg q(x, y)], \forall x[p(x) \rightarrow \exists y[q(y, x)]\}$ is unsatisfiable, using the tableau method for first-order formulae. a is a predefined constant in the signature.
 - (a) Find the flaw in the tableau.
 - (b) Is the set Φ really unsatisfiable? If so, construct a corrected tableau.

32.1. First-order Analytic Tableau with unification

In the case of propositional logic all tableaux for a finite set of propositional sentences are finite since every formula has only a finite degree and may be broken up according to the degree. However in the case of first-order tableaux, the rules for \forall . and $\neg\exists$. are highly non-deterministic and unless guided properly can lead to infinite tableaux that never close.

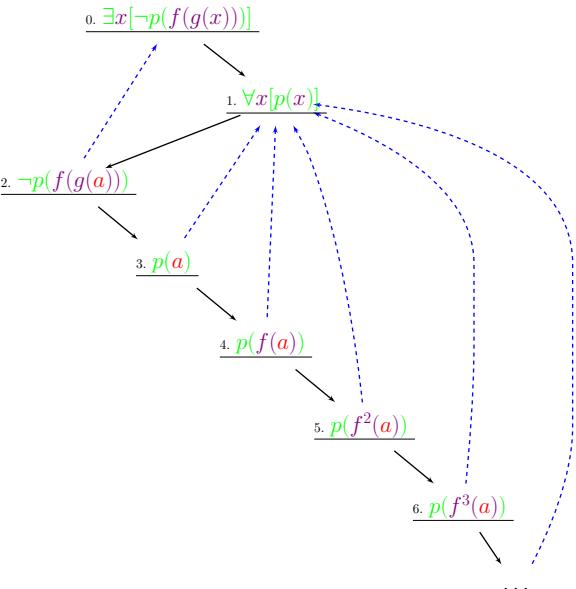
Example 32.3

Let $\Sigma = \{f, g : s \rightarrow s; p : s\}$. Now consider the set of formulae $\Phi = \{\exists x[\neg p(f(g(x)))] \wedge \forall x[p(x)]\}$. It is obvious that Φ is unsatisfiable. However we show that

1. there exist infinite tableaux which never close,
2. We need to construct a closed tableau, that we could do by guiding the tableau construction using unification and a trivial form of Skolemization.

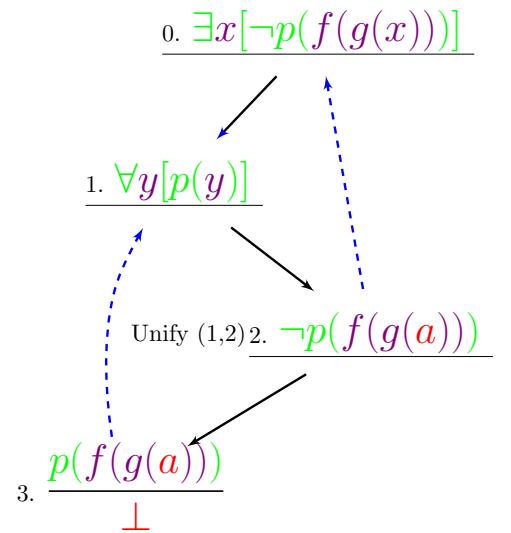
1. An infinite open tableau for Φ of example 32.3. There are an infinite number of such infinite tableau possible as the readers may verify for themselves. The problem is of finding a finite unsatisfiable subset which requires guidance.

[Zoom in](#)



2. A closed tableau for Φ of example 32.3 using unification to guide the choice of universal instantiation. To ensure that unification does not fail due to an occur check we perform appropriate α -conversions.

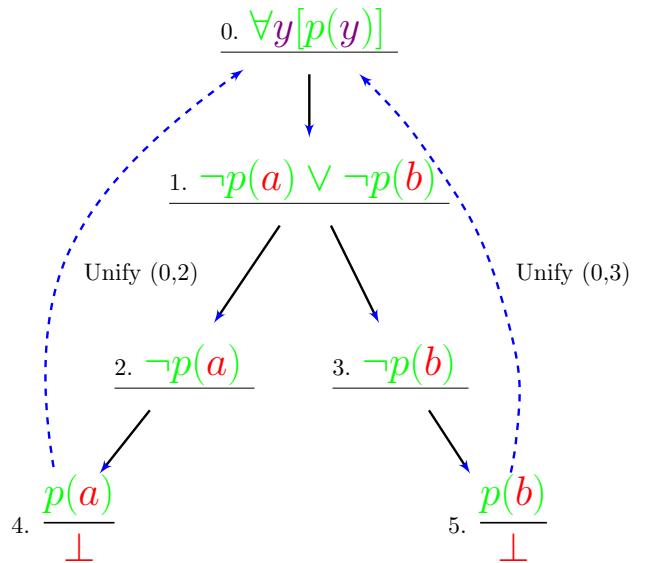
[Zoom in](#)



Example 32.4

Now consider the unsatisfiable set $\Phi_2 = \{\forall \textcolor{violet}{x}[p(\textcolor{violet}{x})], \neg p(\textcolor{red}{a}) \vee \neg p(\textcolor{red}{b})\}$, where $\textcolor{red}{a}$ and $\textcolor{red}{b}$ are constant symbols belonging to the signature. The following is a closed tableau for this set.

[Zoom in](#)



33. Completeness of First-order Tableaux

33: Completeness of First-order Tableaux

1. First-order Hintikka Sets
2. Hintikka's Lemma for FOL
3. First-order tableaux and Hintikka sets
4. Soundness of First-order Tableaux
5. Completeness of First-order Tableaux

First-order Hintikka Sets

Definition 33.1: First-order Hintikka sets

A finite or infinite set Γ is a **first-order Hintikka set** with respect to $\mathcal{P}_1(\Sigma)$ if

- 0-3. Γ is a (propositional) Hintikka set (definition 10.4) such that all the atomic propositions are ground.
- 4. For *every* $t \in \mathbb{T}_\Sigma$ (ground term).
 - $\forall x[\phi] \in \Gamma$ implies $\{t/x\}\phi \in \Gamma$,
 - $\neg\exists x[\phi] \in \Gamma$ implies $\neg\{t/x\}\phi \in \Gamma$
- 5. For *at least one* $t \in \mathbb{T}_\Sigma$ (ground term).
 - $\exists x[\phi] \in \Gamma$ implies $\{t/x\}\phi \in \Gamma$,
 - $\neg\forall x[\phi] \in \Gamma$ implies $\neg\{t/x\}\phi \in \Gamma$

Hintikka's Lemma for FOL

Lemma 33.1: Hintikka's lemma for FOL

If Σ contains at least one constant symbol, then every first-order Hintikka set with respect to $\mathcal{P}_1(\Sigma)$ is satisfiable in a Herbrand model.

Proof: We define a Herbrand interpretation of the formulae as follows. For each n -ary atomic predicate symbol p , $p(t_1, \dots, t_n)$ for ground terms t_1, \dots, t_n is true if and only if $p(t_1, \dots, t_n) \in \Gamma$. By the definition of a Hintikka set we know $\{p(t_1, \dots, t_n), \neg p(t_1, \dots, t_n)\} \not\subseteq \Gamma$. Hence all the atomic sentences in Γ are satisfiable under any valuation v_H . We may then proceed to show by structural induction on each $\phi \in \mathcal{P}_1(\Sigma)$ that $\phi \in \Gamma$ implies $H \Vdash \phi$. QED

First-order tableaux and Hintikka sets

Lemma 33.2: First-order tableaux and Hintikka sets

If a tableau rooted at a closed formula ϕ has an open path then the set of formulae on the path form a first-order Hintikka set.

Proof: We may prove that each rule in [Tableaux Rules](#) and [FOL: Tableaux Rules](#) creates a path for the construction of Hintikka sets. QED

Soundness of First-order Tableaux

Theorem 33.1: Soundness of First-order Tableau Rules

If there is a *closed tableau* rooted at a closed formula $\neg\phi$ then $\models \phi$.

Proof: Suppose there is a closed tableau rooted at $\neg\phi$. Then every path in the tableau is closed because of the occurrence of a complementary pair in the path. On the other hand if $\not\models \phi$, i.e. ϕ is not valid, then $\neg\phi$ is satisfiable, which implies that there is an open path in the tableau rooted at $\neg\phi$, clearly a contradiction. QED

Completeness of First-order Tableaux

Theorem 33.2: Completeness of First-order Tableaux

If a closed formula ϕ is valid, then there exists a closed tableau rooted at $\neg\phi$.

Proof: If there is no closed tableau rooted at $\neg\phi$ then there exists at least one *open* path in each such tableau. The set of formulae on this path form a Hintikka set and hence they are all simultaneously satisfiable, which implies there is a **Herbrand model** satisfying $\neg\phi$, in which case $\not\models \phi$. QED

34. Completeness of the Hilbert System

34: Completeness of the Hilbert System

1. Deductive Consistency
2. Models of Deductively Consistent Sets
3. Deductive Completeness
4. The Completeness Theorem

Deductive Consistency

Definition 34.1: Deductive consistency

A set $\Phi \subseteq \mathcal{L}_1(\Sigma)$ is deductively consistent iff there does not exist a formula ϕ such that $\Phi \vdash_{\mathcal{H}_1} \phi$ and $\Phi \vdash_{\mathcal{H}_1} \neg\phi$.

This definition is equivalent to other possible definitions such as those given below which may all be derived from rule \perp .

Lemma 34.1

The following statements are equivalent.

1. $\Phi \subseteq \mathcal{L}_1(\Sigma)$ is deductively consistent.
2. There does not exist a formula ψ such that $\Phi \vdash_{\mathcal{H}_1} \neg(\psi \rightarrow \psi)$
3. There exists a formula which is not provable.

34.1. Model-theoretic and Proof-theoretic Consistency

We have earlier defined the notion of consistency of a set of formulae in [propositional logic](#) (see also lemma 12.1) leading to the notions of [maximal consistency](#) and [Lindenbaum's theorem](#) which enabled us to extend a consistent set of propositions to a maximally consistent one. Later we have also defined the notion of [consistency of sets of predicate logic formulae](#) in definition 21.2. Notice that definition 10.1 though worded differently, also states that a set of propositions is consistent only if it has a model. The notion of a model in sentential logic however, refers to the existence of a truth assignment under which all the sentences are true (simultaneously). Hence both in sentential and predicate logic the notion of consistency refers to the existence of a model. These notions of consistency are [model-theoretic](#) since they are intimately associated with the existence of a model.

We have reserved the term "[deductive consistency](#)" (definition 34.1) to a [proof-theoretic](#) notion obtained from deductions rather than models. *A priori* there is no reason to believe that the two notions are equivalent unless we can prove that our deductive system is sound and complete. While [soundness](#) has been proven we need to prove completeness before claiming that the model-theoretic notion of consistency and the proof-theoretic one are equivalent.

We need to carry our analogies between model-theory and proof theory a little further to the domain of maximal consistent sets (indeed some of the proof ideas would be analogous too!) in order to be able to prove the completeness of the system \mathcal{H}_1 . We refer to such maximally consistent sets obtained through deductions as being [deductively complete](#). The main difference however, is that we restrict ourselves to only closed formulae as will be evident soon.

Models of Deductively Consistent Sets

Lemma 34.2

If $\Phi \subseteq \mathcal{L}_1(\Sigma)$ has a model then it is deductively consistent.

Proof: Let $\mathbf{A} \Vdash \psi$ for each $\psi \in \Phi$. If Φ is not deductively consistent, then there exists ϕ such that $\Phi \vdash_{\mathcal{H}_1} \phi$ and $\Phi \vdash_{\mathcal{H}_1} \neg\phi$. However since \mathcal{H}_1 is **sound** it follows that $\mathbf{A} \Vdash \phi$ and $\mathbf{A} \Vdash \neg\phi$ which is a contradiction. QED

Deductive Completeness

Lemma 34.3

For any $\Phi \subseteq \mathcal{L}_1(\Sigma)$, $\Phi \vdash_{\mathcal{H}_1} \phi$ iff $\Phi \vdash_{\mathcal{H}_1} \vec{\forall}[\phi]$ iff $\Phi \cup \{\neg \vec{\forall}[\phi]\}$ is not deductively consistent.



We restrict our attention to only deductively consistent and complete sets.

Definition 34.2: Deductive completeness

A (deductively consistent) set $\Phi \subseteq \mathcal{L}_1(\Sigma)$ is **deductively complete** iff for every *closed* formula ϕ , $\Phi \vdash_{\mathcal{H}_1} \phi$ or $\Phi \vdash_{\mathcal{H}_1} \neg\phi$.

Proof of lemma 34.3

Proof:

- $\Phi \vdash_{\mathcal{H}_1} \phi$ iff $\Phi \vdash_{\mathcal{H}_1} \vec{\forall}[\phi]$ is obvious from rules $\forall I$ and $\forall E$.
- (\Rightarrow) Suppose $\Phi \vdash_{\mathcal{H}_1} \phi$. Then by monotonicity (theorem 14.1) $\Phi \cup \{\neg \vec{\forall}[\phi]\} \vdash_{\mathcal{H}_1} \phi$ and hence $\Phi \cup \{\neg \vec{\forall}[\phi]\} \vdash_{\mathcal{H}_1} \vec{\forall}[\phi]$. Further since $\Phi \cup \{\neg \vec{\forall}[\phi]\} \vdash_{\mathcal{H}_1} \neg \vec{\forall}[\phi]$, it follows that $\Phi \cup \{\neg \vec{\forall}[\phi]\}$ is not deductively consistent.
- (\Leftarrow) Suppose $\Phi \cup \{\neg \vec{\forall}[\phi]\}$ is not deductively consistent. Then there exists a formula (by lemma 34.1) ψ such that $\Phi \cup \{\neg \vec{\forall}[\phi]\} \vdash_{\mathcal{H}_1} \neg(\psi \rightarrow \psi)$. From $\vdash_{\mathcal{H}_1} \psi \rightarrow \psi$ and \perp we obtain $\Phi \cup \{\neg \vec{\forall}[\phi]\} \vdash_{\mathcal{H}_1} \vec{\forall}[\phi]$. By corollary 24.2 (Deduction theorem for closed formulae) we obtain $\Phi \vdash_{\mathcal{H}_1} \neg \vec{\forall}[\phi] \rightarrow \vec{\forall}[\phi]$. It follows from the derived axiom C2 that $\Phi \vdash_{\mathcal{H}_1} (\neg \vec{\forall}[\phi] \rightarrow \vec{\forall}[\phi]) \rightarrow \vec{\forall}[\phi]$ from which we obtain $\Phi \vdash_{\mathcal{H}_1} \vec{\forall}[\phi]$ by a single application of MP.

QED

Notes on proof of lemma 34.3.

1. The universal closure is required in the lemma, because in general the inconsistency of $\Phi \cup \{\neg \phi\}$ does not imply $\Phi \vdash_{\mathcal{H}_1} \phi$.

Example 34.1

Let $\Phi = \{\neg \forall x [\neg p(x)]\}$ and $\phi \equiv p(x)$. Then $\Phi \cup \{\neg p(x)\}$ is inconsistent. However, it is not possible to prove $\Phi \vdash_{\mathcal{H}_1} p(x)$, though it is possible to prove $\Phi \vdash_{\mathcal{H}_1} p(a)$ for some particular constant a .

2. Hence the maximally consistent sets of propositional logic translate into deductively complete sets in FOL. And this maximal completeness can only be shown for closed formulae and not for arbitrary formulae with free variables.

3. Clearly deductive completeness therefore is restricted to closed formulae.

The following is the proof-theoretic analogue of Lindenbaum's Theorem (theorem 12.3). Even the proof of the theorem mirrors the **alternative proof** of Lindenbaum's theorem.

Theorem 34.1: The Extension Theorem

Every deductively consistent set may be extended to a deductively complete set.

Proof: Let Φ be a nonempty deductively consistent set of Σ -formulae. For any enumeration of **closed** Σ -formulae

$$\psi_1, \psi_2, \psi_3, \dots \quad (71)$$

define the chain of sets $\Phi_0 \subseteq \Phi_1 \subseteq \Phi_2 \subseteq \dots$ starting with $\Phi_0 = \Phi$ as follows:

$$\Phi_{i+1} = \begin{cases} \Phi_i & \text{if } \Phi_i \vdash_{\mathcal{H}_1} \psi_i \\ \Phi_i \cup \{\neg \psi_i\} & \text{otherwise} \end{cases}$$

Claim. Each Φ_i is deductively consistent.

⊤ By induction on i . For $i = 0$, $\Phi_0 = \Phi$ is given to be deductively consistent. Assuming Φ_i is deductively consistent, we have that if $\Phi_{i+1} = \Phi$ it is obviously deductively consistent. Otherwise $\Phi_{i+1} = \Phi \cup \{\neg \psi_i\}$ and by lemma ?? since $\Phi_i \not\vdash_{\mathcal{H}_1} \psi_i$ and ψ is a closed formula, Φ_{i+1} must be deductively consistent. ⊥

Then $\Phi_\infty = \bigcup_{i \geq 0} \Phi_i$ is the desired set.

Claim. Φ_∞ is deductively consistent.

⊤ Suppose not. Then by lemma 34.1, for some formula ϕ , we have $\Phi_\infty \vdash_{\mathcal{H}_1} \neg(\phi \rightarrow \phi)$. However since such a proof is finite there exists a

finite subset $\Psi \subseteq_f \Phi$, such that $\Psi \vdash_{\mathcal{H}_1} \neg(\phi \rightarrow \phi)$. Since Ψ is finite there exists a $k \geq 0$ such that $\Psi \subseteq \Phi_k$ which implies $\Phi_k \vdash_{\mathcal{H}_1} \neg(\phi \rightarrow \phi)$ contradicting the previous claim that Φ_k is deductively consistent. \dashv

Claim. Φ_∞ is deductively complete.

\vdash For any arbitrary closed formula ψ , ψ occurs in the enumeration (71) at some position, say $\psi \equiv \psi_m$ for some $m \geq 0$. If $\Phi_m \vdash_{\mathcal{H}_1} \psi_m$ then $\Phi_\infty \vdash_{\mathcal{H}_1} \psi_m$. Otherwise $\Phi_{m+1} = \Phi_m \cup \{\neg\psi_m\}$ and $\Phi_\infty \vdash_{\mathcal{H}_1} \neg\psi_m$. By definition 34.2 Φ_∞ is deductively complete. \dashv

QED

The Completeness Theorem

Theorem 34.2: Gödel's Completeness Theorem

$\Phi \models \phi$ implies $\Phi \vdash_{\mathcal{H}_1} \phi$. When $\Phi = \emptyset$ we have that all valid formulae of $\mathcal{L}_1(\Sigma)$ are theorems of \mathcal{H}_1 .



Proof: Assume $\Phi \models \phi$ and suppose $\Phi \not\vdash_{\mathcal{H}_1} \phi$. Then by lemma 34.3 $\Phi \cup \{\neg \forall[\phi]\}$ is deductively consistent and hence possesses a model \mathbf{A} . But that implies $\mathbf{A} \Vdash \Phi$ but $\mathbf{A} \not\Vdash \phi$, which by definition means $\Phi \not\models \phi$, a contradiction. QED

35. First-Order Theories

35: First-order Theories

1. (Simple) Directed Graphs
2. (Simple) Undirected Graphs
3. Irreflexive Partial Orderings
4. Irreflexive Linear Orderings
5. (Reflexive) Preorders
6. (Reflexive) Partial Orderings
7. (Reflexive) Linear Orderings
8. Equivalence Relations
9. Peano's Postulates
10. The Theory of The Naturals
11. Notes and Explanations
12. Finite Models of Arithmetic
13. A Non-standard Model of Arithmetic
14. Z-Chains
15. First-order Mathematical Induction
16. First-order Arithmetic
17. Extending First-order Arithmetic

35.1. First-order Theories and Models

Recall that a *sentence* in first-order logic is a closed formula (one with no free variables).

Definition 35.1: First-order theories of structures

For any Σ -structure \mathbf{A} the **first-order theory of \mathbf{A}** , denoted $\text{Th}(\mathbf{A})$ is the set of all sentences true in \mathbf{A} .

$$\text{Th } \mathbf{A} = \{\phi \in \mathcal{P}_1(\Sigma) \mid FV(\phi) = \emptyset, \mathbf{A} \models \phi\}$$

For any class \mathfrak{A} of Σ -structures,

$$\text{Th } \mathfrak{A} = \{\phi \in \mathcal{P}_1(\Sigma) \mid FV(\phi) = \emptyset, \mathfrak{A} \models \phi\}$$

Note. Let Σ be a nonempty signature.

1. There is a nonempty *smallest* first-order theory $\text{Th}(\mathbb{PC}_1)$ consisting of all the *logically valid* sentences of first-order logic.
2. If Σ contains $=$, then there is a nonempty first-order theory $\text{Th}(\mathbb{PC}_1 =)$ which includes $\text{Th}(\mathbb{PC}_1)$.
3. The first-order theory $\text{Th}(\mathbf{A})$ of a Σ -structure \mathbf{A} includes $\text{Th}(\mathbb{PC}_1)$ (and includes $\text{Th}(\mathbb{PC}_1 =)$ if $= \in \Sigma$).
4. The *largest* first-order theory consisting of all the sentences in the language is *unsatisfiable* since ϕ and $\neg\phi$ both belong to the theory (for any sentence ϕ).

Definition 35.2: Logical consequences

For any set Γ of sentences in $\mathcal{P}_1(\Sigma)$ the set

$$\text{LC } \Gamma = \{\phi \in \mathcal{P}_1(\Sigma) \mid FV(\phi) = \emptyset, \Gamma \models \phi\}$$

is the set of all **logical consequences** of Γ .

Definition 35.3: Deductive consequences

Given a proof system \mathcal{D} , (which may or may not be complete) and a set Γ of sentences, we may define the set

$$\mathcal{DC} \Gamma = \{\phi \in \mathcal{P}_1(\Sigma) \mid FV(\phi) = \emptyset, \Gamma \vdash_{\mathcal{D}} \phi\}$$

of all **deductive consequences** of Γ .

Fact 35.1

For any set of closed Σ -formulae Γ ,

1. $\Gamma \subseteq \text{LC } \Gamma$
2. $\Gamma \subseteq \mathcal{DC} \Gamma$

Definition 35.4: Models of postulates

For any set Γ of Σ -sentences, the set of **models of Γ** is defined as the set

$$\text{Mod } \Gamma = \{\mathbf{A} \mid \mathbf{A} \Vdash \phi, \text{ for every } \phi \in \Gamma\}$$

We then have the following lemma which connects up the various sets that we have defined above.

Lemma 35.1

Th ($\text{Mod } \Gamma$) = $\text{LC } \Gamma$ = $\mathcal{DC} \Gamma$ provided \mathcal{D} is sound and complete.

Proof: For any $\phi \in \text{Th} (\text{Mod } \Gamma)$, we have $\text{Mod } \Gamma \Vdash \phi$. Hence for any $\mathbf{A} \in \text{Mod } \Gamma$ we have $\mathbf{A} \Vdash \phi$. If $\phi \in \Gamma \subseteq \text{LC } \Gamma$ there is nothing to prove. Suppose $\phi \notin \Gamma$, then since ϕ is true in every model of Γ , we have $\Gamma \models \phi$ and hence $\phi \in \text{LC } \Gamma$.

This proves that $\text{Th} (\text{Mod } \Gamma) \subseteq \text{LC } \Gamma$. The second part of the containment may be proven using the same argument backwards. QED

Hence it suffices to specify a (finite or infinite) set of first-order sentences in order to capture the models (which may well be a figment of our imagination) of interest to us. The first-order theory of these models would be the set of theorems that can be deduced from these axioms under a sound and complete deductive system. This first-order theory will in turn capture all the logical consequences of the set Γ of axioms.

Very often we may be interested in only a single structure \mathbf{A} and we may define axioms that we consider are fundamental properties of \mathbf{A} . However it may turn out that there may be other structures (with the same signature) that we never dreamed of, which satisfy the same set of axioms under a suitable interpretation of the signature. These other structures

need not be isomorphic to the structure \mathbf{A} . However they would also satisfy all properties that are logical consequences of the axioms too. What we envisaged as a first-order theory of \mathbf{A} might very well turn out to be a first order theory of a class of structures \mathfrak{A} of which \mathbf{A} is just one member.

On the other hand if the set Γ is a set of axioms which lead to a contradiction, there would be no models at all. In this case every first-order sentence over the signature would be a logical consequence of the axioms and the set of theorems would include every first-order sentence too.

(Simple) Directed Graphs

$$\begin{aligned}\Sigma &= \{ ; e : s^2 \} \\ \phi_{e-\text{irreflexivity}} &\stackrel{df}{=} \forall x [\neg e(x, x)] \\ \Phi_{DG} &\stackrel{df}{=} \{\phi_{e-\text{irreflexivity}}\}\end{aligned}$$

(Simple) Undirected Graphs

 Σ $= \{; e : s^2\}$ $\phi_{e-\text{irreflexivity}}$ $\stackrel{df}{=} \forall x[\neg e(x, x)]$ $\phi_{e-\text{symmetry}}$ $\stackrel{df}{=} \forall x, y[e(x, y) \rightarrow e(y, x)]$ Φ_{UG} $\stackrel{df}{=} \{\phi_{e-\text{irreflexivity}}, \phi_{e-\text{symmetry}}\}$

Equivalently $\phi'_{e-\text{symmetry}} \stackrel{df}{=} \forall x, y[e(x, y) \leftrightarrow e(y, x)]$ (see exercise ???.2) may be used in place of $\phi_{e-\text{symmetry}}$.

Irreflexive Partial Orderings

$$\begin{aligned}\Sigma &= \{ ; < \} \\ \phi_{<-irreflexivity} &\stackrel{df}{=} \forall x [\neg(x < x)] \\ \phi_{<-transitivity} &\stackrel{df}{=} \forall x, y, z [(x < y) \wedge (y < z)) \rightarrow (x < z)] \\ \Phi_{IPO} &\stackrel{df}{=} \{\phi_{<-irreflexivity}, \phi_{<-transitivity}\}\end{aligned}$$

Irreflexive Linear Orderings

$$\begin{aligned}\Sigma &= \{ ; < \} \\ \phi_{<-irreflexivity} &\stackrel{df}{=} \forall x[\neg(x < x)] \\ \phi_{<-transitivity} &\stackrel{df}{=} \forall x, y, z[((x < y) \wedge (y < z)) \rightarrow (x < z)] \\ \phi_{<-trichotomy} &\stackrel{df}{=} \forall x, y[(x < y) \vee (x = y) \vee (y < x)] \\ \Phi_{ILO} &\stackrel{df}{=} \{\phi_{<-irreflexivity}, \phi_{<-transitivity}, \\ &\quad \phi_{<-trichotomy}\}\end{aligned}$$

(Reflexive) Preorders

$$\begin{aligned}\Sigma &= \{ ; \leq : s^2 \} \\ \phi_{\leq-\text{reflexivity}} &\stackrel{df}{=} \forall x [x \leq x] \\ \phi_{\leq-\text{transitivity}} &\stackrel{df}{=} \forall x, y, z [(x \leq y) \wedge (y \leq z)) \rightarrow (x \leq z)] \\ \Phi_{Pre} &\stackrel{df}{=} \{\phi_{\leq-\text{reflexivity}}, \phi_{\leq-\text{transitivity}}\}\end{aligned}$$

(Reflexive) Partial Orderings

 Σ $= \{; \leq : \mathcal{S}^2\}$ $\phi_{\leq-\text{reflexivity}}$ $\stackrel{df}{=} \forall x [x \leq x]$ $\phi_{\leq-\text{transitivity}}$ $\stackrel{df}{=} \forall x, y, z [(x \leq y) \wedge (y \leq z) \rightarrow (x \leq z)]$ $\phi_{\leq-\text{antisymmetry}}$ $\stackrel{df}{=} \forall x, y [((x \leq y) \wedge (y \leq x)) \rightarrow x = y]$ Φ_{PO} $\stackrel{df}{=} \{\phi_{\leq-\text{reflexivity}}, \phi_{\leq-\text{transitivity}}, \phi_{\leq-\text{antisymmetry}}\}$

(Reflexive) Linear Orderings

$$\Sigma = \{; \leq : s^2\}$$

$$\phi_{\leq-\text{reflexivity}} \stackrel{df}{=} \forall x[x \leq x]$$

$$\phi_{\leq-\text{transitivity}} \stackrel{df}{=} \forall x, y, z[((x \leq y) \wedge (y \leq z)) \rightarrow (x \leq z)]$$

$$\phi_{\leq-\text{dichotomy}} \stackrel{df}{=} \forall x, y[(x \leq y) \vee (y \leq x)]$$

$$\Phi_{LO} \stackrel{df}{=} \{\phi_{\leq-\text{reflexivity}}, \phi_{\leq-\text{transitivity}}, \phi_{\leq-\text{dichotomy}}\}$$

Equivalence Relations

$$\Sigma = \{; \sim : s^2\}$$

$$\phi_{\sim-\text{reflexivity}} \stackrel{df}{=} \forall x [x \sim x]$$

$$\phi_{\sim-\text{symmetry}} \stackrel{df}{=} \forall x, y [(x \sim y) \rightarrow (y \sim x)]$$

$$\phi_{\sim-\text{transitivity}} \stackrel{df}{=} \forall x, y, z [((x \sim y) \wedge (y \sim z)) \rightarrow (x \sim z)]$$

$$\Phi_{Equiv} \stackrel{df}{=} \{\phi_{\sim-\text{reflexivity}}, \phi_{\sim-\text{symmetry}}, \phi_{\sim-\text{transitivity}}\}$$

Peano's Postulates

P1. 0 is a natural number.

P2. If x is a natural number then x^{+1} (called the *successor* of x) is a natural number.

P3. $0 \neq x^{+1}$ for any natural number x .

P4. $x^{+1} = y^{+1}$ implies $x = y$

P5. Let P be a property that may or may not hold of every natural number.
If

Basis. 0 has the property P and

Induction Step. whenever a natural number x has the property P , x^{+1} also has the property P .

then all natural numbers have the property P .

The Theory of The Naturals

$$\Sigma_S = \{0 : \longrightarrow s, \quad +1 : s \longrightarrow s; = : s^2\} \quad \text{P1, P2}$$

$$\phi_{0-not successor} \stackrel{df}{=} \forall x[\neg(x^{+1} = 0)] \quad \text{P3}$$

$$\phi_{+1-injective} \stackrel{df}{=} \forall x \forall y[x^{+1} = y^{+1} \rightarrow x = y] \quad \text{P4}$$

$$\phi_{\neq 0 \rightarrow successor} \stackrel{df}{=} \forall y[\neg(y = 0) \rightarrow \exists x[y = x^{+1}]]$$

$$\phi_{(+1)^n-distinct} \stackrel{df}{=} \forall x[\neg(x^{(+1)^n} = x)]$$

$$\Phi_{(+1)^\infty-distinct} \stackrel{df}{=} \{\phi_{(+1)^n-distinct} \mid n > 0\}$$

$$\begin{aligned} \Phi_S \stackrel{df}{=} & \{\phi_{0-not successor}, \phi_{+1-injective}, \phi_{\neq 0 \rightarrow successor}\} \\ & \cup \Phi_{(+1)^\infty-distinct} \end{aligned}$$

Notes and Explanations

1. $x^{(+1)^n}$ denotes the n -fold application of $+1$ to x .
2. $\phi \neq 0 \rightarrow \text{successor}$ says that every “non-zero” element must have a “predecessor”.
3. $\mathbf{N}_S = \langle \mathbb{N}, \Sigma_S \rangle$ is a model of the axioms Φ_S .
4. The infinite collection of axioms $\Phi_{(+1)^n\text{-}distinct}$ is necessary to obtain models that are countable.
5. The axioms $\Phi_{(+1)^\infty\text{-}distinct}$ ensure that there are no finite models of Φ_S .

Finite Models of Arithmetic

1. If the infinite collection $\Phi_{(+1)^\infty\text{-}distinct}$ is replaced by a finite collection for some $m > 0$ i.e.

$$\Phi_{(+1)^{m>n>0}\text{-}distinct} = \{\phi_{(+1)^n\text{-}distinct} \mid m > n > 0\}$$

then both finite and countable models are possible.

2. If in addition to $\Phi_{(+1)^{m>n>0}\text{-}distinct}$ we also include the axiom

$$\psi_{modulo_m} \stackrel{df}{=} \forall x [x^{(+1)^m} = x]$$

we get models $\mathbf{Z}_{S_m} = \langle \mathbb{Z}_m, \Sigma_S \rangle$ for the integers modulo m and there are no infinite models.

A Non-standard Model of Arithmetic

Consider the model $\mathbf{N}_S = \langle \mathbb{N}, \Sigma_S \rangle$ of the axioms of number theory.

- We add a new element $0' \neq 0$.
- This implies adding an infinite number of new elements $0^{(+1)^n}$ one for each $n > 0$. For simplicity let us call these elements $1', 2', 3', \dots$. Each of these new elements is different from every element in \mathbb{N} .
- Since $0' \neq 0$, it must have a “predecessor” say $-1'$ which again leads to the addition of all the elements $-2', -3', -3', \dots$ each of which is distinct and different from all other elements. Let us call this set of elements \mathbb{Z}' .

$\mathbf{N}'_S = \langle \mathbb{N} \cup \mathbb{Z}', \Sigma_S \rangle$ is a model of Φ_S and is said to be *non-standard*.

Z-Chains

- \mathbb{Z}' is called a *Z-Chain*.
- $\mathbf{N}'_S = \langle \mathbb{N} \cup \mathbb{Z}', \Sigma_S \rangle$ is also a *countable* model of the axioms Φ_S .
- Further \mathbf{N}_S and \mathbf{N}'_S are **not isomorphic**
- We could add a countable number of distinct *Z-chains*, \mathbb{Z}'' , \mathbb{Z}''' , \mathbb{Z}'''' , etc. to obtain other distinct and mutually non-isomorphic models.
- Each of the models obtained above is also a *countable* model of Φ_S .
- Each of these models is also *non-standard*.

First-order Mathematical Induction

The principle of induction is really an inference rule (sometimes expressed as an axiom schema)

$$\text{Ind1. } \frac{\{0/x\}X, \forall x[X \rightarrow \{x^{+1}/x\}X]}{\forall x[X]}$$

where

- x is the only free variable of the predicate X .
- $\{0/x\}X$ is the basis of the induction,
- $\forall x[X \rightarrow \{x^{+1}/x\}X]$ is the induction step and
- X the antecedent in the induction step, is the induction hypothesis

Note.

- Notice that we have used syntactic substitutions to express the rule within our notation. But this rule is exactly the first-order rendering of the **Principle of Mathematical Induction – Version 1**.
- X in rule **Ind1** is a meta-variable on first-order predicates in **Peano arithmetic**. Rule **Ind1** by its very syntactic structure therefore is restricted to only the first-order properties of numbers whereas the principle **PMI1'** (and each of its equivalents) holds generally for properties of any order.

First-order Arithmetic

In addition to Peano's postulates first order arithmetic contains the addition and multiplication operations, which are axiomatized as follows. The axioms clearly reflect the definitions by induction of these operations.

$$\Sigma_A = \Sigma_S \cup \{ +, . : s^2 \rightarrow s \}$$

$$\phi_{+0} \stackrel{df}{=} \forall x[x + 0 = x]$$

$$\phi_{+1} \stackrel{df}{=} \forall x \forall y [x + (y^{+1}) = (x + y)^{+1}]$$

$$\phi_{.0} \stackrel{df}{=} \forall x[x.0 = 0]$$

$$\phi_{.+1} \stackrel{df}{=} \forall x \forall y [x.(y^{+1}) = (x.y) + x]$$

Example 35.1: Left additive identity

Let Φ_A be the axioms of first order arithmetic, we already have the axiom ϕ_{+0} which defines 0 to be the right identity for addition. We prove that it is also the left identity. That is, the formula $\phi_{0+} \stackrel{df}{=} \forall x[0 + x = x]$ may be formally deduced from the axioms Φ_A . More formally we show

$$\Phi_A \vdash_{\mathcal{H}_1} \phi_{0+} \quad (72)$$

Proof: The proof requires the use of the rule **Ind1**. So we define the first-order predicate $\psi(x) \stackrel{df}{=} 0 + x = x$. It is clear that $\phi_{0+} \equiv \forall x[\psi(x)]$. The proof proceeds through the following sub-proofs.

1. $\Phi_A \vdash_{\mathcal{H}_1} 0 + 0 = 0 \equiv \{0/x\}\psi(x)$
2. $\Phi_A \vdash_{\mathcal{H}_1} \forall x[\psi(x) \rightarrow \{x^{+1}/x\}\psi(x)]$

from which by the rule **Ind1**, the result (72) immediately follows.

1. The proof of the first step is fairly straightforward as shown below.

$$\text{MP} \frac{\phi_{+0} \quad \frac{\Phi_A \vdash \forall x[x + 0 = x]}{\Phi_A \vdash \forall x[x + 0 = x] \rightarrow \{0/x\}(x + 0 = x)}}{\Phi_A \vdash \{0/x\}(x + 0 = x) \equiv 0 + 0 = 0 \equiv \{0/x\}\psi(x)}$$

2. Since some of the sequents and the formulae are fairly long we revert to the **sequential style** of proof rather than construct a proof tree. Readers may convince themselves that the following proof indeed represents a valid proof tree.

(a) $\Phi_A \vdash \{0/x\}\psi(x)$

1

HOME PAGE

◀◀

◀

▶

▶▶

LCS August 5, 2021

Go BACK

FULL SCREEN

CLOSE

886 OF 1035

QUIT

- (b) $\Phi_A \vdash 0 + x = x \rightarrow (0 + x)^{+1} = x^{+1}$ (b), (DT \Leftarrow)
- (c) $\Phi_A, \psi(x) \vdash (0 + x)^{+1} = x^{+1}$
- (d) $\Phi_A, \psi(x) \vdash \forall x \forall y [x + (y^{+1}) = (x + y)^{+1}]$ ϕ_{++1}
- (e) $\Phi_A, \psi(x) \vdash \forall x \forall y [x + (y^{+1}) = (x + y)^{+1}] \rightarrow \{0/x\} \forall y [x + (y^{+1}) = (x + y)^{+1}]$ $\forall E$
- (f) $\Phi_A, \psi(x) \vdash \{0/x\} \forall y [x + (y^{+1}) = (x + y)^{+1}] \equiv \forall x [0 + (x^{+1}) = (0 + x)^{+1}]$ (e), (d), MP
- (g) $\Phi_A, \psi(x) \vdash \forall x [0 + (x^{+1}) = (0 + x)^{+1}] \rightarrow (0 + (x^{+1}) = (0 + x)^{+1})$ $\forall E$
- (h) $\Phi_A, \psi(x) \vdash 0 + (x^{+1}) = (0 + x)^{+1}$ (g), (f), MP
- (i) $\Phi_A, \psi(x) \vdash 0 + (x^{+1}) = x^{+1}$ (h), (c), =T
- (j) $\Phi_A \vdash \psi(x) \rightarrow \{x^{+1}/x\} \psi(x)$ (i), (DT \Rightarrow)
- (k) $\Phi_A \vdash \forall x [\psi(x) \rightarrow \{x^{+1}/x\} \psi(x)]$ (j), $\forall I$
- (l) $\Phi_A \vdash \forall x [\psi(x)] \equiv \phi_{0+}$ (a), (k), Ind1

QED

Extending First-order Arithmetic

The signature Σ_A may be extended by other operations and relations.

Linear order. Define the (infix) $<$: s^2 as

$$x < y \stackrel{df}{=} \exists z[\neg(z = 0) \wedge x + z = y]$$

Subtraction. Define the (infix) partial function $- : s^2 \rightarrow s$ as one satisfying the axiom

$$\forall x, y, z[(z = y - x) \leftrightarrow (x + z = y)]$$

Exercise 35.1

1. Express the associativity properties of addition and multiplication as first-order formulae.
2. Prove the associativity properties for addition and multiplication in First-order arithmetic from the non-logical axioms and the axioms of $\mathbb{PC} =$.
3. Express the commutativity of addition and multiplication operations in First-order arithmetic.
4. Prove the commutativity properties of addition and multiplication.
5. Prove that $\forall x, y, z[(z = y - x) \rightarrow \neg(z = 0) \rightarrow (x < y) \wedge (z < y)]$
6. Express the quotient and remainder operations on the naturals as axioms in First order arithmetic.

36. Towards Logic Programming

36: Towards Logic Programming

Computer science is no more about computers than astronomy is about telescopes.

Edsger W. Dijkstra

1. Logic Programming
2. Reversing the Arrow
3. Arrow Reversal
4. Horn Clauses
5. Program or Rule Clause
6. Facts: Unit Clauses
7. Goal clauses
8. Logic Programs
9. Prolog: Abstract Interpreter 0
10. Sorting in Logic
11. Prolog: Abstract Interpreter 1

Logic Programming

A program is a theory (in some logic) and computation is deduction from the theory.

J. A. Robinson

Reversing the Arrow

Let

$$\phi \leftarrow \psi \stackrel{df}{=} \psi \rightarrow \phi$$

Consider any clause $C = \{\pi_1, \dots, \pi_p\} \cup \{\neg\nu_1, \dots, \neg\nu_n\}$ where π_i , $1 \leq i \leq p$ are *positive literals* and $\neg\nu_j$, $1 \leq j \leq n$ are the *negative literals*. Since a clause in FOL with free variables represents the **universal closure** of the **disjunction of its literals**, we have

Arrow Reversal

$$\begin{aligned} C &\Leftrightarrow \vec{\forall}[(\bigvee_{1 \leq i \leq p} \pi_i) \vee (\bigvee_{1 \leq j \leq n} \neg \nu_j)] \\ &\Leftrightarrow \vec{\forall}[(\bigvee_{1 \leq i \leq p} \pi_i) \vee \neg(\bigwedge_{1 \leq j \leq n} \nu_j)] \\ &\Leftrightarrow \vec{\forall}[(\bigwedge_{1 \leq j \leq n} \nu_j) \rightarrow (\bigvee_{1 \leq i \leq p} \pi_i)] \\ &\equiv \vec{\forall}[(\bigvee_{1 \leq i \leq p} \pi_i) \leftarrow (\bigwedge_{1 \leq j \leq n} \nu_j)] \\ &\stackrel{df}{=} \pi_1, \dots, \pi_p \leftarrow \nu_1, \dots, \nu_n \end{aligned}$$

Horn Clauses

Definition 36.1: Horn clauses

Given a clause

$$C \stackrel{df}{=} \pi_1, \dots, \pi_p \leftarrow \nu_1, \dots, \nu_n$$

- Then C is a **Horn clause** if $0 \leq p \leq 1$.
- C is called a
 - **program clause or rule clause** if $p = 1$,
 - **fact or unit clause** if $p = 1$ and $n = 0$,
 - **goal clause or query** if $p = 0$,
- Each ν_j is called a **sub-goal** of the goal clause.

Program or Rule Clause

$$\begin{aligned} P &\stackrel{df}{=} \pi \leftarrow \nu_1, \dots, \nu_n \\ &\equiv \vec{\forall}[\pi \vee (\bigvee_{1 \leq j \leq n} \neg \nu_j)] \\ &\equiv \vec{\forall}[\pi \vee \neg(\bigwedge_{1 \leq j \leq n} \nu_j)] \end{aligned}$$

and is read as “ π if ν_1 and ν_2 and ... and ν_n ”.

Facts: Unit Clauses

$$\begin{aligned} F &\stackrel{df}{=} \pi \\ &\equiv \vec{\forall}[\pi] \end{aligned}$$

Goal clauses

Given a goal clause

$$\begin{aligned} G &\stackrel{df}{=} \leftarrow \nu_1, \dots, \nu_n \\ &\Leftrightarrow \vec{\forall}[\neg\nu_1 \vee \dots \vee \neg\nu_n] \\ &\Leftrightarrow \neg\vec{\exists}[\nu_1 \wedge \dots \wedge \nu_n] \end{aligned}$$

If $\vec{y} = FV(\nu_1 \wedge \dots \wedge \nu_n)$ then the goal is to prove that there exists an assignment to \vec{y} which makes $\nu_1 \wedge \dots \wedge \nu_n$ true.

Logic Programs

Definition 36.2: Logic programs

A logic program is a finite set of Horn clauses, i.e. it is a set of rules $P = \{h^1, \dots, h^k\}$, $k \geq 0$ with $h^l \equiv \pi^l \leftarrow \nu_1^l, \dots, \nu_{n_l}^l$, for $0 \leq l \leq k$. π^l is called the head of the rule and $\nu_1^l, \dots, \nu_{n_l}^l$ is the body of the rule.

Given a logic program P and a goal clause $G = \{\nu_1, \dots, \nu_n\}$ the basic idea is to show that

$$\begin{aligned} & P \cup \{G\} \text{ is unsatisfiable} \\ \Leftrightarrow & \vec{\forall}[\neg\nu_1 \vee \dots \vee \neg\nu_n] \text{ is a logical consequence of } P \\ \Leftrightarrow & \vec{\exists}[\nu_1 \wedge \dots \wedge \nu_n] \text{ is a logical consequence of } P \end{aligned}$$

Effectively showing $P \models \vec{\exists}[\nu_1 \wedge \dots \wedge \nu_n]$ implies that we need to find values for the variables $X = \bigcup_{j=1}^n FV(\nu_j)$ which ensure

that $P \cup \{G\}$ is unsatisfiable. By **Herbrand's theorem** this reduces to the problem of finding **substitutions of ground terms** in the Herbrand base for variables in such a manner as to ensure unsatisfiability of $P \cup \{G\}$. This substitution is also called a *correct answer substitution*.

We may regard a logic program therefore as a set of postulates of a family of models (represented by a Herbrand model) and any correct answer substitution that may be derived (through resolution refutation) as a proof of the Goal as a logical consequence of the postulates. Since resolution refutation is **sound** and **complete** we effectively show $P \vdash_{\mathcal{R}} \vec{\exists}[\nu_1 \wedge \dots \wedge \nu_n]$ where $\vdash_{\mathcal{R}}$ denotes a proof by resolution refutation.

A propositional logic program is one in which there are no variables either in P or in the goal clause G and the execution is a pure application of the rule **Res0** to obtain a contradiction.

Prolog: Abstract Interpreter 0

Algorithm = logic + control

R. Kowalski

We describe below an abstract interpreter (algorithm 36.1) for *propositional* logic programs.

Algorithm

Algorithm 36.1 $\text{INTERPRET0 } (P, G) \stackrel{df}{=}$

```
{ requires: A propositional logic program  $P$  and propositional goal  $G$ 
  goals := { $G$ }
  while  $goals \neq \emptyset$ 
    do { Choose some goal  $A \in goals$ 
         Choose a clause  $A' \leftarrow B_1, \dots, B_k \in P : A \equiv A'$ 
         if  $A'$  does not exist
             then exit
         else  $goals := (goals - \{A\}) \cup \{B_1, \dots, B_k\}$ 
    if  $goals = \emptyset$ 
        then return ( $yes$ )
        else return ( $no$ )
    ensures:  $yes$  if  $P \vdash G$  else  $no$ 
```

Sorting in Logic

$sort(x, y)$	$\leftarrow perm(x, y), ordered(y)$
$ordered(nil)$	\leftarrow
$ordered(x.nil)$	\leftarrow
$ordered(x.y.z)$	$\leftarrow lesseq(x, y), ordered(y.z)$
$lesseq(x, x)$	\leftarrow
$lesseq(x, y)$	$\leftarrow x < y$
$perm(nil, nil)$	\leftarrow
$perm(x.y, u.v)$	$\leftarrow delete(u, x.y, z), perm(z, v)$
$delete(x, x.y, y)$	\leftarrow
$delete(x, y.z, y.w)$	$\leftarrow delete(x, z, w)$
	$\leftarrow sort([2, 8, -1, 10, 4, 2], x)$

Example 36.1

Here is the corresponding Prolog encoding.

```
isort(X, Y) :- permutation(X, Y),  
              ordered(Y).  
  
ordered([]).  
ordered([H|T]).  
ordered([F|S|T]) :- lesseq(F, S),  
                  ordered(S|T).  
  
lesseq(F, S) :- F=S.  
lesseq(F, S) :- F<S.  
  
permutation([], []).  
permutation([H|T], [F|R]) :- delete(F, H|T, Z),  
                           permutation(Z, R).  
  
delete(H, H|T, T).  
delete(X, H|T, H|U) :- delete(X, T, U).  
  
/* isort([2,8,-1,10,4,2], Y). */
```

Example 36.2

Notice that merge-sort closely follows the corresponding functional implementation.

```
mergeSort([], []).  
mergeSort(H[], H[]).  
mergeSort(F.S.T, Sall) :- split(F.S.T, Left, Right),  
                      mergeSort(Left, Sleft),  
                      mergeSort(Right, Sright),  
                      merge(Sleft, Sright, Sall).  
  
split([], [], []).  
split(H[], H[], []).  
split(F.S.T, F.U, S.V) :- split(T, U, V).  
merge([], L, L).  
merge(L, [], L).  
merge(F.B, H.T, F.U) :- F=<H, merge(B, H.T, U).  
merge(F.B, H.T, H.V) :- H<F, merge(F.B, T, V).
```

Example 36.3

And so does quick-sort.

```
%consult(+/home/sak/prolog/ordered.P,/home/sak/prolog/permute.P).  
['permutation.P'].  
['ordered.P'].
```

```

quicksort([] , []).
quicksort(H.[] , H.[]).
quicksort(H.T, S) :- partition(H, T, L, G),
                  quicksort(L, Ls),
                  quicksort(G, Gs),
                  append(Ls, H.[] , Lsh),
                  append(Lsh, Gs, S).

partition(M, [] , [] , []).
partition(M, H.T, H.Lesser , Greater) :- H<=M,
                                         partition(M, T, Lesser , Greater).
partition(M, H.T, Lesser , H.Greater) :- M<H,
                                         partition(M, T, Lesser , Greater).

append([], L, L).
append(H.T, L, H.A) :- append(T, L, A).

/* testing
quicksort([-2, 10,3,5,-3,8,22],X).

sorted(X,Y):- quicksort(X,Y), ordered(Y), permutation(X,Y).

sorted([-2, 10,3,5,-3,8,22],X).

*/

```

Example 36.4

Here is a simple problem in *alphametics*

$$SEND + MORE = MONEY$$

which is a constrained equation solving problem. The goal is to be able to assign a digit to each letter so as to satisfy the given identity. The general constraints are that

1. every letter in the equation represents a different digit (0 – 9).
2. the leading digit in any multi-digit number is non-zero.

This rules out trivial solutions such as $0000 + 0000 = 00000$.

```
% SEND + MORE = MONEY
smm :- 
    L = [S,E,N,D,M,O,R,Y] ,
    Digits = [0,1,2,3,4,5,6,7,8,9] ,
    assign_digits(L, Digits) ,
    M > 0 ,
    S > 0 ,
    1000*S + 100*E + 10*N + D +
    1000*M + 100*O + 10*R + E ===
    10000*M + 1000*O + 100*N + 10*E + Y,
    write(' ') , write(S) , write(E) , write(N) , write(D) , nl ,
    write(' + ') , write(M) , write(O) , write(R) , write(E) , nl ,
    write(' -----') , nl ,
    write(' = ') , write(M) , write(O) , write(N) , write(E) , write(Y) , nl .
```

```
select(Z, [Z|R], R).
select(Z, [Y|Zs], [Y|Ys]):- select(Z, Zs, Ys).
```

```
assign_digits([], _List).
assign_digits([D|Ds], List):-
    select(D, List, NewList),
    assign_digits(Ds, NewList).
```

Example 36.5: Prolog: Naturals

An example of a (symbolic) rewrite system to compute normal forms.

```
isnf(z).
isnf(s(X)) :- isnf(X).
rewrite(X, X) :- isnf(X).
rewrite(s(X), s(Y)) :- rewrite(X, Y), isnf(Y).
rewrite(a(z, Y), Y) :- isnf(Y).
rewrite(a(Y, z), Y) :- isnf(Y).
rewrite(a(s(X), Y), s(Z)) :- rewrite(a(X, Y), Z).
rewrite(a(X, s(Y)), s(Z)) :- rewrite(a(X, Y), Z).
rewrite(a(X, Y), Z) :- rewrite(X, U), rewrite(Y, V), rewrite(a(U, V), Z).
even(z).
even(s(s(X))) :- rewrite(X, Y), even(Y).
odd(X) :- not even(X). % negation as failure
/* rewrite(a(a(s(z), s(s(z))), a(s(z), s(s(z)))), X).
X = s(s(s(s(s(z)))))) */
```

Example 36.6

In this example we give a prolog implementation of double-ended queues of integers using constructors.

```
deq(nullq).  
deq(fnq(A, D)) :- integer(A), deq(D).  
deq(rnq(B, D)) :- integer(B), deq(D).  
  
nonnull(fnq(A, D)) :- integer(A), deq(D).  
nonnull(rnq(B, D)) :- integer(B), deq(D).  
  
deq(fdq(D)) :- nonnull(D).  
deq(rdq(D)) :- nonnull(D).  
  
nf(nullq).  
nf(fnq(A, D)) :- integer(A), nf(D).  
  
rewrite(D, D) :- nf(D).  
%induction step for normal forms  
rewrite(fnq(A, D), fnq(A, E)) :- integer(A), rewrite(D, E).  
  
% for all forms other than normal forms  
rewrite(rnq(B, nullq), fnq(B, nullq)):- integer(B). % basis of induction  
rewrite(rdq(fnq(A, nullq)), nullq). % basis of induction  
  
% rewrite(fdq(fnq(A, nullq)), nullq) follows from the more general rewrite  
rewrite(fdq(fnq(A, D)), E):- integer(A), rewrite(D, E). % fdq for all nonnull
```

```

rewrite(rnq(B, fnq(A, D)), fnq(A, E)) :- % for rnq on normal forms
    integer(A), integer(B),
    rewrite(D, F),
    rewrite(rnq(B, F), E), nf(E).

rewrite(rnq(B, D), E) :- % for rnq on other forms
    integer(B),
    rewrite(D, F),
    rewrite(rnq(B, F), E).

rewrite(rdq(fnq(A, D)), fnq(A, E)) :- % for rdq on normal forms
    integer(A),
    rewrite(D, F), nonnull(F),
    rewrite(rdq(F), E).

rewrite(rdq(D), E) :- % for rdq on other forms
    rewrite(D, F), rewrite(rdq(F), E).

% rewrite(rdq(rnq(B, D)), D) follows by induction from the various rewrites above
fv(fnq(A, D), A) :- integer(A), deq(D). % value at the front of the dequeue
fv(D, B) :- rewrite(D, E), fv(E, B).
rv(fnq(A, nullq), B) :- integer(A), A=B.
rv(fnq(A, D), B) :- integer(A), rewrite(D, E), rv(E, B).
rv(D, B) :- rewrite(D, E), rv(E, B).

/* Testing

```

```
% Restoring file /usr/local/lib/Yap/startup
YAP version Yap-5.1.1
?- % reconsulting /home/sak/prolog/deques.P...
% reconsulted /home/sak/prolog/deques.P in module user , 0 msec 4096 bytes
yes
?- rewrite(fnq(2, fnq(1, nullq)), X).
X = fnq(2,fnq(1,nullq)) ?
yes
?- rv(fnq(1, fnq(2, nullq)), B).
B = 2 ?
yes
?- rv(rnq(3, fnq(1, fnq(2, nullq))), B).
B = 3 ?
yes
?- rewrite(rnq(4, fdq(fnq(1, fnq(2, rnq(3, nullq))))), X).
X = fnq(2,fnq(3,fnq(4,nullq))) ?
yes
?- rv(rnq(4, fdq(fnq(1, fnq(2, rnq(3, nullq))))), B).
B = 4 ?
yes
?- rv(rdq(rnq(4, fdq(fnq(1, fnq(2, rnq(3, nullq)))))), B).
B = 3 ?
yes
?- fv(rnq(4, fdq(fnq(1, fnq(2, rnq(3, nullq))))), B).
B = 2 ?
yes
*/
```

Example 36.7: Occurs-check

The occurs-check problem in Prolog.

```
/* To determine whether Prolog does the "occur–check".  
** In the following program it is clear that "test" is not a logical  
** consequence of "p(X, f(X))." and "test :- p(Y,Y)." since p(Y,Y) can  
** never be unified with "p(X, f(X))", sinc ethe "occur–check" is not  
** satisfied. Once the substitution {Y/X} has been done, the next  
** disagreement set is {Y, f(Y)}. However since Y occurs in f(Y) a  
** substitution such as {f(Y)/Y} can never lead to a unification of  
** p(Y, Y) with p(Y, f(Y)). However since this occur check is not  
** performed and the resulting unification is not verified in any  
** other way, Prolog returns the answer "yes" even though "test" is  
** not a logical consequence
```

```
*/
```

```
p(X, f(X)).  
test :- p(Y,Y).
```

```
/* Here is the XSB Prolog session.  
xsb  
[ xsb_configuration loaded]  
[ sysinitrc loaded]  
[ packaging loaded]
```

XSB Version 2.4 (Bavaria) of July 13, 2001
[i686–pc–linux–gnu; mode: optimal; engine: chat; gc: copy; scheduling: local]

```
| ?- [occurcheck].  
[Compiling ./occurcheck]  
[occurcheck compiled, cpu time used: 0.0590 seconds]  
[occurcheck loaded]
```

```
yes  
| ?- test.
```

```
yes  
| ?-
```

```
*/
```

```
/* The following example shows that if Prolog does the occur check  
** then it could easily return the answer "no" for the following.  
*/
```

```
test2 :- q(X, X).  
q(Y, f(Y)) :- q(Y, Y).
```

```
/* However Prolog goes off into an infinite loop in the trying to unify,  
** because it does not perform the "occur–check"  
*/
```

/*

Difference lists in Prolog

```
** A difference list is a Prolog structure consisting of two lists , (with
** possibly uninstantiated components such that the second list is or can be
** a suffix of the first one.
*/
```

```
/* With difference lists the append function can sometimes be done in constant
** time , since Prolog uses the uninstantiated variable as endmarker of the
** difference list
*/
```

```
append_dl(X\Y, Y\Z, X\Z).
```

```
/* Look at the following execution
| ?- append_dl([1,2,3|X]\X, [4,5]\[], Y).
```

```
X = [4,5]
Y = [1,2,3,4,5] \ []
```

```
*/
```

```
/* Now consider the following
```

** An implementation without "occur check would allow us to conclude
** that concat([], [], [a]) holds true

*/

```
test3 :- concat(U\U, V\V, [a|W]\W).  
concat(X\Y, Y\Z, X\Z).
```

/* as indeed it does!

| ?- test3.

yes
| ?-

*/

Prolog: Abstract Interpreter 1

We give an abstract interpreter for Prolog programs which refines the earlier abstract interpreter for the *propositional* version.

The following algorithm 36.2 is a refinement of algorithm 36.1 for general logic programs with variables and predicates and it uses the unification algorithm 28.2.

Algorithm

Algorithm 36.2

INTERPRET1 $(P, G) \stackrel{df}{=}$

```

    { requires: A logic program  $P$  and goal  $G$ 
      Standardize variables apart in  $\mathbf{P} \cup \{G\}$ 
       $goalStack := emptyStack$ 
       $\theta := 1$ 
       $push(goalStack, \theta G)$ 
      while  $\neg empty(goalStack)$ 
        do {  $A := pop(goalStack)$ 
              if  $\exists A' \leftarrow B_1, \dots, B_k \in P : unifiable(A, A')$ 
                then {  $\tau := UNIFY(A, A') // algorithm 28.2$ 
                       $\theta := \tau \circ \theta$ 
                      else exit
                      if  $k > 0$ 
                        then  $push(goalStack, \theta B_k, \dots, \theta B_1)$ 
              if  $empty(goalStack)$ 
                then return ( $\theta$ )
                else return ( $no$ )
            ensures: if  $P \vdash G$  then  $\theta$  else  $no$ 
}
}

```


37. Verification of Imperative Programs

37: Verification of Imperative Programs

Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W. Dijkstra

1. The WHILE Programming Language
2. Programs As State Transformers
3. The Semantics of WHILE
4. Programs As Predicate Transformers
5. Correctness Assertions
6. Total Correctness of Programs
7. Total Correctness = Partial Correctness + Termination
8. Examples: Factorial 1
9. Examples: Factorial 1'
10. Examples: Factorial 2

The WHILE Programming Language

Definition 37.1: Syntax of $\mathcal{WH}(\Sigma)$

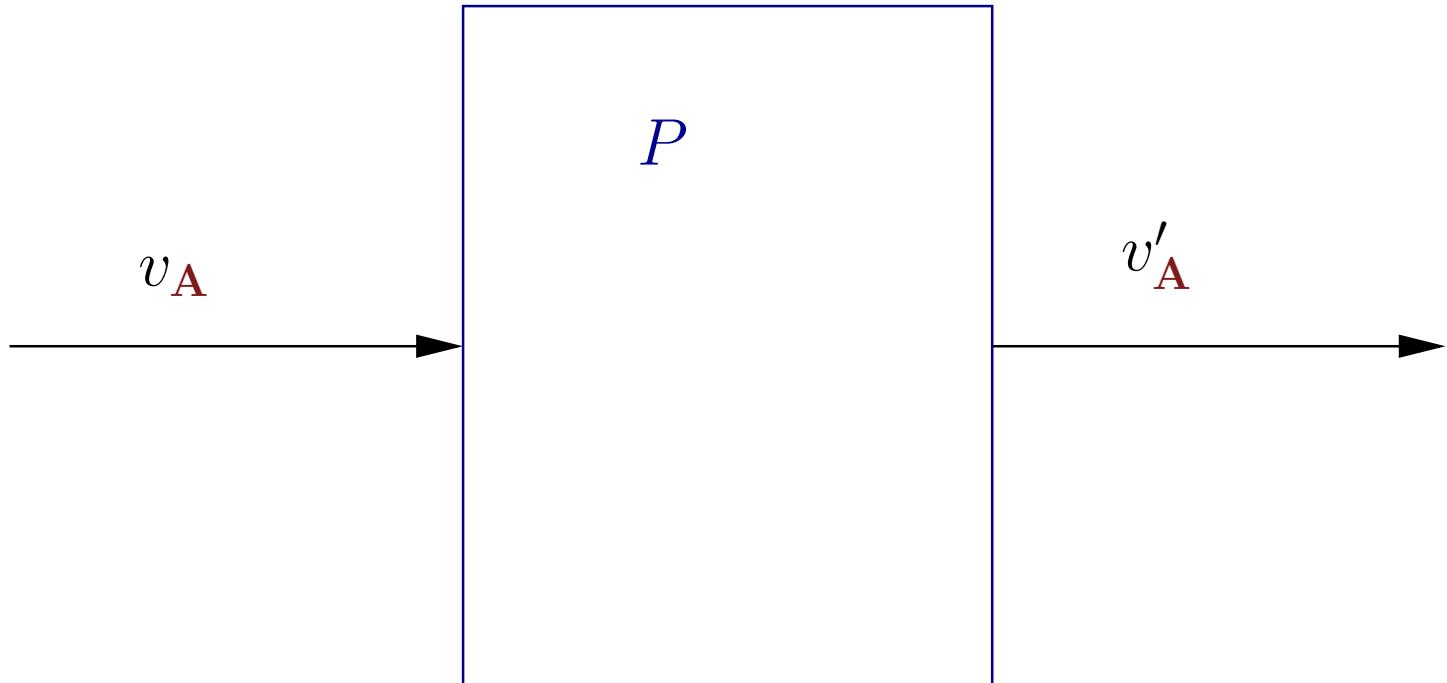
Let Σ be any signature. Then the programming language $\mathcal{WH}(\Sigma)$ is defined by the following BNF.

$$\begin{aligned} P, Q ::= & \epsilon && (\text{Skip}) \\ & | \quad x := t && (\text{Assignment}) \\ & | \quad P; Q && (\text{Composition}) \\ & | \quad [P] && (\text{Block}) \\ & | \quad \chi?P : Q && (\text{Conditional}) \\ & | \quad \{\chi?P\} && (\text{While}) \end{aligned}$$

where $t \in \mathbb{T}_\Sigma(V)$ and $\chi \in \mathcal{QF}_1(\Sigma)$.

Programs As State Transformers

A program is a *state transformer* which takes a valuation (also called a **state**) v_A of variables and yields another valuation v'_A .



The Semantics of WHILE

Definition 37.2: Semantics of $\mathcal{WH}(\Sigma)$

Let \mathbf{A} be a Σ -algebra. $\mathcal{M}_{\mathbf{A}}[\![P]\!]: \mathbf{V}_{\mathbf{A}} \rightarrow \mathbf{V}_{\mathbf{A}}$

$$\begin{aligned}
 \mathcal{M}_{\mathbf{A}}[\![\epsilon]\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} v_{\mathbf{A}} \\
 \mathcal{M}_{\mathbf{A}}[\![x := t]\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} v_{\mathbf{A}}[x := \mathcal{V}_{\mathbf{A}}[\![t]\!]_{v_{\mathbf{A}}}] \\
 \mathcal{M}_{\mathbf{A}}[\![P]\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} \mathcal{M}_{\mathbf{A}}[\![P]\!]_{v_{\mathbf{A}}} \\
 \mathcal{M}_{\mathbf{A}}[\![P; Q]\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} (\mathcal{M}_{\mathbf{A}}[\![Q]\!] \circ \mathcal{M}_{\mathbf{A}}[\![P]\!])_{v_{\mathbf{A}}} \\
 \mathcal{M}_{\mathbf{A}}[\![\chi?P : Q]\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} \begin{cases} \mathcal{M}_{\mathbf{A}}[\![P]\!]_{v_{\mathbf{A}}} & \text{if } \mathcal{T}_{\mathbf{A}}[\![\chi]\!]_{v_{\mathbf{A}}} = 1 \\ \mathcal{M}_{\mathbf{A}}[\![Q]\!]_{v_{\mathbf{A}}} & \text{if } \mathcal{T}_{\mathbf{A}}[\![\chi]\!]_{v_{\mathbf{A}}} = 0 \end{cases} \\
 \mathcal{M}_{\mathbf{A}}[\!\{\chi?P\}\!]_{v_{\mathbf{A}}} &\stackrel{df}{=} \begin{cases} \mathcal{M}_{\mathbf{A}}[\![P; \{\chi?P\}]\!]_{v_{\mathbf{A}}} & \text{if } \mathcal{T}_{\mathbf{A}}[\![\chi]\!]_{v_{\mathbf{A}}} = 1 \\ v_{\mathbf{A}} & \text{if } \mathcal{T}_{\mathbf{A}}[\![\chi]\!]_{v_{\mathbf{A}}} = 0 \end{cases}
 \end{aligned}$$

Notes:

1. Except for the *While* command, all other commands are well-defined.
2. Notice that the last construct viz the *While* command with its meaning defined in terms of the meaning of itself and the *Composition* command inherently makes the meaning function partial (definition -4.4).
3. The meaning of the *While* command corresponds to the usual meaning associated with a (possibly indefinite) “loop-unrolling”.
4. While we have employed structural induction (see section -3.3) to define the semantics of our term languages and logical languages, there is no guarantee that this definition is inductive¹⁶. Hence any program employing this command is potentially undefined at least for certain valuations.
5. The correctness of any program employing the *While* construct needs to establish its “well-defined-ness” by proving an appropriate property of the initial state v_A and all subsequent states (obtained through *Composition*) that the program may traverse.

We could alternatively consider the graph (see -4.3) of the function $\mathcal{M}_A[\![P]\!]$ as a binary relation on V_A . This yields the relational semantics of programs which ignores all the “input” states for which the output state is undefined.

Definition 37.3: Relational semantics

$$\mathcal{R}_A[\![P]\!] = \text{Graph}(\mathcal{M}_A[\![P]\!]) = \{\langle v_A, v'_A \rangle \mid \mathcal{M}_A[\![P]\!]v_A = v'_A\}$$

¹⁶In fact the definition of the *While* command is structurally more complex than the *While* command itself.

However this is also not structurally inductive. But it restricts attention to only those input states from which program execution is guaranteed to terminate in a well-defined state.

Exercise 37.1: State transformer Semantics of $\mathcal{WH}(\Sigma)$

We may define an ordering on programs as follows.

Definition 37.4: Program ordering

$P \sqsubseteq_{WH} Q$ iff for each Σ -algebra \mathbf{A} and each valuation $v_{\mathbf{A}}$, either $\mathcal{M}_{\mathbf{A}}[\![P]\!]_{v_{\mathbf{A}}}$ is undefined or $\mathcal{M}_{\mathbf{A}}[\![P]\!]_{v_{\mathbf{A}}} = \mathcal{M}_{\mathbf{A}}[\![Q]\!]_{v_{\mathbf{A}}}$

1. Prove that \sqsubseteq_{WH} is a partial order on programs.
2. Prove that $P \sqsubseteq_{WH} P'$ implies for any program Q ,
 - (a) $[P] \sqsubseteq_{WH} [P']$.
 - (b) $P; Q \sqsubseteq_{WH} P'; Q$
 - (c) $\chi?P : Q \sqsubseteq_{WH} \chi?P' : Q$
 - (d) $\{\chi?P\} \sqsubseteq_{WH} \{\chi?P'\}$

Exercise 37.2: Program equivalence

We may define an equivalence relation on programs in $\mathcal{WH}(\Sigma)$ as follows:

Definition 37.5: Program equivalence

$$P =_{WH} Q \text{ iff for each } \Sigma\text{-algebra } \mathbf{A} \text{ and each valuation } v_{\mathbf{A}}, \mathcal{R}_{\mathbf{A}}[P]_{v_{\mathbf{A}}} = \mathcal{R}_{\mathbf{A}}[Q]_{v_{\mathbf{A}}}$$

1. Prove that $P =_{WH} Q$ iff $P \sqsubseteq_{WH} Q$ and $Q \sqsubseteq_{WH} P$.
2. For $t, u \in \mathbb{T}_\Sigma$, $t = u$ implies for all variables x , $x := t =_{WH} x := u$.
3. For $\chi, \theta \in \mathcal{QF}_1(\Sigma)$, $\chi \Leftrightarrow \theta$ implies $\chi?P : Q =_{WH} \theta?P : Q$.
4. Prove the following for all programs P , Q and R in $\mathcal{WH}(\Sigma)$.
 - (a) $P; \epsilon =_{WH} P =_{WH} \epsilon; P$.
 - (b) $P; [Q; R] =_{WH} [P; Q]; R$.
 - (c) $\chi?P : Q =_{WH} \neg\chi?Q : P$.
 - (d) $\{\chi?P\} =_{WH} \chi?[P; \{\chi?P\}] : \epsilon$.
5. Prove that $=_{WH}$ is a congruence relation on $\mathcal{WH}(\Sigma)$.

37.1. Programs as Predicate Transformers

Since $\mathcal{M}_A[\![P]\!]$ for any program P is a partial function (see part 2 of definition -4.4) we may make it total by defining the function over sets of source states yielding target states. Of course, in the process, we lose the exact correspondence between input and output states. Let $V_A = \{v_A \mid v_A : V \rightarrow |A|\}$ be the set of all valuations over A . Then

$$\mathcal{M}'_A[\![P]\!] : \mathcal{P}^{V_A} \longrightarrow \mathcal{P}^{V_A} \quad (73)$$

such that for any set $S_A \subseteq V_A$

$$\mathcal{M}'_A[\![P]\!]_{S_A} \stackrel{df}{=} T_A \quad (74)$$

where

$$T_A = \{v'_A \mid v'_A = \mathcal{M}_A[\![P]\!]_{v_A}, v_A \in S_A\}$$

Notes:

- V_A could be uncountable since V is countable and $|A|$ could be at least countable. However, since a program P has only a finite number of variables, the set of tuples of values that represent a *state* of the program is at most countable if $|A|$ is countable.
- For any program P let ϕ be any predicate whose free variables are (some superset of) the variables of P . We may then regard the predicate ϕ as defining a property which relates the values of the variables in the program. Indeed ϕ is a property of certain states of the program P . By abuse of notation we write $\{\phi\}$ to denote the set of all states that

satisfy the predicate. In other words,

$$\{\phi\} = \{v_A \mid (A, v_A) \Vdash \phi\} \quad (75)$$

and we refer to $\{v_A \mid (A, v_A) \Vdash \phi\}$ as the **characteristic set** of ϕ .

- In general, even if V_A is countable there is no guarantee that every subset of V_A can be defined by a predicate. In fact, since the number of possible subsets of a countable set V_A is uncountable, there are only at most a countable number of them which can be characterised by predicates. However, every predicate ϕ whose free variables are drawn from only a finite superset of the variables of the program, does have a unique characteristic set $\{\phi\}$ given by equation (75).
- In particular if $\{\phi\} = S_A$ and $\{\psi\} = T_A$ in equation (74) then we may claim that **P transforms** the predicate ϕ into the predicate ψ and denote this fact by the identity.

$$\{\phi\}P = \{\psi\} \quad (76)$$

- More generally the identity (76) may be regarded as a relation that connects the program P with its **pre-condition** (ϕ) and its **post-condition** (ψ).

$$\{\phi\} P \{\psi\} \quad (77)$$

The triple (77) is also called a **Hoare triple**.

Program Specification

There are several different ways in which we could regard the relationships that exist between the three components of the triple (77).

State transformer. P transforms a state in $\{\phi\}$ into a state in $\{\psi\}$.

Predicate transformer. The triple (77) allows us to regard the program P as one that given an *input* state v_A satisfying the predicate ϕ may¹⁷ produce an *output* state v'_A that satisfies the predicate ψ . That is, P **transforms** the predicate ϕ into the predicate ψ . That is,

$$P : \mathcal{P}_1(\Sigma) \longrightarrow \mathcal{P}_1(\Sigma) \quad (78)$$

and $P(\{\phi\}) = \{\psi\}$.

Specification and Implementation. We may also think of the pair $\langle\phi, \psi\rangle$ as a **specification** that requires a program P to transform the predicate ϕ into ψ . Any such program then is an **implementation** of the specification. We may also say that P **realises** the specification $\langle\phi, \psi\rangle$. In this context we also note the following.

- If $\{\phi\} P \{\psi\}$ and $\{\phi'\} \subseteq \{\phi\}$ then $\{\phi'\} P \{\psi\}$ also holds.
- If $\{\phi\} P \{\psi\}$ and $\{\psi\} \subseteq \{\psi'\}$ then $\{\phi\} P \{\psi'\}$ also holds.
- Hence if $\{\phi\} P \{\psi\}$, $\{\phi'\} \subseteq \{\phi\}$ and $\{\psi\} \subseteq \{\psi'\}$ then $\{\phi'\} P \{\psi'\}$ also holds.

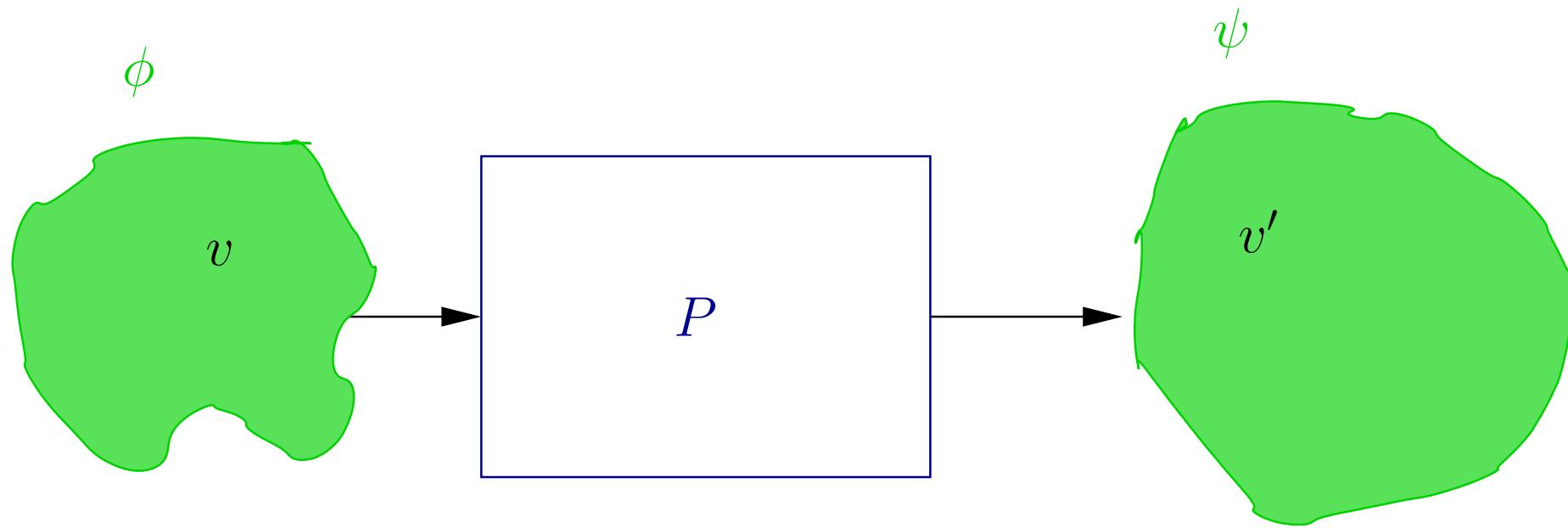
Hence program P may realise an infinite collection of specifications!

Correctness Assertion. The triple (77) may be regarded as a correctness assertion which states that P is correct with respect to the specification $\langle\phi, \psi\rangle$, and hence potentially requires a proof of correctness. This also implies that one requires a collection of axioms and inference rules from which the triple $\{\phi\} P \{\psi\}$ may be deduced.

¹⁷or may fail to terminate.

Programs As Predicate Transformers

We may also view a program as a transformer of properties. Consider a predicate ϕ to represent a set of possible valuations i.e. $V_A(\phi) = \{v_A \mid (A, v_A) \Vdash \phi\}$. Then a program transforms a state satisfying ϕ into a state satisfying some other predicate ψ .



Correctness Assertions

Definition 37.6: Partial correctness assertion

A **partial correctness assertion** (also called a **Hoare-triple**) is a triple of the form $\{\phi\} P \{\psi\}$ where ϕ is a formula called the **precondition**, P is a program and ψ is the **postcondition**.

Definition 37.7

- $\{\phi\} P \{\psi\}$ holds in a state v_A (denoted $(A, v_A) \Vdash \{\phi\} P \{\psi\}$), if $(A, v_A) \Vdash \phi$ and $\mathcal{M}_A[P]_{v_A} = v'_A$ implies $(A, v'_A) \Vdash \psi$
- $\{\phi\} P \{\psi\}$ is valid in A , (denoted $A \Vdash \{\phi\} P \{\psi\}$) if $(A, v_A) \Vdash \{\phi\} P \{\psi\}$ for every state v_A .

Total Correctness of Programs

Definition 37.8: Total correctness assertion

A total correctness assertion is a triple of the form $\{\phi\} P \{\psi\}!$ where ϕ is a formula called the precondition, P is a program and ψ is the postcondition.

Definition 37.9

- $\{\phi\} P \{\psi\}!$ holds in a state v_A (denoted $(A, v_A) \Vdash \{\phi\} P \{\psi\}!$), if $(A, v_A) \Vdash \phi$ implies for some v'_A , $M_A[P]_{v_A} = v'_A$ and $(A, v'_A) \Vdash \psi$.
- $\{\phi\} P \{\psi\}!$ is valid in A , (denoted $A \Vdash \{\phi\} P \{\psi\}!$) if $(A, v_A) \Vdash \{\phi\} P \{\psi\}!$ for every state v_A .

Total Correctness = Partial Correctness + Termination

Fact 37.1: Total Correctness

$(A, v_A) \Vdash \{\phi\} P \{\psi\}!$ iff $(A, v_A) \Vdash \{\phi\} P \{\psi\}$ and $\mathcal{M}_A[P]_{v_A} \in V_A$.

Termination depends upon the interpretation of the terms in $T_\Sigma(V)$ and the carrier set of the Σ -algebra.

Examples: Factorial 1

Example 37.1

Let $\Sigma \supseteq \mathbb{N} \cup \{! : s \longrightarrow s, +, -, * : s^2 \longrightarrow s; =, > : s^2\}$ where

$$x! \stackrel{df}{=} \begin{cases} 1 & \text{if } x=0 \\ x * (x - 1)! & \text{otherwise} \end{cases}$$

Let $\mathbf{N} = \langle \mathbb{N}, \Sigma \rangle$.

Then for $P_1 \stackrel{df}{=} p := 1; \{\neg(x = 0)?[p := p * x; x := x - 1]\}$.

we have

1. $\mathbf{N} \Vdash \{x = x_0\} P_1 \{p=x_0!\}$, and
2. $\mathbf{N} \Vdash \{x = x_0\} P_1 \{p=x_0!\}!$ since P_1 will always terminate for all values of x . Further,
3. $\mathbf{N} \Vdash \{x = x_0\} P_1 \{x_0 \geq 0 \rightarrow p=x_0!\}$ and
4. $\mathbf{N} \Vdash \{x = x_0\} P_1 \{x_0 \geq 0 \rightarrow p=x_0!\}!$ since $x_0 \geq 0$ always holds.

Examples: Factorial 1'

Now consider instead

Example 37.2

$\Sigma \supseteq \mathbb{Z} \cup \{! : s \rightarrow s, +, -, * : s^2 \rightarrow s; =, > : s^2\}$ where

$$x! \stackrel{df}{=} \begin{cases} 1 & \text{if } x=0 \\ x * (x-1)! & \text{otherwise} \end{cases}$$

In this case $!$ is a partial function. Let $\mathbf{Z} = \langle \mathbb{Z}, \Sigma \rangle$.

But for the same program

$P_1 \stackrel{df}{=} p := 1; \{\neg(x = 0)?[p := p * x; x := x - 1]\}$ we have

1. $\mathbf{Z} \Vdash \{x = x_0\} P_1 \{x_0 \geq 0 \rightarrow p = x_0!\}$
2. However $\mathbf{Z} \not\Vdash \{x = x_0\} P_1 \{x_0 \geq 0 \rightarrow p = x_0!\}$ since P_1 will not terminate for negative values of x .

Examples: Factorial 2

Example 37.3

Let $P_2 \stackrel{df}{=} p := 1; \{x > 0? [p := p * x; x := x - 1]\}$ and

$\mathbf{Z} = \langle \mathbb{Z}, \Sigma \rangle$. Then

1. $\mathbf{Z} \Vdash \{x = x_0 \geq 0\} P \{p = x_0!\}$
2. $\mathbf{Z} \Vdash \{x = x_0 \geq 0\} P \{p = x_0!\}!$
3. $\mathbf{Z} \Vdash \{x = x_0\} P \{(x_0 \geq 0 \rightarrow p = x_0!) \wedge (x_0 < 0 \rightarrow p = 1)\}$
4. $\mathbf{Z} \Vdash \{x = x_0\} P \{(x_0 \geq 0 \rightarrow p = x_0!) \wedge (x_0 < 0 \rightarrow p = 1)\}!$

Exercise 37.3

1. For any program P defined on a signature Σ what do the following correctness formulae mean?

- (a) $\{\top\} P \{\top\}$
- (b) $\{\top\} P \{\perp\}$
- (c) $\{\perp\} P \{\top\}$
- (d) $\{\perp\} P \{\perp\}$
- (e) $\{\top\} P \{\top\}!$
- (f) $\{\top\} P \{\perp\}!$
- (g) $\{\perp\} P \{\top\}!$
- (h) $\{\perp\} P \{\perp\}!$

2. Which of the correctness formulae given in problem 1 are

- (a) always valid?
- (b) always unsatisfiable?

38. Verification of WHILE Programs

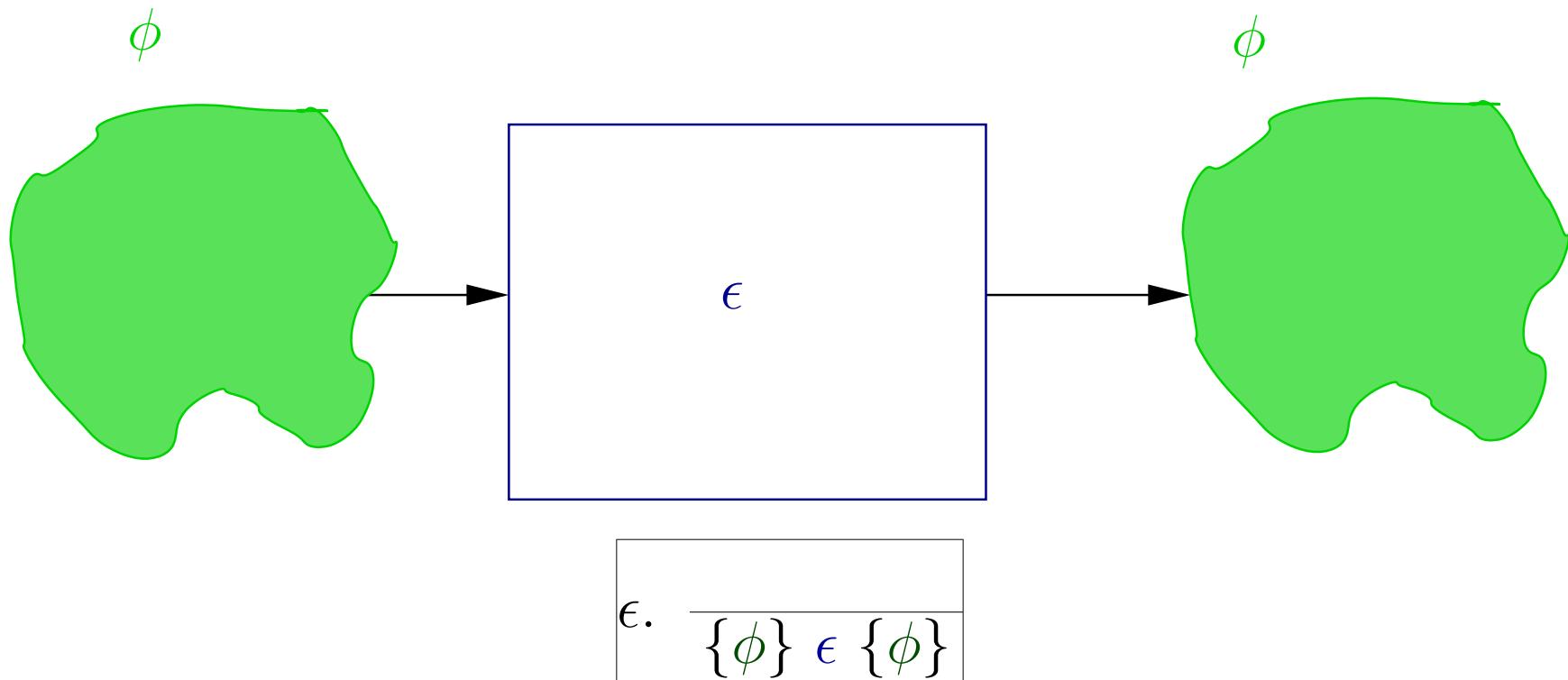
38: Verification of WHILE Programs

If debugging is the process of removing software bugs, then programming must be the process of putting them in.

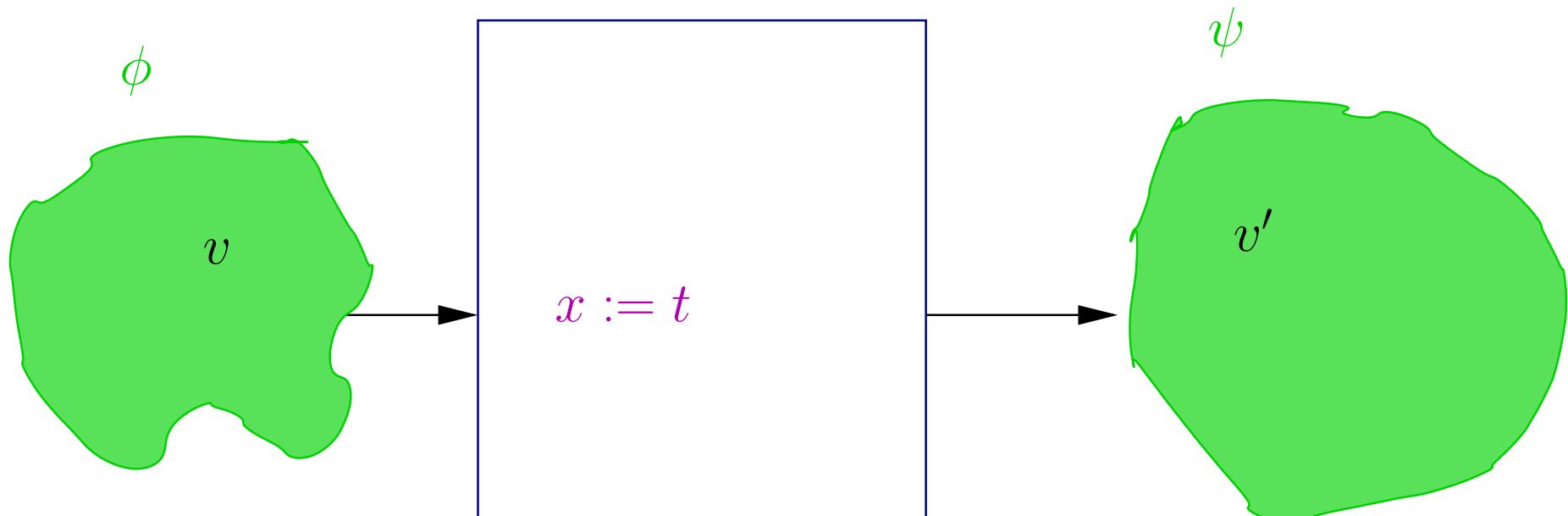
Edsger W. Dijkstra

1. Proof Rule: Epsilon
2. Proof Rule: Assignment
3. Proof Rule: Composition
4. Proof Rule: The Conditional
5. Proof Rule: The While Loop
6. The Consequence Rule
7. Proof Rules for Partial Correctness
8. Example: Factorial 1
9. Towards Total Correctness
10. Termination and Total Correctness
11. Example: Factorial 2
12. Notes on Example: Factorial
13. Example: Factorial 2 Made Complete
14. An Open Problem: Collatz

Proof Rule: Epsilon

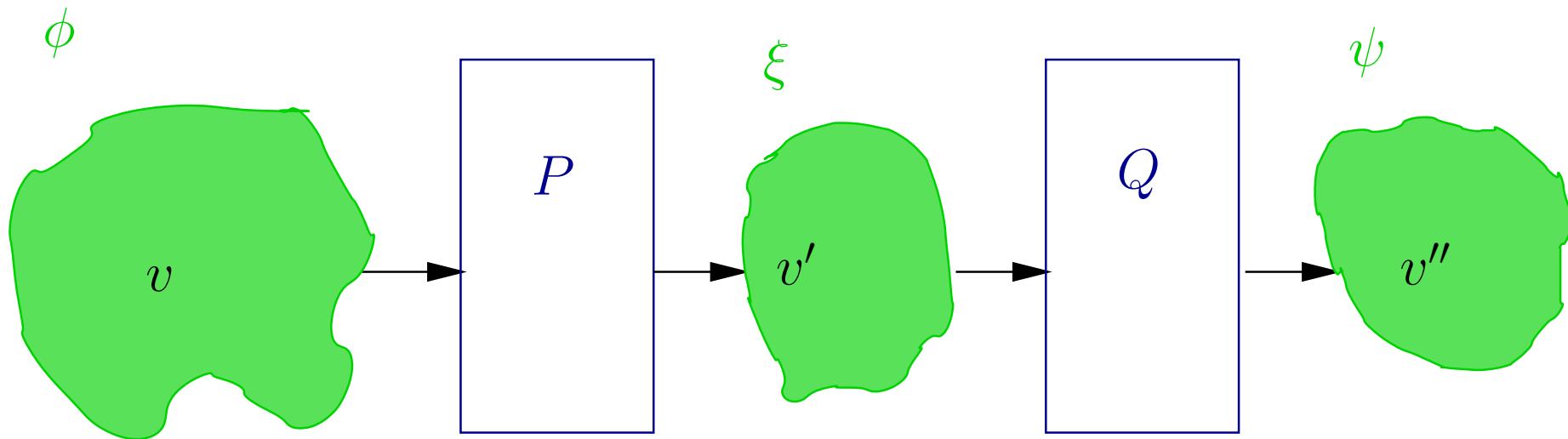


Proof Rule: Assignment



$$\text{:=} \ . \ \frac{}{\{\phi\} \ x := t \ \{\psi\}} \ (\phi \equiv \{t/x\}\psi)$$

Proof Rule: Composition



$$\frac{\{\phi\} \ P \ \{\xi\} \\ ; . \quad \{\xi\} \ Q \ \{\psi\}}{\{\phi\} \ P; Q \ \{\psi\}}$$

Swap Programs

Example 38.1: Swap1

Consider the following program which may be used to swap the values of a pair of variables x and y . It uses a temporary variable to “remember” one of the values which gets “over-written”.

$$\text{Swap1} \stackrel{\text{df}}{=} t := x; x := y; y := t$$

This program is not specific to any particular signature. Hence its correctness is independent of the signature and hence of the interpretation.

Proof of Swap1:0

We begin by specifying the precondition and the post-condition.

$\vdash \{x = x_0 \wedge y = y_0\}$

$t := x;$

$x := y;$

$y := t \quad \{x = y_0 \wedge y = x_0\}$

The **assignment rule** is most convenient when applied “backwards”.

Proof of Swap1:1

We begin by specifying the precondition and the post-condition.

$$\vdash \{x = x_0 \wedge y = y_0\}$$

$$t := x;$$

$$x := y; \quad \{x = y_0 \wedge t = x_0\}$$

$$y := t \quad \{x = y_0 \wedge y = x_0\}$$

The **assignment rule** is most convenient when applied “backwards”.

Proof of Swap1:2

We begin by specifying the precondition and the post-condition.

$$\begin{array}{ll} \vdash & \{x = x_0 \wedge y = y_0\} \\ t := x; & \{y = y_0 \wedge t = x_0\} \\ x := y; & \{x = y_0 \wedge t = x_0\} \\ y := t & \{x = y_0 \wedge y = x_0\} \end{array}$$

The **assignment rule** is most convenient when applied “backwards”.

Swap2

Example 38.2: swap2

However the following program specifically uses operations that are specific to a data type that allows for (operations akin to) addition and subtraction on the integers.

$$\text{Swap2} \stackrel{\text{df}}{=} u := u + v; v := u - v; u := u - v$$

Proof of Swap2:0

Again we specify the precondition and the postcondition. However we restrict our interpretation of the operations $+$ and $-$ to the corresponding operations on integers.

$\mathbf{Z} \Vdash \{u = u_0 \wedge v = v_0\}$

$u := u + v;$

$v := u - v;$

$u := u - v \quad \{u = v_0 \wedge v = u_0\}$

The **assignment rule** is most convenient when applied “backwards”.

Proof of Swap2:1

Again we specify the precondition and the postcondition. However we restrict our interpretation of the operations $+$ and $-$ to the corresponding operations on integers.

$$\mathbf{Z} \Vdash \{u = u_0 \wedge v = v_0\}$$

$$u := u + v;$$

$$v := u - v; \quad \{u - v = v_0 \wedge v = u_0\}$$

$$u := u - v \quad \{u = v_0 \wedge v = u_0\}$$

The **assignment rule** is most convenient when applied “backwards”.

Proof of Swap2:2

Again we specify the precondition and the postcondition. However we restrict our interpretation of the operations $+$ and $-$ to the corresponding operations on integers.

$$\begin{array}{ll} \mathbf{Z} \Vdash & \{u = u_0 \wedge v = v_0\} \\ u := u + v; & \{u - (u - v) = v_0 \wedge u - v = u_0\} \\ v := u - v; & \{u - v = v_0 \wedge v = u_0\} \\ u := u - v & \{u = v_0 \wedge v = u_0\} \end{array}$$

The **assignment rule** is most convenient when applied “backwards”.

Proof of Swap2:3

Again we specify the precondition and the postcondition. However we restrict our interpretation of the operations $+$ and $-$ to the corresponding operations on integers.

$$\begin{array}{ll} \mathbf{Z} \Vdash & \{u = u_0 \wedge v = v_0\} \\ & = \{(u + v) - ((u + v) - v) = v_0 \wedge (u + v) - v = u_0\} \\ u := u + v; & \{u - (u - v) = v_0 \wedge u - v = u_0\} \\ v := u - v; & \{u - v = v_0 \wedge v = u_0\} \\ u := u - v & \{u = v_0 \wedge v = u_0\} \end{array}$$

The assignment rule is most convenient when applied “backwards”.

Proof of Swap2:4

Again we specify the precondition and the postcondition. However we restrict our interpretation of the operations $+$ and $-$ to the corresponding operations on integers.

$$\begin{aligned} \mathbf{Z} \Vdash & \quad \{u = u_0 \wedge v = v_0\} \\ &= \{v = v_0 \wedge u = u_0\} \\ &= \{(u + v) - ((u + v) - v) = v_0 \wedge (u + v) - v = u_0\} \\ u := u + v; & \quad \{u - (u - v) = v_0 \wedge u - v = u_0\} \\ v := u - v; & \quad \{u - v = v_0 \wedge v = u_0\} \\ u := u - v & \quad \{u = v_0 \wedge v = u_0\} \end{aligned}$$

The **assignment rule** is most convenient when applied “backwards”.

Swap1 vs. Swap2

- The proof of Swap1 (example 38.1) required no properties particular to any signature. Hence we have

$$\Vdash \{x = x_0 \wedge y = y_0\} \text{Swap1}\{x = y_0 \wedge y = x_0\}$$

- The proof of Swap2 (example ??) clearly requires certain operations like $+$ and $-$ as part of the signature. These operations need to be total. Moreover

$$- \{v = v_0 \wedge u = u_0\} = \{(u + v) - ((u + v) - v) = v_0 \wedge (u + v) - v = u_0\}$$

would hold only if the two operations $+$ and $-$ provably satisfy properties such as

$$- (u + v) - v = u$$

$$- (u + v) - ((u + v) - v) = v$$

However the first equality $\{u = u_0 \wedge v = v_0\} = \{v = v_0 \wedge u = u_0\}$ is a purely logical one (commutativity of \wedge) and is independent of interpretation. Hence

$$\mathbf{Z} \Vdash \{x = x_0 \wedge y = y_0\} \text{Swap2}\{x = y_0 \wedge y = x_0\}$$

But

$$\nVdash \{x = x_0 \wedge y = y_0\} \text{Swap2}\{x = y_0 \wedge y = x_0\}$$

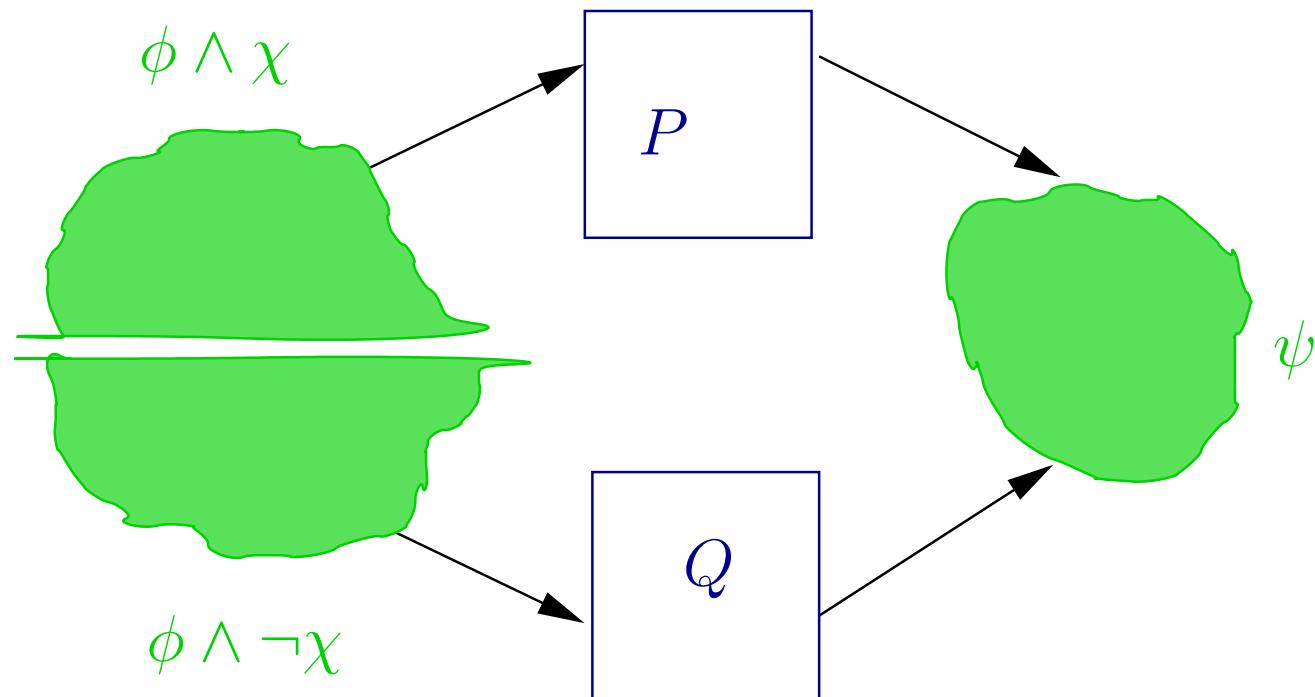
- Suppose we consider (finite-length) strings over a non-empty finite alphabet \mathbb{A} . Now consider the signature Σ consisting of the operations $+$ denoting concatenation of strings and $x - y = z$ if and only if $x = z + y$. Consider the structure

$\mathbf{A}^* = \langle \mathbb{A}^*, \{+, -, =\} \rangle$. It is easy to see that

$$\mathbf{A}^* \Vdash \{x = x_0 \wedge y = y_0\} Swap2 \{x = y_0 \wedge y = x_0\}$$

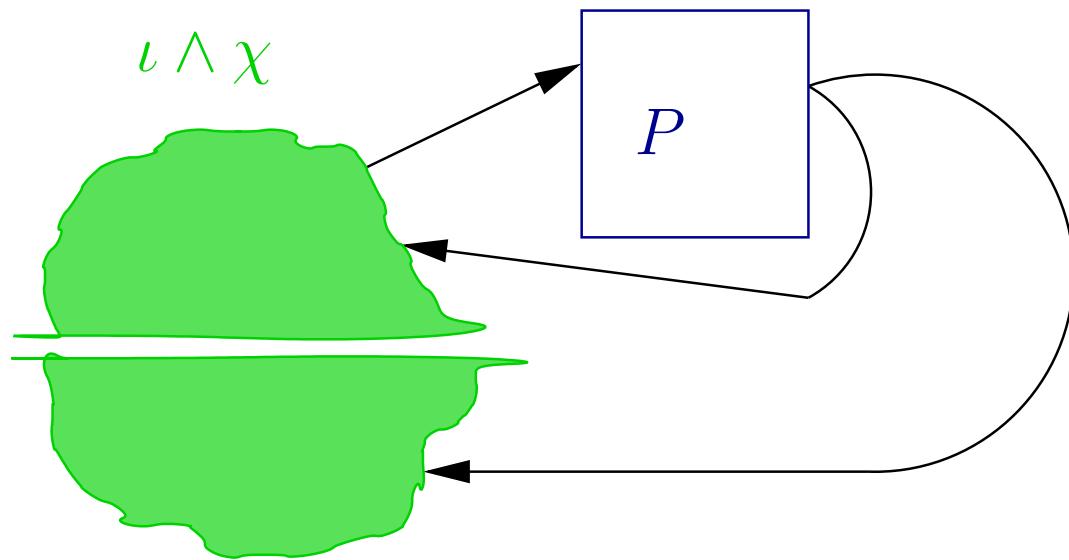
also holds.

Proof Rule: The Conditional



$$\frac{\begin{array}{c} \{\phi \wedge \chi\} P \{\psi\} \\ ? : . \quad \{\phi \wedge \neg\chi\} Q \{\psi\} \end{array}}{\{\phi\} \chi?P : Q \{\psi\}}$$

Proof Rule: The While Loop

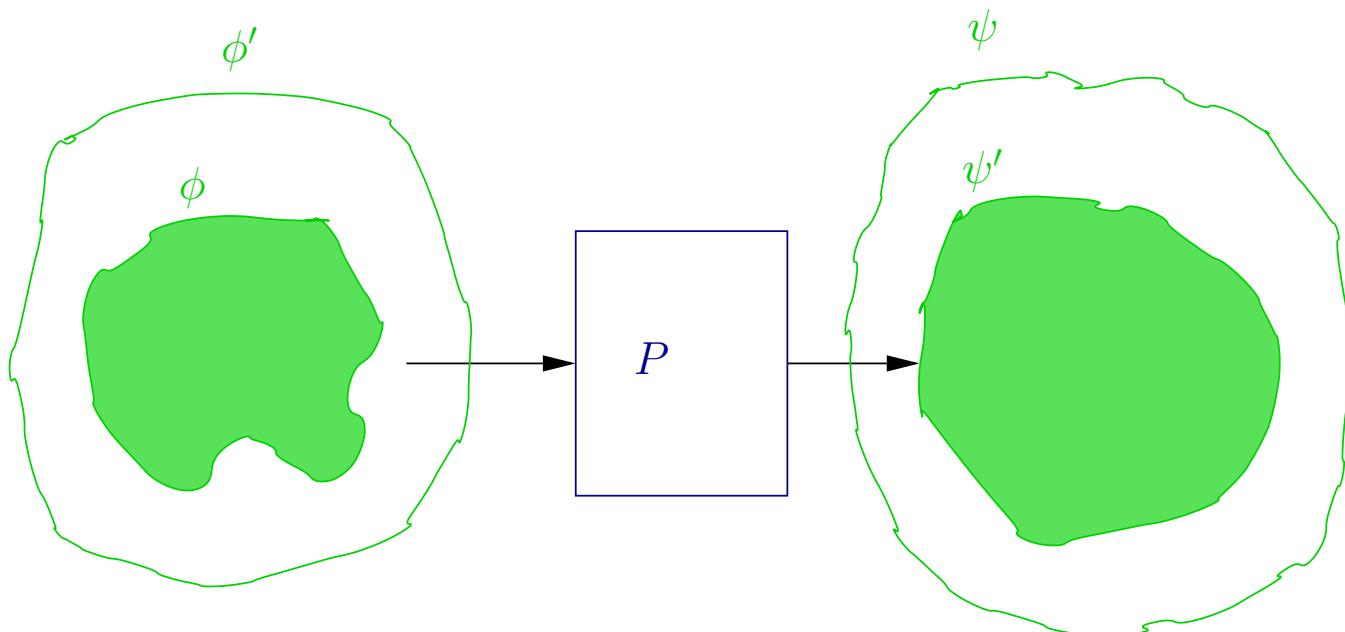


$\iota \wedge \neg\chi$

$$\{\} \cdot \frac{\{\iota \wedge \chi\} \ P \ \{\iota\}}{\{\iota\} \ \{\chi?P\} \ \{\iota \wedge \neg\chi\}}$$

ι in the while rule is called the **loop invariant**.

The Consequence Rule



$$\Rightarrow \cdot \frac{\begin{array}{c} \{\phi\} \subseteq \{\phi'\} \\ \{\phi'\} P \{\psi'\} \\ \{\psi'\} \subseteq \{\psi\} \end{array}}{\{\phi\} P \{\psi\}}$$

Proof Rules for Partial Correctness

$$\epsilon. \frac{}{\{\phi\} \in \{\phi\}}$$

$$:= . \frac{}{\{\{t/x\}\psi\} \ x := t \ \{\psi\}}$$

$$[] . \frac{\{\phi\} \ P \ \{\psi\}}{\{\phi\} [P] \ \{\psi\}}$$

$$\{\}. \frac{\{\iota \wedge \chi\} \ P \ \{\iota\}}{\{\iota\} \ \{\chi?P\} \ \{\iota \wedge \neg\chi\}}$$

$$; . \frac{\{\phi\} \ P \ \{\xi\} \quad \{\xi\} \ Q \ \{\psi\}}{\{\phi\} \ P; Q \ \{\psi\}}$$

$$? : . \frac{\{\phi \wedge \chi\} \ P \ \{\psi\} \quad \{\phi \wedge \neg\chi\} \ Q \ \{\psi\}}{\{\phi\} \ \chi?P : Q \ \{\psi\}}$$

$$\Rightarrow . \frac{\{\phi\} \subseteq \{\phi'\} \quad \{\phi'\} \ P \ \{\psi'\} \quad \{\psi'\} \subseteq \{\psi\}}{\{\phi\} \ P \ \{\psi\}}$$

the while rule is called the **loop invariant**.

ι in

Example: Factorial 1

$$\begin{aligned}
 & \{x = x_0\} \\
 p := 1; & \quad \{x = x_0 \wedge p = 1\} \\
 & \subseteq \{(x_0 = x) \wedge (x \geq 0 \rightarrow p * x! = x_0!)\} \\
 & \subseteq \{\phi_0 \wedge (x \geq 0 \rightarrow p * x! = x_0!)\} \equiv \{\iota\} \\
 \{\neg(x = 0)? & \quad \{\phi_0 \wedge (x \geq 0) \wedge (p * x! = x_0!) \wedge \neg(x = 0)\} \\
 & \subseteq \{\phi_0 \wedge (x > 0) \wedge (p * x! = x_0!)\} \\
 [p := p * x; & \quad \{\phi_0 \wedge (x > 0) \wedge (p * (x - 1)! = x_0!) \wedge\} \\
 x := x - 1] & \quad \{\phi_0 \wedge (x \geq 0) \wedge (p * x! = x_0!)\} \\
 & \subseteq \{\iota\} \\
 \} & \quad \{\iota \wedge (x = 0)\} \\
 & \subseteq \{\phi_0 \wedge (x = 0) \wedge (p * x! = x_0!)\} \\
 & \subseteq \{(x_0 \geq 0 \rightarrow p = x_0!)\}
 \end{aligned}$$

where $\phi_0 \equiv (x_0 \geq 0)$

Towards Total Correctness

1. The only construct which may not terminate is the while loop.
2. Termination of the while loop: Define a “measure” called the **bound function**, $\beta : \text{Dom}(\beta) \rightarrow W$ where
 - $\langle W, < \rangle$ is a well-ordered set (a set with no infinite descending sequences $w > w' > w'' > \dots$) with a least element 0 such that $w < 0$ does not hold true for any $w \in W$.
 - $\text{Dom}(\beta)$ is the tuple of possible values of the program variables (\vec{v}).
 - Often $\langle W, < \rangle = \langle \mathbb{N}, < \rangle$
 - $\iota \wedge \chi \Rightarrow \beta(\vec{v}) > 0$ and $\beta(\vec{v}) = 0 \Rightarrow \iota \wedge \neg \chi$
 - Each execution of the body of the loop decreases the value of the bound function.

Termination and Total Correctness

$$\epsilon! \quad \frac{}{\{\phi\} \in \{\phi\}!}$$

$$:=! \quad \frac{}{\{\{t/x\}\psi\} \ x := t \ \{\psi\}!}$$

$$[]! \quad \frac{\{\phi\} \ P \ \{\phi\}!}{\{\phi\} [P] \ \{\phi\}!}$$

$$;! \quad \frac{\{\phi\} \ P \ \{\xi\}! \quad \{\xi\} \ Q \ \{\psi\}!}{\{\phi\} \ P; Q \ \{\psi\}!}$$

$$? :: \frac{\{\phi \wedge \chi\} \ P \ \{\psi\}! \quad \{\phi \wedge \neg \chi\} \ Q \ \{\psi\}!}{\{\phi\} \ \chi?P : Q \ \{\psi\}!}$$

$$\Rightarrow ! \quad \frac{\{\phi\} \subseteq \{\phi'\} \quad \{\phi'\} \ P \ \{\psi'\}! \quad \{\psi'\} \subseteq \{\psi\}}{\{\phi\} \ P \ \{\psi\}!}$$

$$\{\}! \quad \frac{\{\iota \wedge \chi \wedge \beta(\vec{v}) = b_0 > 0\} \ P \ \{\iota \wedge \beta(\vec{v}) < b_0\}!}{\{\iota\} \ \{\chi?P\} \ \{\iota \wedge \neg \chi\}!}$$

where

- β is a bound function and
- ι is the loop invariant.

c.f. Proof Rules for Partial Correctness

Example: Factorial 2

$$\begin{aligned}
 & \{x = x_0\} \\
 p := 1; & \quad \{x = x_0 \wedge p = 1\} \\
 & \subseteq \{(x_0 = x) \wedge (x \geq 0 \rightarrow p * x! = x_0!) \wedge (x_0 < 0 \rightarrow p = 1)\} \\
 & \subseteq \{\phi_0 \wedge (x \geq 0 \rightarrow p * x! = x_0!)\} = \{\iota\} \\
 \{x > 0? & \quad \{\phi_0 \wedge (x > 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x = \beta_0 > 0\} \\
 [p := p * x; & \quad \{\phi_0 \wedge (x > 0) \wedge (p * (x - 1)! = x_0!) \wedge \\
 & \quad \beta(x, p) = x = \beta_0 > 0\} \\
 x := x - 1] & \quad \{\phi_0 \wedge (x \geq 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x < \beta_0\}] \\
 & \subseteq \{\iota\} \\
 \} & \quad \{\phi_0 \wedge (x = 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x = 0\}! \\
 & \subseteq \{(x_0 \geq 0 \rightarrow p = x_0!) \wedge (x_0 < 0 \rightarrow p = 1) \wedge x \leq 0\} = \{\iota \wedge x \leq 0\}
 \end{aligned}$$

where $\phi_0 = (x_0 \geq 0) \wedge (x_0 < 0 \rightarrow p = 1)$

c.f. Partial correctness proof

Notes on Example: Factorial

- The loop invariant ι is a conjunction of
 - ϕ_0 whose truth is trivially unaffected by the changes in state produced by the loop body, and
 - the formula $(x \geq 0 \rightarrow p * x! = x_0!)$ which
 - * *holds initially* before control enters the loop,
 - * *holds after* the condition has been checked,
 - * *fails to hold* after the first command of the body has been executed,
 - * *is restored* at the end of the loop body, and
 - * *holds after exiting* the loop
- Notice the progress of the **bound function** which is completely internal to the working of the loop and is never part of the specification of the program.

Example: Factorial 2 Made Complete

A more complete proof including the use of the assignment clearly delineated is given below.

$$\begin{aligned}
 & \{x = x_0\} = \{\phi_0\} \\
 & \subseteq \{x = x_0 \wedge 1 = 1\} = \{\{1/p\}\phi_1\} \\
 p := 1; & \quad \{x = x_0 \wedge p = 1\} = \{\phi_1\} \\
 & \subseteq \{(x_0 = x) \wedge (x \geq 0 \rightarrow p * x! = x_0!) \wedge (x_0 < 0 \rightarrow p = 1)\} \\
 & \subseteq \{\phi_0 \wedge (x \geq 0 \rightarrow p * x! = x_0!)\} = \{\iota\} \\
 \{x > 0? & \quad \{\phi_0 \wedge (x > 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x = \beta_0 > 0\} = \{\phi'_1\} \\
 & \subseteq \{\{p * x/p\}\psi_1\} \\
 [p := p * x; & \quad \{\phi_0 \wedge (x > 0) \wedge (p * (x - 1)! = x_0!) \wedge \beta(x, p) = x = \beta_0 > 0\} = \{\psi_1\} \\
 & \subseteq \{\{x - 1/x\}\psi_2\} \\
 x := x - 1 & \quad \{\phi_0 \wedge (x \geq 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x < \beta_0 > 0\} = \{\psi_2\} \\
] & \quad [\psi_2] \\
 & \subseteq \{\iota\} \\
 \} & \quad \{\iota \wedge \neg(x > 0) \wedge \beta(x, p) = x = 0\}! \\
 & \subseteq \{\phi_0 \wedge (x = 0) \wedge (p * x! = x_0!) \wedge \beta(x, p) = x = 0\}! \\
 & \subseteq \{(x_0 \geq 0 \rightarrow p = x_0!) \wedge (x_0 < 0 \rightarrow p = 1) \wedge x \leq 0\} = \{\iota \wedge x \leq 0\}
 \end{aligned}$$

An Open Problem: Collatz

Consider the following specification where Σ includes the quotient (denoted $\circ/$) and remainder (denoted $/\circ$) functions on positive integers.

$$\mathbf{Z} \Vdash \{x > 0\} \text{ Collatz } \{x = 1\}!$$

where

$$\text{Collatz} \stackrel{df}{=} \{x > 1? [x / \circ 2 = 0? x := x \circ/ 2 : x := (3.x + 1) \circ/ 2]\}$$

Example 38.3: GCD

As a simple example we write an annotated program to compute the greatest common divisor of two natural numbers (at least one of which is non-zero). Our signature is that of the natural numbers extended to include the remainder operator /_o . We implement Euclid's algorithm which relies on the following facts about first order number theory which we take for granted.

1. $\neg(b = 0) \rightarrow \text{gcd}(0, b) = b$
2. $\neg(a = 0) \rightarrow \text{gcd}(a, b) = \text{gcd}(b \text{/_o} a, a)$

Consequently we have the loop invariant

$$\iota \stackrel{\text{df}}{=} \text{gcd}(a_0, b_0) = \text{gcd}(a, b) \quad (79)$$

The bound function $\beta = a$ decreases in the loop because either $b = 0$ or because by the division algorithm on non-zero numbers $0 \leq b \text{/_o} a < a$.

$$\begin{aligned}
 GCD(a, b) &\stackrel{df}{=} \{a = a_0 \wedge b = b_0 \wedge \neg(a = 0 \wedge b_0 = 0) \wedge \iota\} \\
 &\subseteq \{\iota\} \\
 \{(a > 0)? &\quad \{\iota \wedge \beta = a = \beta_0 > 0\} \\
 c := a; &\quad \{gcd(a_0, b_0) = gcd(b /_o a, c)\} \\
 a := b /_o a; &\quad \{gcd(a_0, b_0) = gcd(a, c)\} \\
 b := c &\quad \{\iota \wedge \beta = a < \beta_0\} \\
 \}; &\quad \{\iota \wedge \beta = a = 0\} \\
 &\subseteq \{b = gcd(a_0, b_0)\} \\
 g := b &\quad \{g = gcd(a_0, b_0)\}
 \end{aligned}$$

Example 38.4: k-th root

For any two integers $n, k > 1$ write an $O(\log_2 n)$ program along with its proof of correctness in Hoare Logic, which computes the number $\lfloor \sqrt[k]{n} \rfloor$. You may assume that there exists a predefined binary operator ' k ', which computes m^k for any integers m and $k > 0$.

We observe that $m = \lfloor \sqrt[k]{n} \rfloor$ really stands for the predicate $m^k \leq n < (m+1)^k$. We use this as the post-condition. Effectively it requires determining the closed interval $[m, m + 1]$ within which the k -th root of n lies.

The invariant is simply obtained from the post-condition by weakening it, which effectively means extending the post-condition to a larger interval $[l, u]$ and bounding the values of l and u to ensure that the interval does not expand arbitrarily. Hence we have the constraint $1 \leq l < u \leq n$ from which the bound function is obtained as either the length of the interval

$[l, u]$ which is bounded below by 2 or equivalently the measure $\beta = u - l$ which will be bounded below by 1.

The algorithmic strategy employs binary search starting with the initial interval $[1, n]$ which satisfies the invariant property trivially and proceeds to halve the interval in each iteration.

$Root(n, k)$	$\stackrel{df}{=} \{n > 1 \wedge k > 1\}$
$l := 1; u := n;$	$\subseteq \{u = n > 1 = l \wedge k > 1\}$
	$\subseteq \{\iota \equiv (1 \leq l < u \leq n) \wedge (l^k \leq n < u^k)\} \cap \{\beta : u - l \geq 1\}$
$\{u > l + 1?$	$\{u \wedge u > l + 1\}$
	$\subseteq \{(1 \leq l < l + 1 < u \leq n) \wedge (l^k \leq n < u^k)\} \cap \{\beta_0 : u - l > 1\}$
$[m := (l + u) \text{ div } 2;$	$\{(1 \leq l < l + 1 < u \leq n) \wedge (l^k \leq n < u^k) \wedge l \leq m < u\}$
$m^k > n?$	$\{(1 \leq l < l + 1 < u \leq n) \wedge (l^k \leq n < m^k < u^k) \wedge l \leq m < u\}$
	$\subseteq \{(1 \leq l < m \leq n) \wedge (l^k \leq n < m^k)\}$
$u := m$	$\{(1 \leq l < u \leq n) \wedge (l^k \leq n < u^k) \equiv \iota\} \cap \{\beta : u - l = m - l < \beta_0\}$
:	$\{\iota \wedge m^k \leq n\}$
	$\subseteq \{(1 \leq m < u \leq n) \wedge (m^k \leq n < u^k)\}$
$l := m$	$\{(1 \leq l < u \leq n) \wedge (l^k \leq n < u^k) \equiv \iota\} \cap \{\beta : u - l = u - m < \beta_0\}$
$]$	$\{\iota\} \cap \{\beta < \beta_0\}$
$\}$	$\{\iota \wedge u \leq l + 1\}!$
	$\subseteq \{(1 \leq l < u \leq n) \wedge u \leq l + 1 \wedge (l^k \leq n < u^k)\} \cap \{\beta : u - l = 1\}$
	$\subseteq \{(1 \leq l < u = l + 1 \leq n) \wedge (l^k \leq n < u^k)\} \cap \{\beta : u - l = 1\}$
	$\subseteq \{(1 \leq l < u = l + 1 \leq n) \wedge (l^k \leq n < (l + 1)^k)\} \cap \{\beta : u - l = 1\}$
	$\subseteq \{(1 \leq l < n) \wedge l = \lfloor \sqrt[k]{n} \rfloor\} \cap \{\beta : u - l = 1\}$

38.1. Nested Loops

Consider an array A of size $n > 1$ containing some values (drawn from some unknown structure with a pre-defined equality predicate “ $=$ ” on the values). We write an annotated (proven) program in the language $\mathcal{WH}(\Sigma)$ which for any k , $1 \leq k < n - 1$, “left-shifts-and-rotates” the elements of the array A by k positions. For instance, if $\langle a_0, a_1, \dots, a_{n-1} \rangle = \langle a_i \mid 0 \leq i < n, n > 1 \rangle$ is the sequence of values representing the initial array then for $k = 2$, the left-shift-and-rotate yields the array with the value sequence $\langle a_2, a_3, \dots, a_{n-1}, a_0, a_1 \rangle = \langle a_{(i+2) \% n} \mid 0 \leq i < n, n > 1 \rangle$. To keep the problem interesting we stipulate that no extra array may be used and that only a small number of fixed variables of the type of elements of the array are permitted to be used in the program.

This problem may be considered to be a generalization of the swap problem (example 38.1). For the values $n = 2$ and $k = 1$, this problem reduces to swapping the values of the two array elements, without using more than one simple variable.

In general for any value of k , $0 < k < n - 1$, the specification of the program we require is given by

$$\begin{aligned} & \{n > 1 \wedge 0 < k < n - 1 \wedge \forall i [0 \leq i < n \rightarrow A[i] = a_i]\} \\ & \quad LSR(A, k) \\ & \{ \forall i [0 \leq i < n \rightarrow A[i] = a_{(i+k) \% n}] \} \end{aligned}$$

We first solve the problem for the case $k = 1$ and iterate the solution for larger values of k . The result is a solution which requires nk movements of array elements. We then propose a more efficient solution which solves the problem in n

movements, but is more complicated to reason about.

Example 38.5: Left-shift-rotate-1

When $k = 1$ the initial sequence of array values $\langle a_0, a_1, a_2, \dots, a_{n-1} \rangle$ becomes $\langle a_1, a_2, \dots, a_{n-1}, a_0 \rangle$. Notice that if $n = 1$ then no left-shift would occur. For such a movement to be possible the array must have at least 2 elements. Hence $n > 2^a$. The specification of the program which we call $LSR1(A)$ for this problem may be specified as follows.

$$\begin{aligned} &\{n > 1 \wedge \forall i[0 \leq i < n \rightarrow A[i] = a_i]\} \\ &LSR1(A) \\ &\{\forall i[0 \leq i < n \rightarrow A[i] = a_{(i+1) \% n}]\}! \end{aligned} \tag{80}$$

The annotated program which implements this is given below. The loop invariant is

$$\iota \stackrel{df}{=} t = a_0 \wedge 0 \leq j < n - 1 \wedge n > 1 \wedge \forall i[0 \leq i < j < n \rightarrow A[i] = a_{i+1}]$$

and the bound function is $\beta = n - 1 - j$.

^aThis problem may be thought of as generalising the problem of swapping the values of two variables, using a temporary variable to “remember” the value of one of the variables while the value of the other variable is copied into it. A simple solution “remembers” the value a_0 while each of the other elements in the indices a_1 to a_{n-1} are copied in order to the indices 0 to $n - 2$ respectively

$$\begin{aligned}
 LSR1(A) &\stackrel{df}{=} \{n > 1 \wedge \forall i[0 \leq i < n \rightarrow A[i] = a_i]\} \\
 j := 0; &\quad \{j = 0 \wedge n > 1 \wedge \forall i[0 \leq i < n \rightarrow A[i] = a_i]\} \\
 t := A[0]; &\quad \{t = a_0 \wedge j = 0 \wedge n > 1 \wedge \forall i[0 \leq i < n \rightarrow A[i] = a_i]\} \\
 &\subseteq \{\iota\} \\
 \{j < n - 1? &\quad \{\iota \wedge j < n - 1 \wedge \beta = n - 1 - j = \beta_0 > 0\} \\
 A[j] := A[j + 1]; &\quad \{\iota \wedge j < n - 1 \wedge A[j] = a_{j+1} \wedge \beta = \beta_0 = n - 1 - j > 0\} \\
 j := j + 1 &\quad \{\iota \wedge j \leq n - 1 \wedge \beta = n - 1 - j < \beta_0\}! \\
 \}; &\quad \{\iota \wedge j = n - 1 \wedge \beta = 0\} \\
 A[j] := t &\quad \{\iota \wedge j = n - 1 \wedge A[n - 1] = a_0\} \\
 &\subseteq \{A[n - 1] = a_0 \wedge \forall i[0 \leq i < n - 1 \rightarrow A[i] = a_{i+1}]\} \\
 &= \{\forall i[0 \leq i < n \rightarrow A[i] = a_{(i+1) \% n}]\}
 \end{aligned}$$

Example 38.6: Left-shift-rotate-k

We may use the program segment $LSR1(A)$ along with its proof to develop the larger program $LSR(A, k)$ which left-shifts-and-rotates the array elements by k positions for $0 < k < n - 1$. To avoid name clashes between the two programs and their specification we rewrite the new specification as

$$\begin{aligned}
 & \{\textcolor{violet}{n} > 1 \wedge 0 < k = k_0 < n - 1 \wedge \forall i[0 \leq i < \textcolor{violet}{n} \rightarrow B[i] = b_i]\} \\
 & \quad LSR(B, k) \\
 & \quad \{\forall i[0 \leq i < \textcolor{violet}{n} \rightarrow B[i] = b_{(i+k) \textcolor{brown}{\circ} \textcolor{violet}{n}}]\}!
 \end{aligned} \tag{81}$$

For the sake of brevity with clarity we define the function $lsr(\textcolor{violet}{B}, m)$ by induction as follows

$$\begin{aligned}
 lsr(\textcolor{violet}{B}, 0) & \stackrel{df}{=} B \\
 lsr(\textcolor{violet}{B}, m + 1) & \stackrel{df}{=} lsr(lsr(\textcolor{violet}{B}, m), 1) = LSR1(lsr(\textcolor{violet}{B}, m))
 \end{aligned} \tag{82}$$

It is now clear that the specifications (80) and (81) may be rewritten in terms of the function lsr as follows.

$$\{\textcolor{violet}{n} > 1 \wedge A = A_0\} \quad LSR1(A) \quad \{\textcolor{violet}{n} > 1 \wedge A = lsr(A_0, 1)\}! \tag{83}$$

$$\{\textcolor{violet}{n} > 1 \wedge 0 < k = k_0 < n - 1 \wedge B = B_0\} \quad LSR(B, k) \quad \{\textcolor{violet}{n} > 1 \wedge B = lsr(B_0, k_0)\}! \tag{84}$$

The loop invariant for the following program is

$$\iota \stackrel{df}{=} \textcolor{violet}{n} > 1 \wedge 0 \leq k \leq k_0 < n - 1 \wedge B = lsr(B_0, k_0 - k)$$

$$\begin{aligned}
 LSR(B, k) &\stackrel{df}{=} \{n > 1 \wedge 0 < k = k_0 < n - 1 \wedge B_0 = B = lsr(B_0, 0)\} \\
 &\subseteq \{\iota \wedge \beta = k = \beta_0\} \\
 \{k > 0? &\quad \{\iota \wedge \beta = k > 0\} \\
 LSR1(B); &\quad \{n > 1 \wedge 0 \leq k \leq k_0 < n - 1 \wedge B = lsr(lsr(B_0, k_0 - k), 1)\} \\
 &= \{n > 1 \wedge 0 \leq k \leq k_0 < n - 1 \wedge B = lsr(B_0, k_0 - (k - 1))\} \\
 k := k - 1 &\quad \{\iota \wedge \beta = k < \beta_0\} \\
 \} &\quad \{\iota \wedge \beta = k = 0\}! \\
 &\subseteq \{n > 1 \wedge B = lsr(B_0, k_0)\}
 \end{aligned}$$

Exercise 38.1

- Even a cursory look at the program $LSR(B, k)$ reveals that there are k invocations of program $LSR1(B)$ and in each invocation $n + 1$ array elements are assigned to either other array elements or to a temporary variable. Hence $LSR(B, k)$ performs $(n + 1).k$ movements of array values. But a look at the specification (81) reveals that it may be possible to directly copy the array element $B[(i + k) /_o n]$ into $B[i]$ for each index i , thus pruning down the number of moves in the program by a factor close to k . Develop this program along with its proof of total correctness.

39. Many-sorted FOL

39: Many-sorted FOL

1. Sortedness
2. Many-Sorted Logic: Symbols
3. Many-Sorted Signatures
4. Many-Sorted Signature: Terms
5. Many-Sorted Predicate Logic
6. Reductions: Guardedness
7. Reductions: Bounded Quantification

Sortedness

- A treatment that works mainly with a 1-sorted term algebra often does not address the interesting problems of a mathematical theory e.g. the first-order theory of directed or undirected graphs.
- To begin to address even the simplest problems of graph theory requires counting and the power of the first order theory of numbers.
- The problems of second-order logic (quantification over first order properties) may be expressed in many-sorted first order logic by allowing the power set of a set to be included in the universe of discourse of a many-sorted first order logic.

Many-Sorted Logic: Symbols

We extend **1-sorted signatures** to the many-sorted case (most programming languages are many-sorted).

Definition 39.1: Sorts

Given a finite nonempty set S of sorts with $S = \{\textcolor{blue}{s}_i \mid 1 \leq i \leq k\}$, a k -sorted logic consists of

- $V = \bigcup_{1 \leq i \leq k} V_i$: A countable union of pair-wise disjoint countable sets V_i of **variables** of sort $\textcolor{blue}{s}_i$ for each $s_i \in S$.
- F : a countable collection of **function symbols**; $f, g, h, \dots \in F$.
- A : a countable collection of **atomic predicate symbols**; $p, q, r, \dots \in A$ and
- Quantifier symbols \forall_i and \exists_i for each sort $s_i \in S$.

Many-Sorted Signatures

Definition 39.2: Many-sorted signature

Given a finite nonempty set S of sorts with $S = \{s_i \mid 1 \leq i \leq k\}$, a S -sorted signature Σ consists of a set of strings of the form

- $f : s_{i_1}, s_{i_2}, \dots, s_{i_m} \longrightarrow s_{i_0}$, $m \geq 0$ if f represents a total function and $f : s_{i_1}, s_{i_2}, \dots, s_{i_m} \rightharpoonup s_{i_0}$, $m \geq 0$ if it is partial,
- $p : s_{i_1}, s_{i_2}, \dots, s_{i_n}$, $n \geq 0$ such that there is exactly one string for each $f \in F$ and each $p \in A^a$.
- A binary equality relation $=_i : s_i^2$ for some of the sorts $s_i \in S$.

^aThese strings refer to the “type” or “sort” of the function or predicate.

If f represents a partial function then predicates involving it may have to be **guarded** by making them conditional.

Many-Sorted Signature: Terms

Definition 39.3: Many-sorted Terms

Given a S -sorted signature Σ , the set $\mathbb{T}_\Sigma(V) = \biguplus_{1 \leq i \leq k} \mathbb{T}_\Sigma^i(V_i)$ of Σ -terms is the disjoint union of the sets of terms $\mathbb{T}_\Sigma^i(V)$ defined inductively such that every term is assigned a sort from S .

$$s, t, u ::= x_i \in V_i \mid f(t_1, \dots, t_m)$$

where $f : s_{i_1}, s_{i_2}, \dots, s_{i_m} \longrightarrow s_{i_0} \in \Sigma$,

- each variable $x_i \in V_i$ is a term in $\mathbb{T}_\Sigma^i(V)$, a fact usually denoted $x_i : s_i$,
- $f(t_1, \dots, t_m) \in \mathbb{T}_\Sigma^{i_0}(V)$, a fact usually denoted $f(t_1, \dots, t_m) : s_{i_0}$.

Many-Sorted Predicate Logic

- The syntax of the logic is the obvious extension to the one defined as before.
- The semantics requires a S -sorted structure with a non-empty domain \mathbb{A}_i for each sort.
- The semantics is then a minor extension of the semantics for the 1-sorted case
- Second-Order logic may simply be considered a 2-sorted logic with predicates parameterised over both individuals and sets of individuals.

Reductions: Guardedness

(see also [guardedness for partial functions](#))

- An S -sorted Predicate Logic may be reduced to a 1-sorted Predicate logic by introducing a fresh set of unary postfix predicates : s_i (one for each sort s_i)
- Every quantified formula of the form $\forall_i x_i[\phi]$ is replaced by the 1-sorted formula $\forall x[(x : s_i) \rightarrow \phi]$ and recursively for each quantifier.
- Every quantified formula of the form $\exists_i x_i[\phi]$ is replaced by the 1-sorted formula $\exists x[(x : s_i) \wedge \phi]$ and recursively for each quantifier,

Reductions: Bounded Quantification

Quantified formulae are often abbreviated as follows:

$$\begin{array}{ll} \forall x : s_i[\phi] & \text{for } \forall x[(x : s_i) \rightarrow \phi] \\ \exists x : s_i[\phi] & \text{for } \exists x[(x : s_i) \wedge \phi] \end{array}$$

Notice that

$$\begin{array}{l} \neg \forall x : s_i[\phi] \Leftrightarrow \exists x : s_i[\neg \phi] \\ \neg \exists x : s_i[\phi] \Leftrightarrow \forall x : s_i[\neg \phi] \end{array}$$

40. Abstract Data Types

40: Abstract Data Types

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

C. A. R. Hoare

1. Pairs
2. The Array Structure: Signature
3. The Array Structure: Typing Axioms
4. The Array Structure: Access and Update Axioms
5. The List Structure: Signature
6. The List Structure Axioms: 1
7. The List Structure Axioms: 2
8. The List Structure: Induction
9. The List Structure: Acyclicity 1
10. The List Structure: Acyclicity 2
11. The Stack Structure: Signature
12. The Stack Structure Axioms: 1
13. The Stack Structure Axioms: 2
14. The Stack Structure: Induction
15. The Queue Structure: Signature
16. The Queue Structure Axioms: 1
17. The Queue Structure Axioms: 2
18. The Queue Structure: Induction

- 19. Binary Trees: Signature
- 20. The Binary Tree Axioms: 1
- 21. The Binary Tree Axioms: 2
- 22. The Binary Tree Axioms: 3
- 23. The Binary Tree Induction Rule

40.1. Introduction: Compound Data Structures

We have so far considered only the first-order theory of elementary data such as numbers. In each case the data have a predefined equality relation. In this section we consider the first-order theory (with equality) of compound data structures encountered in computer science books on programming, data structures and algorithms.

Compound data structures such as strings, lists and trees are built using some more elementary data such as numbers and characters. However it is also possible to construct compound data structures over other compound data structures. For example, we could construct lists of trees of numbers, lists of strings, or even trees over lists of strings, trees over trees, lists of lists etc. There is literally no limit to such complex constructions. But in each case we would require to consider at least two sorts. Hence lists of numbers would take the numbers as a parameter. Similarly lists of strings would take the strings over an alphabet as a parameter.

Many compound data structures may be built on more than one more elementary data structure. This is especially true of many user-defined data structures such as tuples, records and variant records. In each of these cases there exists a first-order theory (with equality) which depends upon the nature of construction of these structures. The equality relations on the individual components induces a naturally defined equality relation (component-wise equality) on the compound structure too.

Since record structures are specifically user-defined and not particularly interesting structures, we will not describe them in great detail. They are not interesting, mainly because they allow for just constructors to build them and deconstructors (or projection functions) which allow us to extract the individual components. The following example of tuples illustrates what we mean by the above.

Example 40.1: Pairs

Let \mathbb{A} and \mathbb{B} be carrier sets of \mathbf{A} and \mathbf{B} , whose signatures are Σ_1 and Σ_2 respectively. Further let $=_1$ and $=_2$ be respective equality relations in Σ_1 and Σ_2 respectively. $\mathbb{A} \times \mathbb{B}$ is the carrier set of the cartesian product of the two sets i.e. $\mathbb{A} \times \mathbb{B} = \{(a, b) \mid a \in \mathbb{A}, b \in \mathbb{B}\}$. Think of $(\ , \)$ as a “mixfix” constructor which constructs ordered pairs of elements belonging to the two sets respectively. We define the projection functions $\#1$ and $\#2$, to extract the components of any ordered pair $(a, b) \in \mathbb{A} \times \mathbb{B}$. That is, $\#1(a, b) = a$ and $\#2(a, b) = b$. These projection functions are the deconstructors of this pairing data structure. If s_1 denotes the sort of \mathbf{A} and s_2 denotes the sort of \mathbf{B} then we have the structure $\mathbf{Pairs}[\mathbf{A}, \mathbf{B}] = \langle \mathbb{A} \times \mathbb{B}; (\ , \), \#1, \#2; =_{12} \rangle$, where the signature of $\mathbf{Pairs}[\mathbf{A}, \mathbf{B}]$ is given by

$$\begin{aligned} (\ , \) &: s_1, s_2 \longrightarrow (s_1, s_2) \\ \#1 &: (s_1, s_2) \longrightarrow s_1 \\ \#2 &: (s_1, s_2) \longrightarrow s_1 \\ =_{12} &: (s_1, s_2)^2 \end{aligned}$$

(s_1, s_2) is the sort of the pairs whose components have sorts s_1 and s_2 respectively and $=_{12}$ is the component-wise equality predicate induced by the equality predicates $=_1$ and $=_2$ respectively.

In the absence of any knowledge of the properties of the individual component structures, the following axiom illustrates a common thread in the axiomatization of every complex data structure.

$$\phi_{Pairs} \stackrel{df}{=} \forall x: (s_1, s_2)[x =_{12} (\#1 x, \#2 x)]$$

i.e. any pair x may be re-constructed by applying the constructor $(\ , \)$ to the pair of elements obtained by applying the deconstructors $\#1$ and $\#2$ to the pair respectively.

Besides the above axiom, the equality of pairs would depend on having equality as a predicate in the signatures Σ_1 and Σ_2 as well. We assume that $=$ is available in both Σ_1 and Σ_2 . Of course, the three equality relations all operate on different signatures (and hence structures).

We disambiguate the three equality relations as follows. Assume $=_1 : s_1^2$, $=_2 : s_2^2$ and $=_{12} : (s_1, s_2)^2$ are the respective equality predicates. The second axiom lifts equality on the individual components to equality on pairs.

$$\phi_{Pairs-equality} \stackrel{df}{=} \forall \mathbf{x}, \mathbf{y} : (s_1, s_2) [\mathbf{x} =_{12} \mathbf{y} \leftrightarrow (\#1 \ x =_1 \ #1 \ y \wedge \#2 \ x =_2 \ #2 \ y)]$$

3-sorted logic of pairs. In a first-order logic of pairing, with the signature $\Sigma_1 \cup \Sigma_2 \cup \Sigma$ we would require that every bound variable used in a predicate be “guarded” by its sort membership predicate.

In summary we have the following formalization of Pairs.

Pairs

Given sorts s_1 and s_2 with the respective equality predicates $=_1 : s_1^2$ and $=_2 : s_2^2$, the theory of pairs is given by

$$\Sigma = \{(\ , \) : s_1 \times s_2 \longrightarrow (s_1, s_2), \\ \#1 : (s_1, s_2) \longrightarrow s_1, \#2 : (s_1, s_2) \longrightarrow s_1; \\ =_{12} : (s_1, s_2)^2\}$$

$$\phi_{Pairing} \stackrel{df}{=} \forall x: (s_1, s_2)[x =_{12} (\#1 x, \#2 x)]$$

$$\phi_{Pairs-=} \stackrel{df}{=} \forall x, y: (s_1, s_2) \\ [x =_{12} y \leftrightarrow ((\#1 x =_1 \#1 y) \wedge (\#2 x =_2 \#2 y))]$$

$$\Phi_{Pairs} \stackrel{df}{=} \{\phi_{Pairing}, \phi_{Pairs-=}\}$$

Exercise 40.1: Pairing and tuples

Let **A**, **B** and **C** be structures with carrier sets \mathbb{A} , \mathbb{B} and \mathbb{C} respectively.

1. From the theory of **pairs** derive first-order axioms for the following structures.
 - (a) **Pairs[A, Pairs[B, C]]**
 - (b) **Pairs[Pairs[A, B], C]**
2. Define the first-order theory of triples $\text{Triples}[\mathbf{A}, \mathbf{B}, \mathbf{C}] = \langle \mathbb{A} \times \mathbb{B} \times \mathbb{C}; (\ , \ , \), \#1, \#2, \#3; = \rangle$ over structures **A**, **B** and **C** respectively.
3. Note that $\mathbb{A} \times \mathbb{B} \times \mathbb{C} \neq \mathbb{A} \times (\mathbb{B} \times \mathbb{C}) \neq (\mathbb{A} \times \mathbb{B}) \times \mathbb{C} \neq \mathbb{A} \times \mathbb{B} \times \mathbb{C}$ since the binary cartesian product operation is neither commutative nor associative. However there are obvious and trivial isomorphisms which associate each triple (a, b, c) with each of the pairs $(a, (b, c))$ and $((a, b), c)$ respectively. Identify the obvious association between the axioms of the first order theory of **Triples[A, B, C]** and the axioms and derived rules of **Pairs[A, Pairs[B, C]]** and **Pairs[Pairs[A, B], C]** respectively

40.2. Arrays as functions

Arrays may be regarded as valuations from a non-zero finite ordinal (see section 21.1) \underline{n} which acts as the *indexing set* of sort \underline{n}) to a set of values drawn from a sort s . We use the usual notation $A[i]$ to denote the array element with index i . We denote array updates by $A\langle i := v \rangle$ to denote a new array which is point-wise equal to the array A except (possibly) for the index i where it has the value v . Unlike the usual imperative programming languages where arrays are often updated in-place, we take a more functional view of arrays. The reader may think of an array in our treatment as an immutable object and consequently an array “update” actually represents a new array that is a variant of an existing array.

For each non-zero finite ordinal \underline{n} , we assume the structure

$$\underline{n} = \langle \underline{n}, ;=_{\underline{n}}, <_{\underline{n}} \rangle$$

where $\underline{n} = \{0, \dots, \underline{n} - 1\}$.

Our arrays deal with the axiomatization of what are known as “functional arrays” in the sense that the update operation $A\langle i := v \rangle$ results in a different array than the original than actually updating only the index element of the array. This is different from the usual array update operation in imperative programming where array updates are actually side-effects. Our view is safer in the context of proving the correctness of programs, though it is often space and time inefficient compared to the usual imperative programming language view of an array as a directly accessible and updatable sequence of values. While this may sound idealistic it is actually useful when trying to prove programs, since “side-effects” are complicated to reason about and to keep track of.

The Array Structure: Signature

Let $\mathbf{A} = \langle \mathbb{A}, \Sigma_s \rangle$ be a structure with sort s such that $=_s$ is the equality predicate in Σ_s . Let \underline{n} be any non-zero finite ordinal (see section 21.1).

$$\mathbf{Arrays}[\mathbf{A}, \underline{n}] = \langle \mathbb{A}rrays[\mathbb{A}, \underline{n}], \Sigma_{s_n_array} \rangle$$

is the structure of arrays of n elements of \mathbb{A} indexed by the elements $\{0, \dots, n - 1\}$ where

$$\begin{aligned}\Sigma_{s_n_array} = & \{ [\] : s_n_array, \underline{n} \longrightarrow s \\ & \langle := \rangle : s_n_array, \underline{n}, s \longrightarrow s_n_array \\ & =_{s_n_array} : s_n_array^2 \}\end{aligned}$$

The Array Structure: Typing Axioms

$$\phi[]: \stackrel{df}{=} \forall A: s_n_array \forall i: \underline{n} [A[i]: s]$$
$$\phi\langle := \rangle: \stackrel{df}{=} \forall A: s_n_array \forall i: \underline{n} \forall v: s [A\langle i := v \rangle[j]: s_n_array]$$

The Array Structure: Access and Update Axioms

$$\phi_{[]} \stackrel{df}{=} \forall A: s_n_array \forall i, j: \underline{n} [i =_n j \rightarrow A[i] =_s A[j]]$$

$$\phi_{=:} \stackrel{df}{=} \forall A: s_n_array \forall i, j: \underline{n} \forall v: s \\ [i =_n j \rightarrow A\langle i := v \rangle[j] =_s v]$$

$$\phi_{\neq(:=)} \stackrel{df}{=} \forall A: s_n_array \forall i, j: \underline{n} \forall v: s \\ [i \neq_n j \rightarrow A\langle i := v \rangle[j] =_s A[j]]$$

$$\phi_{=s_n_array} \stackrel{df}{=} \forall A, B: s_n_array [A =_{s_n_array} B \leftrightarrow \\ \forall i: \underline{n} [A[i] =_s B[i]]]$$

40.3. Inductively Defined Data-types

The axiomatizations of the above structures was rather simple. What is more interesting are inductively defined (often called recursive) data structures and the axioms of their first-order theories. We devote the rest of our treatment to such inductively defined data structures viz. lists, stacks, queues and binary trees. Inductively defined data-types are characterised by two other important considerations (besides all that characterises the non-inductively defined structures).

Induction rule(s). Any inductively defined data-type needs to have a basis and induction step. Consequently to prove general properties of the data-type we need to be able to use the inductive construction process to prove many properties.

Acyclicity axiom(s). Any inductively defined data-type also needs to have distinctness axioms. One way of understanding the need for this is to hark back to the **inductive definition** of the naturals which require an infinite number of **distinctness axioms** $\Phi_{(+1)^{\infty}-\text{distinct}}$ to ensure that the construction does indeed yield a countably infinite number of distinct elements as the “intended interpretation” (points 4 and 5). We define the distinctness axioms for lists only and leave the the distinctness axioms for the other data-types as exercises (see problem 4 in exercise 40.3, problem 4 in 40.4 and problem 1 in 40.6).

40.4. Lists: The First-order Theory

A list in ML-like languages conforms to the following data-type definition.

```
datatype 'a list = []
           | :: of 'a * 'a list -> 'a list
infix ::
```

In effect, an '`'a list`' of elements of type '`'a`' is the smallest set (of terms) containing the empty list (`[]`) and closed under the (infix) cons `::` operation. This is the “functional” view of a list viz. as a term of an expression language. This view of a list is very unlike the notions of singly linked lists and doubly linked lists commonly treated in textbooks on data structures. To understand the relative differences refer to figure 12.

We may regard singly-linked lists and doubly-linked lists as particular implementations of the abstract data-type (ADT) of lists defined as terms of an expression language. In fact, in the terminology of logic, these implementations would be models of the axiomatic definition of the ADT lists defined as expressions, provided they satisfy all the axioms of the ADT.

Generally speaking, we could define list structures **Lists[A]** which refers to the lists of elements over **A**, where **A** could itself be a simple or compound structure. In other words, we treat this matter through a form of “parametric polymorphism” by making no assumptions about the first order theory (with equality) of the structure **A**, except that an equality relation does exist on **A** and may be used to induce or define an equality relation on **Lists[A]** as well. Further, being a many-sorted first-order logic, not all functions and predicates are total. Hence it is necessary to make several predicates conditional by guarding them with a type-predicate. Lastly, for any sort *s* of elements which make up a list, we use *s.list* as the sort of elements of **|Lists[A]|** and use this sort name for the infix type predicate `:s.list`. Similar remarks also apply to other

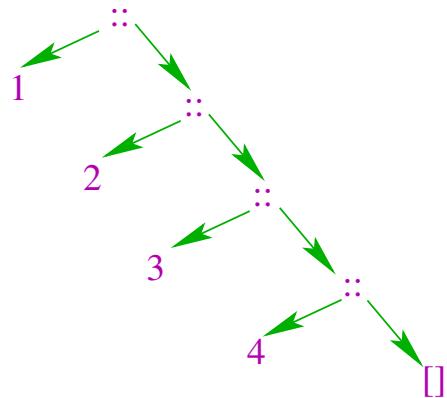
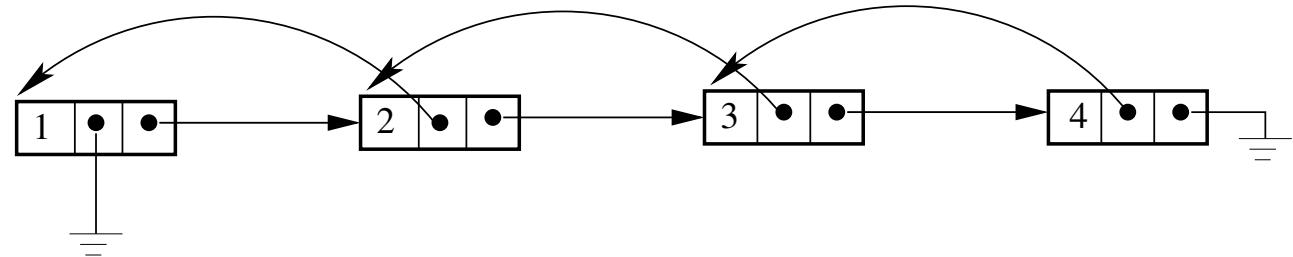


Figure 12: Schematic of a singly-linked list, a doubly-linked list and a list as term of an expression language

inductively defined data-types such as **BinTrees[A]** that we define in the sequel. In that sense our treatment of these structures follows the rigorous type-safe approach used in the ML-type systems as opposed to the more loosely defined type systems of Lisp, C and other languages, old and new.

A signature for such a structure would consist of

Constructors. $[s]$ and $::$

Deconstructors. hd and tl

Distinguished functions and predicates. $null$ which defines when a list is empty.

Extension functions and predicates. Open the ML structure **List** to see the various useful functions and predicates which are present to make the structure useful and easy to work with.

A first-order axiomatization of the theory of such structures would consist typically of

Typing axioms. These define the types of the various constants, constructors, functions and predicates. Since the constructors hd and tl are only partial functions it is important to introduce a “guard” for predicates involving these functions. For example the defining equation (85) would then be a guarded first-order predicate as shown in the axiom ϕ_{decomp} .

Structural axioms. These define the inter-relationships between the constructors and deconstructors. In the case of lists we have the following

$$L = [] \vee L = hd(L) :: tl(L) \quad (85)$$

Distinguished predicates. *null* to check whether a list is empty. This is especially useful to guard the application of the *hd* and *tl* functions.

Equality axioms. Equality of compound structures is often “induced” (usually point-wise as we have seen in the case of pairs) by the equality relation on the parameters of the compound structure and needs to be defined through axioms.

Other predicates. Usually the extension of the signature may require other axioms to define the behaviours of the extension functions and predicates.

We know from the theory of lists in ML-like languages, (due to problems relating to undecidability or representation) that not all structures **A** may have an equality relation defined on them. But in principle, an equality relation always exists. Further many structures do admit an equality relation and the corresponding equality relation on **Lists[A]** is induced by the equality relation on **A**.

Notation 40.1: Meta-variables and sorts

- As a matter of notational convenience we deviate from the rigid **notational formalities defined earlier** and instead use more intuitively obvious variable names such as L , M etc. to range over $\text{Lists}[\mathbb{A}]$ while retaining variable symbols x , y to denote elements of \mathbb{A} .
- Further as a matter of naming convenience we introduce sort names which are derived from the names of sort parameters suffixed by the name of the structure itself. Hence the sort of $\text{Lists}[\mathbb{A}]$ would be s_list where s is the sort of elements of \mathbb{A} . This allows for an easy extension of sort names (analogous to the type names in ML-like languages). Hence the sort of $\text{Lists}[\text{Lists}[\mathbb{A}]]$ would be s_list_list and the sort of $\text{Stacks}[\text{Lists}[\mathbb{A}]]$ would be named s_list_stack and so on.

The List Structure: Signature

Let $\mathbf{A} = \langle \mathbb{A}, \Sigma_s \rangle$ be a structure with sort s such that $=_s$ is the equality predicate in Σ_s . The structure

$$\text{Lists}[\mathbf{A}] = \langle \text{Lists}[\mathbb{A}], \Sigma_{s_list} \rangle$$

is the structure of lists of elements of \mathbb{A} where

$$\begin{aligned}\Sigma_{s_list} = \{ & [] : s_list, \\ & :: : s, s_list \longrightarrow s_list, \\ & hd : s_list \rightarrow s, \\ & tl : s_list \rightarrow s_list; \\ & null : s_list, \\ & =_{s_list} : s_list^2 \}\end{aligned}$$

Note.

1. Even though we are considering a single structure **Lists[A]**, the fact that it is built up from **A** implies that any axiomatization of such a structure has to respect sorts. For instance, the equality predicate $=_{s_list}$ is “induced” on **Lists[A]** by the equality predicate $=_s$ on **A**.
2. In general, any predicate on a compound structure that is defined using predicates on the components would necessarily require that sorts be respected. Hence the first-order theory of compound structures is in general, a many-sorted one.
3. While respecting sorts, sometimes it becomes rather cumbersome to mention the sort at each stage. We will often be guilty of omitting the sort where the context makes it obvious and unambiguous. For instance, equality predicates will often be “overloaded” in this fashion.
4. It frequently happens that our signature consists of operations or constructors or functions which are only partially defined. It is necessary in such cases to specify axioms which clearly delimit the conditions under which the operation is defined. We will refer to these axioms as “typing axioms”. For example the functions hd and tl in the signature Σ_{s_list} are not total. However the signature itself does not specify for what values of the domain these functions are defined. We need to specify in first order logic the sub-domain on which these functions are well-defined.

The List Structure Axioms: 1

$$\phi_{[]} \stackrel{df}{=} [] : s_list$$

$$\phi_{null} \stackrel{df}{=} null([])$$

$$\phi_{cons} \stackrel{df}{=} \forall x: s \ \forall L: s_list [(x :: L) : s_list]$$

$$\phi_{\neg null} \stackrel{df}{=} \forall x: s \ \forall L: s_list [\neg null(x :: L)]$$

$$\phi_{hd} \stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow (hd(L) : s)]$$

$$\phi_{tl} \stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow (tl(L) : s_list)]$$

The List Structure Axioms: 2

$$\begin{aligned}\phi_{hd} &\stackrel{df}{=} \forall x: s \ \forall L: s_list [hd(x :: L) =_s x] \\ \phi_{tl} &\stackrel{df}{=} \forall x: s \forall L: s_list [tl(x :: L) =_{s_list} L] \\ \phi_{decomp} &\stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow (L =_{s_list} hd(L) :: tl(L))] \\ \phi_{=}-inj &\stackrel{df}{=} \forall x, y: s \ \forall L, M: s_list [(x :: L =_{s_list} y :: M) \leftrightarrow \\ &\quad (x =_s y \wedge L =_{s_list} M)]\end{aligned}$$

Explanations.

1. The formulae $\phi_{[],:}$, $\phi_{cons,:}$, $\phi_{hd,:}$ and $\phi_{tl,:}$ are the typing axioms. They are analogous to Peano's postulates P1 and P2 which are the typing axioms for the naturals in Peano arithmetic.
2. The axioms ϕ_{null} and $\phi_{\neg null}$ distinguish between empty and non-empty lists and clearly show that the empty list is unique in the sort *s_list*.
3. The axiom ϕ_{decomp} actually defines the relation between the constructors and the deconstructor operations on lists.
4. The axiom $\phi_{=_{inj}}$ defines the equality relation on lists as being induced by the point-wise equality relation on the underlying elements.
5. Lastly ϕ_{hd} and ϕ_{tl} define the partial functions *hd* and *tl* respectively.

The List Structure: Induction

Analogous to the **induction principle** in Peano Arithmetic we may define an induction principle for List structures in general.

$$\text{IndList. } \frac{\{[]/L\}X, \forall x: s \forall L: s_list[X \rightarrow \{(x :: L)/L\}X]}{\forall L: s_list[X]}$$

where

- L is the only free variable of X ,
- $\{[]/L\}X$ is the basis of the induction,
- $\forall x: s \forall L: s_list[X \rightarrow \{(x :: L)/L\}X]$ is the induction step and
- X the antecedent in the induction step, is the induction hypothesis

The List Structure: Acyclicity 1

Our intended interpretation requires of the “cons” operation that each such operation yields a list that is different from the list on which the operation was performed.

$$\phi_{(::)^1-distinct} \stackrel{df}{=} \forall L: s_list \forall x_1: s[L \neq x_1 :: L]$$

$$\phi_{(::)^2-distinct} \stackrel{df}{=} \forall L: s_list \forall x_1, x_2: s[L \neq x_1 :: (x_2 :: L)]$$

⋮ ⋮ ⋮

$$\phi_{(::)^n-distinct} \stackrel{df}{=} \forall L: s_list \forall x_1, \dots, x_n: s[L \neq x_1 :: (\dots :: (x_n :: L))]$$

⋮ ⋮ ⋮

The List Structure: Acyclicity 2

Our intended interpretation requires that no list equals its tail, or the tail of its tail or the the tail of the tail of its tail, and so on.

$$\phi_{(tl)^1\text{-}distinct} \stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow L \neq tl(L)]$$

$$\begin{aligned}\phi_{(tl)^2\text{-}distinct} &\stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow \neg null(tl(L)) \rightarrow \\ &\quad L \neq tl(tl(L))]\\ &\vdots \quad \vdots \quad \vdots\end{aligned}$$

$$\begin{aligned}\phi_{(tl)^n\text{-}distinct} &\stackrel{df}{=} \forall L: s_list [\neg null(L) \rightarrow \dots \neg null(tl^n(L)) \rightarrow \\ &\quad L \neq tl^n(L)]\\ &\vdots \quad \vdots \quad \vdots\end{aligned}$$

Exercise 40.2

1. We have specified certain axioms and an induction principle for ML-like lists via the **signature** Σ_{s_list} and the **axioms** and an **inference rule**. The “intention” is that $\text{Lists}[\mathbb{A}]$ is the (smallest) inductively defined set of terms containing $[]$ and closed under the infix binary operation $::$.
 - (a) Which axioms actually support the above claim?
 - (b) Are there other sets (analogous to the non-standard models of Peano arithmetic) which also satisfy the same set of axioms and inference rule? If so find examples.
 - (c) Consider a set containing all finite length lists as well as infinite length lists. Does this set satisfy all the **axioms** and the **inference rule**? Justify your answer.
2. The (infix) append operation $@$ on ML-like lists is defined programmatically as follows.

```
fun [] @ M = M
  | (a :: L) @ M = a :: (L @ M)
```

- (a) Define the infix append operation on $\text{Lists}[\mathbb{A}]$.
- (b) Prove that $\langle \text{Lists}[\mathbb{A}]; [], @, = \rangle$ is a monoid. That is, you need to prove the following.
 - i. $[] @ L =_{s_list} L =_{s_list} L @ []$
 - ii. $L @ (M @ N) =_{s_list} (L @ M) @ N$
- (c) Write an iterative program in $\mathcal{WH}(\Sigma_{s_list})$ which takes as input a list L and produces its reverse as list M . Prove the correctness of your program.

40.5. Stacks: The First-order Theory

Fundamentally stacks are a different abstract data-type than lists. So we have used symbols that are distinct for the operations of each data-type. Using different symbols ensures that there is no confusion¹⁸.

1. $[>]$ denotes the empty stack
2. \downarrow denotes the (infix) “push” operation on stacks. Often when we are interested in the “values” in the stack we could abbreviate for example, the complex expression

$$((((>\downarrow 1) \downarrow 2) \downarrow 3) \downarrow 4)$$

by the simpler notation (see problem 3 in exercise 40.3)

$$[1, 2, 3, 4 >]$$

3. \wedge denotes the (prefix) “top of stack” function. Hence $\wedge[1, 2, 3, 4 >]$ is 4.
4. \uparrow denotes the (postfix) “pop” operation. Its effect is to remove the “top of stack” element from the stack. Hence $[1, 2, 3, 4 > \uparrow]$ yields the stack $[1, 2, 3 >]$.

While it is regarded good programming practice to use names that are self-explanatory in nature, it is also at the cost of brevity and compactness of notation. We have chosen to use compact symbols to denote the various constructors, functions and operators to allow ease of composition and readability of compound expressions involving them.

¹⁸Especially when you consider operations over lists of stacks or stacks of lists etc. or when the operations of one data type are expressed (“implemented”) in terms of the operations of another abstract data type.

The Stack Structure: Signature

Let $\mathbf{A} = \langle \mathbb{A}, \Sigma_s \rangle$ be a structure with sort s such that $=_s$ is the equality predicate in Σ_s . The structure

$$\text{Stacks}[\mathbf{A}] = \langle \text{Stacks}[\mathbb{A}], \Sigma_{s_stack} \rangle$$

is the structure of stacks of elements of \mathbb{A} where

$$\begin{aligned}\Sigma_{s_stack} = \{ &[> : s_stack, \\ &\downarrow : s_stack, s \longrightarrow s_stack, \\ &\uparrow : s_stack \rightharpoonup s_stack, \\ &\wedge : s_stack \rightharpoonup s; \\ &\text{nulls} : s_stack, \\ &=_s : s_stack \times s_stack\}\end{aligned}$$

The Stack Structure Axioms: 1

$$\begin{aligned}\phi[>:] &\stackrel{df}{=} [>: s_stack] \\ \phi_{nulls} &\stackrel{df}{=} nulls([>]) \\ \phi_{\downarrow}: &\stackrel{df}{=} \forall x: s \ \forall S: s_stack [(S \downarrow x): s_stack] \\ \phi_{\neg nulls} &\stackrel{df}{=} \forall x: s \ \forall S: s_stack [\neg nulls(S \downarrow x)] \\ \phi^{\wedge}: &\stackrel{df}{=} \forall S: s_stack [\neg nulls(S) \rightarrow (\wedge S): s] \\ \phi^{\wedge} &\stackrel{df}{=} \forall x: s \ \forall S: s_stack [\wedge (S \downarrow x) =_s x]\end{aligned}$$

The Stack Structure Axioms: 2

$$\phi_{\uparrow} \stackrel{df}{=} \forall S: s_stack [\neg nulls(S) \rightarrow (S \uparrow): s_stack]$$

$$\phi_{\uparrow} \stackrel{df}{=} \forall x: s \forall S: s_stack [(S \downarrow x) \uparrow =_{s_stack} S]$$

$$\phi_{decomp} \stackrel{df}{=} \forall S: s_stack [\neg nulls(S) \rightarrow S =_{s_stack} (S \uparrow) \downarrow (\wedge S)]$$

$$\phi_{= - inj} \stackrel{df}{=} \forall x, y: s \forall S, T: s_stack [(S \downarrow x =_{s_stack} T \downarrow y) \leftrightarrow x =_s y \wedge S =_{s_stack} T]$$

The Stack Structure: Induction

Analogous to the **induction principle** for lists we have

$$\text{IndStack. } \frac{\{[>/S]X, \forall x: s \forall S: s_stack[X \rightarrow \{(S \downarrow x)/S\}X]\}}{\forall S: s_stack[X]}$$

where

- S is the only free variable of X ,
- $\{[>/S]X$ is the basis of the induction,
- $\forall x: s \forall S: s_stack[X \rightarrow \{(S \downarrow x)/S\}X]$ is the induction step and
- X the antecedent in the induction step, is the induction hypothesis.

Exercise 40.3

1. Use the **list structure** to define a **stack structure** in terms of lists.
2. Use the **first-order theory of lists** to prove that your stack structure does indeed satisfy all the **axioms** of the first-order theory of stacks.
3. Prove that for every stack expression $S: s_stack$, $S =_{s_stack} T$ where $T: s_stack$ is a stack expression with no occurrences of operations \uparrow or \wedge .
4. State the acyclicity axioms for stacks.

40.6. Queues: The First Order Theory

We proceed in a manner analogous to stacks here.

1. $\langle\langle$ denotes the empty queue.
2. \swarrow is the infix binary operation to “enqueue” an element to a given queue. Often when we are interested only in the “sequence of values” in a queue, we could abbreviate (for example) the complex expression

$$(((\langle\langle \swarrow 1) \swarrow 2) \swarrow 3) \swarrow 4)$$

by the simpler notation (see problem 1 in exercise 40.4)

$$<1, 2, 3, 4 <$$

3. \curvearrowleft denotes the prefix “head of the queue” function. The head of the queue $<1, 2, 3, 4 <$ is the value 1.
4. \nwarrow denotes the (prefix) “dequeue” operation. Hence $\nwarrow <1, 2, 3, 4 <$ yields the queue $<2, 3, 4 <$. The effect of a dequeue operation is to “remove” the head of the queue.

The Queue Structure: Signature

Let $\mathbf{A} = \langle \mathbb{A}, \Sigma_{\mathcal{S}} \rangle$ be a structure with sort s such that $=_s$ is the equality predicate in $\Sigma_{\mathcal{S}}$. The structure

$$\text{Queues}[\mathbf{A}] = \langle \text{Queues}[\mathbb{A}], \Sigma_{s_queue} \rangle$$

is the structure of queues of elements of \mathbb{A} where

$$\begin{aligned}\Sigma_{s_queue} = & \{ << : s_queue, \\ & \leftarrow : s_queue, s \rightarrow s_queue, \\ & \nwarrow : s_queue \rightarrow s_queue, \\ & \curvearrowright : s_queue \rightarrow s; \\ & nullq : s_queue, \\ & =_{s_queue} : s_queue^2 \} \end{aligned}$$

The Queue Structure Axioms: 1

$$\phi_{<<} \stackrel{df}{=} << : s_queue$$

$$\phi_{\text{nullq}} \stackrel{df}{=} \text{nullq}(<<)$$

$$\phi_{\swarrow} \stackrel{df}{=} \forall x: s \ \forall Q: s_queue [(Q \swarrow x) : s_queue]$$

$$\phi_{\neg \text{nullq}} \stackrel{df}{=} \forall x: s \ \forall Q: s_queue [\neg \text{nullq}(Q \swarrow x)]$$

$$\phi_{\curvearrowright} \stackrel{df}{=} \forall Q: s_queue [\neg \text{nullq}(Q) \rightarrow (\curvearrowright Q) : s]$$

$$\phi_{\curvearrowright <<} \stackrel{df}{=} \forall x: s [\curvearrowright (<< \swarrow x) =_s x]$$

$$\begin{aligned} \phi_{\curvearrowright \neg \text{nullq}} \stackrel{df}{=} & \forall x: s \forall Q: s_queue \\ & [\neg \text{nullq}(Q) \rightarrow (\curvearrowright (Q \swarrow x) =_s \curvearrowright Q)] \end{aligned}$$

The Queue Structure Axioms: 2

$$\begin{aligned}\phi_{\nwarrow} &\stackrel{df}{=} \forall Q: s_queue [\neg nullq(Q) \rightarrow ((\nwarrow Q): s_queue)] \\ \phi_{\nwarrow \neg nullq} &\stackrel{df}{=} \forall x: s \forall Q: s_queue \\ &[\neg nullq(Q) \rightarrow (\nwarrow (Q \swarrow x) =_{s_queue} (\nwarrow Q) \swarrow x)] \\ \phi_{=_{-inj}} &\stackrel{df}{=} \forall x, x': s \forall Q, Q': s_queue \\ &[((Q \swarrow x) =_{s_queue} (Q' \swarrow x')) \leftrightarrow \\ &(Q =_{s_queue} Q' \wedge x =_s x')]\end{aligned}$$

The Queue Structure: Induction

$$\text{IndQueue. } \frac{\{ \ll / Q \} X, \forall x: s \forall Q: s_queue[X \rightarrow \{ (Q \swarrow x) / Q \} X]}{\forall Q: s_queue[X]}$$

where

- Q is the only free variable of X ,
- $\{ \ll / Q \} X$ is the basis of the induction,
- $\forall x: s \forall Q: s_queue[X \rightarrow \{ (Q \swarrow x) / Q \} X]$ is the induction step and
- X the antecedent in the induction step, is the induction hypothesis.

Exercise 40.4

1. Prove that for every queue expression $Q: s_queue$, $Q =_{s_queue} R$ where $R: s_queue$ is a queue expression with no occurrences of operations ↪ or ↵.
2. Let \mathbf{A} be any structure. Express the operations of **Queues[A]** in terms of the operations of **Lists[A]** and prove that the queue axioms are satisfied.
3. We may implement a queue as a pair of stacks or as a pair of lists. This is often useful since it allows both enqueueing and dequeuing to be performed as amortised constant time operations. Prove the correctness of your implementation.
4. State the acyclicity axioms for queues.

Exercise 40.5

1. The abstract data type for “double-ended queues” $\mathbf{DEQueues[A]} = \langle \mathbf{DEQueues[A]}, \Sigma_{s_deq} \rangle$ of values drawn from a structure \mathbf{A} is given by

$$\begin{aligned}\Sigma_{s_deq} = & \{ >< : s_deq, \\ & \searrow : s, s_deq \rightarrow s_deq, \swarrow : s_deq, s \rightarrow s_deq, \\ & \nwarrow : s_queue \rightarrow s_deq, \nearrow : s_deq \rightarrow s_deq, \\ & \curvearrowleft : s_deq \rightarrow s, \curvearrowright : s_deq \rightarrow s; \\ & nullq : s_deq, =_{s_deq} : s_deq^2 \} \end{aligned}$$

The various functions and constructors are briefly described below

- $><$: the empty double-ended queue.
- $x \searrow D$: the *infix* operation to enqueue element x at the *front* of the double-ended queue D .
- $D \swarrow y$: the *infix* operation to enqueue element y at the *rear* of the double-ended queue D .
- $\curvearrowleft D$: the *prefix* function which yields the *first* element of the double-ended queue D .
- $D \curvearrowright$: the *postfix* function which yields the *last* element of the double-ended queue D .
- $\nwarrow D$: the *prefix* operation which dequeues the *first* element from the double-ended queue D .
- $D \nearrow$: the *postfix* operation which dequeues the *last* element from the double-ended queue D .
- $nulldeq(D)$: test of emptiness of the double-ended queue D .
- $D =_{s_deq} D'$: equality predicate for double-ended queues.

Write a first-order axiomatization of the structure $\mathbf{DEQueues[A]}$ (Hint: Also see example 36.6).

40.7. Binary Trees: The First-order Theory

We define an inductive user-defined data-type of binary trees in ML-like languages conforming to the following data-type definition.

```
datatype 'a bintree = Empty | Node of 'a * 'a bintree * 'a bintree
```

where each node is labelled by a value from the sort parameter. In this case a leaf-node is actually a (sub-)tree with empty sub-trees as children (nodes 4,5, and 6 in figure 13). Further any internal node with only one child (either left or right) has an empty tree as the other child underneath it (node 2 in figure 13).

Since we require a linear representation we prefer to grow our trees from left to right.

A brief explanation of the signature of this data-type is given below:

1. \triangleleft denotes the empty binary-tree.
2. $(x \begin{array}{c} \nearrow T_R \\ \searrow T_L \end{array})$ denotes a binary tree with root labelled x and left subtree T_L and right subtree T_R .
3. $rt(T)$, $lst(T)$ and $rst(T)$ denote respectively the root, the left subtree and the right subtree of the non-empty binary tree T .

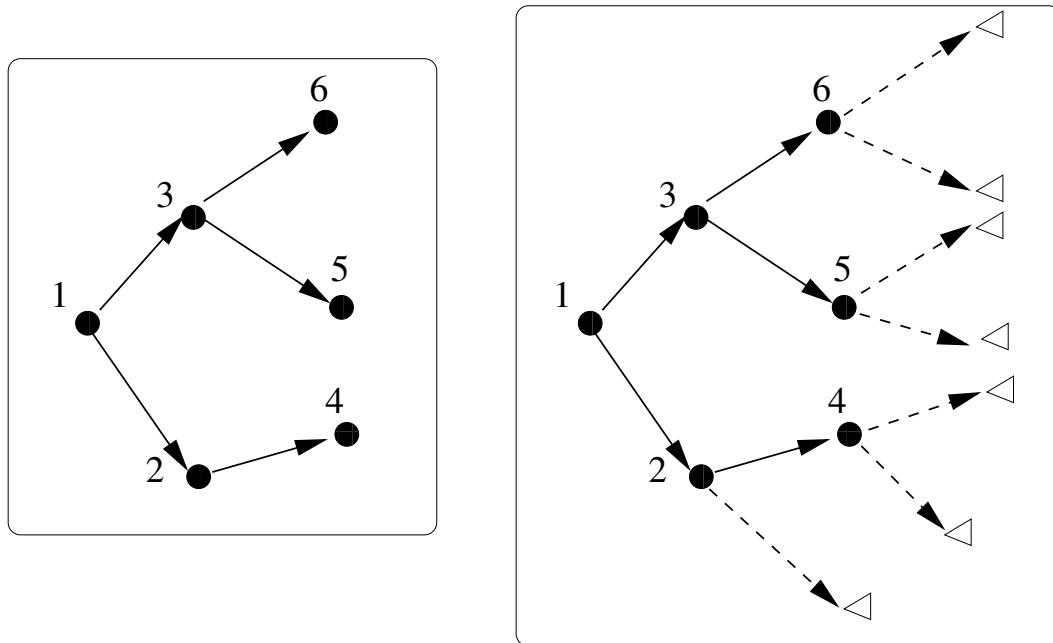


Figure 13: Example binary tree with 6 nodes drawn left to right

Binary Trees: Signature

Let $\mathbf{A} = \langle \mathbb{A}, \Sigma_{\mathcal{S}} \rangle$ be a structure with sort \mathcal{S} such that $=_{\mathcal{S}}$ is the equality predicate in $\Sigma_{\mathcal{S}}$. The structure

$$\text{BinTrees}[\mathbf{A}] = \langle \text{BinTrees}[\mathbb{A}], \Sigma_{\mathcal{S}_bt} \rangle$$

is the structure of binary trees of elements of \mathbb{A} where

$$\begin{aligned}\Sigma_{\mathcal{S}_bt} = \{ & \triangleleft : \mathcal{S}_bt, \\ & lst : \mathcal{S}_bt \rightarrow \mathcal{S}_bt, \\ & rst : \mathcal{S}_bt \rightarrow \mathcal{S}_bt, \\ & rt : \mathcal{S}_bt \rightarrow \mathcal{S}; \\ & nullbt : \mathcal{S}_bt, \\ & =_{\mathcal{S}_bt} : \mathcal{S}_bt^2 \}\end{aligned}$$

The Binary Tree Axioms: 1

$$\phi_{\triangleleft} \stackrel{df}{=} \triangleleft : s_bt$$

$$\phi_{nullbt} \stackrel{df}{=} nullbt(\triangleleft)$$

$$\phi_{node} \stackrel{df}{=} \forall x: s \forall T_L, T_R: s_bt [\begin{pmatrix} & \nearrow^{T_R} \\ x & \downarrow \\ & \searrow^{T_L} \end{pmatrix} : s_bt]$$

$$\phi_{\neg nullbt} \stackrel{df}{=} \forall x: s \forall T_L, T_R: s_bt [\neg nullbt(\begin{pmatrix} & \nearrow^{T_R} \\ x & \downarrow \\ & \searrow^{T_L} \end{pmatrix})]$$

The Binary Tree Axioms: 2

$$\phi_{rt} \stackrel{df}{=} \forall T: s_bt [\neg \text{nullbt}(T) \rightarrow rt(T): s]$$

$$\phi_{lst} \stackrel{df}{=} \forall T: s_bt [\neg \text{nullbt}(T) \rightarrow lst(T): s_bt]$$

$$\phi_{rst} \stackrel{df}{=} \forall T: s_bt [\neg \text{nullbt}(T) \rightarrow rst(T): s_bt]$$

$$\phi_{rt} \stackrel{df}{=} \forall x: s \forall T_L, T_R: s_bt [rt(\begin{pmatrix} & \nearrow^{T_R} \\ x & \downarrow \\ & \searrow^{T_L} \end{pmatrix}) =_s x]$$

$$\phi_{lst} \stackrel{df}{=} \forall x: s \forall T_L, T_R: s_bt [lst(\begin{pmatrix} & \nearrow^{T_R} \\ x & \downarrow \\ & \searrow^{T_L} \end{pmatrix}) =_{s_bt} T_L]$$

$$\phi_{rst} \stackrel{df}{=} \forall x: s \forall T_L, T_R: s_bt [rst(\begin{pmatrix} & \nearrow^{T_R} \\ x & \downarrow \\ & \searrow^{T_L} \end{pmatrix}) =_{s_bt} T_R]$$

The Binary Tree Axioms: 3

$$\begin{aligned}
 \phi_{decomp} &\stackrel{df}{=} \forall T: s_bt [\neg nullbt(T) \rightarrow (T =_{s_bt} \left(rt(T) \begin{array}{c} \nearrow r st(T) \\ \searrow l st(T) \end{array} \right))] \\
 \phi_{=_{s_bt}} &\stackrel{df}{=} \forall x, x': s \forall T_L, T'_L, T_R, T'_R: s_bt \\
 &\quad [(\left(x \begin{array}{c} \nearrow T_R \\ \searrow T_L \end{array} \right) =_{s_bt} \left(x' \begin{array}{c} \nearrow T'_R \\ \searrow T'_L \end{array} \right)) \rightarrow \\
 &\quad (x =_s x' \wedge T_L =_{s_bt} T'_L \wedge T_R =_{s_bt} T'_R)]
 \end{aligned}$$

The Binary Tree Induction Rule

We may define an induction principle for binary trees.

$$\frac{\text{IndBinTrees.} \quad \begin{array}{c} \{ \triangleleft / T \} X, \\ \forall x: s \forall T_L, T_R: s_bt[(\{ T_L / T \} X \wedge \{ T_R / T \} X) \rightarrow \\ \{ \left(x \begin{array}{c} \nearrow T_R \\ \searrow T_L \end{array} \right) / T \} X] \end{array}}{\forall T: s_bt[X]}$$

Explanation. In the induction rule

- T is the only free variable of X ,
- $\{\triangleleft/T\}X$ is the basis of the induction,
- $\forall x: s \forall T_L, T_R: s_bt[(\{T_L/T\}X \wedge \{T_R/T\}X) \rightarrow \{x \begin{array}{c} \nearrow^{T_R} \\ \searrow_{T_L} \end{array} /T\}X]$ is the induction step and
- $(\{T_L/T\}X \wedge \{T_R/T\}X)$ the antecedent in the induction step, is the induction hypothesis.

Exercise 40.6

1. State the acyclicity axioms for binary trees (Hint: You need to specify that no binary tree equals any of its proper sub-trees).
2. It is possible to define a different data-type for binary trees which is more in keeping with the requirements of algorithms on binary-trees. In the usual construction, leaves are treated separately from interior nodes of the tree and there are no empty trees. Such a binary tree may be defined in ML-like languages as

```
datatype 'a bintree' = Leaf of 'a | Node of 'a * 'a bintree' * 'a bintree',
```

Define the modified signature and axioms (and an induction rule) for the modified data-type

$$\text{BinTrees}'[\mathbf{A}] = \langle \text{BinTrees}'[\mathbf{A}], \Sigma_{s_bt} \rangle$$

3. Our construction of binary trees may be criticised for creating an “artificial” difference between the trees $\binom{x}{\nearrow^T \searrow}$ and $\binom{x}{\nearrow^{\triangleleft} \searrow^T}$.
 - (a) Define a new data-type definition to eliminate this distinction for nodes of out-degree 1.
 - (b) Define the signature and the structure of this new data-type, $\text{BinTrees}''[\mathbf{A}]$ without including concepts like out-degree or numbers in any way.
 - (c) Define the modified set of first-order axioms and the induction rule for this new data-type.

(Hint: Define two different constructors for nodes of out-degree 1 and out-degree 2.)

Exercise 40.7: Multi-way trees

1. Multiway trees in ML may be defined by the ML datatype

```
datatype 'a multree = Empty | Node of 'a * 'a multree list
```

Notice that the criticism (problem 3 of exercise 40.6) no longer holds.

- Define a signature for the multi-way trees of ML. In particular, what operations, constructors, deconstructors and atomic predicates are required in the signature to be able to provide useful axioms for reasoning about them?
- A binary tree is merely a particular case of a multi-way tree. Implement the data-type **BinTrees**"[A] of problem 3 of exercise 40.6 using the operations of multi-way trees.
- State the typing axioms for the operations, constructors and deconstructors on multi-way trees.
- State the other axioms that are required to relate the various operations on multi-way trees.
- State an induction rule for multi-way trees.
- What are the acyclicity axioms that will be required to ensure that there are no cycles in multi-way trees.

References

- [1] A. R. Bradley and Z. Manna. *The Calculus of Computation*. Springer-Verlag, Berlin, Germany, 2007.
- [2] I. M. Copi. *Symbolic Logic*. Macmillan, London, UK, 1979.
- [3] H. D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer-Verlag, New York, USA, 1994.
- [4] H. B. Enderton. *A Mathematical Introduction to Logic*. Elsevier India, New Delhi, India, 2001.
- [5] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, USA, 1990.
- [6] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, UK, 2000.
- [7] John Kelly. *The Essence of Logic*. Prentice-Hall India, New Delhi, India, 1997.
- [8] S. C. Kleene. *Mathematical Logic*. Dover Publications Inc., New York, USA, 1967.
- [9] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming, Vol. I*. Addison-Wesley Publishing Company, U. S. A., 1985.
- [10] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming, Vol. II*. Addison-Wesley Publishing Company, U. S. A., 1990.
- [11] E. Mendelson. *Introduction to Mathematical Logic*. D. Van Nostrand Co. Inc., Princeton, New Jersey, USA, 1963.
- [12] Anil Nerode and R. Shore. *Logic for Applications*. Springer-Verlag, New York, USA, 1993.
- [13] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlin, Germany, 1968.
- [14] V. Sperschneider and G. Antoniou. *Logic: A Foundation for Computer Science*. Addison-Wesley Publishing Company, Reading, UK, 1991.

Thank You!

Any Questions?