CHAPTER 13
PREDICATE LANGUAGES

1 Predicate Languages

Propositional Languages are also called Zero Order Languages, as opposed to Predicate Languages that are called First Order Languages. The same applies to the use of terms Propositional and Predicate Logic; they are often called zero Order and First Order Logics and we will use both terms equally.

We will work with several different predicate languages, depending on what applications we have in mind. All of those languages have some common features, and we begin with these.

Propositional connectives We define the set of propositional connectives

$$CON$$

in the same way as in the case of the propositional languages. It means that we assume the following.

1. The set of connectives is non-empty and finite, i.e.

$$0 < cardCON < \aleph_0.$$

2. We consider only the connectives with one or two arguments.

Quantifiers We adopt two quantifiers; $\forall$ (for all, the universal quantifier) and $\exists$ (there exists, the existential quantifier), i.e. we have the following set of quantifiers

$$\mathbf{Q} = \{\forall, \exists\}.$$

In a case of the classical logic and the logics that extend it, it is possible to adopt only one quantifier and to define the other in terms of it and propositional connectives. It is impossible in a case of some non-classical logics, for example the intuitionistic logic. But even in the case of classical logic two quantifiers express better common intuition, so we assume that we have two of them.

Parenthesis. As in the propositional case, we adopt the signs (and) for our parenthesis., i.e. we define a set PAR as

$$PAR = \{(,)\}.$$

Variables We assume that we always have a countably infinite set VAR of variables, i.e. we assume that

$$cardVAR = \aleph_0.$$

We denote variables by $x, y, z, ...$, with indices, if necessary, what we often express by writing

$$VAR = \{x_1, x_2, ...\}.$$

The set of propositional connectives CON defines a propositional part of the predicate logic language. What really differ one predicate language from the other is the choice of additional symbols to the symbols described above. These are called predicate symbols, function symbols, and constant symbols. i.e. a particular predicate language is determined by specifying the following sets of symbols.

Predicate symbols Predicate symbols represent relations. We assume that we have an non empty, finite or countably infinite set

$$\mathbf{P}$$

of predicate, or relation symbols. i.e. we assume that

$$0 < card\mathbf{P} \leq \aleph_0.$$

We denote predicate symbols by $P, Q, R, ...$, with indices, if necessary.

Each predicate symbol $P \in \mathbf{P}$ has a positive integer $\#P$ assigned to it; if $\#P = n$ then say P is called an n-ary (n - place) predicate (relation) symbol.

Function symbols We assume that we have a finite (may be empty) or countably infinite set

$$\mathbf{F}$$

of function symbols. I.e. we assume that

$$0 \leq card\mathbf{F} \leq \aleph_0.$$

When the set $\mathbf{F}$ is empty we say that we deal with a language without functional symbols.

We denote functional symbols by $f, g, h, ...$, with indices, if necessary.

Similarly, as in the case of predicate symbols, each function symbol $f \in \mathbf{F}$ has a positive integer $\#f$ assigned to it; if $\#f = n$ then say f is called an n-ary (n - place) function symbol.

Constant symbols We also assume that we have a finite (may be empty) or countably infinite set

$$\mathbf{C}$$

of constant symbols. I.e. we assume that

$$0 \leq card\mathbf{C} \leq \aleph_0.$$

The elements of $\mathbf{C}$ are denoted by $c, d, e...$, with indices, if necessary, what we often express by writing

$$\mathbf{C} = \{c_1, c_2, ...\}.$$

When the set $\mathbf{C}$ is empty we say that we deal with a language without constant symbols.

Sometimes the constant symbols are defined as 0-ary function symbols, i.e. $\mathbf{C} \subset \mathbf{F}$. We single them out as a separate set for our convenience.

Disjoint sets We assume that all of the above sets are disjoint.

Alphabet The union of all of above disjoint sets is called the *alphabet* $\mathcal{A}$ of the predicate language, i.e.

$$\mathcal{A} = VAR \cup CON \cup PAR \cup \mathbf{Q} \cup \mathbf{P} \cup \mathbf{F} \cup \mathbf{C}.$$

Observe, that once the set of propositional connectives is fixed, the predicate language is determined by the sets **P, F** and **C**, so we use the notation

$$\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$$

for the predicate language $\mathcal{L}$ determined by **P, F** and **C**. If there is no danger of confusion, we may abbreviate $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$ to just $\mathcal{L}$. If for some reason we need to stress the set of propositional connectives involved, we will also use the notation

$$\mathcal{L}_{CON}(\mathbf{P}, \mathbf{F}, \mathbf{C})$$

to denote the predicate language $\mathcal{L}$ determined by **P, F, C** and the set of propositional connectives CON.

We sometimes allow the same symbol to be used as an n-place relation symbol, and also as an m-place one; no confusion should arise because the different uses can be told apart easily. Similarly for function symbols.

Having defined the basic elements of syntax, the alphabet, we can now complete the formal definition of the predicate language by defining two more complex sets: the set T of all terms and the set $\mathcal{F}$ of all well formed formulas of the language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$.

Terms The set

$$T$$

of terms of the predicate language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$ is the smallest set $T \subset \mathcal{A}^*$ meeting the conditions:

1. any variable is a term, i.e. $VAR \subseteq T$;
2. any constant symbol is a term, i.e. $\mathbf{C} \subseteq T$;
3. if f is an nplace function symbol, i.e. $f \in \mathbf{F}$ and $\#f = n$ and $t_1, t_2, ..., t_n \in T$, then $f(t_1, t_2, ..., t_n) \in T$.

Example If $f \in \mathbf{F}, \#f = 1$, i.e. f is a one place function symbol, x, y are variables, c, d are constants, i.e. $x, y \in VAR, c, d \in \mathbf{C}$, then the following are terms:

$$x,\ y,\ f(x),\ f(y),\ f(c),\ f(d),\ ff(x),\ ff(y),\ ff(c),\ ff(d), ...etc.$$

Example If $\mathbf{F} = \emptyset, \mathbf{C} = \emptyset$, then the set T of terms consists of variables only, i.e.

$$T = VAR = \{x_1, x_2, ...\}.$$

From the above we get the following observation.

Remark 1.1 *For any predicate language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$, the set T of its terms is always non-empty.*

Example If $f \in \mathbf{F}, \#f = 1$, $g \in \mathbf{F}, \#g = 2$, $x, y \in VAR, c, d \in \mathbf{C}$, then some of the terms are the following:

$$f(g(x, y)),\ f(g(c, x)),\ g(ff(c), g(x, y)),\ g(c, g(x, f(c))).$$

From time to time, the logicians are and we may be informal about how we write terms. For instance, if we denote a two place function symbol g by $+$, we may write $x + y$ instead $+(x, y)$. Because in this case we can think of $x + y$ as an unofficial way of designating the "real" term $+(x, y)$, or even $g(x, y)$.

Before we define the set of formulas, we need to define one more set; the set of atomic, or elementary formulas. They are the "smallest" formulas as were the propositional variables in the case of propositional languages.

Atomic formulas An atomic formula of a predicate language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$ is any element of $\mathcal{A}^*$ of the form

$$R(t_1, t_2, ..., t_n),$$

where $R \in \mathbf{P}, \#R = n$, i.e. R is n-ary relational symbol and $t_1, t_2, ..., t_n$ are terms. The set of all atomic formulas is denoted by $\mathcal{AF}$ and is defines as

$$\mathcal{AF} = \{R(t_1, t_2, ..., t_n) \in \mathcal{A}^* : R \in \mathbf{P},\ t_1, t_2, ..., t_n \in T,\ \#R = n,\ n \geq 1\}.$$

Example Consider a language

$$\mathcal{L}(\emptyset, \{P\}, \emptyset),$$

for #P = 1, i.e. a language without neither functional, nor constant symbols, and with one, one-place predicate symbol P. The set of atomic formulas contains all formulas of the form $P(x)$, for x any variable, i.e.

$$\mathcal{AF} = \{P(x) : x \in VAR\}.$$

Example Let now

$$\mathcal{L} = \mathcal{L}(\{f, g\}, \{R\}, \{c, d\}),$$

for #f = 1, #g = 2 , #R = 2, i.e. $\mathcal{L}$ has two functional symbols: one -place symbol f and two-place symbol g; one two-place predicate symbol R, and two constants: c,d. Some of the atomic formulas in this case are the following.

$$R(c, d),\ R(x, f(c)),\ R(f(g(x, y)), f(g(c, x))),\ R(y, g(c, g(x, f(c)))).$$

Now we are ready to define the set $\mathcal{F}$ of all well formed formulas of the language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$.

Formulas The set

$$\mathcal{F}$$

of all well formed formulas, called shortly set of formulas, of the language $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$ is the smallest set meeting the following conditions:

1. any atomic formula of $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$ is a formula, i.e.

$$\mathcal{AF} \subseteq \mathcal{F};$$

2. if A is a formula of $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$, $\bigtriangledown$ is a one argument propositional connective, then $\bigtriangledown A$ is a formula of $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$, i.e. if the following recursive condition holds

$$if\ A \in \mathcal{F}, \bigtriangledown \in C_1,\ then\ \bigtriangledown A \in \mathcal{F};$$

3. if A, B are formulas of $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$, $\circ$ is a two argument propositional connective, then $(A \circ B)$ is a formula of $\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C})$, i.e. if the following recursive condition holds

$$if\ A \in \mathcal{F}, \bigtriangledown \in C_2,\ then\ (A \circ B) \in \mathcal{F};$$

4. if A is a formula of $\mathcal{L}(\mathbf{P},\mathbf{F},\mathbf{C})$ and x is a variable, then $\forall xA, \exists xA$ are formulas of $\mathcal{L}(\mathbf{P},\mathbf{F},\mathbf{C})$, i.e. if the following recursive condition holds

$$if\ A \in \mathcal{F},\ x \in VAR,\ \forall, \exists \in \mathbf{Q}\ then\ \forall xA,\ \exists xA \in \mathcal{F}.$$

Scope of the quantifier In $\forall xA,\ \exists xA$, A is in the scope of the quantifier $\forall,\ \exists$, respectively.

Example Let $\mathcal{L}$ be a language of the previous example, with the set of connectives $\{\cap, \cup, \Rightarrow, \neg\}$ i.e.

$$\mathcal{L} = \mathcal{L}_{\{\cap,\cup,\Rightarrow,\neg\}}(\{f,g\},\{R\},\{c,d\}),$$

for #f = 1, #g = 2 , #R = 2. Some of the formulas of $\mathcal{L}$ are the following.

$$R(c,d), \quad \exists xR(x,f(c)), \quad \neg R(x,y), \quad (\exists xR(x,f(c)) \Rightarrow \neg R(x,y)),$$
$$(R(c,d) \cap \exists xR(x,f(c))), \quad \forall yR(y,g(c,g(x,f(c)))), \quad \forall y\neg\exists xR(x,y).$$

The formula $R(x,f(c))$ is in a scope of the quantifier $\exists x$ in $\exists xR(x,f(c))$. The formula $(\exists xR(x,f(c)) \Rightarrow \neg R(x,y))$ isn't in a scope of any quantifier. The formula $(\exists xR(x,f(c)) \Rightarrow \neg R(x,y))$ is in the scope of $\forall$ in $\forall z(\exists xR(x,f(c)) \Rightarrow \neg R(x,y))$.

Now we are ready to define formally a predicate language.

Predicate language Let $\mathcal{A}, T, \mathcal{F}$ be the alphabet, the set of terms and the set of formulas as defined above. A predicate language $\mathcal{L}$ is a triple

$$\mathcal{L} = (\mathcal{A}, T, \mathcal{F}).$$

As we have said before, the language $\mathcal{L}$ is determined by the choice of the symbols of its alphabet, namely of the choice of connectives, predicate, function, and constant symbols. If we want to specifically mention this choice, we write

$$\mathcal{L} = \mathcal{L}_{CON}(\mathbf{P},\mathbf{F},\mathbf{C}) \ \text{ or } \ \mathcal{L} = \mathcal{L}(\mathbf{P},\mathbf{F},\mathbf{C}).$$

Given a predicate language $\mathcal{L} = (\mathcal{A}, T, \mathcal{F})$, we must distinguish between formulas like

$P(x,y), \quad \forall xP(x,y)$ and $\forall x\exists yP(x,y)$.

This is done by introducing the notion of free and bound variables, open and closed formulas (sentences).

Informally, in the formula

$$P(x, y)$$

both variables x and y are called *free* variables. They are not in the scope of any quantifier. The formula of that type (without quantifiers) is an open formula.

In the formula

$$\forall y P(x, y)$$

the variable x is free, the variable y is *bound* . The variable y is in the scope, is bounded by the quantifier $\forall$.

In the formula

$$\forall z P(x, y)$$

both x and y are free. In the formulas

$$\forall z P(z, y), \quad \forall x P(x, y)$$

only the variable y is free.

In the formula

$$\forall x (P(x) \Rightarrow \exists y Q(x, y))$$

there is no free variables, but in

$$(\forall x P(x) \Rightarrow \exists y Q(x, y))$$

the variable x (in $Q(x, y)$) is free.

Sometimes in order to distinguish more easily which variable is free and which is bound in the formula we might use the bold face type for the quantifier bound variables, i.e. to write the last formulas as

$$(\forall \mathbf{x} P(\mathbf{x}) \Rightarrow \exists \mathbf{y} Q(x, \mathbf{y})).$$

The formal definition of the set of free variables of a formula is the following.

Free variables The set $FV(A)$ of free variables of a formula A is defined by the induction of the degree of the formula as follows.

1. If A is an atomic formula, i.e. $A \in \mathcal{AF}$, then $FV(A)$ is just the set of variables appearing in the expression A;
2. for any unary propositional connective, i.e any $\bigtriangledown \in C_1$,
   $FV(\bigtriangledown A) = FV(A)$,
   i.e. the free variables of $\bigtriangledown A$ are the free variables of A;

3. for any binary propositional connective, i.e any $\circ \in C_2$,
   $FV(A \circ B) = FV(A) \cup FV(B)$,
   i.e. the free variables of $(A \circ B)$ are the free variables of A together with the free variables of B;
4. $FV(\forall xA) = FV(\exists xA) = FV(A) - \{x\}$,
   i.e. the free variables of $\forall xA$ and $\exists xA$ are the free variables of A, except for x.

Bound variables A variable is called bound if it is not free.

Sentence A formula with no free variables is called a sentence

Open formula A formula with no bound variables is called an open formula.

Example The formulas

$$\exists xQ(c, g(x, d)), \quad \neg\forall x(P(x) \Rightarrow \exists y(R(f(x), y) \cap \neg P(c)))$$

are sentences.

Example The formulas

$$Q(c, g(x, d)), \quad \neg(P(x) \Rightarrow (R(f(x), y) \cap \neg P(c)))$$

are open formulas.

Example The formulas

$$\exists xQ(c, g(x, y)), \quad \neg(P(x) \Rightarrow \exists y(R(f(x), y) \cap \neg P(c)))$$

are neither sentences nor open formulas. They contain some free and some bound variables; the variable y is free in the first formula, the variable x is free in the second.

It is common practice to use the notation

$$A(x_1, x_2, ..., x_n)$$

to indicate that $FV(A) \subseteq \{x_1, x_2, ..., x_n\}$ without implying that all of $x_1, x_2, ..., x_n$ are actually free in A. This is similar to the practise in algebra of writing $p(x_1, x_2, ..., x_n)$ for a polynomial p in the variables $x_1, x_2, ..., x_n$ without implying that all of them have nonzero coefficients.

Replacing x by t in A If $A(x)$ is a formula, and t is a term then

$$A(t/x)$$

or, more simply,

$$A(t)$$

denotes the result of replacing all occurrences of the free variable x by the term t throughout.

Notation When using the notation

$$A(t)$$

we always assume that none of the variables in t occur as bound variables in A.

The assumption that none of the variables in t occur as bound variables in A is essential, otherwise by substituting t on the place of x we would distort the meaning of $A(t)$.

Example Let $t = y$ and $A(x)$ is $\exists y(x \neq y)$, i.e. the variable y in t is bound in A. The substitution of t for x produces a formula $A(t)$ of the form $\exists y(y \neq y)$, which has a different meaning than $\exists y(x \neq y)$.

But if $t = z$, i.e. the variable z in t is not bound in A, then $A(t/x) = A(t)$ is $\exists y(z \neq y)$ and express the same meaning as $A(x)$.

Remark that if for example $t = f(z, x)$ we obtain $\exists y(f(z, x) \neq y)$ as a result of substitution of $t = f(z, x)$ for x in $\exists y(x \neq y)$.

This notation is convenient because we can agree to write as

$$A(t_1, t_2, ..., t_n) \quad or \quad A(t_1/x_1, t_2/x_2, ..., t_n/x_n)$$

a result of substituting in A the terms $t_1, t_2, ..., t_n$ for all free occurrences (if any) of $x_1, x_2, ..., x_n$, respectively.

But when using this notation we always assume that none of the variables in $t_1, t_2, ..., t_n$ occur as bound variables in A.

The above assumption that none of the variables in $t_1, t_2, ..., t_n$ occur as bound variables in A is often expressed using the notion: $t_1, t_2, ..., t_n$ *are free for all theirs variables in* A which is defined formally as follows.

Term t is free for y in A

If $A \in \mathcal{F}$ and t is a term, then t is said to be free for y if no free occurrence of y lies within the scope of any quantifier bounding variables in t.

Example Let A , B be the formulas

$$\forall y P(f(x, y), y), \quad \forall y P(f(x, z), y),$$

respectively. The term $t = f(x, y)$ is free for x and is not free for y in A. The term $t = f(x, z)$ is free for x and z in B. The term $t = y$ is not free neither for x nor for z in A, B.

Example Let A be a formula

$$(\exists x Q(f(x), g(x,z)) \cap P(h(x,y), y)).$$

The term $t_1 = f(x)$ is not free for x in A; the term $t_2 = g(x,z)$ is free for z only, term $t_3 = h(x,y)$ is free for y only because x occurs as a bound variable in A.

Notation If $A(x), A(x_1, x_2, ..., x_n) \in \mathcal{F}$ and $t, t_1, t_2, ..., t_n \in T$, then

$$A(t/x), \quad A(t_1/x_1, t_2/x_2, ..., t_n/x_n)$$

or, more simply just

$$A(t), \quad A(t_1, t_2, ..., t_n)$$

denotes the result of replacing all occurrences of the free variables $x, x_1, x_2, ..., x_n$, by the terms t, $t, t_1, t_2, ..., t_n$, respectively, assuming that $t, t_1, t_2, ..., t_n$ are free for all theirs variables in A.

2 Gentzen Style Proof System for Classical Predicate Logic - The System QRS

System QRS Definition

Let $\mathcal{F}$ denote a set of formulas of a Predicate (first Order) Logic Language

$$\mathcal{L}(\mathbf{P}, \mathbf{F}, \mathbf{C}) = \mathcal{L}_{\{\cap, \cup, \Rightarrow, \neg\}}(\mathbf{P}, \mathbf{F}, \mathbf{C})$$

for **P, F, C** countably infinite sets of predicate, functional, and constant symbols respectively.

The rules of inference of our system **QRS** will operate, as in the propositional case, on *finite sequences of formulas*, i.e. elements of $\mathcal{F}^*$, instead of just plain formulas $\mathcal{F}$, as in Hilbert style formalizations. We will denote the sequences of formulas by Γ, Δ, Σ, with indices if necessary.

that the truth assignment v makes it true if and only if it makes the formula of the form of thetrue.

The intuitive meaning of a sequence $\Gamma \in \mathcal{F}^*$ is that it represents a disjunction of all formulas of Γ, i.e. if Γ is a sequence

$$A_1, A_2, ..., A_n$$

then by δ_Γ we will understand the disjunction of all formulas of Γ.

As we know, the disjunction in classical logic is commutative, i.e., for any formulas A, B, C, $A \cup (B \cup C) \equiv (A \cup B) \cup C$, we w will denote any of those formulas by $A \cup B \cup C = \delta_{\{A,B,C\}}$. Similarly, we will write $\delta_\Gamma = A_1 \cup A_2 \cup ..., \cup A_n$.

The sequence Γ is said to be *satisfiable* (*falsifiable*) if the formula $\delta_\Gamma = A_1 \cup A_2 \cup ..., \cup A_n$ is satisfiable (falsifiable).

The sequence Γ is said to be *a tautology* if the formula $\delta_\Gamma = A_1 \cup A_2 \cup ..., \cup A_n$ is a tautology.

The system **QRS** consists of *one axiom* and *eleven rules* of inference. They form two groups. First is similar to the propositional case and called propositional connectives group. Each rule of this group introduces a new logical connective or its negation, so we will name them, as in the propositional case: $(\cup), (\neg\cup), (\cap), (\neg\cap), (\Rightarrow), (\neg\ \Rightarrow)$, and $(\neg\neg)$. The second group deals with the quantifiers. It consists of four rules. Two of them introduce the universal and existential quantifiers, and are named $(\forall)$ and $(\exists)$, respectively. The two others correspond to the De Morgan Laws and deal with the negation of the universal and existential quantifiers, and ere named $(\neg\forall)$ and $(\neg\exists)$, respectively.

As the *axiom* we adopt, as in propositional case, any sequence which contains any formula and its negation, i.e any sequence of the form

$$\Gamma_1, A, \Gamma_2, \neg A, \Gamma_3$$

or of the form

$$\Gamma_1, \neg A, \Gamma_2, A, \Gamma_3,$$

for any formula $A \in \mathcal{F}$ and any sequences of formulas $\Gamma_1, \Gamma_2, \Gamma_3 \in \mathcal{F}^*$.

We will denote the axioms by

$$\mathcal{AX}^*.$$

The proof system

$$\mathbf{QRS} = (\mathcal{F}^*, \mathcal{AX}^*, (\cup), (\neg\cup), (\cap), (\neg\cap), (\Rightarrow), (\neg\ \Rightarrow), (\neg\neg), (\neg\forall), (\neg\exists), (\forall), (\exists))$$

will be called a *Gentzen- style formalization* of classical predicate calculus.

In order to define the rules of inference of **QRS** we need to introduce some definitions. They are straightforward modification of the corresponding definitions for the propositional logic.

We will form now, as in the propositional case, a special subset $\mathcal{LIT} \subseteq \mathcal{F}$ of formulas, called a set of all *literals*, which is defined now as follows.

$$\mathcal{LIT} = \{A \in \mathcal{F} : A \in \mathcal{AF}\} \cup \{\neg A \in \mathcal{F} : A \in \mathcal{AF}\},$$

where $\mathcal{AF} \subseteq \mathcal{F}$ is the set of all atomic (elementary) formulas of the first order language, i.e. $\mathcal{AF} = \{P(t_1, ..., t_n) : P \in \mathbf{P}$ is any n-argument predicate symbol, and $t_i \in T$ are terms $\}$.

The elements of the first set of the above union are called *positive literals* and the elements of the second set of the above union are called *negative literals*. I.e atomic (elementary) formulas are called positive literals and the negation of an atomic (elementary) formula is called a negative literal.

Indecomposable formulas

Literals are also called the indecomposable formulas.

Now we form *finite sequences* out of formulas (and, as a special case, out of literals). We need to distinguish the sequences formed out of literals from the sequences formed out of other formulas, so we adopt exactly the same notation as in the propositional case. We will denote by $\Gamma^{'}$, $\Delta^{'}$, $\Sigma^{'}$ finite sequences (empty included) formed out of *literals* i.e. out of the elements of $\mathcal{LIT}$ i.e. we assume that $\Gamma^{'}$, $\Delta^{'}$, $\Sigma^{'} \in \mathcal{LIT}^{*}$.

We will denote by Γ, Δ, Σ the elements of $\mathcal{F}^{*}$ i.e the finite sequences (empty included) formed out of elements of $\mathcal{F}$.

We define the inference rules of **QRS** as follows.

Group 1: Propositional Inference rules

Disjunction rules

$$(\cup)\ \frac{\Gamma^{'}, A, B, \Delta}{\Gamma', (A \cup B), \Delta}, \qquad (\neg\cup)\ \frac{\Gamma^{'}, \neg A, \Delta \quad : \quad \Gamma^{'}, \neg B, \Delta}{\Gamma', \neg(A \cup B), \Delta}$$

Conjunction rules

$$(\cap)\ \frac{\Gamma^{'}, A, \Delta \quad ; \quad \Gamma^{'}, B, \Delta}{\Gamma', (A \cap B), \Delta}, \qquad (\neg\cap)\ \frac{\Gamma^{'}, \neg A, \neg B, \Delta}{\Gamma', \neg(A \cap B), \Delta}$$

Implication rules

$$(\Rightarrow)\ \frac{\Gamma^{'}, \neg A, B, \Delta}{\Gamma', (A \Rightarrow B), \Delta}, \qquad (\neg \Rightarrow)\ \frac{\Gamma^{'}, A, \Delta \quad : \quad \Gamma^{'}, \neg B, \Delta}{\Gamma', \neg(A \Rightarrow B), \Delta}$$

Negation rule

$$(\neg\neg)\ \frac{\Gamma', A, \Delta}{\Gamma', \neg\neg A, \Delta}$$

where $\Gamma' \in \mathcal{F}^*, \Delta \in \mathcal{F}'^*, A, B \in \mathcal{F}$.

Group 2: Quantifiers Rules

$(\exists)$

$$\frac{\Gamma', A(t), \Delta, \exists x A(x)}{\Gamma', \exists x A(x), \Delta}$$

where t is an arbitrary term.

$(\forall)$

$$\frac{\Gamma', A(y), \Delta}{\Gamma', \forall x A(x), \Delta}$$

where y is a free individual variable which does not appear in any formula in the conclusion, i.e. in the sequence $\Gamma', \forall x A(x), \Delta$.

$(\neg\forall)$

$$\frac{\Gamma', \exists x \neg A(x), \Delta}{\Gamma', \neg\forall x A(x), \Delta}$$

$(\neg\exists)$

$$\frac{\Gamma', \forall x \neg A(x), \Delta}{\Gamma', \neg\exists x A(x), \Delta}$$

$\Gamma' \in \mathcal{LIT}^*, \Delta \in \mathcal{F}^*, A, B \in \mathcal{F}$.

Note that $A(t), A(y)$ denotes a formula obtained from $A(x)$ by writing t, y, respectively, in place of all occurrences of x in A. The variable y in $(\forall)$ is called the *eigenvariable*. The condition: *where y is a free individual variable which does not appear in any formula in the conclusion* is called the *eigenvariable condition*.

All occurrences of y in $A(y)$ of the rule $(\forall)$ are fully indicated.

We define the notion of a *formal proof* in **QRS** as in any proof system, i.e., by a formal proof of a sequence Γ in the proof system

$$\mathbf{QRS} = (\mathcal{F}^*, \mathcal{A}\mathcal{X}^*, (\cup), (\neg\cup), (\cap), (\neg\cap), (\Rightarrow), (\neg \Rightarrow), (\neg\neg), (\neg\exists), (\neg\forall), \exists), (\forall))$$

we understand any sequence $\Gamma_1\Gamma_2....\Gamma_n$ of sequences of formulas (elements of $\mathcal{F}^*$, such that $\Gamma_1 \in \mathcal{A}\mathcal{X}^*$, $\Gamma_n = \Gamma$, and for all i $(1 < i \leq n)$ $\Gamma_i \in \mathcal{A}\mathcal{X}^*$, or Γ_i is a conclusion of one of the inference rules of **QRS** with all its premisses placed in the sequence $\Gamma_1\Gamma_2....\Gamma_{i-1}$.

As the proof system under consideration is fixed, we will write, as usual,

$$\vdash \Gamma$$

to denote that Γ has a formal proof in **QRS**.

As the proofs in **QRS** are sequences (definition of the formal proof) of sequences of formulas (definition of **GQ**) we will not use "," to separate the steps of the proof, i.e. will write the sequence the formal proof as a sequence $\Gamma_1\Gamma_2....\Gamma_n$ instead of $\Gamma_1, \Gamma_2, ..., \Gamma_n$, but usually we will use, as in the propositional case, the *proof trees* to represent the formal proofs. The *leafs* of the proof-tree are axioms, *nodes* are sequences such that each sequence on the tree follows from the ones immediately preceding it by one of the rules. The *root* is a sequence (formula). We will picture, and write our proof-trees with the node on the top, and leafs on the very bottom, instead of more common way, where the leafs are on the top and root is on the bottom of the tree.

In particular cases, as in the propositional case, we will write our proof- trees indicating additionally the name of the inference rule used at each step of the proof. For example, if the proof of a *theorem* from 3 *axioms* used subsequently the rules $(\cap), (\exists), (\forall), (\cap), (\neg\exists), (\neg\neg)$, and $(\Rightarrow)$, we will represent it as the following tree

$$Sequence(Formula)$$

$$| \, (\Rightarrow)$$

$$conclusion\ of\ (\neg\neg)$$

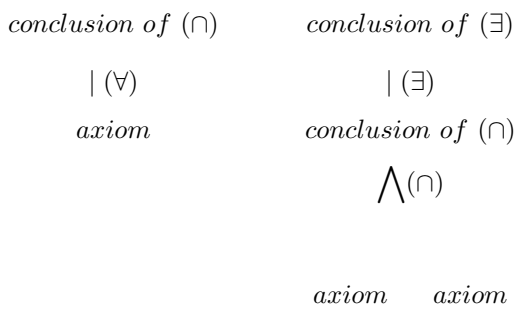
$$| \, (\neg\neg)$$

$$conclusion\ of\ (\neg\exists)$$

$$| \, (\neg\exists)$$

$$conclusion\ of\ (\cap)$$

$$\bigwedge(\cap)$$



Remark that the derivation trees don't represent a different *definition* of a formal proof. This remains the same in the Gentzen - style systems. Trees represent a certain *visualization* for those proofs and any formal proof in any system can be represented in a tree form. It is easy to define the tree-proofs precisely, as well as a general transformation procedure between the tree and the sequence form of the proofs, but we will explain it here on few examples only.

2.1 QRS Decomposition Trees

Given a formula $A \in \mathcal{F}$, we define its decomposition tree $\mathcal{T}_A$ in a similar way as in the propositional case. Observe that the inference rules of **QRS** can be divided in two groups: propositional connectives rules and quantifiers rules. The propositional connectives rules are: $(\cup), (\neg\cup), (\cap), (\neg\cap), (\Rightarrow), (\neg \Rightarrow)$, and $(\neg\neg)$. The quantifiers rules are: $(\forall), (\exists), (\neg\forall)$ and $(\neg\exists)$. We define the decomposition tree in the case of the propositional rules and the rules $(\neg\forall)$, $(\neg\exists)$ in the exactly the same way as in the propositional case. The case of the rules $(\forall)$ and $(\exists)$ is more complicated, as the rules contain the specific conditions under which they are applicable.

To define the way of decomposing the sequences of the form $\Gamma', \forall x A(x), \Delta$ or $\Gamma', \exists x A(x), \Delta$, i.e. to deal with the rules $(\forall)$ and $(\exists)$ in a way which would preserve the property of the *uniqueness* of the decomposition tree, we assume that all terms form a one-to one sequence

$$t_1, t_2, ..., t_n, ... \tag{1}$$

Observe, that by the definition, all free variables are terms, hence all free variables appear in the sequence 1. Let Γ be a sequence on the tree with $\forall$ as a main connective, i.e. Γ is of the form $\Gamma', \forall x A(x), \Delta$. We write a sequence $\Gamma', A(x), \Delta$ below it on the tree, as its child, where the variable x has to fulfill the following

[$\forall$]: x is the first free variable in the sequence 1 such that x does not appear in any formula in $\Gamma', \forall x A(x), \Delta$.

Observe, that the condition 2.1 corresponds to the restriction put on the application of the rule $(\forall)$.

If the main connective of Γ, i.e. the main connective of the first formula in Γ which is not an literal, is $(\exists)$. In this case Γ is of the form $\Gamma^{'}, \exists x A(x), \Delta$, we write a sequence $\Gamma^{'}, A(t), \Delta$ as its child, where the term t has to fulfill the following

[$\exists$]:

t is the first term in the sequence 1 such that the formula A(t) does not appear in any sequence which is placed above $\Gamma^{'}, A(t), \Delta$ on the tree.

The fact that the sequence 1 is one- to - one and the fact that, by the conditions 2.1 and 2.1, we always chose the first appropriate term (variable) from this sequence, guarantee that the decomposition process is also unique in the case of the quantifiers rules $(\forall)$ and $(\exists)$.

From all above, and we conclude the following.

Theorem 2.1 (Uniqueness) *For any formula $A \in \mathcal{F}$, its decomposition tree $\mathcal{T}_A$ is unique. If $\mathcal{T}_A$ is finite and all its leaves are axioms, then $\vdash A$ and $\mathcal{T}_A$ is a tree-proof of A in* **QRS**. *If $\mathcal{T}_A$ is finite and contains a non-axiom leaf or is infinite, then $\nvdash A$.*

2.1.1 Examples of Decomposition Trees

In all the examples below, the formulas $A(x), B(x)$ represent any formula. But there is no indication about their particular components, so they are treated as indecomposable formulas.

The decomposition tree of the de Morgan Law $(\neg\forall x A(x) \Rightarrow \exists x \neg A(x))$ is the following.

$$
\begin{array}{c}
(\neg\forall x A(x) \Rightarrow \exists x \neg A(x)) \\
| \, (\Rightarrow) \\
\neg\neg\forall x A(x), \exists x \neg A(x) \\
| \, (\neg\neg) \\
\forall x A(x), \exists x \neg A(x) \\
| \, (\forall) \\
A(x_1), \exists x \neg A(x)
\end{array}
$$

where x_1 is a first free variable in the sequence 1 such that x_1 does not appear in $\forall x A(x), \exists x \neg A(x)$

$$| \ (\exists)$$

$$A(x_1), \neg A(x_1), \exists x \neg A(x)$$

where x_1 is the first term (variables are terms) in the sequence 1 such that $\neg A(x_1)$ does not appear on a tree above $A(x_1), \neg A(x_1), \exists x \neg A(x)$

Axiom

The above tree ended with an axiom, so it represents a proof of $(\neg \forall x A(x) \Rightarrow \exists x \neg A(x))$ in **QRS**, i.e.

$$\vdash (\neg \forall x A(x) \Rightarrow \exists x \neg A(x))$$

The decomposition tree of $(\forall x A(x) \Rightarrow \exists x A(x))$ is the following.

$$(\forall x A(x) \Rightarrow \exists x A(x))$$

$$| \ (\Rightarrow)$$

$$\neg \forall x A(x), \exists x A(x)$$

$$| \ (\neg \forall)$$

$$\neg \forall x A(x), \exists x A(x)$$

$$\exists x \neg A(x), \exists x A(x)$$

$$| \ (\exists)$$

$$\neg A(t_1), \exists x A(x), \exists x \neg A(x)$$

where t_1 is the first term in the sequence 1, such that $\neg A(t_1)$ does not appear on the tree above $\neg A(t_1), \exists x A(x), \exists x \neg A(x)$

$$| \ (\exists)$$

$$\neg A(t_1), A(t_1), \exists x \neg A(x), \exists x A(x)$$

where t_1 is the first term in the sequence 1, such that $A(t_1)$ does not appear on the tree above $\neg A(t_1), A(t_1), \exists x \neg A(x), \exists x A(x)$

Axiom

The above tree also ended with the axiom, hence

$$\vdash (\forall x A(x) \Rightarrow \exists x A(x))$$

The decomposition tree of $(\exists x A(x) \Rightarrow \forall x A(x))$ is the following.

$$(\exists x A(x) \Rightarrow \forall x A(x))$$

$$| \ (\Rightarrow)$$

$$\neg \exists x A(x), \forall x A(x)$$

$$| \ (\neg \exists)$$

$$\forall x \neg A(x), \forall x A(x)$$

$$| \ (\forall)$$

$$\neg A(x_1), \forall x A(x)$$

where x_1 is a first free variable in 1 such that x_1 does not appear in $\forall x \neg A(x), \forall x A(x)$

$$| \ (\forall)$$

$$\neg A(x_1), A(x_2)$$

where x_2 is a first free variable in 1 such that x_2 does not appear in $\neg A(x_1), \forall x A(x)$, the sequence 1 is one-to- one, hence $x_1 \neq x_2$

Non - axiom

The decomposition tree, for any formula A is *unique*, so we conclude from the fact that the above tree has a non-axiom branch that

$$\nvdash (\exists x A(x) \Rightarrow \forall x A(x)).$$

The decomposition tree of $\exists x A(x)$ is the following.

$$\exists x A(x)$$

$$| \ (\exists)$$

$$A(t_1), \exists x A(x)$$

where t_1 is the first term in the sequence 1, such that $A(t_1)$ does not appear on the tree above $A(t_1), \exists x A(x)$

$$| \ (\exists)$$

$$A(t_1), A(t_2), \exists x A(x)$$

where t_2 is the first term in the sequence 1, such that $A(t_2)$ does not appear on the tree above $A(t_1), A(t_2), \exists x A(x)$, i.e. $t_2 \neq t_1$

$$| \ (\exists)$$

$$A(t_1), A(t_2), A(t_3), \exists x A(x)$$

where t_3 is the first term in the sequence 1, such that $A(t_3)$ does not appear on the tree above $A(t_1), A(t_2), A(t_3), \exists x A(x)$, i.e. $t_3 \neq t_2 \neq t_1$

$$| \ (\exists)$$

$$A(t_1), A(t_2), A(t_3), A(t_4), \exists x A(x)$$

$$| \ (\exists)$$

$$.....$$

$$| \ (\exists)$$

$$.....$$

Obviously, the above decomposition tree is infinite, what proves that

$$\not\vdash \exists x A(x).$$

We will find now the proof of the distributivity law $(\exists x(A(x) \cap B(x)) \Rightarrow (\exists x A(x) \cap \exists x B(x)))$ and show that we can't prove in **QRS** the inverse implication $((\exists x A(x) \cap \exists x B(x)) \Rightarrow \exists x(A(x) \cap B(x)))$. The decomposition tree of the first formula is the following.

$$(\exists x(A(x) \cap B(x)) \Rightarrow (\exists x A(x) \cap \exists x B(x)))$$

$$| \ (\Rightarrow)$$

$$\neg \exists x(A(x) \cap B(x)), (\exists x A(x) \cap \exists x B(x))$$

$$| (\neg\exists)$$
$$\forall x\neg(A(x) \cap B(x)), (\exists x A(x) \cap \exists x B(x))$$
$$| (\forall)$$
$$\neg(A(x_1) \cap B(x_1)), (\exists x A(x) \cap \exists x B(x))$$

where x_1 is a first free variable in the sequence 1 such that x_1 does not appear in $\forall x\neg(A(x) \cap B(x)), (\exists x A(x) \cap \exists x B(x))$

$$| (\neg\cap)$$
$$\neg A(x_1), \neg B(x_1), (\exists x A(x) \cap \exists x B(x))$$
$$\bigwedge(\cap)$$

$$\neg A(x_1), \neg B(x_1), \exists x A(x)$$
$$| (\exists)$$
$$\neg A(x_1), \neg B(x_1), A(t_1), \exists x A(x)$$

where t_1 is the first term in the sequence 1, such that $A(t_1)$ does not appear on the tree above $\neg A(x_1), \neg B(x_1), A(t_1), \exists x A(x)$ Observe, that it is possible that $t_1 = x_1$, as $A(x_1)$ does not appear on the tree above. By the definition of the sequence 1, x_1 is placed somewhere in it, i.e. $x_1 = t_i$, for certain $i \geq 1$. It means that after i applications of the step $(\exists)$ in the decomposition tree, we will get a step:

$$| (\exists)$$
$$\neg A(x_1), \neg B(x_1), ...A(x_1), \exists x A(x)$$

$$\neg A(x_1), \neg B(x_1), \exists x B(x)$$
$$| (\exists)$$
$$\neg A(x_1), \neg B(x_1), B(t_1), \exists x B(x)$$
$$| (\exists)$$
$$...$$
$$| (\exists)$$
$$\neg A(x_1), \neg B(x_1), ...B(x_1), \exists x B(x)$$

All leaves of the above tree are axioms, what means that

$$\vdash(\exists x(A(x) \cap B(x)) \Rightarrow (\exists x A(x) \cap \exists x B(x))).$$

Let's now construct, as the last example, a decomposition tree of

$$((\exists x A(x) \cap \exists x B(x)) \Rightarrow \exists x(A(x) \cap B(x))).$$

We will adopt, as on the right branch of the above tree, the shorthand notation used on this branch instead of the reasoning performed on the left branch, when the reasoning is similar to the one presented above. The decomposition tree is the following.

$$((\exists x A(x) \cap \exists x B(x)) \Rightarrow \exists x(A(x) \cap B(x)))$$

$$| \ (\Rightarrow)$$

$$\neg(\exists x A(x) \cap \exists x B(x)) \exists x(A(x) \cap B(x))$$

$$| \ (\neg\cap)$$

$$\neg\exists x A(x), \neg\exists x B(x), \exists x(A(x) \cap B(x))$$

$$| \ (\neg\exists)$$

$$\forall x \neg A(x), \neg\exists x B(x), \exists x(A(x) \cap B(x))$$

$$| \ (\forall)$$

$$\neg A(x_1), \neg\exists x B(x), \exists x(A(x) \cap B(x))$$

$$| \ (\neg\exists)$$

$$\neg A(x_1), \forall x \neg B(x), \exists x(A(x) \cap B(x))$$

$$| \ (\forall)$$

$$\neg A(x_1), \neg B(x_2), \exists x(A(x) \cap B(x))$$

By the reasoning similar to the reasonings in the previous examples we get that $x_1 \neq x_2$

$$| \ (\exists)$$

$$\neg A(x_1), \neg B(x_2), (A(t_1) \cap B(t_1)), \exists x(A(x) \cap B(x))$$

where t_1 is the first term in the sequence 1, such that $(A(t_1) \cap B(t_1))$ does not appear on the tree above $\neg A(x_1), \neg B(x_2), (A(t_1) \cap B(t_1)), \exists x(A(x) \cap B(x))$ Observe, that it is possible that $t_1 = x_1$, as $(A(x_1) \cap B(x_1))$ does not appear on the tree above. By the definition of the sequence 1, x_1 is placed somewhere in it, i.e. $x_1 = t_i$, for certain $i \geq 1$. For simplicity, we assume that $t_1 = x_1$ and get the sequence:

$$\neg A(x_1), \neg B(x_2), (A(x_1) \cap B(x_1)), \exists x(A(x) \cap B(x))$$

$$\bigwedge(\cap)$$

$$\neg A(x_1), \neg B(x_2),$$
$$A(x_1), \exists x(A(x) \cap B(x))$$
$$Axiom$$

$$\neg A(x_1), \neg B(x_2),$$
$$B(x_1), \exists x(A(x) \cap B(x))$$
$$| \ (\exists)$$
$$\neg A(x_1), \neg B(x_2), B(x_1),$$
$$(A(x_2) \cap B(x_2)), \exists x(A(x) \cap B(x))$$

where $x_2 = t_2$ ($x_1 \neq x_2$) is the first term in the sequence 1, such that $(A(x_2) \cap B(x_2))$ does not appear on the tree above $\neg A(x_1), \neg B(x_2), (B(x_1), (A(x_2) \cap B(x_2)), \exists x(A(x) \cap B(x))$. We assume that $t_2 = x_2$ for the reason of simplicity.

$$\bigwedge(\cap)$$

$\neg A(x_1),$	$\neg A(x_1),$
$\neg B(x_2),$	$\neg B(x_2),$
$B(x_1), A(x_2),$	$B(x_1), B(x_2),$
$\exists x(A(x) \cap B(x))$	$\exists x(A(x) \cap B(x))$
$\mid (\exists)$	$Axiom$

$$...$$
$$\bigwedge(\cap)$$
$$...$$
$$| \ (\exists)$$
$$...$$
$$| \ (\exists)$$
$$Infinite\ branch$$

The above decomposition tree contains an infinite branch what means that

$$\nvdash ((\exists x A(x) \cap \exists x B(x)) \Rightarrow \exists x(A(x) \cap B(x))).$$

2.2 Completeness Theorem for QRS

Given a first order language $\mathcal{L}$ with the set of variables VAR and the set of formulas $\mathcal{F}$. We have defined a notion of a model and counter- model of a formula A of $\mathcal{L}$ as follows.

Definition 2.1 (Model) *A structure $\mathcal{M} = [M, I]$ is called a model of $A \in \mathcal{F}$ if and only if*

$$(\mathcal{M}, v) \models A$$

for all valuations $v : VAR \longrightarrow M$.

M is called a universe of the model, I the interpretation.

Definition 2.2 (Counter - Model) *A structure $\mathcal{M} = [M, I]$ is called a counter-model of $A \in \mathcal{F}$ if and only if there is a valuation $v : VAR \longrightarrow M$, such that*

$$(\mathcal{M}, v) \not\models A.$$

The definition of the first order logic tautology is the following.

Definition 2.3 (Tautology) *For any $A \in \mathcal{F}$, A is called a tautology and denoted by $\models A$, if and only if all structures $\mathcal{M} = [M, I]$ ate models of A, i.e.*

$$\models A \quad if\ and\ only\ if \quad (\mathcal{M}, v) \models A$$

for all structures $\mathcal{M} = [M, I]$ and all valuations $v : VAR \longrightarrow M$.

Directly from the above definition we get the following, simple fact.

Fact 2.1 (Not Tautology) *For any $A \in \mathcal{F}$, A is not a tautology ($\not\models A$) if and only if there is a counter - model $\mathcal{M} = [M, I]$ of A, i.e. we can define M, I, and v such that $([M, I], v) \not\models A$.*

As our proof system is fixed, we will continue to use the notation $\vdash A$ ($\vdash \Gamma$) to denote that a formula A (a sequence Γ) has a proof in **QRS**.

Our goal now is to prove the Completeness Theorem for **QRS**. We do it, as in the propositional case, in two steps. First, we will prove the Soundness Lemmma:

Lemma 2.1 (Soundness Lemmma for QRS) *For any $\Gamma \in \mathcal{F}^*$,*

$$if \quad \vdash \Gamma \quad then \quad \models \Gamma,$$

and in particular, for any $A \in \mathcal{F}$,

$$if \quad \vdash A \quad then \quad \models A.$$

The proof is by step by step verification, similar to the propositional case and is left as an exercise. To complete the proof of the following

Theorem 2.2 (Completeness Theorem for QRS) *For any* $\Gamma \in \mathcal{F}^*$,

$$\vdash \Gamma \quad if \text{ and only } if \quad \models \Gamma,$$

and in particular, for any $A \in \mathcal{F}$,

$$\vdash A \quad if \text{ and only } if \quad \models A.$$

we have to prove the inverse implication to the Soundness Lemmma. We prove the formula case only and show that the case of sequences can be reduced to the formula case. I.e. we prove that the implication: $If \quad \models A \quad then \quad \vdash A$ is true. We do it, as in the propositional case, by proving the opposite implication to it, instead. I. e. we prove that the implication:

$$If \quad \nvdash A \quad then \quad \not\models A$$

is true. This means that we prove that for any formula A, if we know that from the fact that A does not have a proof in **QRS** ($\nvdash A$), we will be able to define its counter- model. The counter- model is defined, as in the propositional case, via the proof search (decomposition) tree. As we know, each formula A, generates its unique decomposition tree $\mathcal{T}_A$ and A has a proof only if this tree is finite and all its end sequences (leaves) are axioms. It means that if $\nvdash A$ then we have two cases to consider: tree $\mathcal{T}_A$ contains a leaf which is not axiom or is infinite. We will show how in both cases to construct a counter- model for A, determined by the infinite branch or non-axiom leaf of the decomposition tree $\mathcal{T}_A$. Before describing a general method of constructing the counter-models determined by the decomposition tree let's look at some examples. **Example 1**

Let's consider a particular case of the formula

$$(\exists x A(x) \Rightarrow \forall x A(x)),$$

i.e. let A be a formula

$$(\exists x(P(x) \cap R(x,y)) \Rightarrow \forall x(P(x) \cap R(x,y)))$$

for P , R one and two argument predicate symbols, respectively. The decomposition tree $\mathcal{T}_A$ is the following:

$$(\exists x(P(x) \cap R(x,y)) \Rightarrow \forall x(P(x) \cap R(x,y)))$$

$$| \ (\Rightarrow)$$

$$\neg\exists x(P(x) \cap R(x,y)), \forall x(P(x) \cap R(x,y))$$

$$| \ (\neg\exists)$$

$$\forall x\neg(P(x) \cap R(x,y)), \forall x(P(x) \cap R(x,y))$$

$$| \ (\forall)$$

$$\neg(P(x_1) \cap R(x_1,y)), \forall x(P(x) \cap R(x,y))$$

where x_1 is a first free variable in 1 such that x_1 does not appear in $\forall x\neg(P(x) \cap R(x,y)), \forall x(P(x) \cap R(x,y))$

$$| \ (\neg\cap)$$

$$\neg P(x_1), \neg R(x_1,y), \forall x(P(x) \cap R(x,y))$$

$$| \ (\forall)$$

$$\neg P(x_1), \neg R(x_1,y), (P(x_2) \cap R(x_2,y))$$

where x_2 is a first free variable in the sequence 1 such that x_2 does not appear in $\neg P(x_1), \neg R(x_1,y), \forall x(P(x) \cap R(x,y))$, the sequence 1 is one-to- one, hence $x_1 \neq x_2$

$$\bigwedge(\cap)$$

$$\neg P(x_1), \neg R(x_1,y), P(x_2) \qquad\qquad \neg P(x_1), \neg R(x_1,y), R(x_2,y)$$

$x_1 \neq x_2$, Non-axiom $\qquad\qquad$ $x_1 \neq x_2$, Non-axiom

There are two non-axiom leaves, to define a counter- model for A we need to chose only one of them, for example, let's choose

$$\neg P(x_1), \neg R(x_1,y), P(x_2).$$

We define a counter - model for A, i.e. a structure $\mathcal{M} = [M, I]$ and a valuation v, such that $(\mathcal{M}, v) \not\models A$ as follows.

1. $M = T$, i.e. the universe is the set of all terms of our language.

2. We define the relations P_I and R_I in the set of all terms T as follows: for any term $t \in T$,

$P_I(t)$ HOLDS iff the negation $\neg P(t)$ of the formula $P(t)$ appears on the non-axiom leaf, and $P_I(t)$ DOES NOT HOLD otherwise.

R_I is defined similarly: for any terms $t, s \in T$,

$R_I(t, s)$ HOLDS iff the negation $\neg R(t, s)$ of the formula $R(t, s)$ appears on the non-axiom leaf, and $R_I(t, s)$ DOES NOT HOLD otherwise.

It is easy to see that in particular case of our non-axiom leaf: $P_I(x_1)$ holds, $R(x_1, y)$ holds for any variable y, and $P(x_2)$ does not hold.

3. We define the valuation $v : VAR \longrightarrow T$ as IDENTITY, i.e., we put $v(x) = x$ for any $x \in VAR$.

Obviously, for such defined structure $[M, I]$ and valuation v we have that $([M, I], v) \models P(x_1)$, $([M, I], v) \models R(x_1, y)$, and $([M, I], v) \not\models P(x_2)$ and hence we obtain that

$$([M, I], v) \not\models \neg P(x_1), \neg R(x_1, y), P(x_2).$$

This proves that such defined structure $[M, I]$ is a counter model for a non-axiom leaf, and hence, by the fact that if one premiss of a rule of inference is false, so is the conclusion, it is a counter-model for all sequences on the branch which ends with this leaf, and hence in particular, it is a counter - model for A.

The case of the infinite tree is similar, even if a little bit more complicated. Observe first that the rule $(\exists)$ is the the only rule of inference (decomposition) which can "produce" an infinite branch. We first show how to construct the counter-model in the case of the simplest application of this rule, i.e. in the case of the formula

$$\exists x A(x)$$

where A is an one argument relational symbol. All other cases are similar to this one. The infinite branch $\mathcal{B}$in this case consists elements of the whole decomposition tree:

$$\exists x A(x)$$

$$| \ (\exists)$$

$$A(t_1), \exists x A(x)$$

where t_1 is the first term in the sequence 1, such that $A(t_1)$ does not appear on the tree above $A(t_1), \exists x A(x)$

$$| \ (\exists)$$

$$A(t_1), A(t_2), \exists x A(x)$$

where t_2 is the first term in the sequence 1, such that $A(t_2)$ does not appear on the tree above $A(t_1), A(t_2), \exists x A(x)$, i.e. $t_2 \neq t_1$

$$| \ (\exists)$$

$$A(t_1), A(t_2), A(t_3), \exists x A(x)$$

where t_3 is the first term in the sequence 1, such that $A(t_3)$ does not appear on the tree above $A(t_1), A(t_2), A(t_3), \exists x A(x)$, i.e. $t_3 \neq t_2 \neq t_1$

$$| \ (\exists)$$

$$A(t_1), A(t_2), A(t_3), A(t_4), \exists x A(x)$$

$$| \ (\exists)$$

$$.....$$

$$| \ (\exists)$$

$$.....$$

i.e.

$$\mathcal{B} = \{\exists x A(x), A(t_1), A(t_2), A(t_2), A(t_4), ...\}$$

where $t_1, t_2, ...$ is a one - to one sequence of all elements of the set of terms T.

This means that the infinite branch $\mathcal{B}$ contains with the formula $\exists x A(x)$ all its instances $A(t)$, for all terms $t \in T$.

We define the structure $[M, I]$ and valuation v in a similar way as in the previous example, i.e. we take as the universe M the set of all terms T, we define A_I as follows: $A_I(t)$ HOLDS if $\neg A(t) \in \mathcal{B}$ and $A_I(t)$ DOES NOT HOLDS if $A(t) \in \mathcal{B}$. We take, as before, the identity function, as the valuation, $v(x) = x$ for any $x \in VAR$ and define the interpretation I for functional symbols as follows. For any constant c, we put $c_I = c$, for any variable x, $x_I = v(x) = x$, and for any n-argument functional symbol f, we have still to define $f_I : T^n \longrightarrow T$. Observe that by definition, the function f_I has to assign a certain term to a sequence of terms $t_1, t_2, ..., t_n$ and the interpretation I says how we do it. Let's define:

$$f_I(t_1, t_2, ..., t_n) = f(t_1, t_2, ..., t_n).$$

It is easy to see that for any formula $A(t) \in \mathcal{B}$,

$$([T, I], v) \not\models A(t).$$

But the $A(t) \in \mathcal{B}$ are all instances $\exists x A(x)$, hence

$$([T, I], v) \not\models \exists x A(x).$$

Problems:

1. Give an example of 4 formulas with finite or infinite proof search trees (decomposition trees). 2. Construct counter-models for the formulas in 1. Do it in two ways: find your own structure, follow the above examples, i.e. give a counter-model determined by the proof search tree.

2. Write the proof of completeness theorem for **QRS**. I.e. Follow the above examples to show the implication:

$$If \quad \nvdash A \quad then \quad \not\models A$$

for ANY formula A.