UNIFICATION OF FUNCTIONAL AND LOGICAL PROGRAMMING

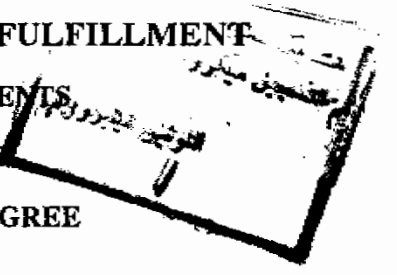
THESIS

SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS

OF THE AWARD

OF THE (M. Sc.) DEGREE



BY

AZZA ABDEL RAHMAN TAHA

SUPERVISED BY

PROF. DR. BAYOUMI IBRAHIM BAYOUMI

DR. SAMEH SAMI DAOUD

SUBMITTED TO

MATHEMATICS DEPARTMENT

FACULTY OF SCIENCE

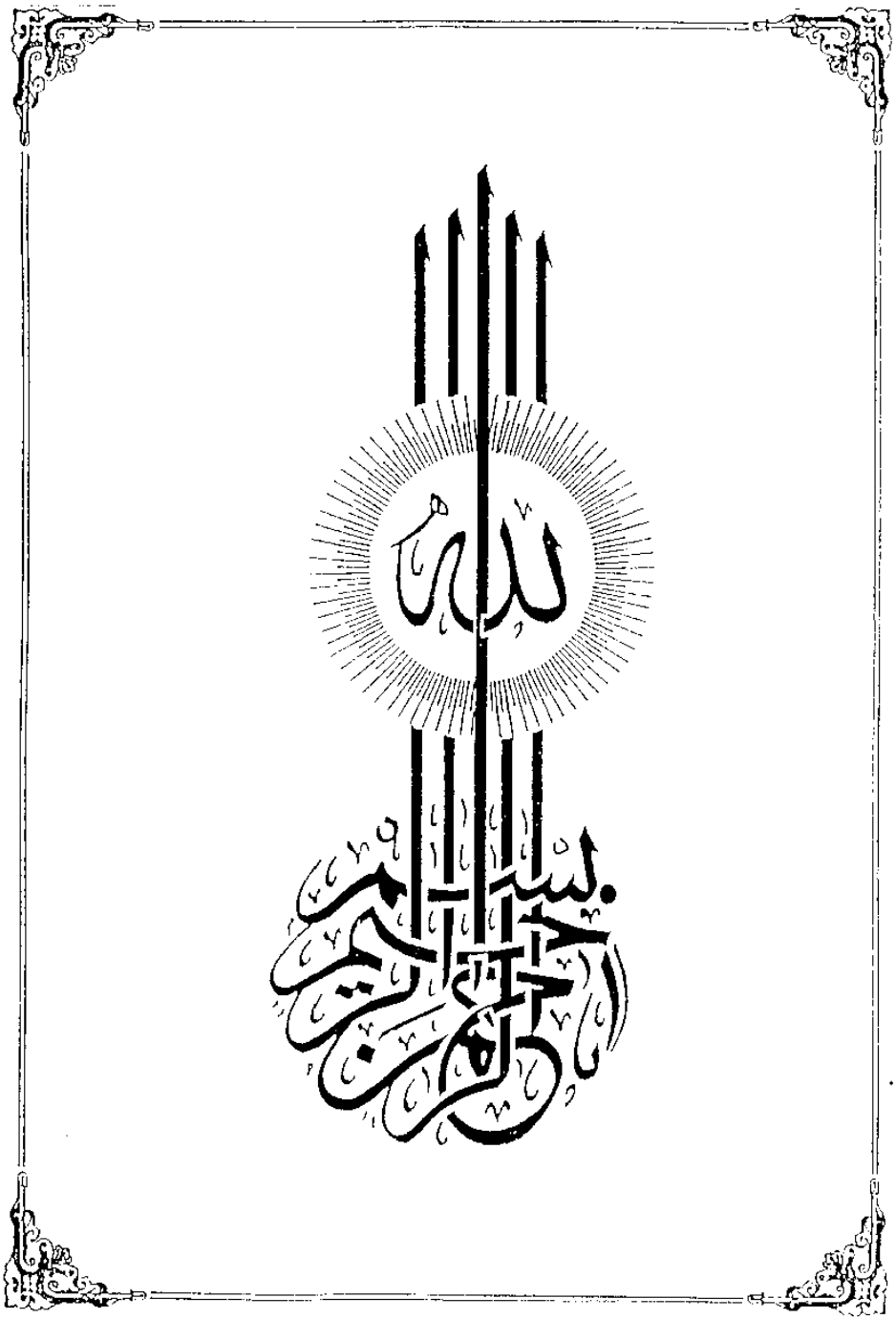
AIN SHAMS UNIVERSITY

(CAIRO)

1994

ABSTRACT

The aim of this thesis is to built an interpreter which evaluates the set of all solutions of an equational programs using paramodulation and reflection inference rules. It's input is an equational program and an equational goal and it returns whether this goal is a logical consequence of the program or not. It returns also a computed answer substitution if it exists.



ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Dr. Sameh Sami Daoud, Associate prof., Mathematics Department, Faculty of science, Ain Shams University, for his valuable guidance, supervision, sincere advice and help during the preparation of the thesis.

I would like to express my gratitude and thanks to Prof. Dr. Bayoumi Ibrahim Bayoumi, Mathematics Department, Faculty of science, Ain Shams University, for his valuable advice and offering every possible help.

I am also grateful to Dr. Mahmoud Khairat, Associate prof., Mathematics Department, Faculty of science, Ain Shams University, for his continuous help, and encouragement.

CONTENTS

SUMMARY

In the recent years many proposals were made to integrate functional and logic programming [2]. Among them equational logic programs were of special interest, since the main semantic properties of logic programs hold also for equational logic programs, such as the existence of a canonical domain of computation, the existence of a least model semantics, or the soundness and completeness for derivations of the implementation model, see [15],[16].

There are different approaches to handle equational theories such as paramodulation, narrowing [7],[23],[24], or complete sets of transformation [8], these transformation rules are an extension of the rules invented by Martelli in [22] to compute the mgu of two expressions. Another approach is to flatten a goal and program clauses and then to apply SLD-resolution [1],[6].

In this thesis, the unification of functional and logic programming is given using the equational logic programming.

The thesis is divided into 5 chapters. Chapter 1, includes the basic definitions for the first order logic and equational logic, the definition of substitution and the unification algorithm, the model and fixpoint semantics, and the SLD-resolution inference rule.

In chapter 2, a universal unification algorithm is defined as a procedure which accepts a Horn equational theory and a set of equations and generates a set of solutions for the given unification problem.

The paramodulation inference rule is given in chapter 3 to deal with the equational theory.

In chapter 4, the complete set of transformations is given to overcome the disadvantage of paramodulation, since in any subgoal many subterms are capable of applying paramodulation and all of them must be investigated to get completeness. Finally lazy resolution inference rule is used with paramodulation or with complete set of transformations to compute goals of an equational logic programs.

An implementation using ada language of the paramodulation inference rule is given in chapter 5.

All the inference rules defined in this thesis are strong complete, i.e. refutation is independent of a selection function, [12],[20].

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

This chapter will discuss the basic definitions which are needed for the theoretical foundations of logic and equational logic programing, its syntax and semantics. The syntax is concerned with well-formed formulae and the first order theory while the semantics is concerned with the meaning of the well-formed formulae and the symbols they contain. This semantics is given by the model theoretic semantics of formulae. Finally inference rule is given to show how new formulae can be derived from the old formulae. The material of this chapter can be found in [12,20,18,19].

1.1 Foundation of Logic Programming

First Order Theories and Basic Definitions

1. An *alphabet* consists of seven classes, namely, variable symbols, constant symbols, function symbols, predicate symbols, connectives, quantifiers and punctuation symbols.

The following notation for the different types of alphabets will be used :

variable symbols	$x,y,z,u,v,w,\ldots,$
constant symbols	$a,b,c,\ldots,$
function symbols	$f,g,h,\ldots,$
predicate symbols	$p,q,r,\ldots,$
connectives	$\wedge,\vee,\neg,\rightarrow,\leftrightarrow,$

quantifiers $\exists, \forall$,
punctuation symbols (,), ','

The last three classes are the same for all alphabets

2. A *term* is defined inductively as follows :

(a) A variable is a term.
(b) A constant is a term.
(c) If f is an n-ary function symbol and $t_1,\ldots,t_n$ are terms then $f(t_1,\ldots,t_n)$ is also a term.

3. A *ground term* is a term which does not contain any variables.

4. A *well-formed formula (simply a formula)* is defined inductively as follows:

(a) If p is an n-ary predicate symbol and $t_1,\ldots,t_n$ are terms then $p(t_1,\ldots,t_n)$ is a formula called atomic formula or atom.

(b) If F and G are formulae then $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$ and $F \leftrightarrow G$ are also formulae.

(c) If F is a formula and x is a variable then $(\forall xF)$ and $(\exists xF)$ are also formulae.

5. A *ground atom* is an atom which does not contain any variables.

The set of all well-formed formulae constructed from the symbols of the alphabets is called a first order language.

$\forall$ is called a universal quantifier,
$\exists$ is called an existential quantifier,
$\wedge$ is called a conjunction and $\vee$ is called a disjunction.

6. The *scope* of $\forall x$ in $\forall xF$ and $\exists x$ in $\exists xF$ is the formula F.

7. Let $Qx_1:F(x_1,\ldots,x_n)$ be a formula, where Q is either $\forall$ or $\exists$. The occurrence of x_1 in $F(x_1,\ldots,x_n)$ is called *bound*. If the occurrence of a variable is not bound then it is a *free occurrence*.

8. A *closed formula* is a formula with no free occurrences of any variable.

9. The *Universal Closure* of a formula F, denoted by $\forall F$, is the closed formula obtained by adding a universal quantifier of any variable having a free occurrence in it, and the *Existential closure* of F, denoted by $\exists F$, is obtained by adding an existential quantifier for every variable having a free occurrence in it.

In all formulae the two quantifiers $\exists,\forall$ have a priority over $(\neg,\wedge,\vee)$, this means that $\exists xp(x,y)\wedge q(x)$ means that $(\exists xp(x,y))\wedge q(x)$

Examples

1. $\exists x\forall y <(x,y)$ is closed while $\exists y<(x,y)$ is not closed since there exist a free occurrence of x.
   The closed formulae have only one value either true or false, for example, if the domain of x is $\mathbb{N}$ then $\forall x<(x,0)$ is always false, while $\forall x(<(x,2x)\wedge>(x,0))$is always true.

2. If F is $p(x,y) \wedge q(x)$ then the universal closure of F is $\forall x \forall y(p(x,y) \wedge q(x))$, and the existential closure of F is $\exists x \exists y(p(x,y),q(x))$.

Definition

A *first order theory* consists of

a) an alphabet,
b) the first order language which is constructed from the symbols of this alphabet,
c) a set, of axioms, which is a designated subset of the well-formed formulae,
d) a set of inference rules that are operations by which new formulae can be generated from the given formulae.

In section 1.6 it will be explained in more details what is meant by an inference rule.

Definitions

1. A *literal* is an atom or the negation of an atom. A positive literal is an atom and a negative literal is the negation of an atom.

2. An *expression* is either a term, literal or a conjunction of literals. A *simple expression* is either a term or an atom.

3. A *clause* is a formula of the form $\forall X_1 \ldots \forall X_s(L_1 \vee \ldots \vee L_m)$ where each L_i is a literal and $X_1, \ldots, X_s$ are the variables occurring in the formula $(L_1 \vee \ldots \vee L_m)$, i.e. a clause is a closed formula of the form $\forall X_1 \ldots \forall X_s(L_1 \vee \ldots \vee L_m)$.

Remark: The clause $\forall X_1 \ldots \forall X_s (A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$ is equivalent to $\forall X_1 \ldots \forall X_s [(A_1 \vee \ldots \vee A_k) \leftarrow (B_1 \wedge \ldots \wedge B_n)]$. If $A_1, \ldots, A_k$, $B_1, \ldots, B_n$ are all atoms, and $X_1, \ldots, X_s$ are all the variables occurring in these atoms then $\forall X_1 \ldots \forall X_s [(A_1 \vee \ldots \vee A_k) \leftarrow (B_1 \wedge \ldots \wedge B_n)]$ will be denoted by $A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$

Thus in any clausal notation all variables are assumed to be universally quantified, the right term $(B_1, \ldots, B_n)$ is called the antecedent or the body of the clause, and the left $(A_1, \ldots, A_k)$ is called the consequent or the head of the clause.

Note that the commas in the body $(B_1, \ldots, B_n)$ denote conjunction and the commas in the head $A_1, \ldots, A_k$ denote disjunction.

Definitions

1. A *definite program clause* is a clause of the form $A \leftarrow B_1, \ldots, B_n$ which contains only one atom in the head.

Any definite program clause is called also a procedure. A is called the procedure name and $(B_1, \ldots, B_n)$ is called the procedure body, each B_i is called a procedure call.
The semantics of this clause is "for all assignment of each variable, if $B_1, \ldots, B_n$ are all true then A is also true".

2. A *unit clause* is a clause of the form $A \leftarrow \square$ i.e. a definite clause with empty body. This means that for each assignment of each variable A is true, it is also called a definite program clause.

3. A *definite program* is a finite set of definite program clauses.

4. A *definite goal* is a clause of the form $\square \leftarrow B_1,\ldots,B_n$ which is a clause with empty head. In this clause each B_i is called a subgoal. If $X_1,\ldots,X_s$ are all the variables of the goal $\square \leftarrow B_1,\ldots,B_n$ then this clause is equivalent to $\forall X1\ldots\forall Xs(\neg B_1 \vee \ldots \vee \neg B_n)$ or $\neg\exists X_1\ldots\neg\exists X_s(B_1 \wedge \ldots \wedge B_n)$.

5. The *empty clause*, denoted by $\square$, is the clause with empty head and empty body.

6. A *Horn clause* is a clause which is either a definite program clause or a definite goal.

In general definite programs are used to compute predicates but they can be also used to compute functions.

Example

Consider the following three predicates :

fact(x,y)=true if x!=y,
fact(x,y)=false if x!≠y,
mult(x,y,z)=true if z= x*y,
mult(x,y,z)=false if z≠ x*y,
sum(x,y,z)=true if z= x+y,
sum(x,y,z)=false if z≠x+y,
and let s(x) be the successor function s(x)=x+1.

Consider the definite program

1. fact(0,s(0))← □
2. fact(s(x),u)← fact(x,v),mult(s(x),v,u)

3. mult(x,0,0)← □
4. mult(x,1,x)← □
5. mult(x,s(y),z)← mult(x,y,v),sum(x,v,z)
6. sum(x,0,x)← □
7. sum(x,s(y),z)← sum(s(x),y,z)

and the goal

□← fact(s(s(s(0))),Y)

This goal can answer the question which element is the factorial of s(s(s(0))).

The formulae 1..7 are called axioms. This program can be regarded as the set of axioms of a first order theory. The language of this theory is given by:

1. The constants *0,s(0).*
2. The variables *x,v,u,z,y.*
3. The function symbol *"s"* which gives the successor of its arguments.
4. The predicate symbols *fact, mult, sum.*

1.2 Foundation of Equational Logic Programming

Basic Definitions

An *alphabet* in the first order theory of an equational logic program is the same as that defined for logic programs except that it includes a new symbol EQ.

Definitions

1. An *equation* has the form EQ(s,t), where s and t are terms. If s=t then EQ(s,t) is a trivial equation,