

# ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՆԱՄԱԼՍԱՐԱՆ

Ինֆորմատիկայի և կիրառական մաթեմատիկայի ֆակուլտետ  
Ծրագրավորման և ինֆորմացիոն տեխնոլոգիաների ամբիոն

Լևոն Արշալույսի Նայկազյան

## Տրամաբանական ծրագրերի համարժեքության պրոբլեմի մասին

“Նաշվողական համալիրների, համակարգերի և ցանցերի մաթեմատիկական  
և ծրագրային ապահովում” մասնագիտությամբ

Մագիստրոսական թեզ

Ղեկավար՝ ֆիզ. մաթ. գիտ. դոկտոր, պրոֆեսոր, Ս. Ա. Նիգիյան

Ընդդիմախոս՝ ֆիզ. մաթ. գիտ. դոկտոր, ՆՆ ԳԱԱ թղթ. անդամ, Ի. Գ. Զապավսկի

Երևան 2009թ.

## **Նամառուր բնութագիր**

Աշխարհանքում ուսումնասիրվում է փրամաբանական ծրագրերի համարժեքությունը և այդ պրոբլեմի լուծելիությունը:

Տրամաբանական ծրագիրն իրենից ներկայացնում է պրեդիկատային հաշվի բանաձև: ՌԻսդի բնական է ծրագրերը համարել համարժեք, եթե նրանք համարժեք են որպես բանաձևեր: Մյուս կողմից, փրամաբանական ծրագիրը գիպելիքների բազա է, և կարելի է ծրագրերը համարել համարժեք, եթե հավասար են հարցումների բազմությունները որոնք փրամաբանորեն հեղուկում են ծրագրերից: Այս համարժեքության հարաբերությունը կոչվում է  $\Delta$ -համարժեքություն:

Աշխարհանքում ուսումնասիրվում է վերոհիշյալ երկու հարաբերությունների լուծելիությունը: Ցույց է արվում, որ ընդհանուր դեպքում դրանք այգորիթմորեն լուծելի չեն: Ապացուցվում է նաև, որ մասնավոր դեպքում, երբ օգտագործվող պրեդիկատային և ֆունկցիոնալ սիմվոլների պրեդայնությունը չի գերազանցում մեկը, երկու հարաբերություններն էլ լուծելի են:

YEREVAN STATE UNIVERSITY

Faculty of Informatics and Applied Mathematics  
Department of Programming and Information Technologies

---

Levon Haykazyan

On Equivalence Problem of Logic Programs

Master Thesis

in “Mathematical and Software Support of Computer Complexes,  
Systems, and Networks”

Supervisor: Doctor of Sc. in Math, Professor, S. A. Nigyan

Opponent: Doctor of Sc. in Math, Corr. mem. of NAS RA, I. D. Zaslavsky

Yerevan, 2009

## **Acknowledgements**

First of all, I would like to thank my supervisor Semyon Nigiyan. He has been a marvelous source of ideas and inspiration for me. Doing research under his supervision was a great pleasure for me.

I would like to thank my family and friends. Without their support and patience this research would be impossible.

## **Abstract**

The thesis is devoted to study of equivalence of logic programs and its decidability.

Logic programs are special cases of first order formulas. Therefore, it is natural to consider them equivalent if they are equivalent as formulas. On the other hand, programs are knowledge bases, and we may consider programs as equivalent if the sets of goals that are logical consequences of the programs are the same. The later is called  $\Delta$ -equivalence.

We study the decidability of above mentioned equivalences. It is shown, that in general case, both equivalences are not decidable. Nevertheless, both relations are decidable for special case of so-called monadic programs (programs that do not use functional and predicate symbols with arity greater than one).

# Contents

<b>1</b>	<b>Preliminaries</b>	<b>5</b>
1.1	First Order Logic . . . . .	5
1.2	Logic Programs . . . . .	11
1.3	Monadic second order theory of successor functions . . . . .	16
<b>2</b>	<b>Monadic first order logic</b>	<b>20</b>
2.1	Herbrand Interpretations . . . . .	20
2.2	Monadic language with a single constant . . . . .	23
2.3	Decidability of monadic first order logic . . . . .	27
<b>3</b>	<b>Equivalence of logic programs</b>	<b>29</b>
3.1	Equivalence as logic formulas . . . . .	29
3.2	$\Delta$ -equivalence . . . . .	31

# Chapter 1

## Preliminaries

This chapter introduces mathematical logic and logic programming to the reader. Logic programs are special formulas of first order logic and their semantics are defined accordingly. While all requisite concepts are discussed, the chapter doesn't tend to be a comprehensive survey neither in mathematical logic nor in logic programming. We suggest reading [Llo84] or [NM95].

### 1.1 First Order Logic

First order logic has two aspects: its *syntax* and *semantics*. The syntactic aspect is concerned with formulas of logic - a class of finite sequences of symbols of formal language, whereas the semantic aspect is concerned with the meaning attached to such sequences. A formula in logic is roughly equivalent to a declarative sentence in an ordinary language.

From the syntactic point of view logic formulas are sequences of symbols.

**Definition 1.1 (Alphabet).** The alphabet of the language of the first order logic consist of the following classes of symbols:

- *variables* - usually denoted by  $x, y, z$  (possibly subscripted).
- *constants* - usually denoted by  $a, b, c$  (possibly subscripted).
- *functional symbols* - usually denoted by  $f, g, h$  (possibly subscripted). Each functional symbol has an associated positive integer called its *arity*. If the arity of a functional symbol is  $n$ , the functional symbol is called  $n$ -ary.

- *predicate symbols* - usually denoted by  $p, q, r$  (possibly subscripted). Each predicate symbol has associated  $> 0$  arity. If the arity of a predicate symbol is  $n$ , the predicate symbol is called  $n$ -ary.
- *logical connectives* -  $\wedge$  (conjunction),  $\vee$  (disjunction),  $\supset$  (implication),  $\sim$  (equivalence) and  $\neg$  (negation).
- *quantifiers* -  $\exists$  (existential) and  $\forall$  (universal).
- *punctuation symbols* -  $(, )$  and  $,$ .

The last three classes of symbols are the same for each language, while the first four may differ. We will usually use terms *nullary*, *unary*, *binary* and *ternary* as 0-ary, 1-ary, 2-ary and 3-ary respectively. Constants are sometimes seen as nullary functional symbols.

Special classes of words of the alphabet are called *terms*. Terms are roughly equivalent to nouns in ordinary language.

**Definition 1.2 (Terms).** The set of terms is the least set such that:

- every constant is a term.
- every variable is a term.
- if  $f$  is an  $n$ -ary functional symbol and  $t_1, \dots, t_n$  are terms, then  $f(t_1, \dots, t_n)$  is a term.

Terms are typically denoted by  $s$  and  $t$ .

As already mentioned *formulas* or so called *well formed formulas* of first order languages are the equivalent of declarative sentences of natural languages.

**Definition 1.3 (Formulas).** The set of formulas is the least set such that:

- if  $p$  is an  $n$ -ary predicate symbol and  $t_1, \dots, t_n$  are terms, then  $p(t_1, \dots, t_n)$  is a formula (called an *atomic formula* or an *atom*).
- if  $F$  and  $G$  are formulas, then so are  $(\neg F)$ ,  $(F \wedge G)$ ,  $(F \vee G)$ ,  $(F \supset G)$  and  $(F \sim G)$ .
- if  $F$  is a formula and  $x$  is a variable then  $(\forall xF)$  and  $(\exists xF)$  are formulas.

The formulas are usually denoted by uppercase Latin letters. We will sometimes write the formula  $F \supset G$  as  $G \leftarrow F$ . To decrease the number of parenthesis, we define the following precedence hierarchy with highest precedence at the beginning

$$\forall, \exists, \neg, \wedge, \vee, \supset, \sim.$$

Whenever no ambiguity arises, we will drop unnecessary parenthesis.

**Example 1.4.** If  $x, y$  and  $z$  are variables,  $a$  is a constant,  $g$  is an unary functional symbol,  $f$  is a binary functional symbol,  $p$  and  $q$  are binary predicate symbols, then

- $x$  and  $y$  are terms.
- $f(x, y), f(x, f(y, g(a)))$  are terms.
- $(\forall x(\exists y(p(f(x, y), z) \supset (\forall xq(g(a), x)))))$  is a formula, shortly denoted as  $\forall x\exists y(p(f(x, y), z) \supset \forall xq(g(a), x))$ .

**Definition 1.5 (First Order Language).** The first order language of given alphabet is the set of all formulas of given alphabet.

We will usually equate the alphabet with the language. That is, instead of referring to the variables, constants, etc. of alphabet we will refer to variables and constants of the language. As in this thesis we will deal mostly with first order languages, the term language means first order language by default.

The *scope* of  $\forall x$  and  $\exists x$  in formulas  $\forall xF$  and  $\exists xF$  respectively is the formula  $F$ . An occurrence of a variable in a formula is called *bound* if it is either directly after a quantifier or within the scope of a quantifier with the same variable after it. Otherwise the occurrence is called *free*. Free variables of a formula are the ones with at least one free occurrence in the formula. A formula containing no free occurrences of any variable is called *closed*. A formula or term containing no variables is called *ground*. If  $x_1, \dots, x_n$  are the free variables of the formula  $F$ , then the formula  $\forall x_1 \dots \forall x_n F$  is called *universal closure* of  $F$  and is abbreviated to  $\forall F$ . The *existential closure* of formula  $F$  is defined analogically and is abbreviated to  $\exists F$ . Note, that the universal and existential closures of a formula are not defined unambiguously. Nevertheless, as we will see, all universal (existential) closures of the formula are in some sense equivalent to each other.

**Example 1.6.** If  $x$  and  $y$  are variables,  $a, b$  and  $c$  are constants and  $p, q$  are predicate symbols of corresponding arity, then

- in the formula  $\exists x p(x, y) \supset q(x)$  the first two occurrence of the variables  $x$  is bound, whereas the third one is free.
- in the formula  $\exists x(p(x, y) \supset q(x))$  all occurrences of the variable  $x$  are bound.
- the formula  $\forall y \exists x(p(x, y) \supset q(x))$  is closed.
- the formula  $p(a, b) \supset q(a)$  is ground.
- the universal closure of the formula  $p(x, y) \supset q(x)$  would the formula  $\forall x \forall y(p(x, y) \supset q(x))$  or the formula  $\forall y \forall x(p(x, y) \supset q(x))$ .

Now we precede to define the meaning of formulas, that is the semantics of the first order logic. As stated above, formulas are somewhat equivalent to declarative statements of natural languages. Formulas express some statements about an abstract world. The meaning of a formula is defined relative to some abstract world called *structure*. The abstract world consist of set of individuals called *domain* and some relationships between them expressing the meaning of constants, functional and predicate symbols. The individuals in the abstract world are referred by nouns, and in our case by terms. Thus the meaning of a term should be an individual in the abstract world and the meaning of a formula should be a truth value.

**Definition 1.7 (Interpretation).** An interpretation of  $\mathcal{I}$  of a first order language  $L$  consist of a non-empty domain  $D$  and the mapping that associates:

- each constant  $c$  of  $L$  with an element  $c_{\mathcal{I}} \in D$ ;
- each  $n$ -ary functional symbol  $f$  of  $L$  with a function  $f_{\mathcal{I}}: D^n \rightarrow D$ ;
- each  $n$ -ary predicate symbol with a relation  $p_{\mathcal{I}} \subseteq D^n$ .

Since interpretation does not assign anything to variables, the auxiliary notation of valuation is required. A *valuation* is a mapping from the set of variables of the language to the domain of the interpretation. If  $\varphi$  is a valuation  $x$  is a variable and  $d \in D$ , then  $\varphi[x \rightarrow d]$  denotes the valuation that assign to each variable the same element that  $\varphi$ , except the variable  $x$  to which it assigns the element  $d$ .

**Definition 1.8 (Semantics of terms).** Let  $L$  be a first order language,  $\mathcal{I}$  be its interpretation and  $\varphi$  - a valuation. The semantics of each term  $t$  of  $L$  (with respect to the interpretation and valuation) is an element in the domain of the interpretation denoted as  $Val_{\varphi}(t)$  and defined as:

- if  $t$  is a constant  $c$  then  $Val_\varphi(t) \stackrel{\text{def}}{=} c_{\mathcal{I}}$ ;
- if  $t$  is a variable  $x$  then  $Val_\varphi(t) \stackrel{\text{def}}{=} \varphi(x)$ ;
- if  $t$  is of the form  $f(t_1, \dots, t_n)$  then  $Val_\varphi(t) \stackrel{\text{def}}{=} f_{\mathcal{I}}(Val_\varphi(t_1), \dots, Val_\varphi(t_n))$ .

**Definition 1.9 (Semantics of formulas).** Let  $L$  be a first order language,  $\mathcal{I}$  be its interpretation with domain  $D$  and  $\varphi$  - a valuation. The semantics of a formula  $A$  (with respect to the interpretation and valuation) is a truth value. The fact that the value is true, is denote as  $\mathcal{I} \models_\varphi A$  and the converse is denoted as  $\mathcal{I} \not\models_\varphi A$ . The semantics is defined as follows:

- $\mathcal{I} \models_\varphi p(t_1, \dots, t_n)$  iff  $\langle Val_\varphi(t_1), \dots, Val_\varphi(t_n) \rangle \in p_{\mathcal{I}}$ ;
- $\mathcal{I} \models_\varphi \neg F$  iff  $\mathcal{I} \not\models_\varphi F$ ;
- $\mathcal{I} \models_\varphi F \wedge G$  iff  $\mathcal{I} \models_\varphi F$  and  $\mathcal{I} \models_\varphi G$ ;
- $\mathcal{I} \models_\varphi F \vee G$  iff  $\mathcal{I} \models_\varphi F$  or  $\mathcal{I} \models_\varphi G$ ;
- $\mathcal{I} \models_\varphi F \supset G$  iff  $\mathcal{I} \models_\varphi G$  whenever  $\mathcal{I} \models_\varphi F$ ;
- $\mathcal{I} \models_\varphi F \sim G$  iff  $(\mathcal{I} \models_\varphi F \text{ and } \mathcal{I} \models_\varphi G)$  or  $(\mathcal{I} \not\models_\varphi F \text{ and } \mathcal{I} \not\models_\varphi G)$ ;
- $\mathcal{I} \models_\varphi \forall x F$  iff  $\mathcal{I} \models_{\varphi[x \rightarrow d]}$  for every  $d \in D$ ;
- $\mathcal{I} \models_\varphi \exists x F$  iff  $\mathcal{I} \models_{\varphi[x \rightarrow d]}$  for some  $d \in D$ .

The semantics of a formula is defined with respect to valuation. Easy to see that the semantics of a formula essentially depends only on values of its free variables. That is, for interpretation  $\mathcal{I}$ , formula  $F$  with  $x_1, \dots, x_n$  as its free variables and valuations  $\varphi$  and  $\psi$ , if  $\varphi(x_i) = \psi(x_i)$  for  $i = 1, \dots, n$ , then  $\mathcal{I} \models_\varphi F$  iff  $\mathcal{I} \models_\psi F$ . As a consequence the semantics of a closed formula does not depend on valuation. That is why we will use the notation  $\mathcal{I} \models F$  instead of  $\mathcal{I} \models_\varphi F$  for ground formulas.

**Example 1.10.** Consider a language with a single constant  $c$ , unary functional symbol  $s$ , binary functional symbol  $f$  and a single unary predicate symbol  $p$ . Consider the following interpretation  $\mathcal{I}$ : the domain of the interpretation is the set of natural numbers  $N = \{0, 1, 2, \dots\}$ , the constant  $c$  is mapped to 0, the functional symbol  $s$  is mapped to the function  $s_{\mathcal{I}}(i) = i + 1$ , the functional symbol  $f$  to multiplication and  $p$  to the unary relation  $\{n^2 \mid n \in N\}$ .

- $\mathcal{I} \models \forall xp(f(f(x, s(c)), x))$  iff  
for every natural number  $i$ ,  $\mathcal{I} \models_{[x \rightarrow i]} p(f(f(x, s(c)), x))$  iff  
for every natural number  $i$ ,  $f_{\mathcal{I}}(f_{\mathcal{I}}(i, s_{\mathcal{I}}(0)), i) \in p_{\mathcal{I}}$   
for every natural number  $i$ ,  $f_{\mathcal{I}}(f_{\mathcal{I}}(i, 1), i) \in p_{\mathcal{I}}$  iff  
for every natural number  $i$ ,  $i^2 \in p_{\mathcal{I}}$ , which is obviously true.  
Therefore  $\mathcal{I} \models \forall xp(f(f(x, s(c)), x))$ .
- $\mathcal{I} \models \exists xp(f(s(s(x)), s(x)))$  iff  
for some natural number  $i$ ,  $\mathcal{I} \models_{[x \rightarrow i]} p(f(s(s(x))), s(x))$  iff  
for some natural number  $i$ ,  $(i + 2) \times (i + 1) \in p_{\mathcal{I}}$ , which is obviously  
false.  
Therefore  $\mathcal{I} \not\models \exists xp(f(s(s(x)), s(x)))$ .

**Definition 1.11 (Model).** Consider a language and a closed formula in it. The interpretation of the language is said to be a model of the formula, if the formula is true in the interpretation.

The concept of a model is widened to the set of closed formulas as follows: an interpretation is called a model of set of closed formulas, if it is a model of each formula in the set. Note that an interpretation will be a model of a set of closed formulas if and only if it is a model of conjunction of the formulas. From this point of view, introduction of a model of a set of closed formulas may seem redundant. However, sometimes speaking about sets are more convenient than speaking about conjunctions.

We will often speak about the interpretation of a formula or set of formulas, instead of language. In such cases it is assumed that there is an underlying first order language, which contains exactly the variables, constants, functional and predicate symbols of the formula. And the interpretation is that of the language.

A formula always has infinitely many interpretations. However, it may happen that none of them is a model of the formula. Such formula is called *unsatisfiable*. A trivial example of an unsatisfiable formula is  $F \wedge \neg F$ . It may also happen that all interpretations are models of the formula. Such formula is called *valid*. A trivial example of a valid formula is  $F \vee \neg F$ . Easy to set that the formula is valid if and only if its negation is unsatisfiable (and vice versa). The formula is called *satisfiable* if it has at least one model.

**Definition 1.12 (Logical consequence).** Consider a first order language  $L$ . A closed formula  $F$  is called a logical consequence of a set of closed

formulas  $S$  (denoted as  $S \models F$ ), if every interpretation of  $L$  that is a model of  $S$  is also a model of  $F$ .

A direct consequence of these definitions is the following proposition.

**Proposition 1.13.** Let  $S$  be a set of closed formulas and  $F$  be closed formula.  $S \models F$  if and only if  $S \cup \{\neg F\}$  is unsatisfiable.

Next we introduce a fundamental concept of logic programming - substitution.

**Definition 1.14 (Substitution).** A substitution is a finite set of pairs  $\{x_1/t_1, \dots, x_n/t_n\}$  ( $n \geq 0$ ), where  $x_i$  is a variable,  $t_i$  is a term such that  $x_i \neq t_i$  and  $x_i \neq x_j$  if  $i \neq j$ .

Let  $\theta$  be a substitution  $\{x_1/t_1, \dots, x_n/t_n\}$  and  $E$  be a formula (term). The application of  $\theta$  to formula (term)  $E$  is a formula (term) denoted as  $E\theta$  and obtained from  $E$  by simultaneously replacing all free occurrences of  $x_i$  by  $t_i$ . (All occurrences of variables in terms are by definition free).  $E\theta$  is called an instance of  $E$ . The empty substitution is denoted by  $\varepsilon$ . For every formula  $F$ ,  $F\varepsilon$  is  $F$ . The substitution is called *ground* if all  $t_i$  are ground.

**Example 1.15.** If  $F$  is the formula  $p(x, y)$  and  $\theta = \{x/f(y), y/f(x)\}$  then

- $F\theta$  is  $p(f(y), f(x))$ ;
- $(F\theta)\theta$  is  $p(f(f(x)), f(f(y)))$ .

## 1.2 Logic Programs

Logic programming, in its broadest sense, is the usage of mathematical logic in computer programming. More specifically, logic programs are special cases of formulas in first order logic. The role of the programmer is to develop such formulas and the role of the computer is to derive conclusions from that.

**Definition 1.16 (Clause).** A clause is a formula of the form  $\forall(L_1 \vee \dots \vee L_n)$ , where  $n \geq 0$  and each  $L_i$  is either an atom (positive literal) or a negation of an atom (negative literal).

The clause

$$\forall(A_1 \vee \dots \vee A_n \vee \neg B_1 \vee \dots \vee \neg B_m),$$

where  $A_1, \dots, A_n, B_1, \dots, B_m$  are atoms will be denoted as

$$A_1, \dots, A_n \leftarrow B_1, \dots, B_m.$$

In case of  $m = 0$  the arrow is usually omitted.

A clause with exactly one positive literal is called *definite* clause. The positive literal of definite clause is called its *head* and the negative ones - its *body*.

**Definition 1.17 (Program).** A (logic) program is a non-empty finite set of definite clauses.

As we have already mentioned, from the model-theoretical point of view the set of closed formulas is identical to the conjunction of them.

**Example 1.18.** The following program is to calculate the addition of two natural numbers. The natural number  $n$  is encoded with the term  $s^n(0)$ .

$$\begin{aligned} &plus(x, 0, x) \\ plus(x, s(y), s(z)) &\leftarrow plus(x, y, z) \end{aligned}$$

The program will be used by computer to derive conclusions. More specifically, the computer is intended to derive formulas which are logical consequences of the program. However, as there are infinitely many of such formulas, the user is expected to query the computer. Such queries are expressed with special types of clauses.

**Definition 1.19 (Goal).** A goal is a clause of the form

$$\leftarrow A_1, \dots, A_n$$

In the case of  $n = 0$ , the goal is called *empty* and is denoted as  $\square$ . This is to be understood as contradiction.

Let us examine the structure of the goal. The goal close above is equivalent to the formula

$$\neg \exists x_1 \dots \exists x_m (A_1 \wedge \dots \wedge A_n),$$

where  $x_1, \dots, x_m$  are the variables of the clause. When we are querying the computer with the goal, we are asking to derive its negation. Moreover, we are interested for the proof to be constructive and to yield the values for  $x_1, \dots, x_m$ .

**Example 1.20.** For the program in the example 1.18 to get the result of addition of three to two we should query the computer with the goal

$$\leftarrow plus(s(s(s(0))), s(s(0)), x)$$

and we are expecting the computer to yield something like  $x = s(s(s(s(s(0))))))$ .

Consider a program  $P$  and a goal  $G$ . To show that  $\neg G$  is a logical consequence of  $P$ , one needs to show that  $P \cup \{G\}$  is unsatisfiable. This means to show that every interpretation of the underlying first order language is not a model of  $P \cup \{G\}$ . No need to say that this is a very difficult task. Luckily, there is a class of interpretation, called Herbrand interpretations, which simplifies this task a lot.

**Definition 1.21 (Herbrand universe).** Let  $L$  be a first order language with at least one constant. The set of all ground terms of  $L$  is called Herbrand universe and is denoted as  $U_L$ .

**Definition 1.22 (Herbrand base).** Let  $L$  be a first order language with at least one constant. The set of all ground atoms of  $L$  is called Herbrand base and is denoted as  $B_L$ .

The precondition for the language to contain at least one constant is needed to make sure that the Herbrand universe and base are not empty. In what follows, whenever we speak about Herbrand universe or base of a language it is assumed that the set of its constants is not empty. As in the case of interpretations we will sometimes speak about the Herbrand universe and base of a formula (program). In such cases it is assumed that there is an underlying first order language, which contains exactly the variables, constants, functional and predicate symbols of the formula. If the formula does not contain any constant, a single one, say  $a$  is added to the language. The Herbrand universe and base are considered as that of the language.

**Example 1.23.** For the program in the example 1.18, the Herbrand universe is

$$\{s^n(0) \mid n \in \mathbb{N}\}$$

and the Herbrand base is

$$\{plus(s^n(0), s^m(0), s^k(0)) \mid n, m, k \in \mathbb{N}\}$$

**Definition 1.24 (Herbrand interpretation).** An interpretation  $\mathcal{I}$  of a first order language  $L$  is called Herbrand interpretation, if

- the domain of the interpretation is  $U_L$ ;
- every constant is mapped to itself;
- every  $n$ -ary functional symbol  $f$  is mapped to the function  $f_{\mathcal{I}}$  such that  $f_{\mathcal{I}}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$ .

Various Herbrand interpretations arise from various interpretations of predicate symbols. Thus, each Herbrand interpretation is identified with a subset of the Herbrand base - the set of ground atoms which are assigned true value by the interpretation. Conversely, each subset of Herbrand base determines a single Herbrand interpretation - the interpretation which assigns true value to the atoms of the subset. Throughout the text, we will refer to Herbrand interpretations as subsets of Herbrand base.

Next we prove an important result which makes Herbrand interpretations so valuable.

**Theorem 1.25.** Let  $S$  be a set of clauses. If  $S$  is satisfiable, then it has a Herbrand model.

*Proof.* Let  $\mathcal{I}$  be a model of  $S$ . Define the Herbrand interpretation  $\mathcal{I}'$  as follows:

$$\mathcal{I}' \stackrel{\text{def}}{=} \{A \in B_L \mid \mathcal{I} \models A\}.$$

Let us prove that  $\mathcal{I}'$  is a model of  $S$ . Suppose the opposite. This means that there is a clause in  $S$  which is false with respect to  $\mathcal{I}'$ . Let the clause be

$$A_1, \dots, A_n \leftarrow B_1, \dots, B_m.$$

Thus, for some ground substitution  $\theta$  of variables of the clause

$$A_i\theta \notin \mathcal{I}' \quad (i = 1, \dots, n)$$

and

$$B_j\theta \in \mathcal{I}' \quad (j = 1, \dots, m).$$

But this means that

$$\mathcal{I} \not\models A_i\theta \quad (i = 1, \dots, n)$$

and

$$\mathcal{I} \models B_j\theta \quad (j = 1, \dots, m).$$

Hence  $\mathcal{I}$  is not a model of  $S$ , which contradicts the assumption. Therefore  $\mathcal{I}'$  is a Herbrand model of  $S$ . □

Now, if we have a program  $P$  and a goal  $G$ , to determine that  $\neg G$  is logical consequence of  $P$ , we need to show that  $P \cup \{G\}$  does not have a Herbrand model.

The theorem however, does not hold for arbitrary formulas. For example the formula  $p(a) \wedge \exists x \neg p(x)$  is satisfiable, but doesn't have a Herbrand model, if the underlying language doesn't contain any constant and functional symbol except  $a$ . To use Herbrand interpretations we need to add symbols to the language. This is discussed in more details in the next chapter.

**Theorem 1.26.** Let  $P$  be a logic program and  $\mathcal{M}_\alpha$  a non-empty family of Herbrand models of  $P$ . Then  $\mathcal{M} = \bigcap_{\alpha} \mathcal{M}_\alpha$  is a Herbrand model of  $P$ .

*Proof.* Let

$$A_0 \leftarrow A_1, \dots, A_n$$

be a ground instance of a clause of  $P$  and  $A_i \in \mathcal{M}$  for every  $i = 1, \dots, n$ . This means that  $A_i \in \mathcal{M}_\alpha$  for every  $i = 1, \dots, n$  and every  $\alpha$ . Therefore  $A_0 \in \mathcal{M}_\alpha$  for every  $\alpha$  and  $A_0 \in \mathcal{M}$ . Hence,  $\mathcal{M}$  is a model of  $P$ . □

Consider a program  $P$ . The set of all Herbrand models of  $P$  is not empty as the Herbrand base is a model of each program. Denote by  $\mathcal{M}_P$  the intersection of all Herbrand models of  $P$ . The previous theorem states that  $\mathcal{M}_P$  is a model of  $P$ .  $\mathcal{M}_P$  is also the least model of  $P$  in the sense that every other model is a superset of it.  $\mathcal{M}_P$  is called the least Herbrand model of the program  $P$ .

**Theorem 1.27.** The least Herbrand model of the program is the set of ground atoms that are logical consequences of the program. That is

$$\mathcal{M}_P = \{A \in B_P \mid P \models A\}.$$

*Proof.* Let  $A \in \mathcal{M}_P$ . Therefore, for every Herbrand model  $\mathcal{M}$  of  $P$ ,  $A \in \mathcal{M}$ . This means that  $P \cup \{\neg A\}$  doesn't have a Herbrand model. As  $P \cup \{\neg A\}$  is a set of clauses, it doesn't have any model. Therefore  $A$  is a logical consequence of  $P$ .

For the converse suppose  $A \in B_P$  and  $P \models A$ . Hence for every Herbrand model  $\mathcal{M}$  of  $P$ ,  $A \in \mathcal{M}$ . Therefore,  $A \in \mathcal{M}_P$ . □

Let us summarise the results, we have obtained. Consider a program  $P$  and a goal:

$$\leftarrow A_1, \dots, A_n.$$

The computer is to prove that the negation of the goal is a logical consequence of the program. The negation of the goal is the following formula:

$$\exists(A_1 \wedge \dots \wedge A_n).$$

It turns out that the computer is to find a ground substitution  $\theta$  of variables of the goal such that  $A_i\theta$  is a logical consequence of the program. By the previous theorem this is equivalent to  $A_i\theta \in \mathcal{M}_P$ .

Therefore, we need some constructive way of describing the least Herbrand model. Consider a program  $P$ . Denote the set of ground instances of clauses of  $P$  as  $ground(P)$ . Define the function  $T_P: 2^{B_L} \rightarrow 2^{B_L}$  as

$$T_P(\mathcal{I}) = \{A_0 \mid A_0 \leftarrow A_1, \dots, A_n \in ground(P) \text{ and } A_1, \dots, A_n \subseteq \mathcal{I}\}.$$

Denote

$$T_P \uparrow 0 \stackrel{\text{def}}{=} \emptyset$$

for every natural number  $i$ ,

$$T_P \uparrow i + 1 \stackrel{\text{def}}{=} T_P(T_P \uparrow i)$$

and

$$T_P \uparrow \omega \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} T_P \uparrow i$$

**Example 1.28.** For the program  $P$  from the example 1.18

$$T_P \uparrow 0 = \emptyset$$

$$T_P \uparrow 1 = \{plus(s^n(0), 0, s^n(0)) \mid n \in \mathbb{N}\}$$

$$T_P \uparrow 2 = \{plus(s^n(0), 0, s^n(0)) \mid n \in \mathbb{N}\} \cup \{plus(s^n(0), s(0), s^{n+1}(0)) \mid n \in \mathbb{N}\}$$

⋮

$$T_P \uparrow i = \{plus(s^n(0), s^m(0), s^{n+m}(0)) \mid n, m \in \mathbb{N} \text{ and } m < i\}$$

⋮

$$T_P \uparrow \omega = \{plus(s^n(0), s^m(0), s^{n+m}(0)) \mid n, m \in \mathbb{N}\}$$

The following theorem, presented here without proof, shows the importance of the mapping  $T_P$ .

**Theorem 1.29.** Let  $P$  be a program. Then

$$T_P \uparrow \omega = \mathcal{M}_P$$

For the proof the reader should consult [Llo84].

## 1.3 Monadic second order theory of successor functions

In this section we will study one second order theory called second order theory of successor function. In general, second order theories are similar to first order theories, except that they allow quantification over predicate symbols. As a result predicate symbols are called predicate (or second order) variables.

Let us fix some natural number  $n$ .

**Definition 1.30 (Alphabet of  $S_nS$ ).** The alphabet of monadic second order theory of  $n$  successor functions, shortly denoted as  $S_nS$ , consists of

- a countable set of *first order variables* - usually denoted as  $x, y, z$ , possibly subscripted;
- a countable set of *second order variables* - usually denoted as  $X, Y, Z$ , possibly subscripted;
- a single *constant*  $\Lambda$ ;
- a set of  $n$  *unary functional symbols*  $r_0, r_1, \dots, r_{n-1}$ ;
- logical connectives, quantifiers and punctuation symbols of a first order language.

**Definition 1.31 (Terms of  $S_nS$ ).** The set of terms of  $S_nS$  is the least set such that

- every first order variable  $x$  is a term;
- the constant  $\Lambda$  is a term;
- if  $s$  is a term and  $r_i$  is a functional symbol, then  $r_i(s)$  is a term.

As we see, the set of terms of  $S_nS$  are precisely those of the first order language with a single constant and  $n$  monadic functional symbols.

Second order variables of  $S_nS$  act as monadic predicate symbols of first order languages, except that quantification over them is allowed. More precise definition of formulas follows.

**Definition 1.32 (Formulas of  $S_nS$ ).** The set of formulas of  $S_nS$  is the least set such that:

- if  $X$  is a second order variable and  $t$  is a term, then  $X(t)$  is a formula;
- if  $F$  and  $G$  are formulas, then  $(\neg F)$ ,  $(F \wedge G)$ ,  $(F \vee G)$ ,  $(F \supset G)$ ,  $(F \sim G)$  are formulas;
- if  $x$  is a first order variable and  $F$  is a formula, then  $(\forall xF)$  and  $(\exists xF)$  are formulas;
- if  $X$  is a second order variable and  $F$  is a formula, then  $(\forall XF)$  and  $(\exists XF)$

Now let us define the semantics of  $SnS$ . The semantics of terms and formulas of  $SnS$  is defined with respect to Herbrand interpretation. As  $SnS$  doesn't contain any predicate symbols there is a single Herbrand interpretation and the semantics of formulas and terms doesn't depend on interpretation at all. The Herbrand universe is denoted by  $T_n$ . Instead the semantics depends on valuation. But as formulas (not terms) contain also second order variables, we need two kinds of valuations: a valuation of first order variables - a mapping from the set of variables to  $T_n$  and a valuation of second order variables - a mapping from the set of second order variables to  $2^{T_n}$ .

**Definition 1.33 (Semantics of terms of  $SnS$ ).** Consider a valuation  $\varphi_1$  of first order variables of  $SnS$ . The semantics of a term  $t$  with respect to the valuation (denoted as  $Val_{\varphi_1}(t)$ ) is a ground term defined as follows:

- if  $t$  is the constant  $\Lambda$ , then  $Val_{\varphi_1}(t) \stackrel{\text{def}}{=} \Lambda$ ;
- if  $t$  is a variable  $x$ , then  $Val_{\varphi_1}(t) \stackrel{\text{def}}{=} \varphi_1(x)$ ;
- if  $t$  is of the form  $r_i(s)$   $Val_{\varphi_1}(t) \stackrel{\text{def}}{=} r_i(Val_{\varphi_1}(s))$ .

**Definition 1.34 (Semantics of formulas of  $SnS$ ).** Consider a valuation  $\varphi_1$  of first order variables, a valuation  $\varphi_2$  of second order variables  $SnS$ . The semantics of a formula  $A$  with respect to both valuation is a truth value. The fact that the value is true is denoted as  $\models_{\varphi_1, \varphi_2} A$  and the converse is denoted as  $\not\models_{\varphi_1, \varphi_2} A$ . The semantics is defined as follows:

- $\models_{\varphi_1, \varphi_2} X(t)$  iff  $Val_{\varphi_1}(t) \in \varphi_2(X)$ ;
- $\models_{\varphi_1, \varphi_2} \neg F$  iff  $\not\models_{\varphi_1, \varphi_2} F$ ;
- $\models_{\varphi_1, \varphi_2} F \wedge G$  iff  $\models_{\varphi_1, \varphi_2} F$  and  $\models_{\varphi_1, \varphi_2} G$ ;

- $\models_{\varphi_1, \varphi_2} F \vee G$  iff  $\models_{\varphi_1, \varphi_2} F$  or  $\models_{\varphi_1, \varphi_2} G$ ;
- $\models_{\varphi_1, \varphi_2} F \supset G$  iff  $\models_{\varphi_1, \varphi_2} G$  whenever  $\models_{\varphi_1, \varphi_2} F$ ;
- $\models_{\varphi_1, \varphi_2} F \sim G$  iff ( $\models_{\varphi_1, \varphi_2} F$  and  $\models_{\varphi_1, \varphi_2} G$ ) or ( $\not\models_{\varphi_1, \varphi_2} F$  and  $\not\models_{\varphi_1, \varphi_2} G$ );
- $\models_{\varphi_1, \varphi_2} \forall x F$  iff  $\models_{\varphi_1[x \rightarrow t], \varphi_2} F$  for every  $t \in T_n$ ;
- $\models_{\varphi_1, \varphi_2} \exists x F$  iff  $\models_{\varphi_1[x \rightarrow t], \varphi_2} F$  for some  $t \in T_n$ ;
- $\models_{\varphi_1, \varphi_2} \forall X F$  iff  $\models_{\varphi_1, \varphi_2[X \rightarrow R]} F$  for every  $R \subseteq T_n$ ;
- $\models_{\varphi_1, \varphi_2} \exists X F$  iff  $\models_{\varphi_1, \varphi_2[X \rightarrow R]} F$  for some  $R \subseteq T_n$ ;

As in case of first order logic, the semantics of a formula depends only on what the valuation assigns to its free variables only. As a consequence, the semantics of a closed formula doesn't depend on any kind of valuation and, therefore indexes  $\varphi_1, \varphi_2$  are usually omitted.

We note that the syntax and semantics of  $SnS$  are quite similar to those of monadic first order logic. In fact the essential difference is the introduction of second order variables and quantification over them. This make the logic essentially more expressive. One major restriction of  $SnS$  is that it uses a fixed interpretation - namely the Herbrand interpretation.

Our interest in the theory  $SnS$  is because of its decidability. More precisely, the following theorem holds.

**Theorem 1.35 (Rabin [Rab69]).** There is an algorithm for deciding whether the given closed formula of  $SnS$  is true or false.

The proof of the theorem involves a notation of finite automata on infinite trees and is not presented here. The details of the proof, the reader can find in [GTW02]. However, the theorem provides a very useful framework for decidability investigation and will repeatedly refer to it in the forthcoming chapters.

# Chapter 2

## Monadic first order logic

In this chapter we explore monadic first order logic. A first order language is called monadic, if all functional and predicate symbols are unary. The logic over monadic first order language is called monadic first order logic. Our primary goal would be to investigate the decidability of the first order logic. That is, the decidability of the following problem: given a formula in a monadic first order language, determine whether it is satisfiable. Easy to see, that the decidability of this problem means the decidability of validity and unsatisfiability problems.

### 2.1 Herbrand Interpretations

Consider an arbitrary first order language  $L$  and a closed formula  $F$  in it. Every interpretation of  $L$  assigns truth value to  $F$ . However, as easily seen from the definition of the semantics of formulas, that value doesn't depend on what the interpretation assigns to constants, functional and predicate symbols of  $L$  that doesn't occur in  $F$ . Therefore, if we add some new symbols to  $L$  or remove some (provided that  $F$  is still a "well formed" formula in the new language), the satisfiability, unsatisfiability and validity of  $F$  wouldn't change. Therefore, we will often speak about satisfiability or validity of a formula without defining a language at all.

The situation is somewhat different when we consider Herbrand interpretations only.

**Example 2.1.** Consider the following formula:

$$p(a) \wedge \exists x \neg p(x).$$

If the underlying language doesn't contain any other functional symbol and constant except  $a$ , then the Herbrand universe is  $\{a\}$  and the Herbrand base

is  $\{p(a)\}$ . Easy to see, that both possible Herbrand interpretations -  $\emptyset$  and  $\{p(a)\}$  are not models of the formula.

However if we add another constant, say  $b$ , to the language, the Herbrand interpretation  $\{p(a)\}$  would be model of the formula.

This leads us to the following definition.

**Definition 2.2 (Satisfiability in language).** Consider a first order language  $L$  and a formula  $F$  in it.  $F$  is called satisfiable in  $L$  or  $L$ -satisfiable if there is a Herbrand interpretation of  $L$  which is a model of  $F$ .

Surely, if a formula is satisfiable in some language, then it is satisfiable. The converse, as the previous example shows, is not true. The following theorem clarifies the situation

**Theorem 2.3.** Consider a first order language  $L_0$  and a formula  $F$  in it. Denote the language  $L$  as acquired from  $L_0$  by adding a new constant  $c$  and a new unary functional symbol  $f$ . If  $F$  is satisfiable, then  $F$  is  $L$ -satisfiable.

*Proof.* Suppose  $F$  is satisfiable. By Löwenheim-Skolem theorem  $F$  has a countable model. Denote that model by  $\mathcal{I}_0$ . Denote the domain of  $\mathcal{I}_0$  by  $D$  and let  $D = \{d_0, d_1, \dots\}$ . First let us extend  $\mathcal{I}_0$  to the interpretation  $\mathcal{I}$  of  $L$  by mapping the constant  $c$  to  $d_0$  and the functional symbol  $f$  to the function  $f_{\mathcal{I}}$  such that  $f_{\mathcal{I}}(d_i) = d_{i+1}$ . As the formula  $F$  doesn't contain the constant  $c$  and the functional symbol  $f$

$$\mathcal{I} \models F.$$

Consider an arbitrary ground term of  $L$ . The definition of the semantics of a term (with respect to an interpretation) relies on the auxiliary notation of valuation. The semantics of ground terms, however, doesn't depend on valuation. Thus, the interpretation  $\mathcal{I}$  assigns to each ground term an element from  $D$  - its semantics, denoted as  $Val_{\mathcal{I}}(t)$ . The addition of  $c$  and  $f$  and their interpretation make the mapping  $Val_{\mathcal{I}}$  surjective. This is a key point of the proof.

Now define the Herbrand interpretation  $\mathcal{I}'$  of  $L$  as

$$\mathcal{I}' \stackrel{\text{def}}{=} \{A \in B_L \mid \mathcal{I} \models A\}.$$

Denote the set of variables of the language  $L$  as  $X$ . To each valuation  $\varphi': X \rightarrow U_L$  of interpretation  $\mathcal{I}'$  we associate the valuation  $\varphi: X \rightarrow D$  of interpretation  $\mathcal{I}$  given as

$$\varphi(x) \stackrel{\text{def}}{=} Val_{\mathcal{I}}(\varphi'(x)).$$

Now, as we have interpretations  $\mathcal{I}$  and  $\mathcal{I}'$  and valuations  $\varphi$  and  $\varphi'$ , we may consider the semantics of each term of  $L$  denoted as  $Val_\varphi$  and  $Val_{\varphi'}$  respectively. We prove that the semantics are somewhat identical. More precisely, for each term  $t$  of  $L$  and each valuation  $\varphi'$  of  $\mathcal{I}'$

$$Val_{\mathcal{I}}(Val_{\varphi'}(t)) = Val_\varphi(t).$$

By induction

- if  $t$  is a constant  $a$ , then  $Val_{\mathcal{I}}(Val_{\varphi'}(a)) = Val_{\mathcal{I}}(a) = a_{\mathcal{I}} = Val_\varphi(a)$ ;
- if  $t$  is a variable  $x$ , then  $Val_{\mathcal{I}}(Val_{\varphi'}(x)) = Val_{\mathcal{I}}(\varphi'(x)) = \varphi(x) = Val_\varphi(x)$
- if  $t$  is of the form  $f(t_1, \dots, t_n)$ , then  $Val_{\mathcal{I}}(Val_{\varphi'}(t)) = Val_{\mathcal{I}}(Val_{\varphi'}(f(t_1, \dots, t_n))) = Val_{\mathcal{I}}(f(Val_{\varphi'}(t_1), \dots, Val_{\varphi'}(t_n))) = f_{\mathcal{I}}(Val_{\mathcal{I}}(Val_{\varphi'}(t_1)), \dots, Val_{\mathcal{I}}(Val_{\varphi'}(t_n))) = f_{\mathcal{I}}(Val_\varphi(t_1), \dots, Val_\varphi(t_n)) = Val_\varphi(t)$

Now we are ready to prove the main result. For each formula  $A$  of  $L$  and for each valuation  $\varphi'$  of  $\mathcal{I}'$

$$\mathcal{I}' \models_{\varphi'} A \iff \mathcal{I} \models_\varphi A.$$

By induction

- $\mathcal{I}' \models_{\varphi'} p(t_1, \dots, t_n)$  iff  
 $\mathcal{I}' \models p(Val_{\varphi'}(t_1), \dots, Val_{\varphi'}(t_n))$  iff  
 $\mathcal{I} \models p(Val_{\varphi'}(t_1), \dots, Val_{\varphi'}(t_n))$  iff  
 $\langle Val_{\mathcal{I}}(Val_{\varphi'}(t_1)), \dots, Val_{\mathcal{I}}(Val_{\varphi'}(t_n)) \rangle \in p_{\mathcal{I}}$  iff  
 $\langle Val_\varphi(t_1), \dots, Val_\varphi(t_n) \rangle \in p_{\mathcal{I}}$  iff  
 $\mathcal{I} \models_\varphi p(t_1, \dots, t_n)$ .
- $\mathcal{I}' \models_{\varphi'} \neg B$  iff  
 $\mathcal{I}' \not\models_{\varphi'} B$  iff  
 $\mathcal{I} \not\models_\varphi B$  iff  
 $\mathcal{I} \models_\varphi \neg B$ .
- $\mathcal{I}' \models_{\varphi'} B \wedge C$  iff  
 $\mathcal{I}' \models_{\varphi'} B$  and  $\mathcal{I}' \models_{\varphi'} C$  iff  
 $\mathcal{I} \models_\varphi B$  and  $\mathcal{I} \models_\varphi C$  iff  
 $\mathcal{I} \models_\varphi B \wedge C$ .

- $B \vee C, B \supset C, B \sim C$  are considered analogically.
- $\mathcal{I}' \models_{\varphi'} \forall x B$  iff
  - $\mathcal{I}' \models_{\varphi'[x \rightarrow t]} B$  for every  $t \in U_L$  iff
  - $\mathcal{I} \models_{\varphi[x \rightarrow \text{Val}_{\mathcal{I}}(t)]} B$  for every  $t \in U_L$  iff
  - $\mathcal{I} \models_{\varphi[x \rightarrow d_i]} B$  for every  $d_i \in D$  iff
  - $\mathcal{I} \models_{\varphi} \forall x B$ .
- $\mathcal{I}' \models_{\varphi'} \exists x B$  iff
  - $\mathcal{I}' \models_{\varphi'[x \rightarrow t]} B$  for some  $t \in U_L$  iff
  - $\mathcal{I} \models_{\varphi[x \rightarrow \text{Val}_{\mathcal{I}}(t)]} B$  for some  $t \in U_L$  iff
  - $\mathcal{I} \models_{\varphi[x \rightarrow d_i]} B$  for some  $d_i \in D$  iff
  - $\mathcal{I} \models_{\varphi} \exists x B$ .

In last two items we essentially use the fact that  $\text{Val}_{\mathcal{I}}$  is surjective.

Particularly if  $A$  is  $F$  (the valuation is of no importance here) we obtain the proof of the theorem. □

Some notes on the importance of the theorem. First of all, this means that  $L$ -satisfiability is not decidable in general, as otherwise we would obtain a decision procedure for satisfiability in contradiction with Church's theorem. Next, the same theorem would also hold if we add to the initial language some other other symbols - particularly infinite number of constants. The advantage of the presented theorem is that the number of symbols of  $L$  will still be finite if it was finite for  $L_0$ .

## 2.2 Monadic language with a single constant

Consider a monadic first order language  $L$  with finite number of constants, functional and predicate symbols. We are going to define a monadic language  $L'$  with a single constant and reduce the problem of  $L$ -satisfiability to the problem of  $L'$ -satisfiability.

In this section we will deal with Herbrand interpretations only. Therefore, if it is not stated explicitly the term interpretation refers to Herbrand interpretation. As the domain of each Herbrand interpretation is fixed (it is the Herbrand universe of the language), the notation of valuation no more depends on interpretation, instead it depends only on language. We will use the term valuation with respect to a language (not interpretation).

Let  $c_1, \dots, c_k$  be the constants,  $f_1, \dots, f_n$  be the unary functional symbols and  $p_1, \dots, p_m$  - the unary predicates of  $L$ .

Let us define the language  $L'$ . First order language is defined with its variables, constants, functional and predicate symbols:

- *variables* of  $L'$  are precisely those of  $L$ ;
- the single *constant* of  $L'$  is denoted by  $a$ ;
- *functional symbols* of  $L'$  are functional symbols of  $L$  plus a new unary functional symbols  $f_c$  for each constant  $c$  of  $L$ ;
- *predicate symbols* of  $L'$  are those of  $L$  with an additional predicate symbol, denoted by  $q$ .

To each term  $t$  of  $L$  we will associate a term  $t'$  of  $L'$  defined as:

- if  $t$  is a constant  $c$ , then  $t' \stackrel{\text{def}}{=} f_c(a)$ .
- if  $t$  is a variable  $x$ , then  $t' \stackrel{\text{def}}{=} x$ .
- if  $t$  is of form  $f(s)$  then  $t' \stackrel{\text{def}}{=} f(s')$ .

Clearly the term  $t'$  would be ground if and only  $t$  is ground. In such case  $t'$  would be called *normal*. Easy to see, that the term  $t'$  would be normal, if and only if it is of the form  $f_{i_1}(\dots f_{i_i}(f_{c_j}(a))\dots)$ . In such case  $t$  is of the form  $f_{i_1}(\dots f_{i_i}(c_j)\dots)$  and is uniquely determined by  $t'$ .

Consider the following formula  $Q$  of  $L'$ :

$$(\neg q(a)) \bigwedge_{i,j} \forall x (\neg q(f_{c_i}(f_j(x)))) \bigwedge_i q(f_{c_i}(a)) \bigwedge_j \forall x (q(f_j(x)) \sim q(x)).$$

Let us study the relation “defined” by the predicate symbol  $q$  in the formula  $Q$ . Consider an arbitrary Herbrand model of  $Q$ . According to the construction of  $Q$  for every functional symbol  $f$  of  $L$  the values of  $q(f(x))$  and  $q(x)$  are the same. Hence, to obtain the value of  $q(t)$  for any ground  $t$  we may remove all functional symbols of  $L$  from the beginning of  $t$  until we meet a newly added functional symbol of  $L'$  or  $a$ . In the former case the truth value of  $q(t)$  depends on weather the remaining term is of the form  $f_c(a)$  for some constant  $c$  of  $L$ , in the later case it is false. Thus, in every Herbrand model of  $Q$ ,  $q(t)$  is true if and only if  $t$  has the form  $f_{i_1}(\dots f_{i_i}(f_{c_j}(a))\dots)$ . Such terms were called normal. Herbrand interpretations of  $L'$  in which  $q(t')$  is true if and only if  $t'$  is normal are called *normal* interpretations. From the above

discussion it is clear that Herbrand models of  $Q$  are precisely the normal interpretations. Valuation  $\varphi'$  of  $L'$  is also called *normal* if  $\varphi'(x)$  is normal for every  $x$ .

To each formula  $A$  of  $L$  we will associate a formula  $A'$  of  $L'$  defined as:

- $(p(t))' \stackrel{\text{def}}{=} p(t')$
- $(\neg F)' \stackrel{\text{def}}{=} \neg F'$
- $(F \vee G)' \stackrel{\text{def}}{=} F' \vee G'$
- $(F \wedge G)' \stackrel{\text{def}}{=} F' \wedge G'$
- $(F \supset G)' \stackrel{\text{def}}{=} F' \supset G'$
- $(F \sim G)' \stackrel{\text{def}}{=} F' \sim G'$
- $(\forall x F)' \stackrel{\text{def}}{=} \forall x(q(x) \supset F')$
- $(\exists x F)' \stackrel{\text{def}}{=} \exists x(q(x) \wedge F')$

To each normal Herbrand interpretation  $\mathcal{I}'$  of  $L'$  we will associate a Herbrand interpretation  $\mathcal{I}$  of  $L$  defined as:

$$\mathcal{I} \stackrel{\text{def}}{=} \{p(t) \mid p(t') \in \mathcal{I}'\}.$$

Easy to see that the mapping from  $\mathcal{I}'$  to  $\mathcal{I}$  is surjective. That is for every interpretation  $\mathcal{I}$  of  $L$  there is a normal interpretation  $\mathcal{I}'$  of  $L'$  to which  $\mathcal{I}$  is associated. However, the interpretation  $\mathcal{I}'$  is not unique for given  $\mathcal{I}$ .

To each valuation  $\varphi$  of  $L$  we will associate a normal valuation  $\varphi'$  of  $L'$  defined as:

$$\varphi'(x) \stackrel{\text{def}}{=} (\varphi(x))'.$$

Clearly, the mapping from valuations of  $L$  to normal valuations of  $L'$  is a one to one mapping.

Note, that we have defined the term  $t'$ , formula  $F'$  and normal valuation  $\varphi'$  of the language  $L'$  from the term  $t$ , formula  $F$  and valuation  $\varphi$  of  $L$  respectively. On the other hand the interpretation  $\mathcal{I}$  of  $L$  was defined with respect to the normal interpretation  $\mathcal{I}'$  of  $L'$ . We will be very careful not to confuse the reader with this.

Now, lets turn our attention to semantics of terms with respect to normal valuations.

**Lemma 2.4.** For each valuation  $\varphi$  of  $L$  and each term  $t$  of  $L$

$$Val_{\varphi'}(t') = (Val_{\varphi}(t))'$$

*Proof.* By induction

- if  $t$  is a constant  $c$  then  $Val_{\varphi'}(t') = f_c(a) = (c)' = (Val_{\varphi}(t))'$
- if  $t$  is a variable  $x$  then  $Val_{\varphi'}(t') = x = (Val_{\varphi}(t))'$
- if  $t$  is of the form  $f(s)$  then  $Val_{\varphi'}(t') = Val_{\varphi'}(f(s')) = f(Val_{\varphi'}(s')) = f((Val_{\varphi}(s))') = (f(Val_{\varphi}(s)))' = (Val_{\varphi}(t))'$

□

We will use this result to prove the following lemma about the semantics of formulas.

**Lemma 2.5.** For each normal interpretation  $\mathcal{I}'$  of  $L'$ , each valuation  $\varphi$  and each formula  $A$  of  $L$

$$\mathcal{I} \models_{\varphi} A \iff \mathcal{I}' \models_{\varphi'} A'.$$

*Proof.* By induction

- $\mathcal{I} \models_{\varphi} p(t)$  iff  
 $p(Val_{\varphi}(t)) \in \mathcal{I}$  iff  
 $p((Val_{\varphi}(t))') \in \mathcal{I}'$  iff  
 $p(Val_{\varphi'}(t')) \in \mathcal{I}'$  iff  
 $\mathcal{I}' \models_{\varphi'} p(t')$ .
- $\mathcal{I} \models_{\varphi} \neg F$  iff  
 $\mathcal{I} \not\models_{\varphi} F$  iff  
 $\mathcal{I}' \not\models_{\varphi'} F'$  iff  
 $\mathcal{I}' \models_{\varphi'} \neg F'$ .
- $F \vee G, F \wedge G, F \supset G, F \sim G$  are considered analogically.
- $\mathcal{I} \models_{\varphi} \forall x F$  iff  
 $\mathcal{I} \models_{\varphi[x \rightarrow t]} F$  for every  $t \in U_L$  iff  
 $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} F'$  for every  $t \in U_L$  iff  
 $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} F'$  for every normal  $t' \in U_{L'}$  iff  
 $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} q(x) \supset F'$  for every  $t' \in U_{L'}$  iff  
 $\mathcal{I}' \models_{\varphi'} \forall x (q(x) \supset F')$ .

- $\mathcal{I} \models_{\varphi} \exists x F$  iff
  - $\mathcal{I} \models_{\varphi[x \rightarrow t]} F$  for some  $t \in U_L$  iff
  - $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} F'$  for some  $t' \in U_L$  iff
  - $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} F'$  for some normal  $t' \in U_{L'}$  iff
  - $\mathcal{I}' \models_{\varphi'[x \rightarrow t']} q(x) \wedge F'$  for some  $t' \in U_{L'}$  iff
  - $\mathcal{I}' \models_{\varphi'} \exists x (q(x) \wedge F')$ .

□

Consider a closed formula  $F$  of  $L$ . The previous lemma suggests that  $F$  is  $L$ -satisfiable if and only if  $F'$  has a normal model. On the other hand, we may use the formula  $Q$  of  $L'$  to “filter out” non normal models of  $F'$ . More precisely, the following theorem holds:

**Theorem 2.6.** A closed formula  $F$  is  $L$ -satisfiable if and only if  $F' \wedge Q$  is  $L'$ -satisfiable.

*Proof.* Suppose that  $F$  is  $L$ -satisfiable. Then it has a Herbrand model  $\mathcal{M}$ . As we have already mentioned there is a normal Herbrand interpretation  $\mathcal{M}'$  of  $L'$  to which  $\mathcal{M}$  is associated. By the previous lemma,  $\mathcal{M}'$  is a model of  $F'$ . As  $\mathcal{M}'$  is a normal interpretation, it is a model of  $Q$ . Therefore  $\mathcal{M}'$  is a model of  $F' \wedge Q$ .

For the converse, suppose  $F' \wedge Q$  has a Herbrand model  $\mathcal{M}'$ . As  $\mathcal{M}'$  is a model of  $Q$ , it is a normal interpretation. Thus, there is a Herbrand interpretation  $\mathcal{M}$  associated to  $\mathcal{M}'$ . As  $\mathcal{M}'$  is a model of  $F'$ ,  $\mathcal{M}$  is a model of  $F$ .

□

## 2.3 Decidability of monadic first order logic

In this section we will prove that monadic second order logic is decidable. More specifically we will prove that there is an algorithm for deciding whether given formula in a first order logic is satisfiable. Easy to see that this means that unsatisfiability and validity problems in monadic first order logic are also decidable. Indeed, the formula is unsatisfiable if and only if it is not satisfiable (a little pun here), and valid if and only if its negation is unsatisfiable.

**Theorem 2.7.** There is an algorithm for deciding whether a given formula of the first order logic is satisfiable or not.

*Proof.* Note that we do not even speak about the language the formula is defined in. The reason is that satisfiability of a first order formula doesn't depend on the language it is defined in. Therefore, the instance of the problem is a formula, rather than a pair of formula and language.

Consider an arbitrary first order formula  $F$ . Denote by  $L_0$  the first order language with constants, functional and predicate symbols of  $F$ . Let the set of variables of  $L_0$  be countable and contain the variables used in  $F$ . Then  $F$  is a "well formed" formula of  $L_0$ . According to the first section of this chapter we may add a new constant and a new unary functional symbol to  $L_0$  and obtain a first order language  $L$  such that  $F$  is satisfiable if and only if it is  $L$ -satisfiable. Obviously  $L_0$  contains a finite number of constants, functional and predicate symbols. From the construction of  $L$  it is clear that it also contains a finite number of them. According to the previous section, we may construct a first order language  $L'$  with a single constant and finite number of functional and predicate symbols and a formula  $F'$  such that  $F$  is  $L$ -satisfiable if and only if  $F'$  is  $L'$ -satisfiable. From the combination of two equivalences it is clear that  $F$  is satisfiable if and only if  $F'$  is  $L'$ -satisfiable.

Let  $a$  be the single constant,  $f_1, \dots, f_n$  be the functional and  $p_1, \dots, p_m$  - the predicate symbols of  $L'$ . Every formula of  $L'$  and in particular  $F'$  may be viewed as formula in  $SnS$  with  $a$  denoting the single constant  $\Lambda$ ,  $f_1, \dots, f_n$  denoting the successor functions and  $p_1, \dots, p_m$  the second order variables (the first order variables are the same). From the semantics of formulas in the first order logic and in second order logic of successor functions it is clear that  $F'$  is  $L'$ -satisfiable if and only if the formula

$$\exists p_1 \dots \exists p_m F'$$

is true in  $SnS$ . Therefore, the problem of satisfiability in monadic first order logic is decidable.

□

# Chapter 3

## Equivalence of logic programs

In this chapter we define and study two notations of equivalence of logic programs.

### 3.1 Equivalence as logic formulas

As we have defined it, a logic program is a non-empty finite set of definite closes. Consider an arbitrary logic program. The program is uniquely identified with the formula that is the conjunction of the closes of the program. Moreover, the set of closed formulas and their conjunction are completely identical from model-theoretic point of view. That is, every model of a set of closed formulas is also a model of their conjunction and vice versa, every formula that is a logical consequence of a set of closed formulas is also a logical consequence of their conjunction and vice versa, etc. Therefore, in this chapter logic programs would be viewed as conjunctions of definite closes. Whenever a distinction arises in a set of closes and their conjunction, we will explicitly state what we mean by term logic program.

**Definition 3.1 (Equivalence as logic formulas).** Two logic programs  $P_1$  and  $P_2$  are said to be equivalent as logic formulas if the formula  $P_1 \sim P_2$  is valid.

Easy to see, that two program would be equivalent as logic formula if and only if every interpretation of an underlying first order language is either a model of both or not a model of any. In this section we only study the equivalence of programs as logic formulas, therefore, the term equivalence as logic formulas is abbreviated to equivalence.

We see, that the problem of equivalence of program is reduced to the problem of validity in the first order logic. The later, however, is not decidable

(Church's theorem). Nevertheless, this doesn't yet mean that the problem of equivalence of programs as logic formulas is not decidable.

To show that the problem of equivalence is undecidable, we will show that from the decision procedure of equivalence problem, we would obtain a decidable interpreter (an algorithm for deciding whether a given goal is a logical consequence of a given program).

**Theorem 3.2.** The problem of whether given two programs are equivalent as logic formulas is not decidable.

*Proof.* Consider a program  $P$  and a goal  $G$ :

$$\leftarrow A_1, \dots, A_n \quad (n > 0).$$

Now consider two programs

$$P'_1 \stackrel{\text{def}}{=} P \cup \{p(a) \leftarrow A_1, \dots, A_n.\}$$

and

$$P'_2 \stackrel{\text{def}}{=} P \cup \{p(a).\}$$

where  $p$  is a new predicate symbol and  $a$  is a new constant.

Easy to check that every model of  $P'_2$  is a model of  $P'_1$ .

Therefore,  $P'_1 \sim P'_2 \iff$  every model of  $P'_1$  is a model of  $P'_2 \iff p(a)$  is true in every model of  $P'_1 \iff \exists(A_1 \wedge \dots \wedge A_n)$  is true in every model of  $P \iff \exists(A_1 \wedge \dots \wedge A_n)$  is a logical consequence of  $P$ .

Thus, from the decision procedure of equivalence problem we would obtain a procedure for deciding whether a given goal is a logical consequence of a given program. However, a well known result is that such procedure doesn't exist (The Post correspondence problem may be "programmed" as a logic program).

□

Now let's turn our attention to monadic logic programs - programs which do not use functional and predicate symbols with arity greater than one. The problem of equivalence of such programs as logic formulas is by definition reduced to the problem of validity in monadic first order language. This problem, on the other hand, is decidable according to the previous chapter. Thus, we obtain the following theorem.

**Theorem 3.3.** The problem of whether given two monadic logic programs are equivalent as logic formulas is decidable.

## 3.2 $\Delta$ -equivalence

The equivalence of programs as logic formulas is the most straightforward, but not the only possible way of defining the notation of equivalence. In a number of cases, while transforming a logical program into another, we are not interested in the programs being equivalent as logic formulas, but rather interested in equality of the sets of goals which are logical consequences of programs. That is, to consider programs equivalent if their least Herbrand models are equal.

**Definition 3.4 ( $\Delta$ -equivalence).** Consider a first order language  $L$  and logic programs  $P_1$  and  $P_2$  in it.  $P_1$  and  $P_2$  are said to be  $\Delta$ -equivalent (denoted as  $P_1 \overset{\Delta}{\sim} P_2$ ) if  $\mathcal{M}_{P_1} = \mathcal{M}_{P_2}$ , where  $\mathcal{M}_P$  denotes the least Herbrand model of the program  $P$ .

Clearly, if two programs are equivalent as logic formulas then they are  $\Delta$ -equivalent. The following example shows that the converse is not true.

**Example 3.5.** Consider a language with unary predicates  $p, q, r$  and a constant  $a$ . Consider a program  $P_1$ :

$$p(x)$$

and a program  $P_2$ :

$$\begin{aligned} p(x) \\ q(x) \leftarrow r(x). \end{aligned}$$

Easy to see, that  $\mathcal{M}_{P_1} = \mathcal{M}_{P_2} = \{p(a)\}$ . Therefore,  $P_1$  and  $P_2$  are  $\Delta$ -equivalent. Nevertheless  $P_1$  and  $P_2$  are not equivalent as logic formulas.

The following example shows that unlike equivalence as logic formula,  $\Delta$ -equivalence depends on underlying first order language.

**Example 3.6.** Consider the following logic programs  $P_1$ :

$$p(x)$$

and  $P_2$ :

$$\begin{aligned} p(a) \\ p(f(x)) \leftarrow p(x). \end{aligned}$$

If the underlying language doesn't contain any other symbol but  $p, f$  and  $a$ , programs  $P_1$  and  $P_2$  are  $\Delta$ -equivalent. However, if we add a constant  $b$  to the language, then  $p(b)$  would be a logical consequence of  $P_1$  but not  $P_2$ . Therefore,  $P_1$  and  $P_2$  will no longer be  $\Delta$ -equivalent.

The relation of  $\Delta$ -equivalence was first defined and studied in [NK97]. The following theorem is taken from that study.

**Theorem 3.7.** The problem of whether two given programs are  $\Delta$ -equivalent in the given language is not decidable.

Here we study the relation of  $\Delta$ -equivalence for monadic programs. We will consider only languages with finite number of constants, functional and predicate symbols. Easy to see that the case of language with infinite number of constants, functional or predicate symbols is reducible to it. Indeed, let the language  $L$  contain infinite number of constants  $c_1, c_2, \dots$ . Consider programs  $P_1$  and  $P_2$ . Let  $c_1, c_2, \dots, c_n$  be the constants used in  $P_1$  and  $P_2$ . From the symmetry, we may state that an atom  $A$  is a logical consequence of  $P_i$  if and only if the atom  $A'$  is a logical consequence of  $P_i$ , where  $A'$  is obtained from  $A$  by substituting every constant  $c_j$  where  $j > n$  with  $c_{n+1}$ . Therefore  $P_1$  and  $P_2$  are  $\Delta$ -equivalent in  $L$ , if and only if they are  $\Delta$ -equivalent in the language  $L'$  acquired from  $L$  by dropping constants  $c_{n+2}, c_{n+3}, \dots$ . The case of infinite functional and predicate symbols is handled similarly.

Consider a monadic language  $L$  with finite number of constants, functional and predicate symbols and a program in it. A program is said to be in *canonical* form, if every variable in the head of every clause (in case of monadic program there may be at most one variable in the head) also occurs in the body of that clause. We prove that every program has a  $\Delta$ -equivalent canonical form.

**Proposition 3.8.** For every program  $P$  in  $L$  there is a program  $P'$  such that  $P'$  is in canonical form and  $P \stackrel{\Delta}{\approx} P'$ .

*Proof.* Suppose  $P$  contains a clause  $A_0 \leftarrow A_1, \dots, A_n$ , where the variable  $x$  occurs in  $A_0$  but not in  $A_1, \dots, A_n$ . The atom  $A$  should be of the form  $p(f_1(\dots f_m(x)\dots))$ . We remove the clause from  $P$  and add the following ones: for every constant  $c$  of  $L$

$$p(f_1(\dots f_m(c)\dots)) \leftarrow A_1, \dots, A_n$$

for every functional symbol  $f$  of  $L$

$$p(f_1(\dots f_m(f(x))\dots)) \leftarrow p(f_1(\dots f_m(x)\dots)), A_1, \dots, A_n.$$

Easy to prove (using fixpoint semantics -  $T_P \uparrow \omega = \mathcal{M}_P$ ) that the new program will be  $\Delta$ -equivalent to the old one. On the other hand, the number of clauses that violates the condition of canonical program is reduced by one. Therefore, by subsequent substitutions we will obtain a program  $P'$ , which is, by transitivity of  $\Delta$ -equivalence relation,  $\Delta$ -equivalent to  $P$ . □

Similarly to the previous chapter, we are going to define a monadic language  $L'$  with a single constant and reduce the problem of  $\Delta$ -equivalence of canonical programs in  $L$  to the problem of  $\Delta$ -equivalence in  $L'$ .

The language  $L'$  is defined as in the previous chapter:

- *variables* of  $L'$  are precisely those of  $L$ ;
- the single *constant* of  $L'$  is denoted by  $a$ ;
- *functional symbols* of  $L'$  are functional symbols of  $L$  plus a new unary functional symbols  $f_c$  for each constant  $c$  of  $L$ ;
- *predicate symbols* of  $L'$  are precisely those of  $L$ .

Let us remind that a ground term  $t'$  of  $L$  of the form  $f_1(\dots f_n(f_c(t))\dots)$  is called normal and the term  $f_1(\dots f_n(c)\dots)$  of  $L$  is the one associated with the normal term.

The program  $P'$  of  $L'$  is constructed from the program  $P$  of  $L$  by replacing every occurrence of a constant  $c$  by the term  $f_c(a)$ .

**Lemma 3.9.** Consider a canonical program  $P$  of  $L$ . A ground term  $p(t')$  is a logical consequence of the program  $P'$  if and only if  $t'$  is normal and  $p(t)$  is a logical consequence of  $P$ .

*Proof.* First let us prove that if  $p(t')$  is a logical consequence of  $P'$ , then  $t'$  is normal and  $p(t)$  is a logical consequence of  $P$ . From the fixpoint semantics we have  $p(t') \in T_{P'} \uparrow n$  for some natural number  $n$ . Let us prove the proposition by induction on  $n$ .

Clearly the proposition holds in case of  $n = 0$ , ( $T_{P'} \uparrow 0 = \emptyset$ ).

Now suppose the proposition holds fore every  $n < k$  and let us prove it for  $n = k$ . Suppose  $p(t') \in T_{P'} \uparrow k$ . Then there is a clause

$$A'_0 \leftarrow A'_1, \dots, A'_m$$

in  $P'$  and a ground substitution  $\theta' = \{x_1/t'_1, \dots, x_r/t'_r\}$  of its variables such that  $A'_i\theta' \in T_{P'} \uparrow k - 1$  for  $i = 1, 2, \dots, m$  and  $A'_0\theta' = p(t')$ . By the induction assumption, for  $i = 1, 2, \dots, m$  the atom  $A_i\theta$  is of the form  $q_i(s'_i)$ , where  $s'_i$  is ground and  $q_i(s_i)$  is a logical consequence of  $P$ . As  $P'$  is in canonical form, all variables of each close occur in its body. Therefore, all  $t'_j$  are normal and hence  $t'$  is also normal. Let the clause

$$A_0 \leftarrow A_1, \dots, A_m$$

be the clause of  $P$  from which  $A'_0 \leftarrow A'_1, \dots, A'_m$  was derived and let  $\theta = \{x_1/t_1, \dots, x_r/t_r\}$ . Obviously  $A_i\theta = q_i(s_i)$  for  $i = 1, 2, \dots, m$ . As all  $A_i\theta$  are logical consequences of the program  $P$ ,  $A_0\theta = p(t)$  is also a logical consequence of  $P$ .

The proof of the converse is very similar. Let us prove by induction that if  $p(t) \in T_P \uparrow n$ , then  $p(t')$  is a logical consequence of  $P'$ . The proposition is obvious for  $n = 0$ . Suppose it holds of every  $n < k$ . If  $p(t) \in T_P \uparrow k$ , then there is clause

$$A_0 \leftarrow A_1, \dots, A_m$$

in  $P$  and a ground substitution  $\theta = \{x_1/t_1, \dots, x_r/t_r\}$ , such that for every  $i = 1, 2, \dots, m$  the atom  $A_i\theta \in T_P \uparrow k - 1$ . Therefore, for the derived clause

$$A'_0 \leftarrow A'_1, \dots, A'_m$$

of  $P'$  and substitution  $\theta' = \{x_1/t'_1, \dots, x_r/t'_r\}$ , atoms  $A'_i\theta'$  are logical consequences of  $P'$  and hence  $p(t') = A'_0\theta'$  is also a logical consequence of  $P'$ .  $\square$

A direct consequence of this lemma is the following

**Theorem 3.10.** Consider a monadic language  $L$  with finite number of constants, functional and predicate symbols and two canonical programs  $P_1$  and  $P_2$  in it. Let the language  $L'$  and programs  $P'_1$  and  $P'_2$  be the ones constructed by above presented procedure from  $L$ ,  $P_1$  and  $P_2$  respectively.  $P_1$  and  $P_2$  are  $\Delta$ -equivalent in  $L$  if and only if  $P'_1$  and  $P'_2$  are  $\Delta$ -equivalent in  $L'$ .

The following theorem concludes the study of  $\Delta$ -equivalence of monadic logic programs.

**Theorem 3.11.** The problem of whether given two programs are  $\Delta$ -equivalent in a given monadic first order language is decidable.

*Proof.* Consider a monadic language  $L$  and programs  $P_1$  and  $P_2$  in it. As we have mentioned, we may consider that  $L$  has finite number of constants, functional and predicate symbols (otherwise we may reduce the problem of  $\Delta$ -equivalence to the one with finite language). First of all, we construct canonical forms of the programs  $P'_1$  and  $P'_2$ . Obviously,

$$P_1 \overset{\Delta}{\sim} P_2 \iff P'_1 \overset{\Delta}{\sim} P'_2,$$

where both equivalences are considered with respect to  $L$ . Next, according to the previous theorem, we construct a monadic language with a single constant  $L'$  and programs  $P''_1$  and  $P''_2$  such that

$$P'_1 \overset{\Delta}{\sim} P'_2 \iff P''_1 \overset{\Delta}{\sim} P''_2,$$

where the first equivalence is considered with respect to  $L$  and the second one - with respect to  $L'$ .

Let  $a$  be the single constant,  $f_1, \dots, f_n$  be the functional and  $p_1, \dots, p_m$  - the predicate symbols of  $L'$ . The programs  $P_1''$  and  $P_2''$  would not be  $\Delta$ -equivalent in  $L'$  if and only if for some predicate symbol  $p_i$  and some ground term  $t$ , the atom  $p_i(t)$  is a logic consequence of  $P_1''$  but not a consequence of  $P_2''$  or vice versa. From the definition of semantics of  $SnS$  it is clear that the former case takes place if and only if the closed formula

$$\exists t(\forall p_1 \dots \forall p_m (P_1'' \supset p_i(t)) \wedge \exists p_1 \dots \exists p_m (P_2'' \wedge \neg p_i(t)))$$

of  $SnS$  is true and the later case takes place if and only if the closed formula

$$\exists t(\forall p_1 \dots \forall p_m (P_2'' \supset p_i(t)) \wedge \exists p_1 \dots \exists p_m (P_1'' \wedge \neg p_i(t)))$$

of  $SnS$  is true.

Therefore, all we need to do to know whether  $P_1''$  and  $P_2''$  are  $\Delta$ -equivalent in  $L'$  is to check the semantics of  $2m$  formulas of  $SnS$  for  $m$  values of  $i$ .

□

# Bibliography

- [GTW02] E. Grädel, W. Thomas, and Thomas Wilke. *Automata, Logics, and Infinite Games*. Number 2500 in Lecture Notes in Computer Science. Springer-Verlag, 2002.
- [Llo84] John W. Lloyd. *Foundations of Logic Programming*. Springer, 1984.
- [NK97] S. A. Nigiyani and L. O. Khachoyan. Transformations of logic programs. *Programming and Computer Software*, 23(6):302–309, 1997.
- [NM95] Ulf Nilsson and Jan Maluszynski. *Logic, Programming, and PROLOG*. John Wiley & Sons, Inc., New York, NY, USA, 1995.
- [Rab69] Michael O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.