Introduction to Logic

A Computational Perspective

Henning Basold

Base revision 383a693, (HEAD -> master) from 2020-06-13

Typeset with XeLaTeX

This is a story of a robot trying to find a heart.
A story of equal rights and hope.

Contents

1. Introduction

2. Introduction to Propositional Logic

2.1. Motivation

2.2. Syntax of Propositional Logic

Definition 2.1

The formulas φ of predicate logic are given by the following context free grammar, in which p ranges over the propositional variables PVar.

$$\varphi ::= p \mid \bot \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \psi \mid (\varphi)$$

Reading conventions:

- $\wedge$ and $\vee$ have precedence over $\to$
- all connective associate to the right

The reading conventions allow us to leave out parentheses:

Formula	With parentheses
$p \to q \to r$	$p \to (q \to r)$
$p \wedge q \wedge r$	$p \wedge (q \wedge r)$
$p \vee q \vee r$	$p \vee (q \vee r)$
$p \wedge q \to r$	$(p \wedge q) \to r$
$p \vee q \to r$	$(p \vee q) \to r$
$p \to q \wedge r$	$p \to (q \wedge r)$

Note that there is no convention about mixing $\wedge$ and $\vee$, as this would cause more confusion than it helps. For example, the formula $p \wedge q \vee r$ is considered to be ambiguous and should be written either as $(p \wedge q) \vee r$ or $p \wedge (q \vee r)$.

Table 2.1 shows all the basic and derived connectives of propositional logic that appear here.

Basic connectives

Connective	Name	Pronunciation
$\bot$	Absurdity/Falsity	
$\wedge$	Conjunction	φ and ψ
$\vee$	Disjunction	φ or ψ
$\rightarrow$	Implication	φ implies ψ

Derived connectives

Connective	Name	Definition
$\neg$	Negation	$\neg\varphi = \varphi \rightarrow \bot$
$\top$	Truth	$\top = \neg\bot$
$\leftrightarrow$	Bi-implication	$\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Table 2.1.: Logical connectives of propositional logic

3. Semantics of Propositional Logic

4. Proof Theory of Propositional Logic

4.1. The Intuitionistic System ND

$$\frac{\varphi : \Gamma}{\Gamma \vdash \varphi}\ (\text{Assum}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ (\bot\text{E})$$

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ (\wedge\text{E}_1) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge\text{E}_2) \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge\text{I})$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_1) \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_2)$$

$$\frac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \delta \quad \Gamma, \psi \vdash \delta}{\Gamma \vdash \delta}\ (\vee\text{E})$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to\text{I}) \qquad \frac{\Gamma \vdash \varphi \to \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to\text{E})$$

Figure 4.1.: Deduction Rules of the natural deduction system $\mathbf{ND}_1$

Definition 4.1: Natural deduction for intuitionistic propositional logic

The system **ND** for propositional logic is given by the rules in fig. 4.1.

5. Normal Forms and Automatic Deduction for Propositional Logic

6. Introduction to First-Order Logic

6.1. The Need for a Richer Language

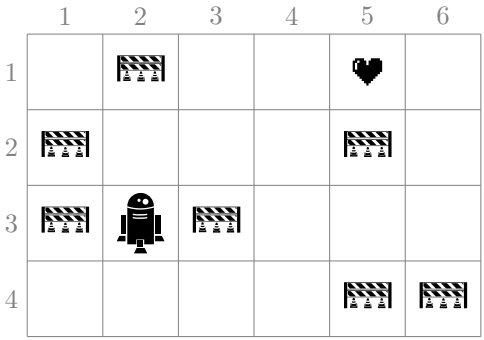


Figure 6.1.: Robot trying to find a heart

6.2. The Language of First-Order Logic

Definition 6.1

A first-order *signature* $\mathcal{L}$ is a triple $\boxed{(\mathcal{F}, \mathcal{R}, \text{ar})}$, where $\mathcal{F}$ and $\mathcal{R}$ are disjoint sets ($\mathcal{F} \cap \mathcal{R} = \emptyset$) and ar is a map $\mathcal{F} \cup \mathcal{R} \to \mathbb{N}$. The elements of $\mathcal{F}$ are called *function symbols* and those of $\mathcal{R}$ *predicate symbols*. The map ar assigns to each symbols its *arity*, which is the number of argument the symbol expects. We write $\boxed{\mathcal{F}_n = \{f \in \mathcal{F} \mid \text{ar}(f) = n\}}$ and $\boxed{\mathcal{R}_n = \{P \in \mathcal{R} \mid \text{ar}(P) = n\}}$. Elements of $\mathcal{F}_0$ are called *constants*.

Basic connectives

Connective	Name	Pronunciation
$\bot$	Absurdity/Falsity	
$\wedge$	Conjunction	φ and ψ
$\vee$	Disjunction	φ or ψ
$\rightarrow$	Implication	φ implies ψ
$\forall x.$	Universal quantifier	for all x, φ holds
$\exists x.$	Existential quantifier	for some x, φ holds

Derived connectives

Connective	Name	Definition
$\neg$	Negation	$\neg\varphi = \varphi \rightarrow \bot$
$\top$	Truth	$\top = \neg\bot$
$\leftrightarrow$	Bi-implication	$\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
$\exists!x.$	Uniqueness quantifier	$\exists!x.\, \varphi = \exists x.\, \varphi \wedge \mathrm{unique}_x(x, \varphi)$

Table 6.1.: Logical connectives of first-order logic

7. Proof Theory of First-Order Logic

7.1. Substitution in First-Order Logic

In these notes, we will discuss an important operation of first-order logic: the substitution of a term for a variable. This operation will replace any occurrence of a variable in a formula by the given term. For instance, we will be able to substitute the term $f(m, x)$ for the variable y in the formula $P(y) \wedge Q(r, y)$ to obtain

$$P(f(m, x)) \wedge Q(r, f(m, x)).$$

However, the substitution operation is surprisingly complex, as we need to deal with variables that are bound by quantifiers. The aim of these notes is to give a rigorous presentation of variables and binding that allows us to safely carry out substitutions.

7.1.1. The Difficulty of Names and Variables

Let us first discuss two questions that arise in FOL:

1. Are the formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$ expressing the same?
2. What is the scope of variables?

Formula Equality and Renaming The first question can be answered by reading the formula without explicitly naming variables:

> "P holds for all objects".

Clearly, this sentence corresponds to both formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$, and we should consider both formulas to be the same, already *syntactically*! This gives us a first rule that we will have to adhere to for FOL:

> Two formulas are considered to be the same, if we can *bijectively rename* the variables of one formula to obtain the other.

For instance, consider the formulas φ and ψ given by $\varphi = \forall x.\, \forall y.\, Q(x, y)$ and $\psi = \forall w.\, \forall z.\, Q(w, z)$. Then, a bijective renaming would be to rename x to w and y to z, which allows us to transform φ into ψ. However, renaming x to r and y to r is not bijective. Thus, we do not consider the formula $\forall r.\, \forall r.\, Q(r, r)$ to be same as φ. Note, that in the latter formula, the r refers to the inner-most quantifier and the outer quantifier has no effect!

Scoping The second question concerns the scope of variables. This requires us to determine which objects a variable refers to. For instance, the variable x in the sub-formula $P(x)$ of $\forall x.\, P(x)$ refers to what the quantifier ranges over. We say that x is *in the scope* of $\forall x$ in this formula. However, the variable y in $\forall x.\, Q(x, y)$ is in the scope of no quantifier and is thus a global reference. If we were now naively substituting x for y in this formula, then we would obtain $\forall x.\, Q(x, x)$. Here, the scope of the second argument of Q, and with it the meaning of the formula, has suddenly changed. This leads to a second rule:

> Substituting a term in a formula should not change the scoping of variables.

7.1.2. De Bruijn Trees

There are several ways to deal with variables, binding and substitution. An intuitively understandable way is to represent terms and formulas as *de Bruijn trees*. The idea is that bound variables are represented by a number that points to the quantifier that binds this variables. All free variables keep their name.

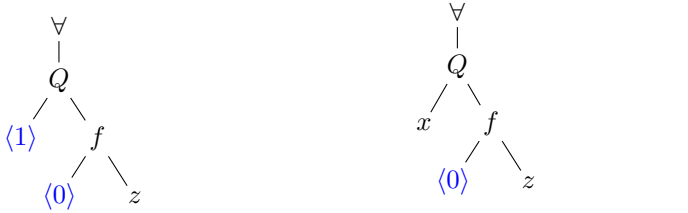
Figure 7.1 shows the (de Bruijn-) tree representation of $\forall x.\, \forall y.\, Q(x, f(y, z))$, where the left figure shows the actual tree and the right figure indicates to which quantifier the numerical name refers. We see that the bound variables

(a) The plain tree (b) Tree with indication of the variable references

Figure 7.1.: de Bruijn-Tree Representations of $\forall x.\, \forall y.\, Q(x, f(y, z))$

are numbered starting from the *inner-most* quantifier. Also note that the variable z is free and thus keeps its name in the tree representation.

Since the nesting depth of quantifiers is important, we also note that tree representations are *not closed under subtrees*! For instance, the tree in fig. 7.2a is not a valid tree representation because $\langle 1 \rangle$ is a dangling reference. Instead,



(a) Invalid de Bruijn-Tree (b) Correct tree representing $\forall y.\, Q(x, f(y, z))$

Figure 7.2.: Attempts of finding trees that represent the subformula $\forall y.\, Q(x, f(y, z))$ of $\forall x.\, \forall y.\, Q(x, f(y, z))$

if we want to remove the outer quantifier, we need to pick a name that we replace $\langle 1 \rangle$ with, say x, and then obtain the tree representation in fig. 7.2b.

In what follows, we will not work explicitly with tree representations, but will use another approach, see section 7.1.3 below. However, the tree representation shows how we can solve the initial problems:

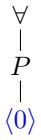


Figure 7.3.: De Bruijn-Tree representing both formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$

(a) Representation before substitution (b) Representation after substitution

Figure 7.4.: Substitution of the free variable x for the free variable y by using the tree representation of $\forall x.\, Q(x, y)$

1. The formulas $\forall x.\, P(x)$ and $\forall y.\, P(y)$ have the same tree representation, see fig. 7.3.

2. Substitutions can be carried out without changing the binding of variables, see fig. 7.4. Note that the *bound* variable x is represented by $\langle 0 \rangle$ and thus there is no danger of substituting the *unbound* variable x for the *unbound* variable y. Hence, the problem of accidentally binding a variable does not exist for the tree representation.

7.1.3. Axiomatising Terms and Substitutions

The tree representations of formulas that we saw in section 7.1.1 solves our problems but it is difficult to work with. In fact, these trees are good way of *implementing* FOL on a computer, but not for humans to work with. The aim of this section is to introduce a bunch of axioms that formulas and substitution have to fulfil. These axioms are fulfilled if we were to represent terms by de Bruijn trees, but for the remainder of the course, we will leave unspecified how formulas are implemented.

We begin with the definition of substitutions and how they can be applied to terms.

Definition 7.1

A *substitution* is a map $\sigma\colon \mathrm{Var} \to \mathrm{Term}$. Given $t \in \mathrm{Term}$ and $x \in \mathrm{Var}$, we write $\sigma[x := t]$ for the *updated* substitution defined by

$$(\sigma[x := t])(y) = \begin{cases} t, & x = y \\ \sigma(y), & x \neq y \end{cases}.$$

Similarly, we write $[x := t]$ for the substitution given by

$$[x := t](y) = \begin{cases} t, & x = y \\ y, & x \neq y \end{cases}.$$

Given a term t, we write $t[\sigma]$ for the *application* of the substitution σ to t, defined by iteration on terms as follows.

$$\begin{aligned} x[\sigma] &= \sigma(x) \\ c[\sigma] &= c \\ f(t_1, \dots, t_n)[\sigma] &= f(t_1[\sigma], \dots, t_n[\sigma]) \end{aligned}$$

The notation $\sigma[x := t]$ to update a substitution σ should be read like an assignment in an imperative programming language: the term that σ assigned to x will be overwritten by t. Similarly, the notation $[x := t]$ starts with a storage in which all variables y have the default value y, except for x which gets t assigned as initial value. It should also be noted that there is a substitution $\eta\colon \mathrm{Var} \to \mathrm{Term}$ that assigns to each variable y the *term* y, that is, we have $\eta(y) = y$. The notation $[x := t]$ is then a shorthand for $\eta[x := t]$.

The following example illustrates these notations.

Example 7.2

Let $g(x, y)$ be a term with two free variables x and y, and one binary function symbol g. Moreover, let $\sigma = [x := y][y := x]$.

1. The substitution σ exchanges the two variables:
$$g(x,y)[\sigma] = g(x[\sigma], y[\sigma]) = g(y,x)$$
2. Let now $\tau = \sigma[y := f(x)]$. In τ, the assignment of x to y is overwritten with the assignment of $f(x)$ to y. We thus have the following.
$$g(x,y)[\tau] = g(x[\tau], y[\tau]) = g(y, f(x))$$
3. It is sometimes convenient to exploit the notation and leave out square brackets. For example, instead of $g(x,y)[[y := f(x)]]$ we could write $g(x,y)[y := f(x)]$ and obtain $g(x,y)[y := f(x)] = g(x, f(x))$.

Next, we need to be able to apply substitutions to formulas. To circumvent the difficulties described in section 7.1.1, we *assume* that there is a set of terms on which we can carry out substitutions. This set of terms can *in principle* be defined by appealing to the tree representation. However, this is tedious and instead we just assume that formulas and the application of substitutions fulfil certain *axioms*.

Assumption 7.3

Given a formula φ, a variable x and a substitution σ, we say that x is *fresh* for σ in φ, if we have for all $y \in \mathrm{fv}(\varphi) \setminus \{x\}$ that $x \notin \mathrm{fv}(\sigma(y))$. We assume that there is an operation $\varphi[\sigma]$ that applies a substitution σ to an FOL formula φ. Further, we assume for all $\Diamond \in \{\forall, \exists\}$ and $\Box \in \{\wedge, \vee, \to\}$ that the equality on formulas fulfils the following six axioms.

$$\varphi \Box \psi = \varphi' \Box \psi' \text{ iff } \varphi = \varphi' \text{ and } \psi = \psi' \quad \text{(EC)}$$
$$\Diamond x.\, \varphi = \Diamond y.\, \psi \text{ iff } y \notin \mathrm{fv}(\varphi) \text{ and } \psi = \varphi[x := y] \quad \text{(EQ)}$$
$$\bot[\sigma] = \bot \quad \text{(SB)}$$
$$P(t_1, \dots, t_n)[\sigma] = P(t_1[\sigma], \dots, t_n[\sigma]) \quad \text{(SP)}$$
$$(\varphi \Box \psi)[\sigma] = \varphi[\sigma] \Box \psi[\sigma] \quad \text{(SC)}$$
$$(\Diamond x.\, \varphi)[\sigma] = \Diamond x.\, \varphi[\sigma[x := x]] \text{ if } x \text{ is fresh for } \sigma \text{ in } \varphi \quad \text{(SQ)}$$

Let us explain the intuition behind these axioms. First of all, there are two groups of axioms: those for the equality on formulas (starting with E) and

those that define the action of substitution on formulas (starting with S). The axiom (EQ) allows us to bijectively rename bound variables without illegally binding other variables. For instance, we have

$$\forall x.\, Q(x, y) = \forall z.\, Q(z, y)$$

because $z \notin \mathrm{fv}(Q(x, y))$. However, we have

$$\forall x.\, Q(x, y) \neq \forall y.\, Q(y, y)$$

because $y \in \mathrm{fv}(Q(x, y))$. The axiom (EC) allows us to carry out this renaming also in complex formulas that involve other connectives than quantifiers. Implicitly, we also use equality on terms and atoms, in the sense that

$$P(t_1, \dots, t_n) = P'(t'_1, \dots, t'_n) \text{ iff } P = P' \text{ and } t_k = t'_k \text{ for all } 1 \leq k \leq n$$

and that equality is an equivalence relation (reflexive, symmetric and transitive).

The second group of axioms describes how substitution can be computed iteratively. Axioms (SB), (SP) and (SC) are doing what we would expect: no action on the atom $\bot$, reduce subsitution on predicates to substitution in terms, and distribute substitution over propositional connectives. Complications arise only in the axiom (SQ), which has to make sure that the *use of a bound variable is not changed* and that *variables are not accidentally bound*. On the one hand, that the use of a bound variable is not changed is achieved by updating the substitution σ to $\sigma[x := x]$. For instance, if $\sigma(x) = g(y)$, then naively carrying out this substitution on $\forall x.\, P(x)$ would lead to $\forall x.\, P(g(y))$, which is certainly not what we want! Instead, we have by (SQ) and (SP)

$$(\forall x.\, P(x))[\sigma] = \forall x.\, P(x)[\sigma[x := x]] = \forall x.\, P(x).$$

Accidental binding is preventing, on the other hand, by the precondition that x must be fresh for σ in φ. This condition ensures that none of the terms we want to substitute for the free variables in φ contains the variable x, which would become then bound by the quantifier.

These rules and their interaction are best illustrated through some examples.

Example 7.4

In the following, we use the substitution σ given by

$$\sigma = [y := x][z := m].$$

1. Let $\varphi = \forall z.\, Q(z, y)$. First, we note that $\mathrm{fv}(Q(z,y)) \setminus \{z\} = \{y\}$ and $\mathrm{fv}(\sigma(y)) = \{x\}$. Thus, $z \notin \mathrm{fv}(\sigma(y))$ and z is fresh. This gives us

$$\begin{aligned} \varphi[\sigma] &= \forall z.\, Q(z,y)[\sigma[z := z]] && \text{(SQ)} \\ &= \forall z.\, Q(z[\sigma[z := z]], y[\sigma[z := z]]) && \text{(SP)} \\ &= \forall z.\, Q(z, x) \end{aligned}$$

   Note that $\sigma[z := z] = [y := x]$ and the substitution of m for z in σ was "forgotten" when we applied the substitution under the quantifier. This is intuitively expected, as the bound variable z in φ is a local reference, while the z in σ refers to a global variable z that has the same name but is distinct from the local variable.

2. Let $\psi = \forall x.\, Q(x, y)$. First, we note that $\mathrm{fv}(Q(x,y)) \setminus \{x\} = \{y\}$ and $\mathrm{fv}(\sigma(y)) = \{x\}$. Thus, $x \in \mathrm{fv}(\sigma(y))$ and x is *not* fresh. However, we have $z \notin \mathrm{fv}(Q(x,y))$ and $Q(x,y)[x := z] = Q(z,y)$. This allows us to rename the bound variable x in ψ to z and then carry out the substitution as above:

$$\begin{aligned} \psi[\sigma] &= (\forall z.\, Q(z,y))[\sigma] && \text{(EQ)} \\ &= \forall z.\, Q(z,x) && \text{by 1.} \end{aligned}$$

   Note that we cannot safely rename z back to x, as we would otherwise illegally bind x.

In the remainder of the course, we will not make explicit use of the axioms provided in assumption 7.3. Instead, we will rename *bound* variables, if necessary, before carrying out substitutions. For instance, we would just write

$$(\forall x.\, Q(x,y))[y := x] = \forall z.\, Q(z,x)$$

without explicitly referring to the axioms. However, we know that in cause of doubt, we can always go back to the axioms and formally carry out the renaming and substitution.

7.2. Natural Deduction for FOL

Similarly to propositional log, we want *syntactic proofs* for FOL. As we reason about objects in FOL, we need a new definition of sequents:

Definition 7.5

A first-order *sequent* for a signature $\mathcal{L}$ is a triple

$$\Delta \mid \Gamma \vdash \varphi\,,$$

where Δ is a list of variables in Var, Γ is a list of $\mathcal{L}$-formulas, φ is a $\mathcal{L}$-formula, and $\mathrm{fv}(\Gamma) \cup \mathrm{fv}(\varphi) \subseteq |\Delta|$ for $|\Delta| = \{x \in \mathrm{Var} \mid x : \Delta\}$. If Δ is the empty list $\cdot$, we write

$$\Gamma \vdash \varphi$$

instead of $\cdot \mid \Gamma \vdash \varphi$.

Example 7.6

- This is a sequent:

  $$x, y \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) = \{x, y\}$.

- This is a sequent:

  $$x, y, z \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) = \{x, y\} \subseteq \{x, y, z\}$.

- This is not a sequent:

  $$x \mid \forall z.\, Q(z, y) \vdash P(x)$$

  because $\mathrm{fv}(\forall z.\, Q(z, y)) \cup \mathrm{fv}(P(x)) \not\subseteq \{x\}$.

$$\frac{\varphi : \Gamma}{\Delta \mid \Gamma \vdash \varphi}\ (\text{Assum}) \qquad \frac{\Delta \mid \Gamma \vdash \bot}{\Delta \mid \Gamma \vdash \varphi}\ (\bot\text{E})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \wedge \psi}{\Delta \mid \Gamma \vdash \varphi}\ (\wedge\text{E}_1) \qquad \frac{\Delta \mid \Gamma \vdash \varphi \wedge \psi}{\Delta \mid \Gamma \vdash \psi}\ (\wedge\text{E}_2)$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \quad \Delta \mid \Gamma \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \wedge \psi}\ (\wedge\text{I})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_1) \qquad \frac{\Delta \mid \Gamma \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \vee \psi}\ (\vee\text{I}_2)$$

$$\frac{\Delta \mid \Gamma \vdash \varphi \vee \psi \quad \Delta \mid \Gamma, \varphi \vdash \delta \quad \Delta \mid \Gamma, \psi \vdash \delta}{\Delta \mid \Gamma \vdash \delta}\ (\vee\text{E})$$

$$\frac{\Delta \mid \Gamma, \varphi \vdash \psi}{\Delta \mid \Gamma \vdash \varphi \to \psi}\ (\to\text{I}) \qquad \frac{\Delta \mid \Gamma \vdash \varphi \to \psi \quad \Delta \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \psi}\ (\to\text{E})$$

$$(x \notin \Delta)\ \frac{\Delta, x \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \forall x.\, \varphi}\ (\forall\text{I}) \qquad \frac{\Delta \mid \Gamma \vdash \forall x.\, \varphi}{\Delta \mid \Gamma \vdash \varphi[x := t]}\ (\forall\text{E})$$

$$\frac{\Delta \mid \Gamma \vdash \varphi[x := t]}{\Delta \mid \Gamma \vdash \exists x.\, \varphi}\ (\exists\text{I}) \qquad (x \notin \Delta)\ \frac{\Delta \mid \Gamma \vdash \exists x.\, \varphi \quad \Delta, x \mid \Gamma, \varphi \vdash \psi}{\Delta \mid \Gamma \vdash \psi}\ (\exists\text{E})$$

Figure 7.5.: Deduction Rules of the natural deduction system $\mathbf{ND}_1$

Definition 7.7: Intuitionistic natural deduction for FOL

The system $\mathbf{ND}_1$ for FOL is given by the rules in fig. 7.5, where the label $x \notin \Delta$ in the rules ($\forall$I) and ($\exists$E) are side-conditions that have to be fulfilled to apply those rules. However, these side-condition will not be displayed in proof trees.

Note about fig. 7.5:

- In ($\forall$E) and ($\exists$I), the definition of a sequent ensures in $\Delta \mid \Gamma \vdash \varphi[x := t]$ that all the free variables of t appear in Δ.

- Similarly, it is ensured in (∃E) that the variable x does not occur freely in ψ because $\Delta \mid \Gamma \vdash \psi$ is a sequent.

7.2.1. The Intuitionistic System $\mathbf{ND}_1$

Empty

7.2.2. Fitch-Style Deduction for $\mathbf{ND}_1$

Empty

7.2.3. The Classical System $\mathbf{cND}_1$

Definition 7.8

The system $\mathbf{cND}_1$ of natural deduction for *classical* first-order logic is given by the system $\mathbf{ND}_1$ together with the contradiction rule:

$$\frac{\Delta \mid \Gamma, \neg\varphi \vdash \bot}{\Delta \mid \Gamma \vdash \varphi}\ \text{(Contra)}$$

7.3. Exercises

The following exercises allow you to practise the material of this chapter.

Exercise 1

Give the de Bruijn-tree representations of the following formulas.

a) $Q(x, g(m))$

b) $\forall x.\, Q(x, g(m))$

c) $(\exists x.\, Q(x, g(m))) \wedge \forall y.\, Q(y, z)$

d) $\exists y.\, Q(y, g(z)) \wedge \forall x.\, Q(y, f(x, z))$

e) $\forall x.\, \exists x.\, P(x)$

Exercise 2

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y].$$

Evaluate the following applications of σ to terms.

a) $x[\sigma]$ b) $f(x, y)[\sigma]$ c) $(y[\sigma])[\sigma]$ d) $x[\sigma[x := x]]$

Exercise 3

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y]$$

and φ by

$$\varphi = \forall x.\, Q(y, z).$$

Determine whether

a) x is fresh for σ in φ.

b) y is fresh for σ in φ.

c) z is fresh for σ in φ.

Exercise 4

Let σ be given by

$$\sigma = [x := g(x)][y := x][z := y].$$

Carry out the following substitutions.

a) $Q(x, g(m))[\sigma]$

b) $(\forall x.\, Q(x, g(m)))[\sigma]$

c) $((\exists x.\, Q(x, g(m))) \land \forall y.\, Q(y, z))[\sigma]$

d) $(\exists y.\, Q(y, g(z)) \land \forall x.\, Q(y, f(x, z)))[\sigma]$

8. Semantics of First-Order Logic

In this chapter, we will discuss the following two questions:

1. When is a formula valid?
2. Are all derivable formulas valid?

The first question is answered by giving semantics to formulas in terms of Boolean values $\mathbb{B}$, as we did for propositional logic. The second question concerns the soundness of our proof systems $\mathbf{ND}_1$ and $\mathbf{cND}_1$, again as for propositional logic.

Before we can give semantics and prove soundness, we need to know when predicates are true. This, in turn, requires us to know which objects terms designates. We obtain both from models of FOL.

8.1. Models of First-Order Logic

Recall that in propositional logic a valuation $v \colon \mathrm{PVar} \to \mathbb{B}$ on propositional variables determined the truth value of formulas. In first-order logic we need to provide interpretations instead for object variables, terms and predicates. Finding the right structures to do so is the subject of this section.

First, we recall some notation.

Notation 8.1
We denote by $\mathbb{1}$ the singleton set $\{*\}$. Given a set A and $n \in \mathbb{N}$, we write A^n for the n-fold product of A defined inductively by the following two equations.

$$A^0 = \mathbb{1}$$
$$A^{n+1} = A \times A^n$$

We identify A^1 with A for simplicity.

For example, we then have

$$A^2 = A \times A^1 = A \times A = \{(a, b) \mid a, b \in A\}.$$

Definition 8.2

Let $\mathcal{L}$ be a signature with $\mathcal{L} = (\mathcal{F}, \mathcal{R}, \mathrm{ar})$. An *$\mathcal{L}$-model* $\mathcal{M}$ consists of

- a non-empty set U, the *universe* of $\mathcal{M}$
- for each $f \in \mathcal{F}$ a map
$$f^{\mathcal{M}} \colon U^{\mathrm{ar}(f)} \to U$$
- for each $P \in \mathcal{R}$ a predicate
$$P^{\mathcal{M}} \subseteq U^{\mathrm{ar}(P)}$$

We also write $|\mathcal{M}|$ for U.

Note that we require in definition 8.2 the universe to be non-empty. This restriction could in principle be dropped but that leads to some technical difficulties, as we need to rule out constants in this case.

Further, observe that if $c \in \mathcal{F}$ is a constant, then $c^{\mathcal{M}}$ is a map of type $\mathbb{1} \to U$. Since $\mathbb{1}$ has one element, providing such a map corresponds to providing one element $a \in U$:

$$c^{\mathcal{M}}(*) = a$$

Let us unfold the definition for some specific arities.

Example 8.3

Suppose our signature $\mathcal{L}$ consists of three function symbols c, f and g with arity 0, 1 and 2, respectively, and three predicate symbols P, Q, R also of arity 0, 1 and 2. An $\mathcal{L}$-model $\mathcal{M}$ consists of a universe U, maps and predicates as in table 8.1.
We see that $c^{\mathcal{M}}$ denotes one element $c^{\mathcal{M}}(*) \in U$, as discussed above; $f^{\mathcal{M}}$ is a unary map; and $g^{\mathcal{M}}$ is a binary map. Moreover, we find that $P^{\mathcal{M}}$ is either the empty set $\emptyset$ or the singleton set $\mathbb{1}$. In other words, P is nothing but a *propositional variable*! Finally, $Q^{\mathcal{M}}$ is a predicate, or unary relation, while $R^{\mathcal{M}}$ is a binary relation.

	Symbol s	ar(s)	Interpretation type
Function Symbols	c	0	$c^{\mathcal{M}} : \mathbb{1} \to U$
	f	1	$f^{\mathcal{M}} : U \to U$
	g	2	$g^{\mathcal{M}} : U^2 \to U$
Predicate Symbols	P	0	$P^{\mathcal{M}} \subseteq \mathbb{1}$
	Q	1	$Q^{\mathcal{M}} \subseteq U$
	R	2	$R^{\mathcal{M}} \subseteq U^2$

Table 8.1.: Data of a model for the indicated signature

?

Let $\mathcal{L} = (\emptyset, \mathcal{R}, \mathrm{ar})$ with $\mathcal{R} = \{P\}$ and $\mathrm{ar}(P) = 0$. How many possibilities are there to make a model for $\mathcal{L}$?

In the next example, we discuss a signature and two models that occur "in the wild".

Example 8.4

Let $\mathcal{L}$ be given as follows.

$$\mathcal{F} = \{\underline{0}, \underline{1}, p\} \qquad \mathcal{R} = \{I, L\}$$
$$\mathrm{ar}(\underline{0}) = \mathrm{ar}(\underline{1}) = 0 \qquad \mathrm{ar}(p) = \mathrm{ar}(I) = \mathrm{ar}(L) = 2$$

The interpretation of this signature could be that $\underline{0}$ and $\underline{1}$ stand for the numbers 0 and 1, p for addition (plus), I for equality (identity), and L for less-than. Indeed, we can give such an interpretation, which leads to the model $\mathcal{M}_a$ in table 8.2 with universe $|\mathcal{M}_a| = \mathbb{N}$.

It is not necessary to interpret $\mathcal{L}$ as in example 8.4 and we can give different meanings to the symbols and even the universe.

Symbol	Interpretation type	Interpretation
$\underline{0}$	$\underline{0}^{\mathcal{M}_a} : \mathbb{1} \to \mathbb{N}$	$\underline{0}^{\mathcal{M}_a}(*) = 0$
$\underline{1}$	$\underline{1}^{\mathcal{M}_a} : \mathbb{1} \to \mathbb{N}$	$\underline{1}^{\mathcal{M}_a}(*) = 1$
p	$p^{\mathcal{M}_a} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$	$p^{\mathcal{M}_a}(n, m) = n + m$
I	$I^{\mathcal{M}_a} \subseteq \mathbb{N} \times \mathbb{N}$	$I^{\mathcal{M}_a} = \{(n, m) \mid n = m\}$
L	$L^{\mathcal{M}_a} \subseteq \mathbb{N} \times \mathbb{N}$	$L^{\mathcal{M}_a} = \{(n, m) \mid n \leq m\}$

Table 8.2.: Arithmetic model $\mathcal{M}_a$ over universe $\mathbb{N}$

Example 8.5

We show in this example how to use the same signature $\mathcal{L}$ as in example 8.4 to reason about formal languages, where we use the notation introduced in appendix A.1: The function symbols $\underline{0}$, $\underline{1}$ and p correspond to, respectively, the empty language, the language containing only the empty and union of languages. The predicate symbols, on the other hand, can be interpreted as language equality and language inclusion. All of this is summed up in table 8.3. Note that that $\underline{0}$ and p behave similarly to 0 and addition under this interpretation. We will discuss this later in more depth.

Symbol	Interpretation type	Interpretation
$\underline{0}$	$\underline{0}^{\mathcal{M}_l} : \mathbb{1} \to \mathcal{P}(A^*)$	$\underline{0}^{\mathcal{M}_l}(*) = \emptyset$
$\underline{1}$	$\underline{1}^{\mathcal{M}_l} : \mathbb{1} \to \mathcal{P}(A^*)$	$\underline{1}^{\mathcal{M}_l}(*) = \{\varepsilon\}$
p	$p^{\mathcal{M}_l} : \mathcal{P}(A^*)^2 \to \mathcal{P}(A^*)$	$p^{\mathcal{M}_l}(L_1, L_2) = L_1 \cup L_2$
I	$I^{\mathcal{M}_l} \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$	$I^{\mathcal{M}_l} = \{(L_1, L_2) \mid L_1 = L_2\}$
L	$L^{\mathcal{M}_l} \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$	$L^{\mathcal{M}_l} = \{(L_1, L_2) \mid L_1 \subseteq L_2\}$

Table 8.3.: Language model $\mathcal{M}_l$ for $\mathcal{L}$ over universe $\mathcal{P}(A^*)$

Clearly, the two models $\mathcal{M}_a$ and $\mathcal{M}_l$ in example 8.4 and example 8.5 are completely different interpretations of $\mathcal{L}$. This illustrates the power of first-order logic: one language we can reason about an enormous variety of different

structures. However, as we will see in chapter 9, this power can also become a weakness of first-order logic.

8.2. Valuations and the Interpretion of FOL

Just as propositional variables in propositional logic, the object variables in first-order formulas have no intrinsic meaning. Instead, we have to give meaning to them externally through valuations, which assign to each variable an element of a given universe.

Definition 8.6

Given a signature $\mathcal{L}$ and an $\mathcal{L}$-model $\mathcal{M}$, an *$\mathcal{M}$-valuation*, or simply *valuation*, is a map v of the following type.

$$v \colon \mathrm{Var} \to |\mathcal{M}|$$

With interpretations of variables at our disposal, we can understand also the meaning of terms.

Definition 8.7

A valuation v in a model $\mathcal{M}$ extends to the *semantics of terms* $[\![-]\!]_v^{\mathcal{M}} \colon \mathrm{Term} \to |\mathcal{M}|$ by iteration on terms as follows.

$$\begin{aligned} [\![x]\!]_v^{\mathcal{M}} &= v(x) \\ [\![c]\!]_v^{\mathcal{M}} &= c^{\mathcal{M}}(*) \\ [\![f(t_1, \dots, t_n)]\!]_v^{\mathcal{M}} &= f^{\mathcal{M}}([\![t_1]\!]_v^{\mathcal{M}}, \dots, [\![t_n]\!]_v^{\mathcal{M}}) \end{aligned}$$

If $\mathcal{M}$ is clear from the context, then we just write $[\![-]\!]_v$ instead of $[\![-]\!]_v^{\mathcal{M}}$.

Let us demonstrate valuations and the term semantics for the models in examples 8.4 and 8.5.

Example 8.8

We begin with the arithmetic model $\mathcal{M}_a$ from example 8.4. Let $x \in \text{Var}$ be some variable and define

$$v(y) = \begin{cases} 5, & x = y \\ 0, & x \neq y \end{cases}$$

Under this valuation, the term $p(\underline{0}, p(x, \underline{1}))$ gets the following semantics assigned.

$$\begin{aligned} [\![p(\underline{0}, p(x, \underline{1}))]\!]_v &= [\![\underline{0}]\!]_v + [\![p(x, \underline{1})]\!]_v \\ &= [\![\underline{0}]\!]_v + ([\![x]\!]_v + [\![\underline{1}]\!]_v) \\ &= \underline{0}^{\mathcal{M}_a}(*) + (v(x) + \underline{1}^{\mathcal{M}_a}(*)) \\ &= 0 + (5 + 1) \\ &= 6 \end{aligned}$$

The next example provides semantics for the same term but in the language model.

Example 8.9

Let $x \in \text{Var}$ be some variable and define

$$v(y) = \begin{cases} A, & x = y \\ \emptyset, & x \neq y \end{cases}$$

Under this valuation, the term $p(\underline{0}, p(x, \underline{1}))$ gets the following semantics assigned.

$$\begin{aligned} [\![p(\underline{0}, p(x, \underline{1}))]\!]_v &= [\![\underline{0}]\!]_v \cup [\![p(x, \underline{1})]\!]_v \\ &= [\![\underline{0}]\!]_v \cup ([\![x]\!]_v \cup [\![\underline{1}]\!]_v) \\ &= \underline{0}^{\mathcal{M}_l}(*) \cup (v(x) \cup \underline{1}^{\mathcal{M}_l}(*)) \\ &= \emptyset \cup (A \cup \{\varepsilon\}) \\ &= \{w \in A^* \mid \text{length}(w) \leq 1\} \end{aligned}$$

Here, $\text{length}(w)$ is the length of the word w.

Now that we have an interpretation of terms, we can further extend it to an

interpretation on formulas. Recall that we needed to update substitutions to eliminate universal quantifiers and introduce existential quantifiers. A similar operation on valuations is necessary in the semantics of FOL formulas.

Definition 8.10

Let v be an $\mathcal{M}$ valuation, $x \in \mathrm{Var}$ and $a \in |\mathcal{M}|$. We define the *update of valuations* on v by the following equation.

$$(v[x \mapsto a])(y) = \begin{cases} a, & y = x \\ v(y), & y \neq x \end{cases}$$

Let us briefly illustrate the update of valuations.

Example 8.11

We start with the valuation $v \colon \mathrm{Var} \to \mathbb{N}$ given by $v(y) = 0$ for all $y \in \mathrm{Var}$. Given variables $x, z \in \mathrm{Var}$ with $x \neq z$, we have

$$\begin{aligned}
(v[x \mapsto 1])(x) &= 1 \\
(v[x \mapsto 1])(z) &= 0 \\
(v[x \mapsto 1][z \mapsto 2])(x) &= 1 \\
(v[x \mapsto 1][z \mapsto 2])(z) &= 2 \\
(v[x \mapsto 1][z \mapsto 2][x \mapsto 3])(x) &= 3 \\
(v[x \mapsto 1][z \mapsto 2][x \mapsto 3])(z) &= 2
\end{aligned}$$

The update operation of valuations allows us now to give semantics to quantifiers and thereby to all formulas.

Definition 8.12

Let $\mathcal{L}$ be a signature and $\mathcal{M}$ an $\mathcal{L}$-model. We define for all $\mathcal{M}$-valuations v a map

$$[\![-]\!]_v \colon \mathrm{Form} \to \mathbb{B}$$

by iteration on formulas.

$$
\begin{aligned}
[\![\bot]\!]_v &= 0 \\
[\![P(t_1, \dots, t_n)]\!]_v &= \begin{cases} 1, & ([\![t_1]\!]_v, \dots, [\![t_n]\!]_v) \in P^{\mathcal{M}} \\ 0, & \text{otherwise} \end{cases} \\
[\![\varphi \wedge \psi]\!]_v &= \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\
[\![\varphi \vee \psi]\!]_v &= \max\{[\![\varphi]\!]_v, [\![\psi]\!]_v\} \\
[\![\varphi \to \psi]\!]_v &= [\![\varphi]\!]_v \implies [\![\psi]\!]_v \\
[\![\forall x.\, \varphi]\!]_v &= \min\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\} \\
[\![\exists x.\, \varphi]\!]_v &= \max\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in |\mathcal{M}|\}
\end{aligned}
$$

Note that the minimum and maximum are always taken over a non-empty subset of $\mathbb{B}$. The possibilities that arise in the semantics of quantifiers are summed up in table 8.4 together with the corresponding quantification. Note

$\{[\![\varphi]\!]_{v[x \mapsto a]} \mid a \in \lvert\mathcal{M}\rvert\}$	$[\![\forall x.\, \varphi]\!]_v$	$[\![\exists x.\, \varphi]\!]_v$	Interpretation
$\{0\}$	0	0	φ holds for no a
$\{1\}$	1	1	φ holds for all a
$\{0, 1\}$	0	1	φ holds for some a

Table 8.4.: Possibilities for quantifier semantics

that if would allow the universe to be empty, then we would have a fourth option in the table, in which the universal quantifier would be always true and the existential quantifier always false.

Let us now calculate the truth value of some formulas.

Example 8.13

Recall the signature $\mathcal{L}$ and arithmetic model $\mathcal{M}_a$ from example 8.4. Consider the formula

$$\forall x.\, I(p(x, \underline{0}), x)$$

that expresses under the arithmetical interpretation that $x + 0 = x$. In other words, the formula should be true for any valuation over $\mathcal{M}_a$.

Indeed, given a valuation v and a natural number n, we have

$$[\![p(x, \underline{0})]\!]_{v[x \mapsto n]} = 0 + n = n$$

and thus

$$\begin{aligned}
[\![I(p(x, \underline{0}), x)]\!]_{v[x \mapsto n]} &= \begin{cases} 1, & \left([\![p(x, \underline{0})]\!]_{v[x \mapsto n]}, [\![x]\!]_{v[x \mapsto n]}\right) \in I^{\mathcal{M}_a} \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & n = n \\ 0, & \text{otherwise} \end{cases} \\
&= 1.
\end{aligned}$$

As expected, this gives us

$$\begin{aligned}
[\![\forall x.\, I(p(x, \underline{0}), x)]\!]_v &= \min\{[\![I(p(x, \underline{0}), x)]\!]_{v[x \mapsto n]} \mid n \in \mathbb{N}\} \\
&= \min\{1\} \\
&= 1
\end{aligned}$$

and thus $\forall x.\, I(p(x, \underline{0}), x)$ is true in $\mathcal{M}_a$.
As a second example, consider the formula

$$\forall x.\, \exists y.\, L(x, y) \wedge \neg I(x, y)$$

that states that for every number there is a strictly larger number. This formula is true in the arithmetic model because for every $n \in \mathbb{N}$ we have that $n < n + 1$. In other words, for every valuation v and $n \in \mathbb{N}$

$$[\![L(x, y) \wedge \neg I(x, y)]\!]_{v[x \mapsto n][y \mapsto n+1]} = 1.$$

Thus,

$$\begin{aligned}
&[\![\exists y.\, L(x, y) \wedge \neg I(x, y)]\!]_{v[x \mapsto n]} \\
&= \max\{[\![L(x, y) \wedge \neg I(x, y)]\!]_{v[x \mapsto n][y \mapsto m]} \mid m \in \mathbb{N}\} \\
&= 1 \qquad \text{because } n + 1 \in \mathbb{N}
\end{aligned}$$

and

$$\begin{aligned}
&[\![\forall x.\, \exists y.\, L(x, y) \wedge \neg I(x, y)]\!]_v \\
&= \min\{[\![\exists y.\, L(x, y) \wedge \neg I(x, y)]\!]_{v[x \mapsto n]} \mid n \in \mathbb{N}\} \\
&= \min\{1\} \\
&= 1
\end{aligned}$$

?

Is the formula $\forall x.\, \exists y.\, L(x,y) \wedge \neg I(x,y)$ from example 8.13 true in the language model $\mathcal{M}_l$?

8.3. Entailment and Satisfiability for FOL

In section 8.2, we have defined the functional interpretation of first-order formulas in terms of the mapping $[\![-]\!]$. As for propositional logic, we can also give a relational interpretation of formulas. This allows us to easily define satisfiability and tautologies.

Definition 8.14

Let φ be a formula and Γ as set of formulas over $\mathcal{L}$. We define *semantic entailment* relations, where $[\![\Gamma]\!]_v = \{[\![\psi]\!]_v \mid \psi \in \Gamma\}$.

$\mathcal{M}, v \vDash \varphi$	if $[\![\varphi]\!]_v^{\mathcal{M}} = 1$	($\mathcal{M}$ and v *satisfy* φ)
$\mathcal{M} \vDash \varphi$	if $\mathcal{M}, v \vDash \varphi$ for all valuations v	($\mathcal{M}$ *satisfies* φ)
$\Gamma \vDash \varphi$	if $\min([\![\Gamma]\!]_v^{\mathcal{M}}) \leq [\![\varphi]\!]_v^{\mathcal{M}}$ for all models $\mathcal{M}$ and valuations v	(Γ *entails* φ)

Let us give some examples in the arithmetic model.

Example 8.15

Let φ_e be the formula $\exists y.\, I(x, p(y,y))$ ("x is even"), and let v_1 and v_2 be given as follows.

$$v_1(z) = \begin{cases} 2, & z = y \\ 1, & z \neq y \end{cases} \qquad v_2(z) = \begin{cases} 3, & z = y \\ 1, & z \neq y \end{cases}$$

Then $\mathcal{M}_a$ and v_1 satisfy φ_e, but $\mathcal{M}_a$ and v_2 do not.
Next, we have seen in example 8.13 that $[\![\forall x.\, I(p(x, \underline{0}), x)]\!]_v^{\mathcal{M}_a} = 1$ for any valuation v. Thus $\mathcal{M}_a \vDash \forall x.\, I(p(x, \underline{0}), x)$. In contrast, not every natural number is even. Thus, $\mathcal{M}_a$ does not satisfy φ_e and $\mathcal{M}_a \vDash \varphi_e$ does not hold.
Finally, let $\Gamma = \{\varphi_e\}$ and $\varphi_o = \exists y.\, I(x, p(p(y,y), \underline{1}))$. Then $\Gamma \vDash$

$\varphi_o[x := p(x, \underline{1})]$ holds: If $\min([\![\Gamma]\!]_v) = 1$, then $v(x)$ must be an even number. Thus, $v(x) + 1$ is an odd number and $[\![\varphi_o[x := p(x, \underline{1})]]\!]_v = 1$. Formally, we have

$$
\begin{aligned}
&\min([\![\Gamma]\!]_v) = 1 \\
&\text{iff } [\![\varphi_e]\!]_v = 1 \\
&\text{iff } \min\{[\![I(x, p(y, y))]\!]_{v[y \mapsto n]} \mid n \in \mathbb{N}\} = 1 \\
&\text{iff } [\![I(x, p(y, y))]\!]_{v[y \mapsto n]} = 1 \text{ for some } n \in \mathbb{N} \qquad \text{(see table 8.4)} \\
&\text{iff } v(x) = n + n \text{ for some } n \in \mathbb{N} \\
&\text{iff } v(x) \text{ even}
\end{aligned}
$$

and under this assumption

$$
\begin{aligned}
[\![\varphi_o[x := p(x, \underline{1})]]\!]_v &= [\![(\exists y.\, I(x, p(p(y, y), \underline{1})))[x := p(x, \underline{1})]]\!]_v \\
&= [\![\exists y.\, I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_v \\
&= \max\{[\![I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_{v[y \mapsto n]} \mid n \in \mathbb{N}\} \\
&\geq [\![I(p(x, \underline{1}), p(p(y, y), \underline{1}))]\!]_{v[y \mapsto v(x)/2]} \\
&= 1,
\end{aligned}
$$

where we use the identity $v(x) + 1 = (v(x)/2 + v(x)/2) + 1$ in the last line.

As in the case of propositional logic, we can designate classes of formulas that hold somewhere or everywhere.

Definition 8.16

Let φ be a formula over $\mathcal{L}$. We say that φ is

- *satisfiable*, if there is a model $\mathcal{M}$ with $\mathcal{M} \models \varphi$.
- *valid* or a *tautology*, written $\models \varphi$, if every model satisfies φ:

$$\models \varphi \quad \text{if} \quad \mathcal{M} \models \varphi \text{ for all models } \mathcal{M}.$$

Note that the definition of tautology corresponds to entailment for the empty

set of assumptions, that is

$$\vDash \varphi \text{ holds if and only if } \emptyset \vDash \varphi.$$

We have seen already some formulas that were satisfied in the arithmetic model, but are not necessarily tautologies.

Example 8.17

Since $\mathcal{M}_a \vDash \forall x.\, I(p(x, \underline{0}), x)$, the formula $\forall x.\, I(p(x, \underline{0}), x)$ is satisfiable, see example 8.15. However, it is not a tautology because it is not satisfied by the model $\mathcal{M}$ with $|\mathcal{M}| = \mathbb{N}$, $\underline{0}^{\mathcal{M}} = 1$ and otherwise the same interpretation as in the arithmetic model $\mathcal{M}_a$.

In chapter 7, we have seen formulas that were derivable in $\mathbf{ND}_1$, which are a good source of tautologies, see also section 8.4 below.

Example 8.18

We claim that the formula $(\forall x.\, \varphi) \to \neg\exists y.\, \neg\varphi$ is a tautology for any formula φ. Indeed, let $\mathcal{M}$ be a model for the signature $\mathcal{L}$, over which φ is a formula, and v a valuation in $\mathcal{M}$. We obtain from table 8.4 that

$$\begin{aligned}
[\![\neg\exists y.\, \neg\varphi]\!]_v = 1 &\text{ iff } [\![\exists y.\, \neg\varphi]\!]_v = 0 \\
&\text{iff } [\![\neg\varphi]\!]_{v[y\mapsto b]} = 0 \text{ for all } b \in |\mathcal{M}| \\
&\text{iff } [\![\varphi]\!]_{v[y\mapsto b]} = 1 \text{ for all } a \in |\mathcal{M}| \\
&\text{iff } [\![\forall x.\, \varphi]\!]_v = 1
\end{aligned}$$

This implies that

$$[\![\forall x.\, \varphi]\!]_v \leq [\![\neg\exists y.\, \neg\varphi]\!]_v$$

and thus

$$\begin{aligned}
&[\![(\forall x.\, P(x)) \to \neg\exists y.\, \neg P(y)]\!]_v \\
&= [\![(\forall x.\, P(x))]\!]_v \implies [\![\neg\exists y.\, \neg P(y)]\!]_v \\
&= 1.
\end{aligned}$$

Hence, $(\forall x.\, \varphi) \to \neg\exists y.\, \neg\varphi$ is a tautology.

8.4. Soundness of Natural Deduction for FOL

We come now to the second initial question: Are all derivable formulas valid? Using definition 8.14, we can state this question precisely by asking: if there is a proof for the sequent $\Gamma \vdash \varphi$ in one of the systems from chapter 7, is φ then entailed semantically by Γ? The answer to this question is the main result of this chapter.

Theorem 8.19: Soundness

For every formula φ and list of formulas Γ over a signature $\mathcal{L}$

1. if $\Gamma \vdash \varphi$ is derivable in $\mathbf{ND}_1$, then $\Gamma \vDash \varphi$.
2. if $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$, then $\Gamma \vDash \varphi$.

Instantiating theorem 8.19 with the empty list of assumptions, we obtain the following corollary.

Corollary 8.20

For every formula φ over a signature $\mathcal{L}$ the following holds.

1. If $\vdash \varphi$ is derivable in $\mathbf{ND}_1$, then φ is a tautology.
2. If $\vdash \varphi$ is derivable in $\mathbf{cND}_1$, then φ is a tautology.

The proof of theorem 8.19 requires some interesting results that essentially show that substitutions are the syntactic counterpart of valuations. This is proved below in lemma 8.21 for terms and in lemma 8.22 for formulas.

Lemma 8.21
For all substitutions σ, terms t, valuations v and variables x, where x is fresh for σ in t, the following equation holds.

$$[\![t[\sigma]]\!]_v = [\![t[\sigma[x := x]]]\!]_{v[x \mapsto [\![\sigma(x)]\!]_v]}$$

Proof. We let $\sigma' = \sigma[x := x]$ and $v' = v[x \mapsto [\![\sigma(x)]\!]_v]$, which means that we have to prove

$$[\![t[\sigma]]\!]_v = [\![t[\sigma']]\!]_{v'}$$

This proof proceeds by induction on the term t.

- In the base case we have for a variable y:

$$\begin{aligned}
[\![y[\sigma]]\!]_v &= [\![\sigma(y)]\!]_v && \text{def. substitution}\\
&= \begin{cases} [\![\sigma(x)]\!]_v, y = x \\ [\![\sigma(y)]\!]_v, y \neq x \end{cases} \\
&= \begin{cases} [\![x]\!]_{v'}, y = x \\ [\![\sigma(y)]\!]_v, y \neq x \end{cases} && \text{def. } v' \\
&= \begin{cases} [\![x]\!]_{v'}, y = x \\ [\![\sigma(y)]\!]_{v'}, y \neq x \end{cases} && x \text{ fresh for } \sigma \\
&= [\![y[\sigma']]\!]_{v'} && \text{def. substitution update}
\end{aligned}$$

- In the induction step, we have

$$\begin{aligned}
&[\![f(t_1, \dots, t_n)[\sigma]]\!]_v \\
&= [\![f(t_1[\sigma], \dots, t_n[\sigma])]\!]_v && \text{def. substitution}\\
&= f^{\mathcal{M}}([\![t_1[\sigma]]\!]_v, \dots, [\![t_n[\sigma]]\!]_v) && \text{def. semantics}\\
&= f^{\mathcal{M}}([\![t_1[\sigma']]\!]_{v'}, \dots, [\![t_n[\sigma']]\!]_{v'}) && \text{induction hypothesis}\\
&= [\![f(t_1, \dots, t_n)[\sigma']]\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

Thus, by induction on t, the sought after identity $[\![t[\sigma]]\!]_v = [\![t[\sigma']]\!]_{v'}$ holds. □

Lemma 8.22
For all substitutions σ, formulas φ, valuations v and variables x, where x is fresh for σ in φ, the following equation holds.

$$[\![\varphi[\sigma]]\!]_v = [\![\varphi[\sigma[x := x]]]\!]_{v[x \mapsto [\![\sigma(x)]\!]_v]}$$

Proof. As in lemma 8.21, we let $\sigma' = \sigma[x := x]$ and $v' = v[x \mapsto [\![\sigma(x)]\!]_v]$, and then prove

$$[\![\varphi[\sigma]]\!]_v = [\![\varphi[\sigma']]\!]_{v'}$$

by induction on φ.

- In the predicate base case, we have

$$\begin{aligned}
[\![P(t_1,\dots,t_n)[\sigma]]\!]_v &= [\![P(t_1[\sigma],\dots,t_n[\sigma])]\!]_v && \text{def. substitution}\\
&= P^{\mathcal{M}}([\![t_1[\sigma]]\!]_v,\dots,[\![t_n[\sigma]]\!]_v) && \text{def. semantics}\\
&= P^{\mathcal{M}}([\![t_1[\sigma']]\!]_{v'},\dots,[\![t_n[\sigma']]\!]_{v'}) && \text{lemma 8.21}\\
&= [\![P(t_1,\dots,t_n)[\sigma']]\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

- The base case for $\bot$ is trivial: $[\![\bot[\sigma]]\!]_v = 0 = [\![\bot[\sigma']]\!]_{v'}$.

- The cases for conjunction, disjunction and implication are immediate by the induction hypothesis. We write $\mathbb{B}_\wedge = \min$, $\mathbb{B}_\vee = \max$ and $\mathbb{B}_\to(x,y) = x \Rightarrow y$, which are the binary Boolean functions for their corresponding connective. This gives us

$$\begin{aligned}
[\![(\varphi_1 \square \varphi_2)[\sigma]]\!]_v &= \mathbb{B}_\square([\![\varphi_1[\sigma]]\!]_v, [\![\varphi_2[\sigma]]\!]_v) && \text{def. subst. and semantics}\\
&= \mathbb{B}_\square([\![\varphi_1[\sigma']]\!]_{v'}, [\![\varphi_2[\sigma']]\!]_{v'}) && \text{induction hyp.}\\
&= [\![(\varphi_1 \square \varphi_2)[\sigma']]\!]_{v'} && \text{def. semantics and subst.}
\end{aligned}$$

- For quantifiers assume that y is fresh for σ in ψ.

  Before we continue, observe that for any $a \in U$, we can define $w = v[y \mapsto a]$ and $w' = w[x \mapsto [\![\sigma(x)]\!]_v]$. By using the induction hypothesis for ψ with w, we obtain $[\![\psi[\sigma]]\!]_w = [\![\psi[\sigma']]\!]_{w'}$. Since x and y are fresh, we have that $w' = v'[y \mapsto a]$. Thus $[\![\psi[\sigma]]\!]_w = [\![\psi[\sigma']]\!]_{v'[y\mapsto a]}$.

  With this observation, we have

$$\begin{aligned}
[\![(\forall y.\, \psi)[\sigma]]\!]_v &= [\![\forall y.\, \psi[\sigma]]\!]_v && \text{(SQ)}\\
&= \min\{[\![\psi[\sigma]]\!]_{v[y\mapsto a]} \mid a \in U\} && \text{def. semantics}\\
&= \min\{[\![\psi[\sigma']]\!]_{v'[y\mapsto a]} \mid a \in U\} && \text{see above}\\
&= [\![(\forall y.\, \psi)[\sigma']]\!]_{v'} && \text{def. semantics and (SQ)}
\end{aligned}$$

  The same reasoning, replacing min by max, gives us the result also for the existential quantifier.

This concludes the induction and thereby the proof. □

Proof of theorem 8.19. We have to generalise the statement and prove that $\Delta \mid \Gamma \vdash \varphi$ implies $\Gamma \vDash \varphi$. To this end, we proceed by induction on the proof tree for $\Delta \mid \Gamma \vdash \varphi$ in $\mathbf{cND}_1$. The statement for $\mathbf{ND}_1$ follows from this.

Thus, assume that we are given a proof tree for $\Delta \mid \Gamma \vdash \varphi$. We have to show for all models $\mathcal{M}$ and valuations v in $\mathcal{M}$ that $\min(\llbracket\Gamma\rrbracket_v) \leq \llbracket\varphi\rrbracket_v$. Most of the cases are dealt with in the same way as for propositional logic and we only treat the rules for quantifier here.

- Suppose the proof tree ends in

$$\frac{\Delta, x \mid \Gamma \vdash \varphi}{\Delta \mid \Gamma \vdash \forall x.\, \varphi}\ (\forall\mathrm{I})$$

  where x does not appear in Δ. We now have

$$\begin{aligned} \llbracket\forall x.\, \varphi\rrbracket_v &= \min\{\llbracket\varphi\rrbracket_{v[x\mapsto a]} \mid a \in |\mathcal{M}|\} && \text{def. semantics} \\ &\geq \min\{\min(\llbracket\Gamma\rrbracket)_{v[x\mapsto a]} \mid a \in |\mathcal{M}|\} && \text{by IH} \\ &= \min(\llbracket\Gamma\rrbracket)_v, && x \text{ not in } \Gamma \end{aligned}$$

  where we use the induction hypothesis (IH) for $\Delta, x \mid \Gamma \vdash \varphi$ with model $\mathcal{M}$ and valuation $v[x \mapsto a]$. This identity gives us $\Gamma \vDash \forall x.\, \varphi$.

- TBD

□

9. Extensions and Limits of First-Order Logic

First-order logic with the formal proof system **cND$_1$** is at the same time a strong and expressive logic, but at the same time also severely limited. In this chapter, we will see where this seeming contradiction comes from.

Before we get to that, we will first extend the syntax of first-order logic by a special predicate that allows us to reason about the identity of objects.

9.1. First-Order Logic with Equality

Recall that we used in example 8.4 a binary predicate symbol I to express that two objects are equal. In the arithmetic model $\mathcal{M}_a$, we also interpreted I as the equality of numbers. However, a general model $\mathcal{M}$ is not forced to give this interpretation to I, it can interpret I as any binary relation. For example, the interpretation as inequality is perfectly valid but is the exact opposite of our intention: $I^{\mathcal{M}} = \{(a, b) \mid a \neq b\}$.

This problem can be fixed by giving equality a special status and making it part of the syntax and proof system of first-order logic. To this end, we extend the first-order formulas by one extra kind atomic formula of the form $t \doteq s$ for terms s and t, which is a logical formula with the intent of expressing that t is equal to s. We use this special notation the to carefully distinguish the syntactic equality $\doteq$ in the logic, from the equality that we use elsewhere to express general equality of mathematical objects. In particular, we previously wrote $s = t$ to say that s and t are equal as terms. For instance, we reasoned about identities like $f(x)[x := c] = f(c)$. The "dotted" notation is different from this equality as it is just a syntactic formula. For example, we can form the formula $f(x) \doteq c$. This formula may be true or not, but the notation with the dot does not assign any intrinsic meaning to such statements, whereas $f(x) = c$ cannot be true as identity of terms. The meaning of $\doteq$ will rather come from the proof system and the semantics. It may happen that $f(x) \doteq c$

becomes true in a model, for example, by interpreting f as the successor map ($f^{\mathcal{M}}(n) = n + 1$), x as the number 1 and c as the number 2. Thus, it is important to keep in mind that $=$ and $\doteq$ express generally different things.

Definition 9.1

Let $\mathcal{L}$ be a signature. *$\mathcal{L}^{=}$-formulas* φ are generated by the following grammar.

$$\varphi ::= \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \to \psi \mid \forall x.\, \varphi \mid \exists x.\, \varphi$$
$$\mid P(t_1, \dots, t_n) \mid t \doteq s \mid \bot \mid (\varphi)$$

where $\mathrm{ar}(P) = n$ and $t_1, \dots, t_n, t$ and s are terms over $\mathcal{L}$. The set of $\mathcal{L}^{=}$-formulas is denoted by $\mathrm{Form}(\mathcal{L}^{=})$ or $\mathrm{Form}^{=}$ if $\mathcal{L}$ is clear from the context.

Since $\mathrm{Form}^{=}$ is defined in the same way as Form with one extra case added for $s \doteq t$, we can easily extend the notions of free variables and substitution to $\mathrm{Form}^{=}$. First of all, the map $\mathrm{fv} \colon \mathrm{Form}^{=} \to \mathcal{P}(\mathrm{Var})$ is defined exactly as the map as on Form, with the following extra case for equality.

$$\mathrm{fv}(s \doteq t) = \mathrm{fv}(s) \cup \mathrm{fv}(t)$$

The axioms for substitution on $\mathrm{Form}^{=}$ are also the same as for Form, but we need the following extra axiom (SE) to handle substitution on formulas that involve equality. In this axiom, it is important that we distinguish clearly between the atomic equality formulas and the equality of formulas themselves.

$$(s \doteq t)[\sigma] = (s[\sigma] \doteq t[\sigma]) \tag{SE}$$

Let us revisit the examples involving statements about natural numbers.

Example 9.2: Arithmetic with equality

In example 8.15, we have reasoned about even and odd numbers by using a predicate I that was supposed to represent the equality of numbers. As we have mentioned in the introduction of this section, this approach does not work very well. Let us instead consider the follow-

ing signature $\mathcal{L}$.

$$\mathcal{F} = \{\underline{0}, \underline{1}, p\} \qquad \mathcal{R} = \{L\}$$
$$\mathrm{ar}(\underline{0}) = \mathrm{ar}(\underline{1}) = 0 \qquad \mathrm{ar}(p) = \mathrm{ar}(L) = 2$$

The symbols in this signature still have the same intent as in example 8.4: $\underline{0}$ and $\underline{1}$ represent the numbers 0 and 1, p addition, and L less-than. We can now express in $\mathrm{Form}(\mathcal{L}^{=})$ the formulas that appeared in examples 8.13 and 8.15 more naturally:

$$\forall x.\, x \doteq p(x, \underline{0}) \qquad (\underline{0} \text{ is neutral})$$
$$\exists y.\, x \doteq p(y, y) \qquad (x \text{ is even})$$
$$\exists y.\, x \doteq p(p(y, y), \underline{1}) \qquad (x \text{ is odd})$$

Recall from table 8.4 that the existential quantifier in $\exists x.\, \varphi$ requires that there is at least one object that has the property φ. Suppose we want to express that there is *exactly one* such object, that is, that there is a unique object with property φ. We can use equality to do exactly that. To say that x is unique with the property φ is expressed by requiring that x is equal to any other object with the same property:

$$\forall y.\, \varphi[x := y] \to x \doteq y$$

Using this expression of uniqueness, we can define unique existential quantification, typically written as $\exists! x.\, \varphi$, by the following formula.

$$\exists x.\, \varphi \wedge (\forall y.\, \varphi[x := y] \to x \doteq y)$$

As uniqueness is one of the most important applications of equality, it deserves its own definition.

Definition 9.3: Uniqueness

Let φ be a formula, t a term and x a free variable in φ. We define a formula $\mathrm{unique}_x(t, \varphi)$ that expresses that t is uniquely among all objects that can be placed in the formula φ for x:

$$\mathrm{unique}_x(t, \varphi) = \forall y.\, \varphi[x := y] \to t \doteq y$$

The variable y is chosen to be fresh for φ and t. Using uniqueness, we

can define the *uniqueness quantifier* as follows.

$$\exists! x.\, \varphi \quad = \quad \exists x.\, \varphi \wedge \mathrm{unique}_x(x, \varphi)$$

Another useful application of equality is that it allows us to count how many things there are with a certain property.

Example 9.4: Expressing finite quantities

Not only can we use equality to enforce uniqueness as in definition 9.3, but we can even state that there must be at least or exactly a certain amount of objects with some property. To do so, we need to express that two objects are not identical, which we define here as the negation of equality:

$$s \not\doteq t \quad = \quad \neg(s \doteq t).$$

We can express that there are *at least two* objects with property φ by

$$\exists x.\, \exists y.\, \varphi[x := x] \wedge \varphi[x := y] \wedge x \not\doteq y$$

and that there must be *exactly two* objects for which φ holds by

$$\exists x.\, \exists y.\, \varphi[x := x] \wedge \varphi[x := y] \wedge x \not\doteq y \wedge (\forall z.\, \varphi[x := z] \to z \doteq x \vee z \doteq y)$$

This is called *counting quantification*.

Besides counting objects, we can also use equality to make sure that objects are given by a certain pattern. Let us demonstrate this by specifying the commands of a simple protocol.

Figure 9.1.: Simple vending machine

Example 9.5: Commands of a vending machine

Suppose we were to specify the interface of the vending machine in fig. 9.1, which has that has a coin slot and button to select the product that we wish to buy. Since vending machines occur everywhere across computer science curricula, it is worthwhile to figure out how to make such a specification formal.

We begin with the signature $\mathcal{L}_v$, where v stands for "vending machine", with predicate symbols Cmd and Prod of arity one for classifying commands, products, a constant coin, and a unary function symbol sel for the product selection command. This model assumes that our vending machine does not distinguish between coins and possible surpluses will be donated. Formally, we define $\mathcal{L}_v = (\mathcal{R}, \mathcal{F}, \mathrm{ar})$, where $\mathcal{R} = \{\mathrm{Cmd}, \mathrm{Prod}\}$, $\mathcal{F} = \{\mathrm{coin}, \mathrm{sel}\}$, $\mathrm{ar}(\mathrm{coin}) = 0$, and $\mathrm{ar}(\mathrm{Cmd}) = \mathrm{ar}(\mathrm{Prod}) = \mathrm{ar}(\mathrm{sel}) = 1$,

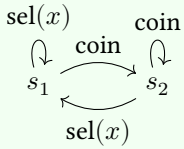
Within the signature $\mathcal{L}_v$, we can now use equality to specify that a command must either be the input of a coin or the selection of a product:

$$\forall x.\, \mathrm{Cmd}(x) \leftrightarrow x \doteq \mathrm{coin} \vee \exists y.\, x \doteq \mathrm{sel}(y) \wedge \mathrm{Prod}(y)$$

This formula can be recognised as a typical data type declaration, here in Haskell-style:

```
data Cmd = Coin | Sel Prod
```

Given that we know how commands look like, we can specify the behaviour of our vending machine. Our vending machine is going to be a bit greedy and not very user friendly: It has two states s_1 and s_2, one that waits for a coin and one that dispenses the chosen product. When the machine is in the first state it will ignore any selection commands, while in the second state it will continue accepting coins, even after already receiving one. This is displayed in the following state diagram.



Before we can reason about this machine, we have to carry out the mundane task of formally specifying this state diagram. To do so, we have to extend the signature $\mathcal{L}_v$ with constants s_1 and s_2, a unary

predicate symbol St to classify states, and a binary function symbol f that represents that transitions of the machine. Our goal is to describe f by a formula, such that

$$\forall x.\, \forall y.\, \mathrm{St}(x) \wedge \mathrm{Cmd}(y) \to \exists! z.\, \mathrm{St}(z) \wedge f(x, y) \doteq z$$

holds. The uniqueness quantifier is what makes f a map, which means that f assigns to every state x and command y a unique state z. Typically, we would specify f by *pattern matching*:

$$\begin{aligned} f(s_1, \mathrm{coin}) &= s_2 & \qquad f(s_1, \mathrm{sel}(x)) &= s_1 \\ f(s_2, \mathrm{coin}) &= s_2 & \qquad f(s_2, \mathrm{sel}(x)) &= s_1 \end{aligned}$$

Such a pattern matching definition ensures automatically that f is well-defined, if we know that there are exactly two states s_1 and s_2, and the two commands coin and sel. We have specified already how commands look like above. Similarly, we can specify that there are only two states:

$$\forall x.\, \mathrm{St}(x) \leftrightarrow x \doteq s_1 \vee x \doteq s_2$$

Such pattern matching definition can be specified in first-order logic with equality as follows.

$$\begin{aligned} \forall x.\, \forall y.\, \mathrm{St}(x) \wedge \mathrm{Cmd}(y) \to \\ \quad & (x \doteq s_1 \wedge y \doteq \mathrm{coin} \to f(x, y) = s_2) \\ \wedge\ & (x \doteq s_2 \wedge y \doteq \mathrm{coin} \to f(x, y) = s_2) \\ \wedge\ & (x \doteq s_1 \wedge \exists z.\, \mathrm{Prod}(z) \wedge y \doteq \mathrm{sel}(z) \to f(x, y) = s_1) \\ \wedge\ & (x \doteq s_2 \wedge \exists z.\, \mathrm{Prod}(z) \wedge y \doteq \mathrm{sel}(z) \to f(x, y) = s_1) \end{aligned}$$

That this formulas specifies f uniquely as a map, follows from the formulas that declare the "data types" Cmd and St. We will, however, refrain from proving this here because such a proof would be quite long and extremely boring. A better approach is to use an extension of first-order logic with types [And02; Jac99], that allows the specification of Cmd, St and f directly by pattern matching. That being said, this example shows that we can *in principle* handle data types and function definitions in first-order logic with equality.

We could now go further and also specify and reason about the behaviour of the vending machine, but at that point we should be really using a computer

and a proof assistant.

9.1.1. Semantics of FOL with Equality

The intention of the new symbol $\doteq$ is that it expresses that two objects are identical. With this in mind, we can easily extend the semantics of formulas to account for equality.

Definition 9.6: Formula semantics with equality

Let $\mathcal{L}$ be a signature and $\mathcal{M}$ an $\mathcal{L}$-model. The $\mathcal{L}^{=}$-*semantics* or just semantics of $\mathcal{L}^{=}$-formulas is given for a valuation $v\colon \mathrm{Var} \to |\mathcal{M}|$ by the map

$$[\![-]\!]_v^{=}\colon \mathrm{Form}^{=} \to \mathbb{B}$$

that is defined by iteration on formulas as follows.

$$\begin{aligned}
[\![\bot]\!]_v^{=} &= 0 \\
[\![P(t_1,\dots,t_n)]\!]_v^{=} &= \begin{cases} 1, & ([\![t_1]\!]_v,\dots,[\![t_n]\!]_v) \in P^{\mathcal{M}} \\ 0, & \text{otherwise} \end{cases} \\
[\![s \doteq t]\!]_v^{=} &= \begin{cases} 1, & [\![s]\!]_v = t_v \\ 0, & \text{otherwise} \end{cases} \\
[\![\varphi \wedge \psi]\!]_v^{=} &= \min\{[\![\varphi]\!]_v^{=}, [\![\psi]\!]_v^{=}\} \\
[\![\varphi \vee \psi]\!]^{=} v &= \max\{[\![\varphi]\!]_v^{=}, [\![\psi]\!]_v^{=}\} \\
[\![\varphi \to \psi]\!]_v^{=} &= [\![\varphi]\!]_v^{=} \implies [\![\psi]\!]_v^{=} \\
[\![\forall x.\, \varphi]\!]_v^{=} &= \min\{[\![\varphi]\!]_{v[x\mapsto a]}^{=} \mid a \in |\mathcal{M}|\} \\
[\![\exists x.\, \varphi]\!]_v^{=} &= \max\{[\![\varphi]\!]_{v[x\mapsto a]}^{=} \mid a \in |\mathcal{M}|\}
\end{aligned}$$

If it is clear from the context that $[\![-]\!]_v^{=}$ is applied to an $\mathcal{L}^{=}$-formula φ, then we just write $[\![\varphi]\!]_v$ instead of $[\![\varphi]\!]_v^{=}$. Semantic entailment for formulas and assumptions in $\mathrm{Form}^{=}$ is adapted accordingly: if Γ is a set of formulas and φ a single formula in $\mathrm{Form}^{=}$, then we write,

$$\Gamma \vDash \varphi \text{ if } \min([\![\Gamma]\!]_v^{=}) \leq [\![\varphi]\!]_v^{=} \text{ for all models } \mathcal{M} \text{ and valuations } v.$$

Note that $[\![-]\!]_v^=$ differs from the semantics of formulas without equality only by the extra case for $s \doteq t$. This means that whenever a formula does not use the equality symbol, then its semantics is given by definition 8.12. The following lemma makes this idea precise.

Lemma 9.7: Preservation of semantics
There is a map $e : \mathrm{Form}(\mathcal{L}) \to \mathrm{Form}(\mathcal{L}^=)$, such that for all models $\mathcal{M}$, valuations v and formulas $\varphi \in \mathrm{Form}(\mathcal{L})$ the following identity holds.

$$[\![e(\varphi)]\!]_v^= = [\![\varphi]\!]_v$$

Proof. The map e is defined by iteratively mapping a formula in $\mathrm{Form}(\mathcal{L})$ to the same formula in $\mathrm{Form}(\mathcal{L}^=)$: $e(\bot) = \bot$, $e(P(t_1, \dots, t_n)) = P(t_1, \dots, t_n)$, $e(\varphi \wedge \psi) = e(\varphi) \wedge e(\psi)$ etc. That the semantics of $e(\varphi)$ and φ agree is easily proved by induction. □

This result allows us also to use table 8.4 to determine the semantics of $\mathcal{L}^=$-formulas with quantifiers.

Let us go through the semantics of some formulas that use the equality predicate.

Example 9.8

Recall that the arithmetic model $\mathcal{M}_a$ in example 8.4 had to account for the predicate I that modelled equality explicitly. We can now forget about the interpretation of I and use $\mathcal{M}_a$ to give semantics to $\mathcal{L}^=$-formulas for the signature $\mathcal{L}$ from example 9.2. For instance, we have for a given valuation $v \colon \mathrm{Var} \to \mathbb{N}$ that

$$\begin{aligned}
[\![\forall x.\, x \doteq p(x, \underline{0})]\!]_v &= \begin{cases} 1, & [\![x \doteq p(x, \underline{0})]\!]_{v[x \mapsto n]} \text{ for all } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 1, & n = n + 0 \text{ for all } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\
&= 1,
\end{aligned}$$

where we used table 8.4 for the first identity. Similarly, we have

$$\begin{aligned}\llbracket \exists y.\, x \doteq p(y,y) \rrbracket_v &= \begin{cases} 1, & \llbracket x \doteq p(y,y) \rrbracket_{v[y \mapsto n]} = 1 \text{ for some } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & v(x) = 2n \text{ for some } n \in \mathbb{N} \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & v(x) \text{ even} \\ 0, & \text{otherwise} \end{cases}\end{aligned}$$

9.1.2. Natural Deduction for FOL with Equality

The last step to complete the picture of first-order logic with equality is to give the corresponding proof system. Keeping lemma 9.7 in mind, we expect that such a proof system has the same rules as $\mathbf{ND}_1$ or $\mathbf{cND}_1$ for all connectives and only adds rules for equality. Thus, let us briefly think about how we use equality and try to deduce the proof rules from this intuition.

A common use of equality is equational reasoning. For instance, if a and b are numbers, then

$$ab + a(-b) = a(b - b) = a0 = 0$$

establishes that $ab + a(-b)$ is identical to 0. In fact, if we calculate with concrete numbers, say 2 and 5, then $2 \cdot 5 + 2 \cdot (-5)$ is just another way of writing 0. This leads us to the most basic identity, the identity of an object with itself:

$$x = x$$

This identity is called *reflexivity* and will be the first rule that we adapt into our proof system.

Reflexivity by itself is, however, not enough. We often replace "equals by equals" in equational reasoning. For instance, if we know that $c = 0$ then we can infer

$$d = 0 + d = c + d.$$

From this and the previous identity, we can infer the following much more complex identity.

$$d = (ab + a(-b)) + d$$

The process of replacing equal objects can be generalised to arbitrary formulas and leads us the so-called *replacement rule*.

Formally, the intuitionistic and classical natural deduction proof systems for first-order logic with equality are given in the following definition.

Definition 9.9: Natural deduction with equality

The systems $\mathbf{ND}_1^=$ and $\mathbf{cND}_1^=$ are given by extending, respectively, $\mathbf{ND}_1$ and $\mathbf{cND}_1$ with the following two rules.

$$\frac{}{\Delta \mid \Gamma \vdash t \doteq t}\ (\text{Refl}) \qquad \frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash \varphi[x := s]}{\Delta \mid \Gamma \vdash \varphi[x := t]}\ (\text{Repl})$$

The rule (Refl) is called *reflexivity* and (Repl) is called *replacement*.

We follow in definition 9.9 the traditional naming for the rules, although we could also use the naming scheme of introduction and elimination rules that we employed for other logical connectives. Under this naming scheme, (Refl) would be an introduction rule, while (Repl) would be an elimination rule.

Surprisingly, the two rules (Refl) and (Repl) are enough to fully characterise equality. To give a flavour of the power of the replacement rule, let us prove that $\doteq$ is symmetric and transitive. These two properties form the basis of equational reasoning.

Lemma 9.10: Equality is an equivalence relation

The following two rules of *symmetry* and *transitivity* are admissible in $\mathbf{ND}_1^=$ and $\mathbf{cND}_1^=$.

$$\frac{\Delta \mid \Gamma \vdash s \doteq t}{\Delta \mid \Gamma \vdash t \doteq s}\ (\text{Sym}) \qquad \frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash s \doteq r}\ (\text{Trans})$$

Proof. To derive transitivity, we will have to use the replacement rule. The difficulty in using this rule lies in finding the correct formula φ, in which we replace equal terms. If we take a look at the conclusion of the transitivity rule, then we have several options of choosing such a formula φ. After a bit of trial and error, we can come up with

$$s \doteq x$$

for φ. This formula works because $\varphi[x := r]$ is the conclusion $s \doteq r$ of (Trans) and $\varphi[x := t]$ is the first premise $s \doteq t$ of (Trans). Thus, (Trans) is given by the following application of (Repl), albeit with swapped premises.

$$\frac{\Delta \mid \Gamma \vdash \varphi[x := t] \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash \varphi[x := r]} \text{ (Repl)}$$

By the above discussion, this a proof tree for

$$\frac{\Delta \mid \Gamma \vdash s \doteq t \qquad \Delta \mid \Gamma \vdash t \doteq r}{\Delta \mid \Gamma \vdash s \doteq r} \text{ (Trans)}$$

by carrying out the substitution. This shows that the transitivity rule is derivable from (Repl). □

?

Can you find a derivation for the symmetry rule in lemma 9.10 in $\mathbf{ND}_1^=$?

At this point, one may wonder why we have to introduce equality as a new symbol and give new proof systems. Why can we not just add equality to the signature and find axioms Γ so that that $\Gamma, \Gamma' \vdash \varphi$ is derivable in $\mathbf{ND}_1$ if and only if $\Gamma' \vdash \varphi$ is derivable in $\mathbf{ND}_1^=$? For instance, the formula $\forall x.\, x \doteq x$ could serve as an axiom that represents reflexivity. The problem is that the replacement rule ranges over all formulas, which would mean that we have to add an axiom for each formula, including our axioms. This leads to a problem that we cannot solve in first-order logic. Thus, we have to give equality a special status in our logic.

Since the unique quantifier in definition 9.3 is a derived logical connective, we can equip it with introduction and elimination rules. Especially the introduction rule shortens proofs considerably.

Lemma 9.11: Rules for uniqueness quantifier

The following rules are admissible in $\mathbf{ND}_1^=$, where the variable y in (∃!I) has to be fresh.

$$\frac{\Delta \mid \Gamma \vdash \varphi[x := t] \qquad \Delta, y \mid \Gamma, \varphi[x := y] \vdash t \doteq y}{\Delta \mid \Gamma \vdash \exists!x.\, \varphi} \text{ (∃!I)}$$

$$\frac{\Delta \mid \Gamma \vdash \exists!x.\, \varphi \qquad \Delta, x \mid \Gamma, \varphi, \mathrm{unique}_x(x, \varphi) \vdash \psi}{\Delta \mid \Gamma \vdash \psi} \text{ (∃!E)}$$

?

Can you derive the two rules in lemma 9.11?

We finish this section with a few example proofs in the systems $\mathbf{ND}_1^=$ and $\mathbf{cND}_1^=$. As before, we will use Fitch-style proofs to make the proofs readable.

Example 9.12

In this example, we will formally prove the incredibly difficult fact that the successor of an even number is odd. To this end, we use the formulas φ_e and φ_o given by

$$\varphi_e = \exists y.\, x \doteq p(y, y) \quad \text{and} \quad \varphi_o = \exists z.\, x \doteq p(p(z, z), \underline{1})$$

to describe even and odd numbers. We then derive the sequent

$$\vdash \forall x.\, \varphi_e \to \varphi_o[x := p(x, \underline{1})]$$

in $\mathbf{ND}_1^=$ by using Fitch-style.

1	x		$\exists y.\, x \doteq p(y, y)$	
2			$y \mid x \doteq p(y, y)$	
3			$p(x, \underline{1}) \doteq p(x, \underline{1})$	Refl
4			$p(x, \underline{1}) \doteq p(p(y, y), \underline{1})$	Repl, 2, 3
5			$\exists z.\, p(x, \underline{1}) \doteq p(p(z, z), \underline{1})$	$\exists$I, 4
6			$\varphi_o[x := p(x, \underline{1})]$	$\exists$E, 2–5
7		$\varphi_e \to \varphi_o[x := p(x, \underline{1})]$		$\to$I, 1–6
8	$\forall x.\, \varphi_e \to \varphi_o[x := p(x, \underline{1})]$			$\forall$I, 1–7

In the proof, we use the reflexivity rule (Refl) in line 3 and the replacement rule (Repl) in line 4. The rule (Repl) is thereby applied to the formula ψ given by $p(x, \underline{1}) \doteq p(u, \underline{1})$ and the substitutions $[u := x]$ and $[u := p(y, y))]$:

$$\frac{\Delta \mid \Gamma \vdash x \doteq p(y, y) \qquad \Delta \mid \Gamma \vdash \psi[u := x]}{\Delta \mid \Gamma \vdash \psi[u := p(y, y)]}$$

where $\Delta = x, y$ and $\Gamma = \varphi_e, x \doteq p(y, y)$ are the context and assumptions introduced in lines 1 and 2.

The next example shows how equality can be used to prove uniqueness of dividers

Example 9.13: Even numbers have unique divider

We all know that division gives a unique results, whenever it is defined. In particular, if a number n is even, then there should be a unique number k with $n = 2k$. Formally, we use the formula

$$\forall x.\, \varphi_e \to \exists!z.\, x \doteq p(z,z) \qquad (*)$$

to express uniqueness of divisors. In this example, we will prove uniqueness of divisors, as expressed by $(*)$, in $\mathbf{ND}^{=}_{\mathrm{I}}$.
Underlying uniqueness of division by two is the fact that doubling a number is an injective function, that is, $2n = 2m$ implies $m = n$ for all natural numbers m and n:

$$\varphi_{\text{double-inj}} \quad = \quad \forall m.\, \forall n.\, p(m,m) \doteq p(n,n) \to m \doteq n$$

Thus, what we will prove is that uniqueness of divisors is derivable from injectivity of doubling:

$$\varphi_{\text{double-inj}} \vdash \forall x.\, \varphi_e \to \exists!z.\, x \doteq p(z,z) \qquad (\dagger)$$

To make injectivity of doubling better usable in the proof of $(\dagger)$ below, let us derive for some context Δ, and terms r, s and t the following sequent.

$$\Delta \mid \varphi_{\text{double-inj}}, r \doteq p(s,s), r \doteq p(t,t) \vdash s \doteq t \qquad (\flat)$$

This sequent allows us to use an intermediate term t to relate the doubling of s and of t to derive the equality of s and t, and is proven as follows. Note that the proof uses symmetry and transitivity that we derived in lemma 9.10, and that the steps 4 and 5 are akin to the chain

of equations $p(s,s) \doteq x \doteq p(t,t)$.

1	$\varphi_{\text{double-inj}}$	
2	$x \doteq p(s,s)$	
3	$x \doteq p(t,t)$	
4	$p(s,s) \doteq x$	Sym, 2
5	$p(s,s) \doteq p(t,t)$	Trans, 4, 3
6	$\forall n.\, p(s,s) \doteq p(n,n) \to s \doteq n$	$\forall$E, 1
7	$p(s,s) \doteq p(t,t) \to s \doteq t$	$\forall$E, 6
8	$s \doteq t$	$\to$E, 7, 5

Using the identity, we can prove the uniqueness of the divisor of even numbers. The derivation of $(*)$ uses a typical combination: assume that an object with some property exists and prove that this object is unique. In the course of this, we use existential elimination (line 4-7) to inspect the object that we know to exist, and then introduce the uniqueness quantifier in line 7 by appealing to the above identity $(\flat)$.

1	$\varphi_{double-inj}$	
2	x	
3	$\exists y.\, x \doteq p(y,y)$	
4	y | $x \doteq p(y,y)$	
5	y' | $x \doteq p(y',y')$	
6	$y \doteq y'$	$(\flat)$, 4, 5
7	$\exists! z.\, x \doteq p(z,z)$	$\exists!$I, 4, 5–6
8	$\exists! z.\, x \doteq p(z,z)$	$\exists$E, 4–7
9	$(\exists y.\, x \doteq p(y,y)) \to \exists! z.\, x \doteq p(z,z)$	$\to$I, 3–8
10	$\forall x.\, \varphi_e \to \exists! z.\, x \doteq p(z,z)$	$\forall$I, 2–9

Example 9.13 shows how $\mathbf{ND}_1^=$ can be used to do equational reasoning, as it occurs in computer science and mathematics. Clearly, the proofs are fairly lengthy because we have to make every step explicit, which we typically not do on paper. However, the amount of boilerplate can be reduced by using a computer to automate some, or even all, of the steps, while still retaining the certainty of a formal proof as the one above.

9.2. Completeness

The strength of $\mathbf{cND}_1$ lies in the completeness theorem. Recall from theorem 8.19 that $\mathbf{cND}_1$ is sound, meaning that any statement that is provable in the formal system $\mathbf{cND}_1$ is also true semantically:

> If $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$, then $\Gamma \vDash \varphi$.

The completeness theorem establishes the other direction of this implication.

Theorem 9.14: Completeness of $\mathbf{cND}_1$

Let $\Gamma \vdash \varphi$ be a first-order sequent. If $\Gamma \vDash \varphi$, then $\Gamma \vdash \varphi$ is derivable in $\mathbf{cND}_1$.

Just as in the case of propositional logic, this remarkable result tells us that there is a proof tree for any semantically true statement. Unfortunately, the proof of this result is not effective, that is, we cannot extract an actual proof tree from the proof of theorem 9.14 and we only know that such a proof tree has to exist. The reason for this is that the proof of theorem 9.14 is a proof by contradiction and therefore uses classical logic in an essential way. This cannot be avoided, which renders the completeness theorem fairly useless from a computational perspective. If we wanted to use it to derive $\Gamma \vdash \varphi$, then we would have to quantify over all models and establish for each model $\mathcal{M}$ that the Γ entails φ in $\mathcal{M}$. As Γ and φ will likely contain quantifiers, we will then have to quantify over all elements of the universe of $\mathcal{M}$, cf. example 8.18. This is not only difficult, but generally undecidable. Even though the soundness and completeness theorems establish $\mathbf{cND}_1$ as a good proof system for first-order logic, finding a proof tree remains a difficult problem. In chapter 10, we will see a proof system that can prove less than $\mathbf{cND}_1$ and $\mathbf{ND}_1$, but allows us to do proof search.

9.3. Compactness and its Consequences

Closely related to completeness is also compactness. Where the completeness theorem 9.14 told us that any semantically true formula has a formal proof, the compactness theorem will establish limits on properties that we can express as formulas in first-order logic.

Note that (first-order) sequents of the form $\Gamma \vdash \varphi$ always require that Γ is a *finite list* of formulas, whereas the semantic entailment $\Gamma \models \varphi$ allows that Γ can be *any set*, even an infinite set, of formulas. Theorem 9.14 can be formulated differently to allow for such a general set Γ of assumption. However, one then has to show that $\Gamma \models \varphi$ uses only a finite number of assumptions from Γ to obtain completeness of $\mathbf{cND}_1$. This is because a the proof trees of $\mathbf{cND}_1$ are finite and can therefore only use finitely many assumptions. As a consequence, we get the following result.

Theorem 9.15: Compactness of first-order logic

Let Γ be an *arbitrary set* of first-order formulas. Then $\Gamma \models \varphi$ if and only if there is a *finite* $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \varphi$.

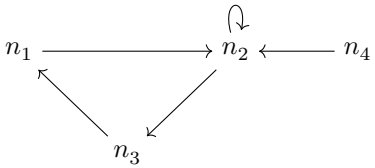
The right-to-left direction of the equivalence between the two conditions is easily established: Observe that if $\Gamma_0 \subseteq \Gamma$, then for any valuation v we have

$$\min([\![\Gamma]\!]_v) \leq \min([\![\Gamma_0]\!]_v) \leq [\![\varphi]\!]_v.$$

This shows that $\Gamma_0 \models \varphi$ implies $\Gamma \models \varphi$ for any (finite) subset Γ_0 of Γ. The other direction is what makes theorem 9.15 so interesting, and it is this direction that we use now to establish some limits of first-order logic with $\mathbf{cND}_1$.

9.3.1. Expressiveness of First-Order Logic

A *(simple directed) graph* G is given by a finite set N of nodes and a relation $E \subseteq N \times N$ of edges. Graphs are typically drawn as diagrams like this:



This graph G_1 is given by the set $N_1 = \{n_1, n_2, n_3, n_4\}$ of nodes and the relation E_1 of edges given by

$$E_1 = \{(n_1, n_2), (n_2, n_2), (n_2, n_3), (n_3, n_1), (n_4, n_2)\}.$$

A typical problem in graph theory is reachability:

> Given a graph G and nodes m and n in G, can we reach n from m by traversing along edges of G?

In the above example, we have that n_3 is reachable from n_1 by using the edges (n_1, n_2) and (n_2, n_3). Contrary to that, n_4 is not reachable from n_1 in G_1.

This problem seems to lend itself to formalisation in first-order logic: Let $\mathcal{L}$ be the signature $(\emptyset, \mathcal{R}, \mathrm{ar})$ with $\mathcal{R} = \{R\}$ and $\mathrm{ar}(R) = 2$. The intention is that the relation symbol R represents a binary relation and therefore the edges of a graph. Indeed, given a graph G with nodes N and edges E, we obtain an $\mathcal{L}$-model $\mathcal{M}_G$ by defining

$$|\mathcal{M}_G| = N \quad \text{and} \quad R^{\mathcal{M}_G} = E.$$

We can now ask if there is a first-order formula φ over $\mathcal{L}$ with two free variables x and y, such that for all graphs G, all nodes m and n in G and valuations v we have

$$[\![\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if } n \text{ is reachable from } m \text{ in } G.$$

It is important to note that we ask for a formula that works for all graphs! Let us try to write down such a formula. How can we reach a node y from x? Either x is already y, or an edge connects them, or we make a transition via other nodes. The problem is that there is a priori no upper bound on the number of nodes that we have to visit to get from x to y because the formula has to work for any graph. Thus, we are left with the following attempt to write down a formula, in which the dots indicate that such a formula would have to be infinitely long, which is clearly not what we consider a formula.

$$\begin{aligned} & x \doteq y \\ & \vee R(x, y) \\ & \vee (\exists z.\, R(x, z) \wedge R(z, y)) \\ & \vee (\exists z_1.\, \exists z_2.\, R(x, z_1) \wedge R(z_1, z_2) \wedge R(z_2, y)) \\ & \vee \cdots \end{aligned}$$

How do we get out of this? In just first-order logic with equality, we don't!

Theorem 9.16: Graph reachability cannot be expressed

There is no $\mathcal{L}^{=}$-formula φ with $\mathrm{fv}(\varphi) = \{x, y\}$, such that φ is true if and only if y is reachable from x via R.

Proof. Assume that there is an $\mathcal{L}^{=}$-formula with $\mathrm{fv}(\varphi) = \{x, y\}$, such that for every graph $G = (N, E)$ with $\mathcal{M}_G$ as above and for all $n, m \in N$

$$[\![\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if } n \text{ is reachable from } m \text{ in } G.$$

We define now formulas φ_k with $\mathrm{fv}\{x, y\}$, such that

$$[\![\varphi_k]\!]^{\mathcal{M}_G}_{[x \mapsto m][y \mapsto n]} = 1 \quad \text{if and only if there is a path from } n \text{ to } m \text{ of length } k.$$

Concretely, we define φ_k by iteration on k:

$$\begin{aligned} \varphi_0 &= x \doteq y \\ \varphi_{k+1} &= \exists z.\, R(x, z) \wedge \varphi_k[x := z] \end{aligned}$$

Let now $\Gamma = \{\neg\varphi_k \mid k \in \mathbb{N}\}$, which expresses that there is no path of any length and we clearly have $\Gamma \vDash \neg\varphi$. By the compactness theorem 9.15, we get a finite $\Gamma_0 \subseteq \Gamma$ with $\Gamma_0 \vDash \neg\varphi$. Let $k \in \mathbb{N}$ be the largest number, such that $\varphi_k \in \Gamma_0$. Thus, any path longer than k is not forbidden by Γ_0!

To use this fact, we define a graph $G = (N, E)$ by $N = \{n_0, n_1, \dots, n_{k+1}\}$ and $E = \{(n_i, n_{i+1}) \mid i = 0, \dots, k\}$. This graph G looks essentially like a list

$$n_0 \longrightarrow n_1 \longrightarrow \cdots \longrightarrow n_{k+1}$$

with only one path from n_0 to n_{k+1}, which is furthermore of length $k + 1$. In other words, for this graph G, we have

$$\min\Big([\![\Gamma_0]\!]^{\mathcal{M}_G}_{[x \mapsto n_0][y \mapsto n_{k+1}]}\Big) = 1$$

but

$$[\![\neg\varphi]\!]^{\mathcal{M}_G}_{[x \mapsto n_0][y \mapsto n_{k+1}]} = 0,$$

which contradicts $\Gamma_0 \vDash \neg\varphi$. Hence, the formula φ cannot exist. □

Theorem 9.16 is a severe limitation of first-order logic. Our robot from section 6.1 was counting on FOL to find a route that leads it to its heart! However, since routing is essentially graph reachability, our robot is now very sad 😕 But there is hope! In the exercises and in chapter 10, we will see that our robot can still be helped to find its heart.

9.4. Exercises

Exercise 5

a) Formalise the sentence

> "Pavel owes money to everyone but himself"

as a formula φ in first-order logic with equality. You need one constant p for "Pavel" and one binary predicate symbol O for "owes to".

b) Derive for your formula φ the following sequent in $\mathbf{ND}_1^=$ using a Fitch-style proof.

$$\varphi \vdash \neg O(p, p)$$

Exercise 6

Let $\mathcal{L}$ be a signature with a unary predicate symbol P. We define P_1 to be the formula

$$P_1 \quad = \quad \forall y.\, \forall z.\, P(y) \land P(z) \to y \doteq z,$$

which expresses that there can be maximally one object that fulfils P. Prove the following logical equivalence in $\mathbf{ND}_1^=$:

$$\vdash (\exists! x.\, P(x)) \leftrightarrow ((\exists x.\, P(x)) \land P_1)$$

To approach the proof of this formula, do both implications in separate proofs and refer to them in the proof of the logical equivalence. Furthermore, use the derived rules for the uniqueness quantifier from lemma 9.11 in the lecture notes.

Exercise 7 *Graph Reachability*

We have seen that compactness prevents us from giving a *formula* that expresses reachability in graphs. In this exercise, we will see that reachability can be expressed by appropriately defining a *predicate*. Let $\mathcal{L}$ be the signature $(\{E, R\}, \{n_1, \dots, n_4\}, \mathrm{ar})$ with $\mathrm{ar}(E) = \mathrm{ar}(R) = 2$ and $\mathrm{ar}(n_k) = 0$ for $k = 1, \dots 4$. The intention is that $E(x, y)$ holds if there is an edge between x and y in a given graph, and $R(x, y)$ holds if the node y is reachable from x. We will use the constants n_k later to model the nodes of a concrete graph.

Reachability R is the reflexive and transitive closure of the edge relation E. In other words, each node x must be related to itself via E (reflexivity), and if there is an edge from x to some z and y is reachable from z, then y is also reachable from x (transitivity).

a) Give two formulas φ_r and φ_t with free variables x and y that express, respectively, reflexivity and transitivity. That is to say, that the formula φ_R given by

$$\forall x.\, \forall y.\, R(x, y) \leftrightarrow \varphi_r \vee \varphi_t$$

expresses that R is the reachability relation in the graph with edges E.

b) Let $\varphi_{R,1}$, $\varphi_{R,2}$ and $\varphi_{R,3}$ be given by

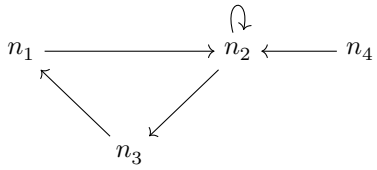
$$\begin{aligned}
\varphi_{R,1} &= \forall x.\, \forall y.\, R(x, y) \to \varphi_r \vee \varphi_t \\
\varphi_{R,2} &= \forall x.\, \forall y.\, \varphi_r \to R(x, y) \\
\varphi_{R,3} &= \forall x.\, \forall y.\, \varphi_t \to R(x, y)
\end{aligned}$$

Derive the following sequent in $\mathbf{ND}_1^=$ using Fitch-style.

$$\varphi_{R,1}, \varphi_{R,2}, \varphi_{R,3} \vdash \varphi_R$$

Note that the proof merely uses the rules for quantifiers and propositional connectives, not those for equality.

c) Consider the following graph G.



Give formulas $\varphi_{E,1}, \dots, \varphi_{E,5}$ that describe the edge of G.

d) Give a proof in $\mathbf{ND}_1^=$ using Fitch-style of $\varphi_R, \varphi_{E,1}, \dots, \varphi_{E,5} \vdash R(n_4, n_3)$. Use b) to simplify the task.

Exercise 8

A *group* G is given by a binary map $\dot{+} \colon G \times G \to G$ and an element $\dot{0}$ of G, such that

1. for all x in G, $x \dot{+} \dot{0} = \dot{0} \dot{+} x = x$,
2. for all x, y, z in G, $x \dot{+} (y \dot{+} z) = (x \dot{+} y) \dot{+} z$, and
3. for every x in G there is a y in G, such that $x \dot{+} y = \dot{0}$ and $y \dot{+} x = \dot{0}$.

The equations 1 - 3 are called the *group axioms*. Groups appear everywhere in computer science and mathematics, with very popular applications in cryptography. The goal of this exercise is to formally reason about groups in first-order logic with equality.

Let $\mathcal{L}$ be the signature with function symbols $\dot{+}$ and $\dot{0}$ of arity 2 and 0, respectively. Let us write, as above, the symbol $\dot{+}$ in infix notation, that is, we write $s \dot{+} t$ instead of $\dot{+}(s, t)$ for terms s and t. We define $\mathcal{L}^=$-formulas $\varphi_{1,l}$ and $\varphi_{1,r}$ by

$$\varphi_{1,l} = \forall x. \, \dot{0} \dot{+} x \doteq x \quad \text{and} \quad \varphi_{1,r} = \forall x. \, x \dot{+} \dot{0} \doteq x$$

that formalise together the first group axiom.

a) Give $\mathcal{L}^=$-formulas φ_2 and φ_3 that formalise the axioms 2 and 3 from above.

b) Give a formula φ_u that expresses the uniqueness of y in the third group axiom. The element y is called the *inverse* of x.

c) Prove in $\mathbf{ND}_1^=$ that the inverse y of x in the third axiom is unique, that is, derive $\vdash \varphi_u$ for your formula in $\mathbf{ND}_1^=$. Use Fitch-style as usual.

10. First-Order Horn Clauses and Automatic Deduction

10.1. Automatic Deduction and the Cut-Rule

When approaching proofs of mathematical theorems, we usually decompose the problem into intermediate results that are easier to prove on their own. In the natural deduction systems, the approach is justified by the so-called *cut rule*. This rule allows one to prove first a formula φ, then prove a formula ψ under the assumption of φ, and finally the conclude that ψ holds on its own. The rule is formulated in the following theorem.

Theorem 10.1: Admissible Cut

The following *cut rule* is admissible in $\mathbf{ND}_1$.

$$\frac{\Gamma \vdash \varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ (\text{Cut})$$

Proof. The cut rule is given by the following proof tree.

$$\dfrac{\Gamma \vdash \varphi \qquad \dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to\text{I})}{\Gamma \vdash \psi}\ (\to\text{E})$$

□

The cut rule is very useful in structuring proofs, but horrendous for automatic deduction because it requires ingenuity for inventing the intermediate lemma φ to prove ψ. There are techniques for generating such lemmas but we will take another route here.

A first step to automatic deduction is to avoid the use of the cut rule and thereby the introduction of an implication that is immediately followed by

an implication elimination, as in the proof of the cut rule. Fortunately, it is possible to avoid such detours and we can proceed without having to make up formulas out of thin air. This is captured by the following theorem, the proof [TS00; TvD88] of which we do not present here due to its complexity.

Theorem 10.2: Cut Elimination

Any proof tree in $\mathbf{ND}_1$ for a sequent $\Gamma \vdash \varphi$ can be transformed into a proof tree in $\mathbf{ND}_1$ for the same sequent, which contains no instances of the cut rule.

This is good news for automatic deduction because we can limit at each step the rules that may be applied by inspecting the formula φ that we have to prove. The approach of inspecting φ in finding a proof for $\Gamma \vdash \varphi$ is also called *goal-oriented* because the search for the proof is driven by the goal φ. However, restricting ourselves to proofs that avoid the cut rule is not enough for fully automatic deduction because there are other choice-points. For instance, if we try to prove $\exists x.\, P(x) \vdash \neg\forall x.\, \neg P(x)$ do we first use the introduction of the negation or the elimination of the existential quantifier? The problem is that the principle formula of elimination rules, the formula that gives a rule its name, appears among the premises of those rules. This means for the goal-oriented construction of a proof that we have to guess the right formula to eliminate from the goal, while there may be a non-trivial relation between the two. In the above example, we have to guess that we have to eliminate $\exists x.\, P(x)$ to prove $\neg\forall x.\, \neg P(x)$, which is difficult for human intelligence, let alone for a computer. One way out of this is to limit the class of formulas that may appear in proofs and thereby reduce the amount of guessing to a manageable amount. Combined with limiting proof rules, we obtain a reasonable fragment of first-order logic for automatic deduction.

10.2. First-Order Horn Clauses and Logic Programming

In chapter 5, we have seen a class of propositional formulas that were called Horn clauses. It turns out that there is an extremely useful generalisation to first-order logic.

Definition 10.3: Horn Clauses and Theories in FOL

A *predicate atom* over a signature $\mathcal{L}$ is a formula of the form $P(t_1, \dots, t_n)$, where P is an n-ary predicate symbol in $\mathcal{L}$ and $t_1, \dots, t_n$ are $\mathcal{L}$-terms. Predicate atoms are denoted by letters $A, B, C, \dots$ A *Horn clause* is a formula of the form

$$\forall x_1. \cdots \forall x_m. A_1 \land \cdots \land A_n \to A_0$$

for $m, n \in \mathbb{N}$ and predicate atoms $A_0, A_1, \dots, A_n$ that use only the variables $x_1, \dots, x_m$. The atom A_0 is called the *head* of the clause and the set $\{A_1, \dots, A_n\}$ is called the *body* of the clause. We call a list Γ of Horn clauses a *Horn clause theory* or *logic program*.

The body of a clause may be empty, that is, n may be zero, in which case we leave out the implication in the Horn clause in definition 10.3 and just write $\forall x_1. \cdots \forall x_m. A_0$. This is convenient when writing logic programs that contain Horn clauses without assumptions, so called *base facts*. In the abstract treatment of Horn clauses later in this chapter it will, however, not be necessary to differentiate between base facts and general Horn clauses.

What makes Horn clause theories so useful? As the name "logic program" indicates, we can use such theories for programming. In fact, we can describe *any* Turing machine by a logic program Γ. A computation corresponds to giving a predicate atom A with free variables $x_1, \dots, x_m$ and asking for a derivation of $\Gamma \vdash \exists x_1. \cdots \exists x_m. A$ in $\mathbf{ND}_1$. If the Turing machine halts, then such a derivation exists. This correspondence makes it possible to automatically find such a derivation if it exists.

Let us demonstrate the programming aspect of Horn clauses. We have come across arithmetic several times in the context of first-order logic. So far, we always relied on the correct interpretation of symbols. The following example shows how we can express arithmetic on natural numbers in terms of Horn clauses.

Example 10.4

In this example, we will use that every natural number is either zero or the successor of another natural number. More specifically, the constant z represents in the following zero and the unary function symbol s represents the successor of a number. With these two symbols, we

can enumerate all the natural numbers:

$$z,\ s(z),\ s(s(z)),\ s(s(s(z))), \ldots$$

Writing natural numbers in this way is called *unary encoding*, which is a very bad encoding from the perspective of complexity but very useful for explanatory purposes.

We will be working in this example with a signature $\mathcal{L}$ that contains a constant z, a unary function symbol s, a unary predicate symbol N, a binary predicate symbol L, and a ternary predicate symbol S. The purpose of z and s to encode natural numbers was already explained above. We use the unary predicate symbol N to pin down the natural numbers, in the sense that $N(t)$ holds if t is the representation of a natural number. The following two Horn clauses implement precisely the initial description of natural numbers.

$$N(z)$$
$$\forall n.\, N(n) \to N(s(z))$$

We can now continue and implement arithmetic for natural numbers. To use Horn clauses for this task, we have to switch from thinking in terms of function to thinking in terms of relations. More specifically, we use a ternary predicate symbol S with the intent that $S(m, n, k)$ holds if k is the sum of m and n. To provide Horn clauses that implement addition, we briefly need to think about the recursive specification of addition. It turns out that it suffices to consider two cases: either m is zero, or m is the successor of some number i. In the first case, k will be n, which corresponds to $0 + n = n$. In the second case, k will be the successor of the sum of i and n, which corresponds to $(p + 1) + n = (i + n) + 1$. Assuming that m and n are given in the above representation of natural numbers, we can represent this recursive specification of addition by the following two Horn clauses.

$$\forall n. \qquad S(z, n, n)$$
$$\forall m.\, \forall n.\, S(m, n, k) \to S(s(m), n, s(k))$$

In a similar spirit, we can also define recursive relations on natural numbers. For instance, to specify that the binary predicate L represents the less or equal relation, we would use the following two Horn clauses.

$$\forall n. \qquad L(n, n)$$
$$\forall m.\, \forall n.\, L(m, n) \to L(m, s(n))$$

The clauses say that every natural number is less or equal to itself (reflexivity) and that increasing the number on the right preserves the less or equal relation.

?

Can you find two Horn clauses that describe the computation of multiplication as ternary predicate symbol M analogously to S in example 10.4?

What makes logic programming so interesting compared to other programming paradigms? The answer lies in its declarative nature, which allows us to specify what the result of a computation should be rather than how the result is computed. In his seminal work, Kowalski [Kow79] has expressed this in the following equation.

Algorithm = Logic + Control

This equation essentially means that we can design algorithms by providing logical formulas that describe the result of an algorithm and a control mechanism that controls the use of these formulas to compute the result. A particular property of this approach is that new algorithms can be obtained by exchanging the control mechanism, while preserving the logical properties and correctness of the computed result. In terms of the above equation, we may have a logical specification L and algorithms A_1 and A_2 given by $A_1 = L + C_1$ and $A_2 = L + C_2$ for some control mechanisms C_1 and C_2. These two algorithms will compute results with the same logical properties, but may have different computational behaviour. For example, A_2 may be more efficient than A_1.

Let us return to concrete logic programming approach and illustrate it on our initial robot example.

Example 10.5: Robot Path Finding

Remember our robot from fig. 6.1 on page 11 trying to find a heart? In section 6.1, we used first-order logic to describe what moves the robot is allowed to make and how a route towards the heart looks like. The aim of this example is to use Horn clauses to give a logical description of routes that will allow us, with an appropriate control mechanism, to

derive an efficient algorithm for routing.

In section 9.3.1, we have seen that reachability in graphs cannot be expressed by a first-order formula. Since routing for the robot can be seen as reachability in a graph, we cannot directly express routes as a formula. In exercise 3 of chapter 9, we saw how graph reachability could still be expressed by introducing a new predicate that represented the reflexive, transitive closure of the edge relation of a graph. And even better, the two formulas that expressed this are Horn clauses and we can therefore formulate graph reachability as a logic program! Let us now use this approach to help the robot break this limitation of first-order logic and to finally find its heart.

For this purpose, we will use the signature $\mathcal{L}$ with constants $\underline{1}, \underline{2}, \ldots$ for all positive natural numbers, one binary function symbol pos, three unary predicate symbols R, H and F, and two binary predicate symbols A and C. Let us explain the intention of all these symbols. We will use pos to describe positions on the field and will denote by $\mathrm{pos}(x, y)$ the position with horizontal coordinate x and vertical coordinate y. For instance, the robot is in fig. 6.1 in position $\mathrm{pos}(\underline{2}, \underline{3})$. The unary predicate symbols R, H and F describe, respectively, the position of the **R**obot, the position of the **H**eart and **F**ree positions. Since the robot may only move between adjacent positions, will use the binary predicate symbol A to express when a position is **A**djacent to another position. Finally, the binary predicate symbol C is what we are after: it will relate two positions if they are **C**onnected by a path via free and adjacent positions.

Our goal is to describe the situation in fig. 6.1 and the predicate symbol C as a logic program, that is, a finite list of Horn clauses. All of the following is provided in appendix B as a Prolog program that can be directly run in your favourite Prolog interpreter.

Let us begin with the easy part: the position of the robot and the heart. These are given by the following two formulas.

$$R(\mathrm{pos}(\underline{2}, \underline{3})) \quad \text{and} \quad H(\mathrm{pos}(\underline{5}, \underline{1})) \tag{10.1}$$

This two formulas are base facts, albeit with no quantifiers and an empty body, that is, m and n in definition 10.3 are both zero.

Next, we describe the free position that our robot may visit by provid-

ing a formula for each free position:

$$\begin{array}{llll} F(\text{pos}(\underline{1},\underline{1})) & F(\text{pos}(\underline{2},\underline{2})) & F(\text{pos}(\underline{3},\underline{1})) & \\ F(\text{pos}(\underline{1},\underline{4})) & F(\text{pos}(\underline{2},\underline{3})) & F(\text{pos}(\underline{3},\underline{2})) & \ldots \\ & F(\text{pos}(\underline{2},\underline{4})) & F(\text{pos}(\underline{3},\underline{4})) & \end{array} \tag{10.2}$$

You may have noticed that we have initially talked in section 6.1 about the obstacles on the field and now we talk about free positions instead. The reason for this is that obstacles are an inherently negative description of the allowed moves that our robot may make. More precisely, we have the relation

$$\forall p.\, F(p) \leftrightarrow \neg O(p)\,,$$

where O was the predicate symbol that we used for describing the positions of obstacles. This negative relation between F and O would become a problem when we ask if the robot can move to a certain position because the first-order Horn clauses in definition 10.3 may not use $\bot$ and therefore no negation! Thus, we would not be able to formulate the routing problem as a logic program if we describe positions with obstacles instead of free positions.

To finish the field description, we have to give all the adjacent positions by providing formulas that specify the predicate symbol A. This predicate should be read as, if $A(p, q)$ holds then position p and q are adjacent. For instance, we should have that $A(\text{pos}(\underline{1},\underline{1}), \text{pos}(\underline{1},\underline{2}))$ holds. However, neither $A(\text{pos}(\underline{1},\underline{1}), \text{pos}(\underline{2},\underline{2}))$ nor $A(\text{pos}(\underline{6},\underline{1}), \text{pos}(\underline{7},\underline{1}))$ should not hold. In the first case, the two positions $\text{pos}(\underline{1},\underline{1})$ and $\text{pos}(\underline{2},\underline{2})$ are not adjacent, as we do not allow diagonal steps. In the second case, the term $\text{pos}(\underline{7},\underline{1})$ is not a valid position on the field because the horizontal coordinates range from 1 to 6 only. Formalising all of this as Horn clauses is a bit tedious because we have to enumerate for each position all its four neighbours, except for positions at the boundary that have two or three neighbours:

$$\begin{array}{llll} A(\text{pos}(\underline{x},\underline{y}), \text{pos}(\underline{x+1},\underline{y})), & 1 \le x \le 5, & 1 \le y \le 4 \\ A(\text{pos}(\underline{x},\underline{y}), \text{pos}(\underline{x},\underline{y+1})), & 1 \le x \le 6, & 1 \le y \le 3 \\ A(\text{pos}(\underline{x},\underline{y}), \text{pos}(\underline{x-1},\underline{y})), & 2 \le x \le 6, & 1 \le y \le 4 \\ A(\text{pos}(\underline{x},\underline{y}), \text{pos}(\underline{x},\underline{y-1})), & 1 \le x \le 6, & 2 \le y \le 4 \end{array} \tag{10.3}$$

The first line of (10.3) states that every position $\text{pos}(\underline{x}, \underline{y})$ is adjacent to its east neighbour $\text{pos}(\underline{x+1}, \underline{y})$. Note that $\text{pos}(\underline{x}, \underline{y})$ only has a neighbour in the east if it is not a boundary position, that is, if $x \leq 5$. In the remaining three lines, we analogously provide the specification of the southern, western and northern neighbours.
With the field layout described through the Horn clauses in eqs. (10.1) to (10.3), we can now try to find a way for our robot to the heart. We achieve this by providing a description of the predicate symbol C with the intention that $C(p, q)$ holds if position q is reachable from p via free and adjacent positions. Reachability can be described by two Horn clauses, one that states that every position can be reached from itself and one for making an intermediate step via a free position:

$$\forall p. \quad C(p, p) \tag{10.4}$$

$$\forall p.\, \forall q.\, \forall r.\, C(q, r) \wedge A(p, q) \wedge F(q) \rightarrow C(p, r) \tag{10.5}$$

These two Horn clauses formulate reachability as backward search. That is to say, we start with the final position we want to reach and then try to find positions that make a path to the initial position.

10.3. Uniform Proofs

In example 10.5, we have used logical formulas to describe how a path from the starting position of the robot to its goal looks like. What is missing to get an algorithm for path finding is the control part of Kowalski's equation, which allows us to search for a path. We will introduce in this section a control mechanism based on proof theory, called uniform proofs.

The idea of uniform proofs is that we devise a proof system tailored towards proving goal formulas of a very specific shape only from Horn clauses. This restriction of goals and assumptions has a two-fold effect:

1. we eliminate all the choice that appears through the cut rule, and
2. we concentrate all the choice of using assumptions in one proof rule.

Eliminating the cut rule removes an infinitude of choices and makes the search for proof rules more goal-oriented. As we will see, the second aspect of concentrating choice, reduces the amount of branching in proof search drastically

to just a few options. All together, uniform proofs provide an operational perspective on proof search and will guide us in the construction of proofs, also in $\mathbf{ND}_1$, relative to logic programs.

Before we come to the actual definition of uniform proofs, we will need to talk about specific terms and substitutions that arise in proof search. In the simplest case, which will be the only one that we treat here, we prove an existentially quantified formula $\exists x.\, \varphi$ by providing a term t, which uses no variables and for which $\varphi[x := t]$ holds. For instance, let φ be $R(\mathrm{pos}(x, \underline{3}))$ and let t be $\underline{2}$, as in example 10.5. Under the assumption of eq. (10.1), we have that $\varphi[x := t]$ holds and thus also $\exists x.\, (\mathrm{pos}(x, \underline{3}))$. Note that $\underline{2}$ is a constant and does not use any variables. Such a term without any variables will be the only kind of term that we will be interested in the following when we prove existential quantifiers.

Definition 10.6: Ground terms and atoms

A *ground term* t is a term over a signature $\mathcal{L}$, such that $\mathrm{fv}(t) = \emptyset$. Similarly, a *ground predicate atom* is a predicate atom $P(t_1, \ldots, t_n)$, where all the terms t_k are ground terms.

Let us provide some more examples of ground terms.

Example 10.7

We have already seen that $\underline{2}$ is a constant and thus ground term over the signature of example 10.5. Other examples of ground terms over that signature are $\mathrm{pos}(\underline{2}, \underline{3})$ or $\mathrm{pos}(\underline{2}, \mathrm{pos}(\underline{2}, \underline{3}))$. However, if $x \in \mathrm{Var}$, then $\mathrm{pos}(x, \underline{3})$ is not a ground terms because $\mathrm{fv}(\mathrm{pos}(x, \underline{3})) = \{x\}$.

?

Do signatures without constants admit ground terms?

Uniform proofs will allow us to introduce existential quantifiers in goals and eliminate the universal quantifiers that appear in Horn clauses. Recall that the natural deduction system $\mathbf{ND}_1$ involved substitutions in the two corresponding rules ($\exists$I) and ($\forall$E). As we aim to only construct proofs that involve ground terms, we need to restrict these two rules to substitutions that replace variables by ground terms. These are the closing substitutions introduced in the following definition.

Definition 10.8: Closing substitution

Let $X \subseteq \text{Var}$ be a set of variables. An *X-closing substitution* is a substitution $\sigma\colon \text{Var} \to \text{Term}$, such that $\sigma(x)$ is a ground term for all $x \in X$.

We mentioned initially that we obtain proof search by restricting the formulas φ that can appear as a goal, that is, in a sequent $\Gamma \vdash \varphi$ that we wish to prove. There are several choices one can make to restrict the formula φ, out of which we will use that given in definition 10.9 below. Before we come to that definition, let us briefly think what would be a reasonable set of formula that we can search proofs for. First of all, we of course want to be able to prove facts about predicates, hence predicate atoms of the form $P(t_1, \dots, t_n)$ should certainly be allowed as goals. Next, conjunction does certainly not pose any problem because finding a proof for $\Gamma \vdash \varphi \wedge \psi$ only requires us to find proofs for $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$ separately by the introduction rule ($\wedge$I). To be able to state and prove more interesting properties, we will also allow disjunction and existential quantification. These two connectives will make proof search more difficult. Recall that the introduction rules ($\vee \text{I}_1$) and ($\vee \text{I}_2$) for disjunction require us to find proofs for *either* $\Gamma \vdash \varphi$ *or* $\Gamma \vdash \psi$ in order to prove $\Gamma \vdash \varphi \vee \psi$. Thus, to find a proof for the disjunction $\varphi \vee \psi$ we have to try to prove one of the options, say φ, and if we fail to prove this option, then we try the other. Trying out these different options is not difficult, but can be costly, depending on how many proof steps we have to carry out before we find out that φ is not provable. Finally, the proof of an existential quantifier $\exists x.\, \varphi$ requires us to find a term t, such that $\varphi[x := t]$ holds. As it turns out, it is possible to devise procedure that construct a ground term t, if it exists, automatically and we see how this can be done in section 10.4. Thus, we will also allow existential quantifiers in proof goals. The following definition 10.9 sums up the formulas that we allow as goals.

Definition 10.9: Goal formulas

The set Goal of all *goal formulas* φ_G is the set of $\mathcal{L}$-formulas over a signature $\mathcal{L}$ generated by the following grammar.

$$\varphi_G ::= P(t_1, \dots, t_n) \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \exists x.\, \varphi_G$$

Let us now come to uniform proofs, which are given by a proof system that allows us to prove goals φ (definition 10.9) from Horn clause theories Γ (definition 10.3). To distinguish sequents for uniform proofs from those of $\mathbf{ND}_1$, we will write $\Gamma \vdash_u \varphi$ for a sequent that we intent to find a uniform proof for.

Introduction Rules

$$\frac{\Gamma \vdash_u \varphi_1 \quad \Gamma \vdash_u \varphi_2}{\Gamma \vdash_u \varphi_1 \wedge \varphi_2}\ (\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash_u \varphi[x := t]}{\Gamma \vdash_u \exists x.\, \varphi}\ (\exists\mathrm{I})$$

$$\frac{\Gamma \vdash_u \varphi_1}{\Gamma \vdash_u \varphi_1 \vee \varphi_2}\ (\vee\mathrm{I}_1) \qquad \frac{\Gamma \vdash_u \varphi_2}{\Gamma \vdash_u \varphi_1 \vee \varphi_2}\ (\vee\mathrm{I}_2)$$

Backchaining Rule

$$\frac{(\forall x_1.\, \cdots \forall x_m.\, A_1 \wedge \cdots A_n \to A_0) : \Gamma \quad \Gamma \vdash_u A_1[\sigma] \ \cdots\ \Gamma \vdash_u A_n[\sigma]}{\Gamma \vdash_u A_0[\sigma]}\ (\mathrm{B})$$

where σ is an $\{x_1, \ldots, x_m\}$-closing substitution.

Figure 10.1.: Rules of Uniform Proofs

We have already explained the general approach to prove $\Gamma \vdash_u \varphi$ above for the case of conjunction, disjunction and existential quantification. This leads us to the introduction rules in the upper part of fig. 10.1, which are exactly the same rules as in $\mathbf{ND}_1$, just restricted to the introduction of goal formulas. Let us consider the goal φ given by $\exists u.\, \exists v.\, R(u) \wedge (A(u, v) \wedge F(v))$ over the signature from the robot path finding example 10.5. This formula says that the robot is in some position on the field that has a free adjacent position. Such positions exist, namely if we pick $p = \mathrm{pos}(\underline{2}, \underline{3})$ for the robot position and the adjacent position $q = \mathrm{pos}(\underline{2}, \underline{4})$. Using only the introduction rules from fig. 10.1, we can start a proof for φ:

1	$\vdots$	
2	$A(p, q)$	??
3	$F(q)$	??
4	$A(p, q) \wedge F(q)$	$\wedge$I, 2, 3
5	$R(p)$	??
6	$R(p) \wedge (A(p, q) \wedge F(q))$	$\wedge$I, 5, 4
7	$\exists v.\, R(p) \wedge (A(p, v) \wedge F(v))$	$\exists$I, 6
8	$\exists u.\, \exists v.\, R(u) \wedge (A(u, v) \wedge F(v))$	$\exists$I, 7

However, we are not able to fill in the question marks, yet. Note that the formulas that we have to prove in lines 2, 3 and 5 are ground predicate atoms. It is easy to see that any proof attempt that starts with a goal formula and only uses the introduction rules, must eventually reach predicate atoms. Thus, we need a rule to prove such predicate atoms from a Horn clause theory. That is the heart of logic programming: given a predicate atom, find a Horn clause that matches the atom, and continue with the premises of the Horn clause. This process is called *backchaining* and is captured by the rule (B) in fig. 10.1. The idea of the backchaining rule is that, whenever we have to prove a predicate atom, then we choose a Horn clause from our logic program in such a way that the head of the clause (definition 10.3) matches the atom. If we find such a matching clause, then we continue by proving all the premises of the chosen Horn clause. Note that the premises of a Horn clause are always predicate atoms, thus we will continue using the backchaining rule until we can finish the proof by selecting a Horn clause without premises, that is, a fact.

Let us use this procedure to fill in the question marks in the proof above. For the purpose of this, let us denote by Γ_1 the logic program that consists of all the Horn clauses in eqs. (10.1) to (10.3). In line 5, we have to prove $R(\mathrm{pos}(\underline{2}, \underline{3}))$, which is exactly the fact that we assumed in (10.1). Thus, we can use the following instance of the backchaining rule, where both n and m are 0 and thus the substitution σ is irrelevant.

$$\frac{R(\mathrm{pos}(\underline{2}, \underline{3})) : \Gamma_1}{\Gamma_1 \vdash_u R(\mathrm{pos}(\underline{2}, \underline{3}))} \text{ (B)}$$

The proofs for lines 2 and 3 are given analogously, which then results in the following completed uniform proof in Fitch-style.

1	Γ_1	
2	$A(p, q)$	B, 1
3	$F(q)$	B, 1
4	$A(p, q) \land F(q)$	∧I, 2, 3
5	$R(p)$	B, 1
6	$R(p) \land (A(p, q) \land F(q))$	∧I, 5, 4
7	$\exists v.\, R(p) \land (A(p, v) \land F(v))$	∃I, 6
8	$\exists u.\, \exists v.\, R(u) \land (A(u, v) \land F(v))$	∃I, 7

Before we continue to explain the backchaining rule for more interesting cases, let us formally define what we mean by a uniform proof.

Definition 10.10: Uniform proofs

Let Γ be a Horn clause theory and φ a goal formula with $\mathrm{fv}(\varphi) = \emptyset$. We say that φ can be proven from Γ by a *uniform proof*, if a proof for $\Gamma \vdash_u \varphi$ can be derived from the rules in fig. 10.1.

Note that the uniform proofs have no separate elimination rules. Instead, backchaining combines the assumption rule (Assum), the universal quantifier elimination (∀E) and the implication elimination (→E) into one rule. This works only because the only assumptions that we can use are Horn clauses.

Let us demonstrate uniform proofs cases, in which we use the backchaining rule on Horn clauses with assumptions and universal quantifiers.

Example 10.11

Suppose we want to show that our robot can reach *some* position, thus we want to find a proof for

$$\Gamma \vdash_u \exists u.\, C(\mathrm{pos}(\underline{2}, \underline{3}), u)\,,$$

where C is the predicate for connected positions from example 10.5 and Γ the logic program that consists of eqs. (10.1) to (10.5). As above, we will refer to the logic program given by eqs. (10.1) to (10.3) as Γ_1. This

allows us refer explicitly to the Horn clauses in eqs. (10.4) and (10.5). There are many choices for the position u, but to make the example non-trivial and not too lengthy, let us use $\mathrm{pos}(\underline{2}, \underline{4})$. The uniform proof for the above sequent goes for this choice as follows.

1	Γ_1	
2	$\forall p.\, C(p,p)$	
3	$\forall p.\, \forall q.\, \forall r.\, C(q,r) \wedge A(p,q) \wedge F(q) \to C(p,r)$	
4	$C(\mathrm{pos}(\underline{2},\underline{4}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 2
5	$A(\mathrm{pos}(\underline{2},\underline{3}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 1
6	$F(\mathrm{pos}(\underline{2},\underline{4}))$	B, 1
7	$C(\mathrm{pos}(\underline{2},\underline{3}), \mathrm{pos}(\underline{2},\underline{4}))$	B, 3, 4, 5, 6
8	$\exists u.\, C(\mathrm{pos}(\underline{2},\underline{3}), u)$	$\exists$I, 7

The reader may notice that there is another choice hidden in example 10.11: To apply the backchaining rule in line 7, we had to choose an "intermediate" position, the variable q in the transitivity rule from line 3. We chose the position $\mathrm{pos}(\underline{2}, \underline{4})$ because it brought us directly to the position that we wanted to go to. However, we could have chosen to take a completely different path, for instance via $\mathrm{pos}(\underline{2}, \underline{2})$, $\mathrm{pos}(\underline{3}, \underline{2})$, $\mathrm{pos}(\underline{4}, \underline{2})$, $\mathrm{pos}(\underline{4}, \underline{3})$, $\mathrm{pos}(\underline{4}, \underline{4})$ and $\mathrm{pos}(\underline{3}, \underline{4})$. This is where we have to find an appropriate control mechanism to guide the search for a path. In appendix B, an implementation of example 10.5 is given in a language called Prolog, a programming language based on Horn clause theories. The control mechanism chosen there is "tabling" combined with the search for a shortest path, which gives a very efficient search strategy for path finding. We will not explain the details of this control mechanism or Prolog here. For our purposes, it suffices to say that uniform proofs provide a foundation for the reasoning steps that Prolog takes and that an algorithm can be obtained from the uniform proof system by complementing it with an appropriate search strategy for the substitution σ in the backchaining rule.

Remark. The uniform proof system has quite a few less rules than $\mathbf{ND}_1$: all the elimination rules and the introduction rules for implication and universal quantification are not present. We have already explained why the general introduction and elimination rules for implication cause troubles. It is pos-

sible to add restricted versions of the introduction rules for implication and universal quantification [MN12; Mil+91]. However, the general elimination rules need to be treated completely differently.

10.4. Unification*

This section will not be relevant for the exam.

Solutions

Answers to the Quizzes of Chapter 8

Answer to quiz on page 27 There are exactly two possibilities because $P^{\mathcal{M}}$ must be a subset of $\mathbb{1}$. Thus, either $P^{\mathcal{M}} = \emptyset$ or $P^{\mathcal{M}} = \mathbb{1}$.

Answer to quiz on page 34 The formula $\forall x.\, \exists y.\, L(x,y) \wedge \neg I(x,y)$ translates in the language model $\mathcal{M}_l$ to "for every language U there is language V that strictly contains U: $U \subset V$." This is not true because the total language A^* is maximal. Indeed, formally we have for all valuations v and $V \subseteq A^*$ with $w = v[x \mapsto A^*]$ that

$$[\![L(x,y) \wedge \neg I(x,y)]\!]_{w[y \mapsto V]} = \begin{cases} 1, & A^* \subseteq U \text{ and } A^* \neq V \\ 0, & \text{otherwise} \end{cases} = 0.$$

This gives $[\![\exists y.\, L(x,y) \wedge \neg I(x,y)]\!]_w = 0$ and thus

$$\begin{aligned} &[\![\forall x.\, \exists y.\, L(x,y) \wedge \neg I(x,y)]\!]_v \\ &= \min\{[\![\exists y.\, L(x,y) \wedge \neg I(x,y)]\!]_{v[x \mapsto U]} \mid U \in A^*\} \\ &\leq [\![\exists y.\, L(x,y) \wedge \neg I(x,y)]\!]_w \\ &= 0 \end{aligned}$$

Answers to the Quizzes of Chapter 9

Answer to quiz on page 51 To derive symmetry, we apply the replacement rule to the formula $x \doteq s$, which we refer to as φ, as follows. First, we note that $\varphi[x := t] = t \doteq s$, which means that we can prove symmetry by showing that $\varphi[x := t]$ is derivable from $s \doteq t$. Second, we note that $\varphi[x := s] = s \doteq s$,

whence $\varphi[x := s]$ is provable by (Refl). Putting this together, we obtain the following derivation for symmetry of $\doteq$.

$$\dfrac{\dfrac{\Delta \mid \Gamma \vdash s \doteq t \qquad \overline{\Delta \mid \Gamma \vdash \varphi[x := s]}\ \text{(Refl)}}{\Delta \mid \Gamma \vdash \varphi[x := t]}\ \text{(Repl)}$$

Answers to the Quizzes of Chapter 10

Answer to quiz on page 67 The idea is that multiplication is given recursively by $0 \cdot n = 0$ and $(m + 1) \cdot n = (m \cdot n) + n$. This can be translated into the following two Horn clauses.

$$\begin{aligned} &\forall n. && M(0, n, 0) \\ &\forall m.\, \forall n.\, \forall i.\, \forall k.\, M(m, n, i) \land S(i, n, k) \to && M(s(m), n, k) \end{aligned}$$

Answer to quiz on page 71 No, if a signature $\mathcal{L}$ has no constants, then it is not possible to obtain an $\mathcal{L}$-term without variables. For example, suppose $\mathcal{L}$ is a signature with only one unary function symbol f. Then the only terms we can build over this signature are of the form $x, f(x), f(f(x)), \ldots$ for variables x. Thus, it is not possible to obtain a ground term without variables from $\mathcal{L}$. This can formally be proven by induction over first-order terms.

Bibliography

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. 2nd ed. Applied Logic Series. Springer Netherlands, 2002. ISBN: 978-1-4020-0763-7. DOI: 10.1007/978-94-015-9934-4. URL: https://www.springer.com/gp/book/9781402007637 (visited on 10/05/2020).

[Jac99] Bart Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. Amsterdam: North Holland, 1999.

[Kow79] Robert A. Kowalski. 'Algorithm = Logic + Control'. In: *Commun. ACM* 22.7 (1979), pp. 424–436. DOI: 10.1145/359131.359136. URL: https://doi.org/10.1145/359131.359136.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012. 306 pp. ISBN: ISBN 978-0-521-87940-8. URL: http://www.cambridge.org/de/academic/subjects/computer-science/programming-languages-and-applied-logic/programming-higher-order-logic?format=HB.

[Mil+91] Dale Miller, Gopalan Nadathur, Frank Pfenning and Andre Scedrov. 'Uniform Proofs as a Foundation for Logic Programming'. In: *Ann. Pure Appl. Logic* 51.1-2 (1991), pp. 125–157. DOI: 10.1016/0168-0072(91)90068-W. URL: https://doi.org/10.1016/0168-0072(91)90068-W.

[TS00] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. 2nd ed. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, 2000. ISBN: 0-521-77911-1.

[TvD88] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction*. Vol. 1 & 2. Studies in Logic and the Foundations of Mathematics 121 & 123. North-Holland, 1988. 384 pp. ISBN: 978-0-444-70266-1.

A. Tools

A.1. Formal Languages

Recall that A^* denotes the set of *words* over an alphabet A. Concretely, the set of words is given by

$$A^* = \{\varepsilon\} \cup \{a_0 a_1 \cdots a_n \mid n \in \mathbb{N}, a_k \in A\},$$

where ε is the empty word. For instance, if $A = \{a, b\}$, then A^* contains the singleton words a and b, and longer words like $abbaa$. The set of *languages* over A is the powerset $\mathcal{P}(A^*)$, that is, the set of all subsets of A^*.

B. Logic Programming

```
% :- table path(_,_,lattice(shortest/3))
% :- table conn/2

% Partial order of lists by length; used in tabled execution
shortest(P1, P2, P) :-
   length(P1, L1),
   length(P2, L2),
   (L1 < L2 -> P = P1; P = P2).

% Right
adjacent(pos(X1,Y1), pos(X2, Y1)) :- succ(X1, X2), X1 < 6.
% Down
adjacent(pos(X1,Y1), pos(X1, Y2)) :- succ(Y1, Y2), Y1 < 4.
% Left
adjacent(pos(X1,Y1), pos(X2, Y1)) :- succ(X2, X1).
% Up
adjacent(pos(X1,Y1), pos(X1, Y2)) :- succ(Y2, Y1).

% Can we go from U to V?
step(U, V) :- adjacent(U, V), free(V).

% conn(U, V) holds if two positions U and V connected.
conn(U, U).
conn(U, V) :-
  conn(W, V),
  step(U, W).

% Can our robot reach the goal?
connr :- robot(U), goal(V), conn(U, V).

% path(U, V, P) holds if P is a path from U to V. A path is here a list of
    positions.
path(U, U, [U]).
path(U, V, [U|P]) :-
  path(W, V, P),
  step(U, W).

% A path P with route(P) leads our robot from the initial position to the
    goal.
route(P) :- robot(U), goal(V), path(U, V, P).

```

```
% Initial position of robot.
robot(pos(2, 3)).

% Position of goal.
goal(pos(5,1)).

% All the positions that do not contain an obstacle.
free(pos(1,1)).
free(pos(1,4)).

free(pos(2,2)).
free(pos(2,3)).
free(pos(2,4)).

free(pos(3,1)).
free(pos(3,2)).
free(pos(3,4)).

free(pos(4,1)).
free(pos(4,2)).
free(pos(4,3)).
free(pos(4,4)).

free(pos(5,1)).
free(pos(5,3)).

free(pos(6,1)).
free(pos(6,2)).
free(pos(6,3)).
```