LOGICS FOR COMPUTER SCIENCE: Classical and Non-Classical Springer 2019

Anita Wasilewska

Chapter 10
Predicate Automated Proof Systems
Completeness of Classical Predicate Logic

CHAPTER 10 SLIDES

Chapter 10
Predicate Automated Proof Systems
Completeness of Classical Predicate Logic

Slides Set 1

PART 1: **QRS** Proof System

PART 2: Proof of **QRS** Completeness

Slides Set 2

PART 3: Skolemization and Clauses

Chapter 10
Predicate Automated Proof Systems
Completeness of Classical Predicate Logic

Slides Set 1

PART 1: **QRS** Proof System

Predicate Automated Proof Systems
Introduction

We define and discuss here Rasiowa and Sikorski Gentzen style proof system **QRS** for classical predicate logic

The propositional version of it, the **RS** proof system, was studied in detail in chapter 6

These both proof systems **RS** and **QRS** admit a constructive proof of **completeness** theorem

Predicate Automated Proof Systems
Introduction

We adopt Rasiowa, Sikorski (1961) technique of construction a counter model determined by a decomposition tree to prove **QRS** completeness theorem

The proof, presented here is a generalization of the completeness proofs of **RS** and other Gentzen style propositional systems presented in details in chapter 6

We refer the reader to the chapter 6 as it provides a good introduction to the subject

Predicate Automated Proof Systems
Introduction

The other Gentzen type predicate proof system, including the original Gentzen proof systems **LK, LI** for classical and intuitionistic **predicate** logics are obtained from their **propositional** versions discussed in detail in chapter 6 by adding the **Quantifiers Rules** to them

It can be done in a similar way as a generalization of the propositional **RS** to the the predicate **QRS** system as presented here

We leave these generalizations as an **exercise** for the reader

Predicate Automated Proof Systems
Introduction

We also leave as an exercise the **predicate language** version of Gentzen proof of the **cut elimination** theorem, Hauptzatz (1935)

The Hauptzatz proof for the **predicate** classical **LK** and intuitionistic **LI** systems is easily obtained from the propositional proof included in chapter 6

There are of course other types of **automated proof** systems based on different methods of deduction

Predicate Automated Proof Systems
Introduction

There is a **Natural Deduction** mentioned by Gentzen in his Hauptzatz paper in 1935

It was later and fully developed by Dag Prawitz 1965)
It is now called Prawitz, or Gentzen-Prawitz **Natural Deduction**

There is a **Semantic Tableaux** deduction method invented by Evert Beth (1955)

It was consequently simplified and further developed by Raymond Smullyan (1968)
It is now often called Smullyan **Semantic Tableaux**

Predicate Automated Proof Systems
Introduction

Finally, there also is a **Resolution**

The **resolution method** can be traced back to Davis and Putnam (1960)
Their work is still known as Davis-Putnam **method**

The difficulties of Davis-Putnam method were eliminated by John Alan Robinson (1965)

He consequently **developed** it into what we call now Robinson **Resolution**, or just the **Resolution**

Predicate Automated Proof Systems
Introduction

The **resolution** proof system for propositional or predicate logic operates on a set of clauses as a basic expressions and uses a resolution rule as the only rule of inference

We define and prove **correctness** of effective procedures of **converting** any formula A into a corresponding set of **clauses** in both propositional and predicate cases

QRS Proof System

QRS Proof System

The components of the **QRS** proof system are as follows

Language $\mathcal{L}$

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

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

Expressions $\mathcal{E}$

Let $\mathcal{F}$ denote a set of formulas of $\mathcal{L}$. We adopt as the set of expressions the set of all finite sequences of formulas, i.e.

$$\mathcal{E} = \mathcal{F}^*$$

We will denote theexpressions of **QRS** by

$$\Gamma, \quad \Delta, \quad \Sigma, \ldots$$

with indices if necessary

Rules of Inference of **QRS**

The system **QRS** consists of two axiom schemas and eleven rules of inference

The rules of inference form **two groups**

First group is similar to the propositional case and contains propositional connectives rules:

$$(\cup), \quad (\neg\cup), \quad (\cap), \quad (\neg\cap), \quad (\Rightarrow), \quad (\neg\Rightarrow), \quad (\neg\neg)$$

Second group deals with the quantifiers and consists of four rules:

$$(\forall), \quad (\exists), \quad (\neg\forall), \quad (\neg\exists)$$

Logical Axioms of **RS**

We adopt as **logical axioms** LA of **QRS** any sequence of formulas which contains a formula and its negation, i.e any sequence

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

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

where $A \in \mathcal{F}$ is any **formula**

Proof System **QRS**

Formally we define the system **QRS** as follows

$$\mathbf{QRS} = (\mathcal{L}_{\{\cap,\cup,\Rightarrow,\neg\}}(\mathbf{P},\mathbf{F},\mathbf{C}),\ \ \mathcal{F}^*,\ \ LA,\ \ \mathcal{R})$$

where the set $\mathcal{R}$ of **inference rules** contains the following rules

$(\cup)$, $(\neg\cup)$, $(\cap)$, $(\neg\cap)$, $(\Rightarrow)$, $(\neg\Rightarrow)$, $(\neg\neg)$, $(\forall)$, $(\exists)$, $(\neg\forall)$, $(\neg\exists)$

and LA is the set of all **logical axioms**

Literals in **QRS**

Definition

Any atomic formula, or a negation of atomic formula is called a **literal**

We form, as in the propositional case, a special subset

$$LT \subseteq \mathcal{F}$$

of formulas, called a **set of all literals** defined now as follows

$$LT = \{A \in \mathcal{F} : A \in A\mathcal{F}\} \cup \{\neg A \in \mathcal{F} : A \in A\mathcal{F}\}$$

The elements of the set $\{A \in \mathcal{F} : A \in A\mathcal{F}\}$ are called **positive literals**

The elements of the set $\{\neg A \in \mathcal{F} : A \in A\mathcal{F}\}$ are called **negative literals**

Sequences of Literals

We denote by

$$\Gamma', \quad \Delta', \quad \Sigma' \ldots$$

finite sequences (empty included) formed out of **literals** i.e

$$\Gamma', \Delta', \Sigma' \in LT^*$$

We will denote by

$$\Gamma, \quad \Delta, \quad \Sigma \ldots$$

the elements of $\mathcal{F}^*$

Connectives Inference Rules of **QRS**

Group 1

Disjunction rules

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

Conjunction rules

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

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

Connectives Inference Rules of **QRS**

Group 1

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

Quantifiers Inference Rules of **QRS**

Group 2: Universal Quantifier rules

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

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

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

The variable y in the rule $(\forall)$ is called the eigenvariable

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

Quantifiers Inference Rules of **QRS**

Group 2: Existential Quantifier rules

$$(\exists)\ \frac{\Gamma^{'},\ A(t),\ \Delta, \exists x A(x)}{\Gamma^{'},\ \exists x A(x),\ \Delta} \qquad (\neg\exists)\ \frac{\Gamma^{'},\ \forall x \neg A(x),\ \Delta}{\Gamma^{'},\ \neg\exists x A(x),\ \Delta}$$

where $t \in T$ is an arbitrary term, $\Gamma^{'} \in LT^{*}$, $\Delta \in \mathcal{F}^{*}$, $A, B \in \mathcal{F}$

Note that $A(t), A(y)$ denotes a formula obtained from $A(x)$ by writing the term t or y, respectively, in place of all occurrences of x in A

Proofs and Proof Trees

By a **formal proof** of a sequence Γ in the proof system **QRS** we understand any sequence

$$\Gamma_1, \ \Gamma_2, ... \ \Gamma_n$$

of sequences of formulas (elements of $\mathcal{F}^*$), such that

1. $\Gamma_1 \in LA, \ \Gamma_n = \Gamma$, and

2. for all i $(1 \leq i \leq n)$, $\Gamma_i \in LA$, or Γ_i is a conclusion of one of the inference rules of **QRS** with all its premisses placed in the sequence $\Gamma_1, \ \Gamma_2, \ .... \ \Gamma_{i-1}$

Proofs and Proof Trees

We write, as usual,

$$\vdash_{QRS} \Gamma$$

to denote that the sequence 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 expressions $\mathcal{E}$) we will not use " ; " to separate the steps of the proof, and write the **formal proof** as

$$\Gamma_1;\ \Gamma_2;\ ....\ \Gamma_n$$

Proofs and Proof Trees

We write, however, the formal proofs in **QRS** as we did the propositional case (chapter 6),

in a form of **trees** rather then in a form of sequences

We adopt hence the following definition

Proof Tree

By a proof tree, or **QRS** - tree proof of Γ we understand a tree $\mathbf{T}_{\Gamma}$ of sequences satisfying the following conditions:

1. The topmost sequence, i.e the **root** of $\mathbf{T}_{\Gamma}$ is Γ,

2. all **leafs** are axioms,

3. the **nodes** are sequences such that each sequence on the tree follows from the ones immediately preceding it by one of the rules

of inference rules

Proof Trees

We picture, and write the proof trees with the **root** on the top, and **leafs** on the very bottom

In particular cases, as in the propositional case, we write the proof trees indicating additionally the **name** of the inference rule used at each step of the proof

For **example**, when in a proof of a formula A we use subsequently the rules

$$(\cap),\ (\cup),\ (\forall),\ (\cap),\ (\neg\neg),\ (\forall),\ (\Rightarrow)$$

we represent the proof of A as the following tree

Proof Trees

$\mathbf{T}_A$

Formula A

$|$ $(\Rightarrow)$

conclusion of $(\forall)$

$|$ $(\forall)$

conclusion of $(\neg\neg)$

$|$ $(\neg\neg)$

conclusion of $(\cap)$

$\bigwedge$ $(\cap)$

conclusion of $(\forall)$ *conclusion of* $(\cup)$

$|$ $(\forall)$ $|$ $(\cup)$

axiom   *conclusion of* $(\cap)$

$\bigwedge$ $(\cap)$

axiom   *axiom*

DecompositionTrees

The main advantage of the Gentzen type proof systems lies in the way we are able to **search** for proofs in them

Moreover, such proof search happens to be **deterministic** and **automatic**

We conduct **proof search** by treating inference rules as decomposition rules (see chapter 6) and by building **decomposition trees**

A general principle of building decomposition trees is the following.

Decomposition Trees

Decomposition Tree $\mathbf{T}_\Gamma$

For each $\Gamma \in \mathcal{F}^*$, a decomposition tree $\mathbf{T}_\Gamma$ is a tree build as follows

Step 1. The sequence Γ is the **root** of $\mathbf{T}_\Gamma$

For any node Δ of the tree we follow the steps bellow

Step 2. If Δ is **indecomposable** or an **axiom**, then Δ becomes a **leaf** of the tree

DecompositionTrees

Step 3. If Δ is **decomposable**, then we traverse Δ from **left** to **right** to identify the first **decomposable** formula B and identify inference rule treated as **decomposition** rule that is determined uniquely by B

We put its premiss as a **node below**, or its left and right premisses as the left and right **nodes below**, respectively

Step 4. We repeat steps **2.** and **3.** until we obtain only **leaves** or an **infinite branch**

In particular case when when Γ has only one element, namely a formula $A \in \mathcal{F}$, we call it a decomposition tree of A and denote by $\mathbf{T}_A$

QRS Decomposition Trees

Given a formula $A \in \mathcal{F}$, we define its **decomposition tree** $\mathbf{T}_A$ as follows

Observe that the inference rules of **QRS** can be divided in two groups: propositional connectives rules

$$(\cup), (\neg\cup), (\cap), (\neg\cap), (\Rightarrow), (\neg \Rightarrow)$$

and quantifiers rules

$$(\forall), \quad (\exists), \quad (\neg\forall) \quad (\neg\exists)$$

We define the **decomposition tree** in the case of the propositional rules and the quantifiers rules $(\neg\forall)$, $(\neg\exists)$ in the same way as for the propositional language (chapter 6)

QRS Decomposition Trees

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

$$\Gamma^{'}, \forall x A(x), \Delta \quad \text{or} \quad \Gamma^{'}, \exists x A(x), \Delta,$$

i.e. to deal with the rules quantifiers rules $(\forall)$ and $(\exists)$ **we assume** that **all terms** form a one-to one sequence

$$ST \qquad t_1, t_2, ..., t_n, ...$$

Observe, that by the definition, all free variables are **terms**, hence all free variables **appear** in the sequence ST

QRS Decomposition Trees

Let Γ be a sequence on the tree in which the first **indecomposable** formula **has** the quantifier $\forall$ as its **main connective**. It means that Γ is of the form

$$\Gamma', \forall_x A(x), \Delta$$

We write a sequence

$$\Gamma', A(y), \Delta$$

below Γ on the tree as its **child**, where the variable y fulfills the following condition

Condition 1 : the variable y is the **first** free variable in the sequence ST of terms such that y **does not** appear in any formula in $\Gamma', \forall x A(x), \Delta$

Observe, that the condition the **Condition 1** corresponds to the restriction put on the **application** of the rule $(\forall)$

QRS Decomposition Trees

Let now the first **indecomposable** formula in Γ **has** the quantifier $\exists$ as its **main** connective. It means that Γ is of the form

$$\Gamma', \exists x A(x), \Delta$$

We write a sequence

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

as its **child**, where the term t **fulfills** the following condition

Condition 2: the term t is the first term in the sequence ST of all terms such that the formula A(t) **does not** appear in any sequence on the tree which is **placed above**

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

QRS Decomposition Trees

Observe that the sequence ST of all terms is one- to - one and by the **Condition 1** and **Condition 2** we always chose the **first** appropriate term (variable) from the sequence ST

Hence the decomposition tree definition guarantees that the **decomposition** process is also unique in the case of the quantifier rules $(\forall)$ and $(\exists)$

From all above, and we **conclude** the following

QRS Decomposition Trees

Uniqueness Theorem

For any formula $A \in \mathcal{F}$,

(i) the decomposition tree $\mathbf{T}_A$ is unique

(ii) Moreover, the following conditions hold

1. If the decomposition tree $\mathbf{T}_A$ is **finite** and all its **leaves** are axioms, then

$$\vdash_{QRS} A$$

2. If $\mathbf{T}_A$ is **finite** and contains a non-axiom leaf, or $\mathbf{T}_A$ is **infinite**, then

$$\nvdash_{QRS} A$$

Examples of Decomposition Trees

In all the examples below, the formulas $A(x)$, $B(x)$ represent any formulas

But as there is no indication about their particular components, they are treated as **indecomposable** formulas

For example, the decomposition tree of the formula A representing the **de Morgan Law**

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

is constructed as follows

Examples of Decomposition Trees

$\mathbf{T}_A$

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

$$\mid (\Rightarrow)$$

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

$$\mid (\neg\neg)$$

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

$$\mid (\forall)$$

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

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

$$\mid (\exists)$$

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

where x_1 is the first term (variables are terms) in the sequence ST 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

Examples of Decomposition Trees

The above tree $\mathbf{T}_A$ ended with one leaf being axiom, so it represents a **proof** in **QRS** of the **de Morgan Law**

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

and . we have proved that

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

The decomposition tree $\mathbf{T}_A$ for a formula

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

is constructed as follows

Examples of Decomposition Trees

$\mathbf{T}_A$

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

$$\mid (\Rightarrow)$$

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

$$\mid (\neg \forall)$$

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

$$\mid (\exists)$$

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

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

$$\mid (\exists)$$

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

where t_1 is the first term in the sequence ST, 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

Examples of Decomposition Trees

The above tree also ended with the only leaf being the axiom, hence we have **proved** that

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

We know that the the inverse implication

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

is not a predicate tautology

Let's now look at its **decomposition tree** $\mathbf{T}_A$

Examples of Decomposition Trees

$\mathbf{T}_A$

$$\exists x A(x)$$

$$\mid (\exists)$$

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

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

$$\mid (\exists)$$

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

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

$$\mid (\exists)$$

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

where t_3 is the first term in the sequence ST, 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. $t_3 \neq t_2 \neq t_1$

$$\mid (\exists)$$

Examples of Decomposition Trees

We continue the decomposition

$$\mid (\exists)$$

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

where t_4 is the first term in the sequence ST, such that $A(t_4)$ does not appear on the tree above $A(t_1), A(t_2), A(t_3), A(t_4), \exists x A(x)$, i.e. $t_4 \neq t_3 \neq t_2 \neq t_1$

$$\mid (\exists)$$

.....

$$\mid (\exists)$$

.....

infinite branch

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

$$\nvdash \ \exists x A(x)$$

Examples of Decomposition Trees

We construct now a **proof** in **QRS** of the quantifiers **distributivity law**

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

and show that the proof in **QRS** of the inverse implication

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

does not exist, i.e. that

$$\nvdash \ ((\exists xA(x) \cap \exists xB(x)) \Rightarrow \exists x(A(x) \cap B(x)))$$

The decomposition tree $\mathbf{T}_A$ of the first formula is the following

Examples of Decomposition Trees

$\mathbf{T}_A$

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

$$\mid (\Rightarrow)$$

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

$$\mid (\neg\exists)$$

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

$$\mid (\forall)$$

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

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

$$\mid (\neg\cap)$$

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

$$\bigwedge (\cap)$$

Examples of Decomposition Trees

$$\bigwedge(\cap)$$

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

$$\mid (\exists)$$

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

where t_1 is the first term in the sequence ST, such that $A(t_1)$ does not appear on the tree above $\neg A(x_1), \neg B(x_1), A(t_1), \exists x A(x)$

$$\mid (\exists)$$

....

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

axiom

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

$$\mid (\exists)$$

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

$$\mid (\exists)$$

...

$$\mid (\exists)$$

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

axiom

Examples of Decomposition Trees

Observe, that it is possible to choose eventually a term $t_i = x_1$, as the formula $A(x_1)$ **does not** appear on the tree above the node

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

By the definition of the sequence ST, the variable 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 an axiom leaf

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

Examples of Decomposition Trees

All leaves of the above tree $\mathbf{T}_A$ are axioms, what means that we proved

$$\vdash_{QRS} (\exists x(A(x) \cap B(x)) \Rightarrow (\exists x A(x) \cap \exists x B(x))).$$

We construct now, as the last example, a decomposition tree $\mathbf{T}_A$ of the formula

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

Examples of Decomposition Trees

$$\mathbf{T}_A$$

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

$$\mid (\Rightarrow)$$

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

$$\mid (\neg\cap)$$

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

$$\mid (\neg\exists)$$

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

$$\mid (\forall)$$

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

$$\mid (\neg\exists)$$

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

$$\mid (\forall)$$

Examples of Decomposition Trees

$$| \, (\forall)$$

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

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 ST 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 ST of terms, 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:

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

$$\bigwedge (\cap)$$

Examples of Decomposition Trees

$$\bigwedge(\cap)$$

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

Axiom

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

see COMMENT

$$\bigwedge(\cap)$$

Examples of Decomposition Trees

COMMENT: where $x_2 = t_2$ $(x_1 \neq x_2)$ is the first term in the sequence ST, 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*
...	
$\mid (\exists)$	
infinite branch	

Examples of Decomposition Trees

The above decomposition tree $\mathbf{T}_A$ contains an **infinite branch** what means that

$$\nvdash_{QRS} ((\exists x A(x) \cap \exists x B(x)) \Rightarrow \exists x (A(x) \cap B(x)))$$

Chapter 10
Predicate Automated Proof Systems

Slides Set 1

PART 2: Proof of **QRS** Completeness

QRS Completeness

Our main goal now is to prove the **Completeness Theorem** for the predicate proof system **QRS**

The **proof** of the **Completeness Theorem** presented here is due to Rasiowa and Sikorski (1961), as is the proof system **QRS**

We adopted Rasiowa - Sikorski proof of **QRS** completeness to propositional case in chapter 6

QRS Completeness

Proofs of the **Completeness Theorem** in the propositional case and in the predicate case, are **both** constructive

Both are based on a direct construction of a **counter model** for any unprovable formula

The construction of the **counter model** for the **unprovable** formula A uses in both cases the **decomposition** tree $\mathbf{T}_A$

Rasiowa-Sikorski type of **constructive proofs** by defining a counter models determined by the decomposition trees relay heavily of the notion of **strong soundness**

QRS Semantics

Given a first order language $\mathcal{L}$

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

with the set *VAR* of variables and the set $\mathcal{F}$ of formulas

We **define**, after chapter 8 a notion of a **model** and a **counter- model** of a formula $A \in \mathcal{F}$

We establish the **semantics** for **QRS** by extending it to the set $\mathcal{F}^*$ of all finite sequences of formulas of $\mathcal{L}$

QRS Semantics

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 assignments $v : VAR \longrightarrow M$

We denote it by

$$\mathcal{M} \models A$$

M is called the **universe** of the model, I the **interpretation**

QRS Semantics

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 variable assignment $v : VAR \longrightarrow M$, such that

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

We denote it by

$$\mathcal{M} \not\models A$$

QRS Semantics

Tautology

A formula $A \in \mathcal{F}$ is called a **predicate tautology**

and is denoted by

$$\models A$$

if and only if **all** structures $\mathcal{M} = [M, I]$ are **models** of A, i.e.

$$\models A \quad \text{if and only if} \quad \mathcal{M} \models A$$

for all structures $\mathcal{M} = [M, I]$ for $\mathcal{L}$

QRS Semantics

For any sequence $\Gamma \in \mathcal{F}^*$, by δ_Γ we understand any **disjunction** of all formulas of Γ

A structure $\mathcal{M} = [M, I]$ is called a **model** of a sequence $\Gamma \in \mathcal{F}^*$ and denoted by

$$\mathcal{M} \models \Gamma$$

if and only if $\mathcal{M} \models \delta_\Gamma$

The sequence $\Gamma \in \mathcal{F}^*$ is a **predicate tautology** if and only if the formula δ_Γ is a predicate tautology, i.e.

$$\models \Gamma \text{ if and only if } \models \delta_\Gamma$$

Strong Soundnesss

Our **goal** now is to prove the Completeness Theorem for **QRS**

The **correctness** of the Rasiowa-Sikorski **constructive proof** depends on the strong soundness of the rules of inference of **QRS**

We define it (in general case) as follows

Strong Soundnesss

Strongly Sound Rules

Given a predicate language proof system

$$S = (\mathcal{L}, \mathcal{E}, LA, \mathcal{R})$$

An inference rule $r \in \mathcal{R}$ of the form

$$(r) \quad \frac{P_1 \;\; ; \;\; P_2 \;\; ; \;\; .... \;\; ; \;\; P_m}{C}$$

is **strongly sound** if the following condition holds for any structure $\mathcal{M} = [M, I]$ for $\mathcal{L}$

$\mathcal{M} \models \{P_1, P_2, .P_m\}$ if and only if $\mathcal{M} \models C$

Strong Soundnesss

A predicate language proof system $S = (\mathcal{L}, \mathcal{E}, LA, \mathcal{R})$ is **strongly sound** if and only if all logical axioms LA are tautologies and all its rules of inference $r \in \mathcal{R}$ are strongly sound

Strong Soundness Theorem
The proof system **QRS** is strongly sound

Proof
We have already proved in chapter 6 strong soundness of the propositional rules. The quantifiers rules are strongly sound by straightforward verification and is left as an exercise

Soundnesss Theorem

The strong soundness property is stronger then soundness property, hence also the following holds

QRS Soundness Theorem

For any $\Gamma \in \mathcal{F}^*$,

$$\text{if} \quad \vdash_{QRS} \ \Gamma, \quad \text{then} \quad \models \ \Gamma$$

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

$$\text{if} \quad \vdash_{QRS} A, \quad \text{then} \quad \models \ A$$

Proof of Completeness Theorem

Completeness Theorem

For any $\Gamma \in \mathcal{F}^*$,

$$\vdash_{QRS} \Gamma \text{ if and only if } \models \Gamma$$

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

$$\vdash_{QRS} A \text{ if and only if } \models A$$

Proof We prove the completeness part. We need to prove the formula A case only because the case of a sequence Γ can be reduced to the formula case of δ_Γ. I.e. we prove the implication:

$$\text{if } \models A, \text{ then } \vdash_{QRS} A$$

Proof of Completeness Theorem

We do it, as in the propositional case, by proving the opposite implication

$$\text{if } \nvdash_{QRS} A \text{ then } \not\models A$$

This means that we want prove that for any formula A, **unprovability** of A in **QRS** allows us to define its **counter- model**

Proof of Completeness Theorem

The counter- model is determined, as in the propositional case, by the decomposition tree $\mathbf{T}_A$

We have proved the following

Tree Theorem

Each formula A, generates its unique decomposition tree $\mathbf{T}_A$

and A **has a proof** if and only if

this tree is finite and all its **leaves** are axioms

Proof of Completeness Theorem

The **Tree Theorem** says says that we have two cases to consider:

(C1) the tree $\mathbf{T}_A$ is **finite** and contains a leaf which is not axiom, or

(C2) the tree $\mathbf{T}_A$ is **infinite**

We will show how to construct a counter- model for A in both cases:

a counter- model determined by a non-axiom leaf of the decomposition tree $\mathbf{T}_A$,

or a counter- model determined by an infinite branch of $\mathbf{T}_A$

Proof of Completeness Theorem

Proof in case **(C1)**

The tree $\mathbf{T}_A$ is **finite** and contains a non- axiom leaf

Before describing a general method of constructing the counter-model determined by the decomposition tree $\mathbf{T}_A$ we describe it, as an example, for a case of a general formula

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

and its **particular case**

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

where P, R are one and two argument predicate symbols, respectively

Proof of Completeness Theorem

First we build its decomposition tree:

$$\mathbf{T}_A$$

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

$$\mid (\Rightarrow)$$

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

$$\mid (\neg\exists)$$

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

$$\mid (\forall)$$

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

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

$$\mid (\neg\cap)$$

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

$$\mid (\forall)$$

Proof of Completeness Theorem

$$| (\forall)$$

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

where x_2 is a first free variable in the sequence of term ST 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 ST is one-to- one, hence $x_1 \neq x_2$

$$\bigwedge (\cap)$$

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

$x_1 \neq x_2$, Non-axiom

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

$x_1 \neq x_2$, Non-axiom

Proof of Completeness Theorem

There are two non-axiom leaves

In order to define a counter-model determined by the tree $\mathbf{T}_A$ we need to chose only one of them

Let's choose the leaf

$$L_A = \neg P(x_1), \neg R(x_1, y), P(x_2)$$

We use the **non-axiom leaf** L_A to define a structure $\mathcal{M} = [M, I]$ and an assignment v, such that

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

Such defined $\mathcal{M}$ is called a **counter - model** determined by the tree $\mathbf{T}_A$

Proof of Completeness Theorem

We take a the **universe** of $\mathcal{M}$ the set **T** of all terms of the language $\mathcal{L}$, i.e. we put $M = \mathbf{T}$.

We define the **interpretation** I as follows.

For any **predicate** symbol $Q \in \mathbf{P}, \#Q = n$ we put that

$Q_I(t_1, \ldots t_n)$ is **true** (holds) for terms $t_1, \ldots t_n$

if and only if

the negation $\neg Q_I(t_1, \ldots t_n)$ of the formula $Q(t_1, \ldots t_n)$ **appears** on the leaf L_A

and $Q_I(t_1, \ldots t_n)$ is **false** (does not hold) for terms $t_1, \ldots t_n$, otherwise

For any **functional** symbol $f \in \mathbf{F}, \#f = n$ we put

$$f_I(t_1, \ldots t_n) = f(t_1, \ldots t_n)$$

Proof of Completeness Theorem

It is easy to see that in particular case of our non-axiom leaf

$$L_A = \neg P(x_1), \ \neg R(x_1, y), \ P(x_2)$$

$P_I(x_1)$ is **true** (holds) for x_1, and **not true** for x_2

$R_I(x_1, y)$ is **true** (holds) for x_1 and for any $y \in VAR$

Proof of Completeness Theorem

We define the assignment $v : VAR \longrightarrow T$ as **identity**,

i.e., we put $v(x) = x$ for any $x \in VAR$

Obviously, for such defined structure $[M, I]$ and the assignment v we have that

$$([\mathbf{T}, I], v) \models P(x_1), \quad ([\mathbf{T}, I], v) \models R(x_1, y), \quad ([\mathbf{T}, I], v) \not\models P(x_2)$$

We hence obtain that

$$([\mathbf{T}, I], \ v) \not\models \neg P(x_1), \neg R(x_1, y), P(x_2)$$

This proves that such defined structure $[\mathbf{T}, I]$ is a **counter model** for a non-axiom leaf L_A and by the **Strong Soundness** we proved that

$$\not\models (\exists x(P(x) \cap R(x, y)) \Rightarrow \forall x(P(x) \cap R(x, y)))$$

C1: Proof of Completeness Theorem

C1: General Method

Let A be any formula such that

$$\nvdash_{QRS} \; A$$

Let $\mathbf{T}_A$ be a decomposition tree of A

By the fact that $\nvdash_{QRS}$ and **C1**, the tree $\mathbf{T}_A$ is **finite** and has a non axiom leaf

$$L_A \subseteq LT^*$$

By definition, the leaf L_A contains only atomic formulas and negations of atomic formulas

C1: Counter Model Definition

We use the **non-axiom leaf** L_A to define a structure $\mathcal{M} = [M, I]$, an assignment $v : VAR \longrightarrow M$, such that

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

Such defined structure $\mathcal{M}$ is called a **counter - model** determined by the tree $\mathbf{T}_A$

C1: Counter Model Definition

Structure $\mathcal{M}$ **Definition**

Given a formula A and a **non-axiom** leaf L_A

We define a structure

$$\mathcal{M} = [M, I]$$

and an assignment $v : VAR \longrightarrow M$ as follows

1. We take a the universe of $\mathcal{M}$ the set **T** of all **terms** of the language $\mathcal{L}$, i.e. we put

$$M = \mathbf{T}$$

C1: Counter Model Definition

2. For any predicate symbol $Q \in \mathbf{P}, \#Q = n$,

$$Q_I \subseteq \mathbf{T}^n$$

is such that $Q_I(t_1, \ldots t_n)$ **holds** (is true) for terms $t_1, \ldots t_n$

if and only if

the **negation** $\neg Q(t_1, \ldots t_n)$ of the formula $Q(t_1, \ldots t_n)$ appears on the leaf L_A and

$Q_I(t_1, \ldots t_n)$ **does not hold** (is false) for terms $t_1, \ldots, t_n$ otherwise

$\mathbf{C1}$: Counter Model Definition

3. For any constant $c \in \mathbf{C}$, we put $c_I = c$

For any variable x, we put $x_I = x$

For any functional symbol $f \in \mathbf{F}$, $\#f = n$

$$f_I : \mathbf{T}^n \longrightarrow \mathbf{T}$$

is **identity** function, i.e. we put

$$f_I(t_1, \dots t_n) = f(t_1, \dots t_n)$$

for all $t_1, \dots t_n \in \mathbf{T}$

4. We define the assignment $v : VAR \longrightarrow \mathbf{T}$ as identity, i.e. we put for all $x \in VAR$

$$v(x) = x$$

C1: Counter Model Definition

Obviously, for such defined structure $[\mathbf{T}, I]$ and the assignment v we have that

$$([\mathbf{T}, I], \; v) \not\models \; P \text{ if formula } P \text{ appears in } \; L_A,$$

$$([\mathbf{T}, I], \; v) \models \; P \text{ if formula } \neg P \text{ appears in } \; L_A$$

This proves that the structure $\mathcal{M} = [\mathbf{T}, I]$ and assignment v are such that

$$([\mathbf{T}, I], v) \not\models \; L_A$$

C1: Counter Model Definition

By the **Strong Soundness Theorem** we have that

$$(([\mathbf{T}, I], v) \not\models A$$

This proves $\mathcal{M} \not\models A$ and we proved that

$$\not\models A$$

This **ends** the proof of the case **C1**

C2: Counter Model Definition

Proof of case **C2**: $\mathbf{T}_A$ is **infinite**

The case of the **infinite tree** is similar to the **C1** case, even if a little bit more complicated

Observe that the rule $(\exists)$ is the **only** rule of inference (decomposition) which can "produces" 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 atomic formula

$$\exists x P(x)$$

for P one argument relational symbol. All other cases are. similar to this one

C2: Particular Case n

The **infinite** branch $\mathcal{B}_A$ in the following

$$\mathcal{B}_A$$

$$\exists x P(x)$$

$$\mid (\exists)$$

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

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

$$\mid (\exists)$$

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

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

$$\mid (\exists)$$

C2: Particular Case

$$| \ (\exists)$$

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

where t_3 is the first term in the sequence of terms, such that $P(t_3)$ does not appear on the tree above $P(t_1), P(_2), P(t_3), \exists x P(x)$, i.e. $t_3 \neq t_2 \neq t_1$

$$| \ (\exists)$$

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

$$| \ (\exists)$$

.....

$$| \ (\exists)$$

.....

The infinite branch $\mathcal{B}_A$, written from the top, in oder of appearance of formulas is

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

where $t_1, t_2, ...$ is a one - to one sequence of **all terms**

C2: Particular Case n

The **infinite** branch

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

contains with the formula $\exists x P(x)$ all its instances $P(t)$, for all terms $t \in \mathbf{T}$

We define the structure $\mathcal{M} = [M, I]$ and the assignment v as we did previously, i.e.

we take as the universe M the set $\mathbf{T}$ of all terms, and define P_I as follows:

$P_I(t)$ **holds** if $\neg P(t) \in \mathcal{B}_A$, and

$P_I(t)$ **does not hold** if $P(t) \in \mathcal{B}_A$

C2: Particular Case

For any constant $c \in \mathbf{C}$, we put $c_I = c$, for any variable x, we put $x_I = x$

For any functional symbol $f \in \mathbf{F}$, $\# f = n$

$$f_I : \mathbf{T}^n \longrightarrow \mathbf{T}$$

is **identity** function, i.e. we put

$$f_I(t_1, \ldots t_n) = f(t_1, \ldots t_n)$$

for all $t_1, \ldots t_n \in \mathbf{T}$

C2: Particular Case

We define the assignment $v : VAR \longrightarrow \mathbf{T}$ as identity, i.e. we put for all $x \in VAR$

$$v(x) = x$$

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

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

But the $P(t) \in \mathcal{B}$ are **all instances** of the formula $\exists x P(x)$, hence

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

and we proved

$$\not\models \exists x P(x)$$

C2: General Method

C2: General Method

Let A be any formula such that

$$\nvdash_{QRS} \ A$$

Let $\mathcal{T}_A$ be an **infinite** decomposition tree of the formula A

Let $\mathcal{B}_A$ be the **infinite branch** of $\mathbf{T}_A$, written from the top, in order of appearance of sequences $\Gamma \in \mathcal{F}^*$ on it, where $\Gamma_0 = A$, i.e.

$$\mathcal{B}_A = \{\Gamma_0, \ \Gamma_1, \ \Gamma_2, \ \ldots \ \Gamma_i, \ \Gamma_{i+1}, \ \ldots\}$$

C2: General Method

Given the infinite branch

$$\mathcal{B}_A = \{\Gamma_0, \ \Gamma_1, \ \Gamma_2, \ \ldots \ \Gamma_i, \ \Gamma_{i+1}, \ \ldots\}$$

We define a set

$$L\mathcal{F} \subseteq \mathcal{F}$$

of all **indecomposable** formulas appearing in at least one sequence $\Gamma_i, \ i \leq j$, i.e. we put

$$L\mathcal{F} = \{B \in LT : \text{ there is } \Gamma_i \in \mathcal{B}_A, \text{ such that } B \text{ iappiears } \Gamma_i\}$$

C2: General Method

Note, that the following holds

(1) If $i \leq i'$ and an **indecomposable** formula appears in Γ_i, then it also appears in $\Gamma_{i'}$

(2) Since **none** of Γ_i is an axiom, for every atomic formula $P \in A\mathcal{F}$, at **most one** of the formulas P and $\neg P$ is in $L\mathcal{F}$

Counter Model Definition

Let **T** be the set of all terms. We define the structure $\mathcal{M} = [\mathbf{T}, I]$, the interpretation **I** of constants and functional symbols, and the assignment v in the set **T**, as in previous cases

We define the interpretation **I** of predicates $Q \in \mathbf{P}$ as follows

For any predicate symbol $Q \in \mathbf{P}, \#Q = n$, we put

(1) $Q_I(t_1, \dots t_n)$ **does not hold** (is false) for terms $t_1, \dots t_n$ if and only if

$$Q_I(t_1, \dots t_n) \in L\mathcal{F}$$

(2) $Q_I(t_1, \dots t_n)$ **does holds** (is true) for terms $t_1, \dots t_n$ if and only if

$$Q_I(t_1, \dots t_n) \notin L\mathcal{F}$$

Counter Model Definition

Directly from the definition we we have that $\mathcal{M} \not\models LF$

Our goal now is to prove that

$$\mathcal{M} \not\models A$$

For this purpose we first introduce, for any formula $A \in \mathcal{F}$, an inductive definition of the **order** $ordA$ of the formula A

(1) If $A \in A\mathcal{F}$, then $ord\ A = 1$

(2) If $ordA = n$, then $ord\neg A = n + 1$

(3) If $ordA \leq n$ and $ordB \leq n$, then
$ord(A \cup B) = ord(A \cap B) = ord(A \Rightarrow B) = n + 1$

(4) If $ordA(x) = n$, then $ord\exists xA(x) = ord\forall xA(x) = n + 1$

Proof of Completeness Theorem

We conduct the proof of $\mathcal{M} \not\models A$ by contradiction.

Assume that

$$\mathcal{M} \models A$$

Consider now a set $M\mathcal{F}$ of all formulas B appearing in one of the sequences Γ_i of the branch $\mathcal{B}_A$, such that

$$\mathcal{M} \models B$$

We write the the set $M\mathcal{F}$ formally as follows

$$M\mathcal{F} = \{B \in \mathcal{F} : \text{for some } \Gamma_i \in \mathcal{B}_A, \ B \text{ is in } \Gamma_i \text{ and } \mathcal{M} \models B\}$$

Proof of Completeness Theorem

Observe that the formula A is in $M\mathcal{F}$ so

$$M\mathcal{F} \neq \emptyset$$

Let B' be a formula in $M\mathcal{F}$ such that

$$ordB' \leq ordB \quad \text{for every} \quad B \in M\mathcal{F}$$

There exists $\Gamma_i \in\in \mathcal{B}_A$ that is of the form Γ', B', Δ with an **indecomposable** Γ'

We have that B' **can not** be of the form

$$(*) \quad \neg\exists x A(x) \quad \text{or} \quad \neg\forall x A(x)$$

for if B' of the $(*)$ form **is** in $M\mathcal{F}$, then also formula $\forall x\neg A(x)$ or $\exists x\neg A(x)$ is in $M\mathcal{F}$ and the **orders** of the two formulas are equal

Proof of Completeness Theorem

We carry the same order argument and show that B' **can not** be of the form

$$(**) \quad (A \cup B), \ \neg(A \cup B), \ (A \cap B), \ \neg(A \cap B),$$

$$(A \Rightarrow B), \quad \neg(A \Rightarrow B), \quad \neg\neg A, \quad \forall x A(x)$$

The formula B' **can not** be of the form

$$(***) \quad \exists x B(x)$$

since then there exists term t and j such that $i \leq j$, and $B'(t)$ **appears** in Γ_j and the formula $B(t)$ is such that

$$\mathcal{M} \models B$$

Proof of Completeness Theorem

Thus $B(t) \in M\mathcal{F}$ and $ordB(t) < ordB'$

This **contradicts** the definition of B'

Since B' **is not** of the forms $(*)$, $(**)$, $(***)$, B' is **indecomposable**. Thus $B' \in L\mathcal{F}$ and consequently

$$\mathcal{M} \not\models B'$$

On the other hand B' is in the set $M\mathcal{F}$ and hence is one of the formulas satisfying

$$\mathcal{M} \models B'$$

This **contradiction** proves that $\mathcal{M} \not\models A$ and hence we proved that

$$\not\models A$$

This **ends** the proof of the Completeness Theorem for **QRS**

Chapter 10
Predicate Automated Proof Systems
Completeness of Classical Predicate Logic

Slides Set 2

PART 3: Skolemization and Clauses

Skolemization and Clauses : Introduction

A **resolution** based proof system for predicate logic operates on sets of **clauses** as a basic expressions and uses a resolution rule as the only rule of inference

The **first goal** of this part is to define an effective process of transformation of any formula A of a predicate language

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

into its **logically equivalent** set of clauses

$$\mathbf{C}_A$$

Skolemization and Clauses: Introduction

This process of transformation is done in two stages

S1. We convert any formula A of the predicate language $\mathcal{L}$ into an **open** formula A^* of a language $\mathcal{L}^*$ by a process of **elimination of quantifiers** from the original language $\mathcal{L}$

The elimination method is due to T. Skolem (1920) and is called Skolemization

Skolem Theorem

The resulting formula A^* is equisatisfiable with A:

it is **satisfiable** if and only if the original one is **satisfiable**

Skolemization and Clauses; Introduction

The stage **S1.** is performed as the first step in a resolution based automated theorem prover

S2. We define a proof system **QRS*** based on the Skolemized language

$$\mathcal{L}^*$$

and use it transform automatically any formula A^* of $\mathcal{L}^*$ into an logically equivalent set of clauses

$$\mathbf{C}_{A^*}$$

Skolemization and Clauses; Introduction

The final result of stages **S1.** and **S2.**, i.e. the set

$$\mathbf{C}_{A^*}$$

of clauses of the Skolemized language $\mathcal{L}^*$ called a **clausal form** of the original formula A of the language $\mathcal{L}$

The **transformation** process for any propositional formula A into its **logically equivalent** set $\mathbf{C}_A$ of clauses follows directly from the use of the propositional system **RS**

Clauses: Definition

Definition

Given a formal language $\mathcal{L}$, propositional or predicate

1. A **literal** as an atomic , or a negation of an atomic formula of $\mathcal{L}$. We denote by *LT* the set of all **literals** of $\mathcal{L}$

2. A **clause** C is a **finite set** of literals

Empty clause is denoted by $\{\}$

3. We denote by **C** any finite set of all **clauses**. For any $n \geq 0$,

$$\mathbf{C} = \{C_1, C_2, \ldots C_n\}$$

Clauses: Definition

Definition

Given a propositional or predicate language L, and a sequence

$$\Gamma \in LT^*$$

determined by Γ is a **set** form out of all elements of the sequence Γ

We we denote it by

$$C_\Gamma$$

Example

In particular,

1. if $\Gamma_1 = a, a, \neg b, c, \neg b, c$ and $\Gamma_2 = \neg b, c, a$, then

$$\mathcal{C}_{\Gamma_1} = \mathcal{C}_{\Gamma_2} = \{a, c, \neg b\}$$

2. If $\Gamma_1 = \neg P(x_1), \neg R(x_1, y), P(x_2), \neg P(x_1), \neg R(x_1, y), P(x_2)$ and $\Gamma_2 = \neg P(x_1), \neg R(x_1, y), P(x_2)$, then

$$\mathcal{C}_{\Gamma_1} = \mathcal{C}_{\Gamma_2} = \{\neg P(x_1), \neg R(x_1, y), P(x_2)\}$$

Clauses Semantics

Given a propositional or predicate language L

We use the following notations

For any **clause** $\mathcal{C}$, write

$$\delta_{\mathcal{C}}$$

for a disjunction of all literals in $\mathcal{C}$

Let $\mathcal{M}$ denote a **structure** $[M, I]$ for a predicate language $\mathcal{L}$, or a **truth assignment** v in case when $\mathcal{L}$ is a propositional language

Clauses Semantics

Definition

$\mathcal{M}$ is called a **model** for a clause C

$$\mathcal{M} \models C, \quad \text{if and only if} \quad \mathcal{M} \models \delta_C$$

$\mathcal{M}$ is called a **model** for a **set** $\mathbf{C}$ of clauses,

$$\mathcal{M} \models \mathbf{C} \quad \text{if and only if} \quad \mathcal{M} \models C \quad \text{for all clauses } C \in \mathbf{C}$$

Clauses Semantics

Definition

A formula A is **equivalent** with a set $\mathbf{C}$ of clauses

$$(A \equiv \mathbf{C}) \quad \text{if and only if} \quad A \equiv \sigma_{\mathbf{C}}$$

where $\sigma_{\mathbf{C}}$ is a conjunction of all formulas δ_{C} for all clauses $C \in \mathbf{C}$

Propositional Formula-Clauses Equivalency

Theorem (Formula-Clauses Equivalency)

For any formula A of a propositional language $\mathcal{L}$, there is an **effective procedure** of generating a corresponding set $\mathbf{C}_A$ of clauses such that

$$A \equiv \mathbf{C}_A$$

Proof

Given a formula A, we first use the **RS** system (chapter 6) to build a **decomposition tree** $\mathbf{T}_A$ of A

We form clauses out of the **leaves** of the tree $\mathbf{T}_A$, i.e. for every leaf L we create a clause C_L determined by L

Propositional Formula-Clauses Equivalency

We put

$$\mathbf{C}_A = \{C_L : \ L \text{ is a leaf of } \ \mathbf{T}_A\}$$

Directly from the **strong soundness** of rules of inference of **RS** we get

$$A \equiv \mathbf{C}_A$$

This ends the **proof** for the propositional case

Example

Example Consider a decomposition tree

$$\mathbf{T}_A$$

$$(((a \Rightarrow b) \cap \neg c) \cup (a \Rightarrow c))$$

$$\mid (\cup)$$

$$((a \Rightarrow b) \cap \neg c), (a \Rightarrow c)$$

$$\bigwedge (\cap)$$

$$(a \Rightarrow b), (a \Rightarrow c)$$

$$\mid (\Rightarrow)$$

$$\neg a, b, (a \Rightarrow c)$$

$$\mid (\Rightarrow)$$

$$\neg a, b, \neg a, c$$

$$\neg c, (a \Rightarrow c)$$

$$\mid (\Rightarrow)$$

$$\neg c, \neg a, c$$

Example

For the formula

$$A = (((a \Rightarrow b) \cap \neg c) \cup (a \Rightarrow c))$$

the leaves of its tree $\mathbf{T}_A$ are

$$L_1 = \neg a, b, \neg a, c \quad \text{and} \quad L_2 = \neg c, \neg a, c$$

The set of clauses determined by them is

$$\mathbf{C}_A = \{\{\neg a, b, c\}, \ \{\neg c, \neg a, c\}\}$$

By the Formula-Clauses Equivalency **Theorem**

$$A \equiv \mathbf{C}_A$$

Semantically it means that

$$A \equiv (((\neg a \cup b) \cup c) \cap ((\neg c \cup \neg a) \cup c))$$

Predicate Clausal Form

Theorem

For any formula A of a **predicate** language $\mathcal{L}$, there is an **effective** procedure of generating an **open** formula A^* of a quantifiers free language $\mathcal{L}^*$ and a set $\mathbf{C}_{A^*}$ of **clauses** such that

$$(*) \quad A^* \equiv \mathbf{C}_{A^*}$$

The set $\mathbf{C}_{A^*}$ of clauses of the language $\mathcal{L}^*$ with the property $(*)$ is called a **clausal form** of the formula A of $\mathcal{L}$

Proof of Theorem

Proof Given a formula A of a language $\mathcal{L}$

The **open** formula A^* of the **quantifiers free** language $\mathcal{L}^*$ is obtained by the Skolemization process

The effectiveness and correctness of the process follows from **PNF Theorem** and **Skolem Theorem** described in the next section

As the next step, we define there a proof system **QRS*** based on the **quantifiers free** language $\mathcal{L}^*$

Proof of Predicate Clausal Form Theorem

The system **QRS*** is a version of the predicate system **QRS** with inference rules restricted to Propositional Rules

At this point we use the system **QRS*** to define in it a decomposition tree $\mathbf{T}_{A^*}$ for any **open** formula A^*

We form **clauses** out of its leaves and we put

$$\mathbf{C}_{A^*} = \{\mathcal{C}_L : \ L \text{ is a leaf of } \mathbf{T}_{A^*}\}$$

This is the **clausal form** of the formula A of $\mathcal{L}$

To complete the proof we develop in the next section all needed **notions** and **results**

Prenex Normal Forms and Skolemization

Some Basic Notions

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

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

denote the result of replacing respectively all occurrences of the free variables x, $x_1, x_2, ..., x_n$, by the terms $t, t_1, t_2, ..., t_n$

We assume that $t, t_1, t_2, ..., t_n$ are **free for** x, $x_1, x_2, ..., x_n$, respectively, **in** A

The assumption that $t \in \mathbf{T}$ is **free for** x **in** $A(x)$ while substituting t for x, is **important** because otherwise we would distort the meaning of $A(t)$

Examples

Example 1

Let $t = y$ and $A(x)$ be

$$\exists y(x \neq y)$$

Obviously t is **not free** for y **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

$$\exists y(x \neq y)$$

Example 2

Let $A(x)$ be a formula

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

and let $t = f(x, z)$

We **substitute** t on a place of x in $A(x)$ and we obtain a formula $A(t)$ of the form

$$(\forall y P(f(x, z), y) \cap Q(f(x, z), z))$$

None of the occurrences of the variables x, z of t is **bound** in $A(t)$, hence we say that $t = f(x, z)$ is **free** for x in

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

Examples

Example 3

Let $A(x)$ be a formula

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

The term $t = f(y, z)$ is **not free** for x in $A(x)$ because **substituting** $t = f(y, z)$ on a place of x in $A(x)$ we obtain now a formula $A(t)$ of the form

$$(\forall y P(fy, z), y) \cap Q(f(y, z), z))$$

which contain a **bound** occurrence of the variable y of t in sub-formula $(\forall y P(f(y, z), y))$

The other occurrence of y in sub-formula $(Q(f(y, z), z))$ is **free**, but it is **not sufficient**, as for term to be **free for** x, **all occurrences** of its variables has to be free in $A(t)$

Similar Formulas

Informally, we say that formulas $A(x)$ and $A(y)$ are **similar** if and only if $A(x)$ and $A(y)$ are the **same** except that $A(x)$ has **free** occurrences of x in **exactly** those places where $A(y)$ has **free** occurrence of of y

We define it formally as follows

Definition

Let x and y be two different variables. We say that the formulas $A(x)$ and $A(y) = A(x/y)$ are **similar** and denote it by

$$A(x) \sim A(y)$$

if and only if y **is free** for x in $A(x)$ and $A(x)$ **has no** free occurrences of y

Similar Formulas Examples

Example 1

The formulas

$$A(x): \ \exists z(P(x, z) \Rightarrow Q(x, y))$$

and

$$A(y): \ \exists z(P(y, z) \Rightarrow Q(y, y))$$

are **not similar**; y is **free for** x in $A(x)$ as **no occurrence** of y becomes a **bound** occurrence in the formula $A(y)$ but the formula $A(x)$ has a **free occurrence** of y

Similar Formulas Examples

Example 2

The formulas

$$A(x):\ \exists z(P(x,z) \Rightarrow Q(x,y))$$

and

$$A(w):\ \exists z(P(w,z) \Rightarrow Q(w,y))$$

are similar; w is **free** for x in $A(x)$ as **no occurrence** of w becomes a **bound** occurrence in the formula $A(w)$ and the formula $A(x)$ **has no free** occurrence of w

Renaming the Variables

Directly from the definition we get the following

Fact (Renaming the Variables)

For any formula $A(x) \in \mathcal{F}$,

if $A(x)$ and $A(y) = A(x/y)$ are similar, i.e.

$$A(x) \sim A(y)$$

then the following logical equivalences hold

$$\forall x A(x) \equiv \forall y A(y)$$

and

$$\exists x A(x) \equiv \exists y A(y)$$

Example

Example 3

We proved in **Example 2** that

$$\exists z(P(x,z) \Rightarrow Q(x,y)) \sim \exists z(P(w,z) \Rightarrow Q(w,y))$$

Hence by the **Fact** we get that

$$\forall x\exists z(P(x,z) \Rightarrow Q(x,y)) \; \equiv \; \forall w\exists z(P(w,z) \Rightarrow Q(w,y))$$

and

$$\exists x\exists z(P(x,z) \Rightarrow Q(x,y)) \; \equiv \; \exists w\exists z(P(w,z) \Rightarrow Q(w,y))$$

Replacement Theorem

We prove, by the **induction** on the number of connectives and quantifiers in a formula A the following

Replacement Theorem

For any formulas $A, B \in \mathcal{F}$,

if B is a **sub-formula** of A, and A^* is the result of **replacing** zero or more occurrences of B in A by a formula C, and $B \equiv C$, then $A \equiv A^*$

Change of Bound VariablesTheorem

Theorem (Change of Bound Variables)

For any formula $A(x), A(y), B \in \mathcal{F}$,

if the formulas $A(x)$ and $A(x/y)$ are **similar**, i.e.

$$A(x) \sim A(y$$

and the formula

$$\forall x A(x) \quad \text{or} \quad \exists x A(x)$$

is a **sub-formula** of B, and the formula B^* is the result of **replacing** zero or more occurrences of $A(x)$ in B by a formula $\forall y A(y)$ or by a formula $\exists y A(y)$, then

$$B \equiv B^*$$

Naming Variables Apart

Definition

We say that a formula B has its variables named apart if no two quantifiers in B **bind** the same variable and no bound variable is also free

We now use the Change of Bound Variables **Theorem** to prove its more general version

Naming Variables Apart

Theorem (Naming Variables Apart)

Every formula $A \in \mathcal{F}$ is logically **equivalent** to one in which all variables are named apart

We use the above theorems plus the **equational laws** for quantifiers to prove, as a next step a so called a **Prenex Form Theorem**

In order to do so we first we define an important notion of prenex normal form of a formula

Closure of a Formula

Here is an important notion we need for future definition

Definition(Closure of a Formula)

By a closure of a formula A we mean a **closed** formula A' obtained from A prefixing in universal quantifiers all those variables that a free in A ; i.e.

if $A(x_1, \ldots.., x_n)$ then $A' \equiv A$ is

$$\forall x_1 \forall x_2 .... \forall x_n A(x_1, x_2, \ldots.., x_n)$$

Example

Let A be a formula $(P(x, y) \Rightarrow \neg \exists z \, R(x, y, z))$. its **closure** $A' \equiv A$ is $\forall x \forall y (P(x, y) \Rightarrow \neg \exists z \, R(x, y, z))$

Prenex Normal Form

PNF Definition

Any formula of the form

$$Q_1 x_1 Q_2 x_2 .... Q_n x_n \ B$$

where each Q_i is a **universal** or **existential quantifier**, i.e. the following holds

for all $1 \leq i \leq n$,

$$Q_i \in \{\exists, \forall\} \quad \text{and} \quad x_i \neq x_j \quad \text{for} \quad i \neq j$$

and the formula B contains **no quantifiers**, is said to be in **Prenex Normal Form (PNF)**

We include the case $n = 0$ when there are no quantifiers at all

Prenex Normal Form Theorem

We assume that the formula A in **PNF** is always **closed**

If it is not closed we form its closure instead

PNF Theorem

There is an effective procedure for transforming any formula $A \in \mathcal{F}$ into a formula B in the prenex normal form **PNF** such that

$$A \equiv B$$

Proof

The procedure uses the Replacement and Naming Variables Apart **Theorems** and and the following Equational Laws of Quantifiers proved in chapter 2

Equational Laws of Quantifiers

For any $A(x), B \in \mathcal{F}$, where B **does not** contain any free occurrence of x the following holds

$$\forall x(A(x) \cup B) \equiv (\forall x A(x) \cup B)$$

$$\forall x(A(x) \cap B) \equiv (\forall x A(x) \cap B)$$

$$\exists x(A(x) \cup B) \equiv (\exists x A(x) \cup B)$$

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

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

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

$$\forall x(B \Rightarrow A(x)) \equiv (B \Rightarrow \forall x A(x))$$

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

PNF Procedure

The general **PNF procedure** is defined by induction on the number k of occurrences of connectives and quantifiers in A

We show here how it works in some particular cases

Exercise Find a prenex normal form **PNF** of a formula

$$A : \quad (\forall x(P(x) \Rightarrow \exists x Q(x))$$

Solution We find **PNF** as follows

Step 1: Naming Variables Apart

We make all **bound variables** in A different, i.e. we transform A into an equivalent formula A'

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

Step 2: Pull Out Quantifiers

We apply the equational law

$(C \Rightarrow \exists y Q(y)) \equiv \exists y\,(C \Rightarrow Q(y))$ to the sub-formula

$$B: \quad (P(x) \Rightarrow \exists y Q(y))$$

of A' for $C = P(x)$, as P(x) **does not** contain the variable y

We get its equivalent formula

$$B^*: \quad \exists y (P(x) \Rightarrow Q(y))$$

We substitute B^* on place of B in A' and get the formula

$$A" \quad \forall x \exists y (P(x) \Rightarrow Q(y))$$

By the Replacement **Theorem** $A'' \equiv A' \equiv A$

The formula A'' is a required prenex normal form **PNF** for A

PNF Procedure

Example

Let's now find **PNF** for the formula A:

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

Step 1: Rename Variables Apart

Take a sub- formula $B(x,y)$: $\forall y \exists x\, R(x,y)$ of A

Rename variables in $B(x,y)$, i.e. get
$B(x/z, y/w) : \forall w \exists z\, R(z,w)$

Replace $B(x,y)$ by $B(x/z, y/w)$ in A and get

$$(\exists x \forall y\, R(x,y) \Rightarrow \forall w \exists z\, R(z,w))$$

PNF Procedure

Step 2: Pull out quantifiers

We use corresponding equational laws for quantifiers to pull out **first** (one by one) quantifiers $\exists x \forall y$ and **then** pulling out one by one the quantifiers $\forall w \exists z$

We get the following **PNF** for A

$$\forall x \exists y \forall w \exists z\, (R(x,y) \Rightarrow R(z,w))$$

Observe we can also perform **Step 2** by pulling out **first** (one by one) the quantifiers $\forall w \exists z$ and **then** pulling out one by one the quantifiers $\exists x \forall y$.

We hence can obtain **another PNF** for A

$$\forall w \exists z \forall x \exists y\, (R(x,y) \Rightarrow R(z,w))$$

Skolem Procedure of Elimination of Quantifiers

Skolemization

We will show now how any formula A already in its prenex normal form **PNF** can be transformed into a certain **open formula** A^*, such that

$$A \equiv A^*$$

The **open formula** A^* belongs to a **richer language** then the initial language $\mathcal{L}$ to which the formula A belongs

Skolemization

This transformation process **adds** new constants to the original language $\mathcal{L}$

They are called **Skolem constants**

The process also **adds** to $\mathcal{L}$ new functions symbols called **Skolem functions**

The whole transformation process is called **Skolemization** of the initial language $\mathcal{L}$

Such build extension of the initial language $\mathcal{L}$ is called the **Skolem extension** of and $\mathcal{L}$ and denoted

$$\mathcal{L}^*$$

Skolem Elimination of Quantifiers

Skolem Procedure of Elimination of Quantifiers

Given a formula A of the language

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

We assume that A is already in its prenex normal form **PNF**

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n B(x_1, x_2, \dots x_n)$$

where each Q_i is a **universal** or **existential** quantifier, i.e. for all $1 \leq i \leq n$, $Q_i \in \{\exists, \forall\}$, $x_i \neq x_j$ for $i \neq j$, and the formula $B(x_1, x_2, \dots x_n)$ contains **no quantifiers**

Skolem Elimination of Quantifiers

We describe now a procedure of **elimination** of all quantifiers from a **PNF** formula A

The procedure transforms **PNF** formula A into a logically equivalent **open formula** A^*

We also assume that the **PNF** formula A is **closed**

If it is not closed we form its closure instead

Closure of a Formula

For any formula A, its **closure** is a formula A' obtained from A by **prefixing** in universal quantifiers all those variables that are **free** in A

Example

Let A be a formula

$$(P(x, y) \Rightarrow \neg \exists z\, R(x, y, z))$$

its **closure** i.e. a formula $A' \equiv A$ is

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

Elimination of Quantifiers

Given a formula A in its closed **PNF** form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n B(x_1, x_2, \ldots x_n)$$

We considerer 3 cases

Case 1

All quantifiers Q_i for $1 \leq i \leq n$ are **universal**, i.e. the formula A is

$$A : \quad \forall x_1 \forall x_2 \ldots \forall x_n B(x_1, x_2, \ldots, x_n)$$

We **replace** the formula A by the **open formula** A^*

$$A^* : \quad B(x_1, x_2, \ldots., x_n)$$

Elimination of Quantifiers

Case 2

All quantifiers Q_i for $1 \leq i \leq n$ are **existential**, i.e. formula A is

$$A : \quad \exists x_1 \exists x_2 .... \exists x_n B(x_1, x_2, \dots x_n)$$

We **replace** the formula A by the **open formula** A^*

$$A^* : \quad B(c_1, c_2, \dots.., c_n)$$

where $c_1, c_2, \dots.., c_n$ and **new** individual constants **added** to our original language $\mathcal{L}$

We call such individual **constants** added to the original language Skolem constants

Elimination of Quantifiers

Case 3

The quantifiers in A are **mixed**

We **eliminate** the mixed quantifiers one by one and step by step depending on first, and then the consecutive quantifiers in the closed **PNF** formula A

$$A: \quad Q_1x_1Q_2x_2\ldots Q_nx_nB(x_1,x_2,\ldots x_n)$$

We have two possibilities for the **first** quantifier Q_1x_1

P1 Q_1x_1 is **universal**

P2 Q_1x_1 is **existential**

Elimination of Quantifiers; Step 1

Step 1 Elimination of Q_1

We consider the two cases for the **first** quantifier

Case **P1**

First quantifier Q_1 is **universal**

This means that A is

$$A: \quad \forall x_1 Q_2 x_2 \dots Q_n x_n B(x_1, x_2, \dots x_n)$$

We **replace** A by the following formula A_1

$$A_1: \quad Q_2 x_2 Q_3 x_3 \dots Q_n x_n B(x_1, x_2, x_3, \dots x_n)$$

We have **eliminated** the quantifier Q_1 in this case

Elimination of Quantifiers; Step 1

Case **P2**

First quantifier Q_1 is **existential**. This means that A is

$$A: \quad \exists x_1 Q_2 x_2 \ldots Q_n x_n B(x_1, x_2, \ldots x_n)$$

We **replace** A by a following formula A_1

$$A_1 \quad Q_2 x_2 \ldots Q_n x_n B(b_1, x_2, \ldots x_n)$$

where b_1 is a new constant symbol **added** to our original language $\mathcal{L}$

We call such constant symbol **added** to the language a Skolem constant

We have **eliminated** the quantifier Q_1 in both cases and this ends the **Step 1**

Elimination of Quantifiers; Step 2

Step 2 Elimination of Q_2

Consider now the **PNF** formula A_1 from **Step1** - case **P1**

$$A_1 \quad Q_2 x_2 \ldots Q_n x_n B(x_1, x_2, \ldots x_n)$$

Remark that the formula A_1 might **not be** closed

We have again two cases for elimination of the quantifier Q_2

P1 Q_2 is **universal**

P2 Q_2 is **existential**

Elimination of Quantifiers; Step 2

Case **P1**

First quantifier in A_1 is **universal**

The formula A_1 is

$$A_1 \quad \forall x_2 Q_3 x_3 \ldots Q_n x_n B(x_1, x_2, x_3, \ldots x_n)$$

We **replace** A_1 by the following A_2

$$A_2 \quad Q_3 x_3 \ldots Q_n x_n B(x_1, x_2, x_3, \ldots x_n)$$

We have **eliminated** the quantifier Q_2 in this case

Elimination of Quantifiers; Step 2

Case **P2**

First quantifier in A_1 is **existential**

The formula A_1 is

$$A_1 \quad \exists x_2 Q_3 x_3 \dots Q_n x_n B(x_1, x_2, x_3, \dots x_n)$$

Observe that now the variable x_1 is a **free** variable in

$$B(x_1, x_2, x_3, \dots x_n)$$

and hence x_1 is a **free** variable in in the formula A_1

Elimination of Quantifiers; Step 2

The variable x_1 is **free** in A_1

$$A_1 \quad \exists x_2 Q_3 x_3 \ldots Q_n x_n B(x_1, x_2, x_3, \ldots x_n)$$

We **replace** A_1 by the following A_2

$$A_2 \quad Q_3 x_3 \ldots Q_n x_n B(x_1, f(x_1), x_3, \ldots x_n)$$

where f is a new **one** argument functional symbol **added** to our original language $\mathcal{L}$

We call such functional symbols **added** to the original language Skolem functional symbols

We have **eliminated** the quantifier Q_2 in this case

Elimination of Quantifiers; Step 2

Consider now the **PNF** formula A_1 from **Step1** - case **P2**

$$A_1 \quad Q_2 x_2 Q_3 x_3 \ldots Q_n x_n B(b_1, x_2, \ldots x_n)$$

Again we have two cases for the quantifier Q_2

Case **P1**

First quantifier Q_2 in A_1 is **universal**

The formula A_1 is

$$A_1 \quad \forall x_2 Q_3 x_3 \ldots Q_n x_n B(b_1, x_2, x_3, \ldots x_n)$$

We **replace** A_1 by the following A_2

$$A_2 \quad Q_3 x_3 \ldots Q_n x_n B(b_1, x_2, x_3, \ldots x_n)$$

We have **eliminated** the quantifier Q_2 in this case

Elimination of Quantifiers; Step 2

Case **P2**

First quantifier in A_1 is **existential**

The formula A_1 is

$$A_1 \quad \exists x_2 Q_3 x_3 \ldots Q_n x_n B(b_1, x_2, x_3, \ldots x_n)$$

We **replace** A_1 by the following A_2

$$A_2 \quad Q_3 x_3 \ldots Q_n x_n B(b_1, b_2, x_3, \ldots x_n)$$

where $b_2 \neq b_1$ is a **new** Skolem constant **added** to the original language $\mathcal{L}$

We have **eliminated** the quantifier Q_2 in this case

We have covered all cases and this ends the **Step 2**

Elimination of Quantifiers; Step 3

Step 3 Elimination of Q_3

Let's now consider, as an **example** a formula A_2 from **Step 2** - case **P1** i.e. the formula

$$Q_3 x_3 \ldots Q_n x_n B(x_1, x_2, x_3, \ldots x_n)$$

We have two cases but we describe only the following

P2 First quantifier in A_2 is **existential**

The formula A_2 is

$$A_2 \quad \exists x_2 Q_4 x_4 \ldots Q_n x_n B(x_1, x_2, x_3, x_4, \ldots x_n)$$

Observe that now the variables x_1, x_2 are **free** variables in

$$B(x_1, x_2, x_3, \ldots x_n)$$

and hence in A_2

Elimination of Quantifiers; Step 2

The the variables x_1, x_2 are **free** in A_2

$$A_2 \quad \exists x_2 Q_4 x_4 \ldots Q_n x_n B(x_1, x_2, x_3, x_4, \ldots x_n)$$

We replace A_2 by the following A_3

$$A_3 \quad Q_4 x_3 \ldots Q_n x_n B(x_1, x_2, g(x_1, x_2), x_4 \ldots x_n)$$

where g is a **new** two argument functional symbol **added** to the original language $\mathcal{L}$

We have **eliminated** the quantifier Q_3 in this case

Elimination of Quantifiers

At each **Step i** for $1 \leq i \leq n$ we build a **binary tree** of cases

P1 Q_i is universal or **P2** Q_i is existential

The result in each case is a formula A_i with one less quantifier

The **elimination** of the proper quantifier **adds** new Skolem constant or Skolem function symbol to the original language $\mathcal{L}$

Elimination of Quantifiers

The **elimination of quantifiers** process builds a sequence of formulas

$$A,\ A_1,\ A_2,\ \ldots,\ A_n = A^*$$

where the formula A belongs to our original language

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

and the **open** formula A^* belongs to its Skolem extension defined as follows

Skolem Extension

Definition

The **Skolem extension** $\mathcal{L}^*$ of a language

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

is the language

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

where the sets $\mathbf{SF}$ and $\mathbf{SC}$ are respectively the sets of Skolem functions and Skolem constants

They are obtained by the **quantifiers elimination procedure**

Elimination of Quantifiers Result

Given a formula A in its closed **PNF** form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n B(x_1, x_2, \dots x_n)$$

Observe that the **elimination** of an universal quantifier Q_i introduces a **free** variable x_i in the formula

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n B(x_1, x_2, \dots x_n)$$

Elimination of Quantifiers Result

The **elimination** of an existential quantifier Q_i that follows universal quantifiers introduces a **new** functional symbol with number of arguments equal the number of universal quantifiers preceding it

The **elimination** of an existential quantifier Q_i that **does not** follows any universal quantifiers introduces a **new** constant symbol

The resulting **open** formula A^* is logically equivalent to the **PNF** formula A

Skolemization

Definition

Given a formula A of $\mathcal{L}$

A formula

$$A^*$$

of the Skolem extension language $\mathcal{L}^*$ obtained from A by the **elimination of quantifiers** process is called a Skolem form of the formula A

The elimination of quantifiers process obtaining it is called **Skolemization**

Example

Example 1

Let A be a closed **PNF** formula

$$A: \quad \forall y_1 \exists y_2 \forall y_3 \exists y_4\ B(y_1, y_2, y_3, y_4, y_4)$$

We **eliminate** $\forall y_1$ and get a formula A_1

$$A_1: \quad \exists y_2 \forall y_3 \exists y_4\ B(y_1, y_2, y_3, y_4)$$

We **eliminate** $\exists y_2$ by **replacing** the variable y_2 by $h(y_1)$

The symbol h is a **new** one argument functional symbol **added** to the language $\mathcal{L}$

We get a formula A_2

$$A_2: \quad \forall y_3 \exists y_4\ B(y_1, h(y_1), y_3, y_4)$$

Example 1

Given the formula A_2

$$A_2 : \quad \forall y_3 \exists y_4\, B(y_1, h(y_1), y_3, y_4)$$

We **eliminate** $\forall y_3$ and get a formula A_3

$$A_3 : \quad \exists y_4\, B(y_1, h(y_1), y_3, y_4)$$

We **eliminate** $\exists y_4$ by replacing y_4 by $f(y_1, y_3)$, where f is a **new** two argument functional symbol **added** to $\mathcal{L}$

We get a formula A_4 that is our resulting **open** formula A^*

$$A^* : \quad B(y_1, h(y_1), y_3, f(y_1, y_3))$$

Example 2

Let A be a closed **PNF** formula

$$A: \quad \exists y_1 \forall y_2 \forall y_3 \exists y_4 \exists y_5 \forall y_6 \; B(y_1, y_2, y_3, y_4, y_4, y_5, y_6)$$

We **eliminate** $\exists y_1$ and get a formula A_1

$$A_1: \quad \forall y_2 \forall y_3 \exists y_4 \exists y_5 \forall y_6 \; B(b_1, y_2, y_3, y_4, y_4, y_5, y_6)$$

where b_1 is a **new** constant **added** to the language $\mathcal{L}$

We **eliminate** $\forall y_2, \forall y_3$ and get formulas A_2, A_3

$$A_2: \quad \forall y_3 \exists y_4 \exists y_5 \forall y_6 \; B(b_1, y_2, y_3, y_4, y_4, y_5, y_6)$$

$$A_3: \quad \exists y_4 \exists y_5 \forall y_6 \; B(b_1, y_2, y_3, y_4, y_4, y_5, y_6)$$

Example 2

We **eliminate** $\exists y_4$ and get a formula A_4

$$A_4 : \quad \exists y_5 \forall y_6 \, B(b_1, y_2, y_3, g(y_2, y_3), y_5, y_6)$$

where g is a **new** two argument functional symbol **added** to the original language $\mathcal{L}$

We **eliminate** $\exists y_5$ and get a formula A_5

$$A_5 : \quad \forall y_6 \, B(b_1, y_2, y_3, g(y_2, y_3), h(y_2, y_3), y_6)$$

where h is a **new** two argument functional symbol **added** to the language $\mathcal{L}$

We **eliminate** $\forall y_6$ and get a formula A_6 that is the resulting **open** formula A^*

$$A^* : \quad B(b_1, y_2, y_3, g(y_2, y_3), h(y_2, y_3), y_6)$$

Skolem Theorem

The **correctness** of the Skolemization process is established by the **Skolem Theorem**

It states informally that the formula A^* obtained from a formula A via the Skolemization process is **satisfiable** if and only if the original formula A is **satisfiable**

We define this notion formally as follows

Skolem Theorem

Definition Equisatisfiable formulas

Given any formulas A of $\mathcal{L}$ and B of the **Skolem extension** $\mathcal{L}^*$ of $\mathcal{L}$

We say that A and B are **equisatisfiable** if and only if the following conditions are satisfied

1. Any structure $\mathcal{M}$ of $\mathcal{L}$ can be **extended** to a structure $\mathcal{M}^*$ of $\mathcal{L}^*$ and following implication holds

$$\text{If } \mathcal{M} \models A, \quad \text{then } \mathcal{M}^* \models B$$

2. Any structure $\mathcal{M}^*$ of $\mathcal{L}^*$ can be **restricted** to a structure $\mathcal{M}$ of $\mathcal{L}$ and following implication holds

$$\text{If } \mathcal{M}^* \models B, \quad \text{then } \mathcal{M} \models A$$

Skolem Theorem

Skolem Theorem

Let $\mathcal{L}^*$ be the **Skolem extension** of a language $\mathcal{L}$
Any formula A of $\mathcal{L}$ and its **Skolem form** A^* of $\mathcal{L}^*$
are **equisatisfiable**

Clausal Form of Formulas

Poof System **QRS***

Let $\mathcal{L}^*$ be the **Skolem extension** of $\mathcal{L}$

By definition, the language $\mathcal{L}^*$ does not contain quantifiers and all its formulas and **open**

We define a proof system **QRS*** as an **open formulas** version of the proof system **QRS** based on the language $\mathcal{L}$

We denote the set of **formulas** of $\mathcal{L}^*$ by $O\mathcal{F}$ to stress the fact that all its formulas are **open**

Let

$$A\mathcal{F} \subseteq O\mathcal{F}$$

be the set of all **atomic** formulas of $\mathcal{L}^*$ and the set

$$LT = \{A : \; A \in A\mathcal{F}\} \cup \{\neg A : \; A \in A\mathcal{F}\}$$

the set of all **literals** of $\mathcal{L}^*$

Poof System **QRS***

We denote by

$$\Gamma', \quad \Delta', \quad \Sigma' \ldots$$

finite sequences (empty included) formed out of **literals**, i.e of the elements of LT^*

We will denote by

$$\Gamma, \quad \Delta, \quad \Sigma \ldots$$

finite sequences (empty included) formed out of **formulas**, i.e of the elements of $\mathcal{OF}^*$

Poof System $\mathbf{QRS}^*$

We define the proof system $\mathbf{QRS}^*$ formally as follows

$$\mathbf{QRS}^* = (\mathcal{L}^*, \ \mathcal{E}, \ LA, \ \mathcal{R})$$

where $\mathcal{E} = \{\Gamma : \ \Gamma \in \mathcal{OF}^*\}$

The set LA of logical axioms contains any sequence $\Gamma' \in LT^*$

which contains an atomic formula and its negation

$\mathcal{R}$ is the set inference rules

$$(\cup), \ (\neg\cup), \ (\cap), \ (\neg\cap), \ (\Rightarrow), \ (\neg \Rightarrow), \ (\neg\neg)$$

defined as follows

Poof System **QRS***

Disjunction rules

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

Conjunction rules

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

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

Poof System **QRS***

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 $\Gamma' \in LT^*,\ \ \Delta \in O\mathcal{F}^*,\ \ A, B \in O\mathcal{F}$

$\mathbf{QRS}^*$ Semantics

Definition

For any sequence Γ of formulas of $\mathcal{L}^*$, any structure $\mathcal{M} = [M, I]$ for $\mathcal{L}^*$,

$$\mathcal{M} \models \Gamma \quad \text{if and only if} \quad \mathcal{M} \models \delta_\Gamma$$

where δ_Γ denotes a **disjunction** of all formulas in Γ

The semantics for **clauses** is basically the same as for the sequences. We define it as follows

Clauses Semantics

Definition

For any finite set of clauses **C** of $\mathcal{L}^*$, any structure $\mathcal{M} = [M, I]$ for $\mathcal{L}^*$, and any clause $C \in \mathbf{C}$,

1. $\mathcal{M} \models C$ if and only if $\mathcal{M} \models \delta_C$

2. $\mathcal{M} \models \mathbf{C}$ if and only if $\mathcal{M} \models \delta_C$ for all $C \in \mathbf{C}$

3. $(A \equiv \mathbf{C})$ if and only if $A \equiv \sigma_{\mathbf{C}}$

where δ_C denotes a disjunction of all literals in C and $\sigma_{\mathbf{C}}$ is a conjunction of all formulas δ_C for all clauses $C \in \mathbf{C}$

Obviously, the rules of inference of **QRS*** are strongly sound and the following holds

Strong Soundness Theorem

The proof system **QRS*** is strongly sound

Formula to Clauses Transformation

We use the **QRS*** system to define an effective procedure that **transforms** any formula A of $\mathcal{L}^*$ into set of clauses and prove correctness of this transformation

We treat the rules of inference of **QRS*** as decomposition rules and use them to **generate** needed set $\mathbf{C}_A$ of **clauses** corresponding to a given formula A

Decomposable, Indecomposable

Definition

A formula that is not a literal, i.e. any formula $A \in \mathcal{OF} - \mathbf{L}$ is called a **decomposable**

Otherwise A is called **indecomposable**

Definition

A sequence Γ that contains a decomposable formula is called a **decomposable** sequence

Definition

A sequence Γ' built only out of literals, i.e. $\Gamma' \in \mathbf{L}^*$ is called an **indecomposable** sequence

Decomposition Tree $\mathbf{T}_A$

Definition

Given a formula $A \in \mathcal{OF}$

We build the **decomposition tree** $\mathbf{T}_A$ of A as follows

Step 1.

The formula A is the **root** of $\mathbf{T}_A$

For any node Δ of the tree $\mathbf{T}_A$ we follow the steps bellow

Step 2.

If Δ is **indecomposable**, then Δ becomes a **leaf** of the tree

Decomposition Tree $\mathbf{T}_A$

Step 3.

If Δ is **decomposable**, then we traverse Δ from left to right to identify the first **decomposable formula** B

In case of a one premiss rule we put is **premise** as a **leaf**

In case of a two premisses rule we put its left and right premisses as the **left** and **right leaves**, respectively

Step 4.

We repeat steps **2.** and **3.** until we obtain only **leaves**

Formula-Clauses Equivalency Theorem

For any formula A of $\mathcal{L}^*$, there is an effective procedure of generating a set of **clauses** $\mathbf{C}_A$ of $\mathcal{L}^*$ such that

$$A \equiv \mathbf{C}_A$$

Proof

Given $A \in \mathcal{OF}$. Here is the two steps procedure

S1. We construct (finite and unique) decomposition tree $\mathbf{T}_A$

S2. We form **clauses** out of the leaves of the tree $\mathbf{T}_A$, i.e. for every **leaf** L we create a clause C_L determined by L and we put

$$\mathbf{C}_A = \{C_L : \ L \text{ is a leaf of } \mathbf{T}_A\}$$

Directly from the **QRS*** **Strong Soundness Theorem** and the semantics for clauses definition we get that

$$A \equiv \mathbf{C}_A$$

Exercise

Exercise

Find the set $\mathbf{C}_A$ of clauses for the following formula A

$$(((P(b, f(x)) \Rightarrow Q(x)) \cup \neg R(z)) \cup (P(b, f(x)) \cap R(z))))$$

Solution

Step **S1.** We construct the decomposition tree $\mathbf{T}_A$ for A

Step **S2.** We form **clauses** out of the leaves of the tree $\mathbf{T}_A$

We put

$$\mathbf{C}_A = \{\mathcal{C}_L : \ L \text{ is a leaf of } \mathbf{T}_A\}$$

Exercise

Step **S1.** The decomposition tree is

$$\mathbf{T}_A$$

$$(((P(b,f(x)) \Rightarrow Q(x)) \cup \neg R(z)) \cup (P(b,f(x)) \cap R(z)))$$

$$\mid (\cup)$$

$$(((P(b,f(x)) \Rightarrow Q(x)) \cup \neg R(z)), (P(b,f(x)) \cap R(z))$$

$$\mid (\cup)$$

$$(P(b,f(x)) \Rightarrow Q(x)), \neg R(z), (P(b,f(x)) \cap R(z))$$

$$\mid (\Rightarrow)$$

$$\neg P(b,f(x)), Q(x), \neg R(z), (P(b,f(x)) \cap R(z))$$

$$\bigwedge (\cap)$$

$$\neg P(b,f(x)), Q(x), \neg R(z), P(b,f(x)) \qquad \neg P(b,f(x)), Q(x), \neg R(z), R(z)$$

$$L_1 \qquad\qquad L_2$$

Exercise

Step **S2.** The leaves of $\mathbf{T}_A$ are

$$L_1 = \neg P(b, f(x)),\ Q(x),\ \neg R(z),\ P(b, f(x))$$

$$L_2 = \neg P(b, f(x)),\ Q(x),\ \neg R(z),\ R(z)$$

The corresponding clauses are

$$C_1 = \{\neg P(b, f(x)), Q(x), \neg R(z), P(b, f(x))\}$$

$$C_2 = \{\neg P(b, f(x)), Q(x), \neg R(z), R(z)\}$$

The set of clauses is

$$\mathbf{C}_A = \{\ C_1,\ \ C_2\ \}$$

Clausal Form of Formulas of $\mathcal{L}$

Definition

Given a formula A of the original language $\mathcal{L}$

Let A^* of $\mathcal{L}^*$ be the **Skolem form** A obtained by the Skolemization process

A a set $\mathbf{C}_{A^*}$ of clauses of $\mathcal{L}^*$ such that

$$A^* \equiv \mathbf{C}_{A^*}$$

is called a **clausal form** of the formula A of the language $\mathcal{L}$

Exercise

Exercise Find the clausal form of a formula A

$$A: \quad (\exists x \forall y\, (R(x,y) \cup \neg P(x)) \Rightarrow \forall y \exists x\, \neg R(x,y))$$

Solution We first find the Skolem form A^* of A

Step 1: We **rename variables** apart in A and get a formula A'

$$A': \quad (\exists x \forall y\, (R(x,y) \cup \neg P(x)) \Rightarrow \forall z \exists w\, \neg R(z,w))$$

Step 2: We use **Equational Laws** of Quantifiers to pull out quantifiers $\exists x$ and $\forall y$ and get a formula A''

$$A'': \quad \forall x \exists y\, ((R(x,y) \cup \neg P(x)) \Rightarrow \forall z \exists w\, \neg R(z,w))$$

Exercise

Step 3 : We use **Equational Laws** of Quantifiers to pull out the quantifiers $\exists z$ and $\forall w$ from the sub formula

$$((R(x,y) \cup \neg P(x)) \Rightarrow \forall z \exists w \, \neg R(z,w))$$

and get a formula A'''

$$A''' : \quad \forall x \exists y \forall z \exists w \, ((R(x,y) \cup \neg P(x)) \Rightarrow \neg R(z,w))$$

This is the Prenex Normal Form **PNF** of A

Exercise

Step 4: We perform the Skolemization Procedure

Observe that the formula

$$\forall x \exists y \forall z \exists w\, ((R(x,y) \cup \neg P(x)) \Rightarrow \neg R(z,w))$$

is of the form of the formulas of the **Examples 1, 2**

We follow them and eliminate $\forall x$ and get a formula A_1

$$A_1 : \quad \exists y \forall z \exists w\, ((R(x,y) \cup \neg P(x)) \Rightarrow \neg R(z,w))$$

We eliminate $\exists y$ by replacing y by $h(x)$ where h is a **new** one argument functional symbol **added** to the language $\mathcal{L}$

We get a formula A_2

$$A_2 : \quad \forall z \exists w\, ((R(x,h(x)) \cup \neg P(x)) \Rightarrow \neg R(z,w))$$

Exercise

We eliminate $\forall z$ and get a formula A_3

$$A_3 : \quad \exists w\, ((R(x,h(x)) \cup \neg P(x)) \Rightarrow \neg R(z,w))$$

We eliminate $\exists w$ by replacing w by $f(x,z)$, where f is a **new** two argument functional symbol **added** to the original language $\mathcal{L}$

We get a formula A_4 that is the resulting **open** formula A^* of $\mathcal{L}^*$

$$A^* : \quad ((R(x,h(x)) \cup \neg P(x)) \Rightarrow \neg R(z,(x,z)))$$

Exercise

Step 5: We build the decomposition tree of A^* as follows

$\mathbf{T}_{A^*}$

$$((R(x,h(x)) \cup \neg P(x)) \Rightarrow \neg R(z,f(x,z)))$$

$$\mid (\Rightarrow)$$

$$\neg(R(x,h(x)) \cup \neg P(x)),\ \neg R(z,f(x,z))$$

$$\bigwedge (\neg\cup)$$

$$\neg R(x,h(x)), \neg R(z,f(x,z) \qquad \qquad \neg\neg P(x),\ \neg R(z,f(x,z))$$

$$\mid (\neg\neg)$$

$$P(x),\ \neg R(z,f(x,z))$$

Exercise

Step 6: The leaves of $\mathbf{T}_{A^*}$ are

$L_1 = \neg R(x, h(x)), \neg R(z, f(x, z)$

$L_2 = P(x),\ \neg R(z, f(x, z))$

The corresponding clauses are

$C_1 = \{\neg R(x, h(x)), \neg R(z, f(x, z)\}$

$C_2 = \{P(x),\ \neg R(z, f(x, z))\}$

Step 7: The **clausal form** of the formula A

$$A: \quad (\exists x \forall y\ (R(x, y) \cup \neg P(x)) \Rightarrow \forall y \exists x\ \neg R(x, y))$$

is the **set of clauses**

$$\mathbf{C}_{A^*} = \{\ \{\neg R(x, h(x)), \neg R(z, f(x, z)\},\ \ \{P(x),\ \neg R(z, f(x, z))\}\ \}$$