Interner Bericht

An Intuitionistic Approach to Logic Programming

Rodrigo Readi-Nasser

Internal Report 296/98, 03.03.1998,
Arbeitsgruppe Professor Dr. Otto Mayer:
Grundlagen der Programmierung
FB Informatik, Universität Kaiserslautern

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-67653 Kaiserslautern

An Intuitionistic Approach to Logic Programming

Rodrigo Readi-Nasser *

Internal Report 296/98, 03.03.1998,
Arbeitsgruppe Professor Dr. Otto Mayer:
Grundlagen der Programmierung
FB Informatik, Universität Kaiserslautern

Abstract

A natural extension of SLD-resolution is introduced as a goal directed proof procedure for the full first order implicational fragment of intuitionistic logic. Its intuitionistic semantic fits a procedural interpretation of logic programming. By allowing arbitrary nested implications it can be used for implementing modularity in logic programs. With adequate negation axioms it gives an alternative to negation as failure and leads to a proof procedure for full first order predicate logic.

1 Introduction

The programming language Prolog originated from the work of Colmerauer [4] and his colleagues at Marseille and Kowalski [10] at Edinburgh in the early 70's. It is based on the idea of using *logic for problem solving*, or more accurate, of expressing computation as the *controlled search* for a derivation of an instance of a given conjunction of atomic goals from a given list of *declarative horn clauses* with *SLD-resolution*. The first Prolog interpreter was written by Philippe Roussel, followed by the Prolog interpreter written in Fortran by Gérard Battani, H. Meloni and R. Bazzoli [3]. D. Warren together with F. Pereira and L. Pereira developed the DEC-10 Prolog system at Edinburgh, the first Prolog compiler to a low level language; its principles were presented in a refined and abstracted form [20] in 1983, providing a standard for implementing efficient Prolog compilers. With their work, the concept of the language as we know it today, the idea of how to successfully use it for symbolic computation and schemata for its efficient implementation were developed until their ripeness.

Many extensions of logic programming were studied. Negative goals cannot be deduced with SLD-resolution. The failure of a search for a derivation of an atom A was then considered to be a derivation of $\neg A$, negative literals in the bodies of the program clauses were accepted, and so the rule of *negation as failure* arose, this rule extends SLD-resolution to SLDNF-resolution and was first studied in detail by Clark [5]. J. W. Lloyd goes further writing just in the preface of his book [13] that "once a single negation [in the bodies] is allowed, one should go all the way and allow arbitrary formulas": remaining in the framework of SLDNF-resolution, he found a way of transforming a program having arbitrary formulas in the bodies of its clauses into an "equivalent" program having only

*Comments to the author are welcome to his address: Fachbereich Informatik,Postfach 3049,67653 Kaiserslautern, Germany. Email: readi@informatik.uni-kl.de

literals, positive or negative, in the bodies. The logical problem with negation as failure is simple: *unprovability in a formal system isn't equivalent to provability of the negation, as also provability isn't equivalent to provability in the formal system*, further, *the rule of negation as failure isn't representable as a typical deduction rule of formal logic with premises and a conclusion.*

There is research on extending Prolog to broader classes of logic. For example, D. Reed and W. Loveland present and compare in [17] three extensions of Prolog: Loveland's near-Horn Prolog, Plaisted's simplified problem reduction format and Gabbay's and Reyle's N-Prolog. The first two ones *extend Prolog to the full first-order classical logic*, N-Prolog is an intuitionistic system that allows *arbitrary nested hypothetical implications.* Gabbay's and Reyle's N-Prolog is a *natural proof procedure* in the case of propositional logic, but for dealing with universal quantifiers they develop a "theory of Skolem functions" in [6]. L. Hallnäs and P. Schroeder-Heister [7] interpret horn clauses as rules of a formal system for the derivation of atomic formulas, SLD-resolution is extended by considering *higher level rules*, and hence also allow nested hypothetical implications. In [7] the basic properties of the language GCLA [2] are treated, this language was developed and implemented by M. Aronsson, L. H. Ericksson, L. Hallnäs, and P. Kreuger in Stockholm, it has a *second control level* that allows search strategies and search behaviour to be expressed. L. Th. McCarty [19] extended horn clauses by allowing nested hypothetical implications of level one and *interpreted them intuitionistically.* D. Miller explores in [14] the *use of hypothetical implication for implementing modules in logic programming.* A *uniform proof* [15] is one in which the principal connective of the formula is introduced in the last step, the existence of such a proof allow a *goal directed* search for a proof; due to this, D. Miller, G. Nadathur, F. Pfenning and A. Scedrov [15] presented *uniform proofs as a basis for logic programming.* J. Harland [8] studies the problem of restricting the class of formulas in order that every provable formula has a uniform proof, his results show that hereditary Harrop formulas may be seen as the *largest fragment* satisfying this property in the framework of intuitionistic logic.

We present in this article a simple and natural extension of SLD-resolution to be called **mj-resolution**. It is based on our calculus mj introduced in [16]. It is a correct and complete proof procedure for the universal-implicational fragment of intuitionistic logic, and hence it provides SLD-resolution with intuitionistic semantic. It can deal with hypothetical implications. It is related with N-Prolog, but it deals with universal quantification without leaving the framework of typical deduction systems for formal logic. It can also deal with classical negation in a more natural way than negation as failure, and hence one can paraphrase all usual logical symbols: this yields a complete proof procedure for classical logic.

As an extension of SLD-resolution and as a proof procedure, mj-resolution could be used to extend Prolog to a *logic programming and theorem proving system.* Logic programming demands a *simple control* because the programmer's intention is expressed by the formulas of his program, theorem proving demands a *flexible control* because the formulas involved are given by the problem to be solved. The control should be developed considering the possible programming strategies for solving problem classes. The efficient implementation using techniques developed for Prolog should be considered. We plan to develop this system to be called mj-Prolog.

2 Classical Model Theory vs. Intuitionistic Formal Logic

A. Heyting states a predicate calculus for intuitionistic mathematics which deals with the same formulas as the classical calculi. Although the meaning of the implication $A \rightarrow B$ differs in intuitionistic and in classical logic, in both cases one can inaccurately say that it means that B follows from A. The intuitionistic meaning is that from any given proof of A one can build a proof of B. A. Kolmogorov creates a *calculus of problem solving*, or "Aufgabenrechnung" in his words, that formally coincides with the propositional segment of Heyting's intuitionistic logic. Kolmogorov interprets formulas as problems to be solved, for him $A \rightarrow B$ is the problem of reducing the solution of B to the solution of A, or the problem of finding a solution of B under the supposition that a solution of A is given. The classical meaning of $A \rightarrow B$ may be found in the writings of the great Polish logician A. Tarski. We can assert that *the intuitionistic interpretation of logic fits a procedural interpretation of logic programming.*

Ordinary Prolog deals with horn clauses A of the form $A_n \wedge \cdots \wedge A_1 \rightarrow A_0$, where the A_i are atomic formulas possibly containing free variables. In case $n = 0$ we have $A = A_0$. This formula, as a *procedure definition* in a logic program, means that a solution for all the problems $A_n,\ldots,A_1$ leads to a solution of the problem A_0, or that the solution of A_0 can be reduced to the solution of all $A_n,\ldots,A_1$. By induction it follows that this clause is equivalent with the formula $(A_n \rightarrow (A_{n-1} \rightarrow (\cdots \rightarrow (A_1 \rightarrow A_0)\cdots)))$ consisting of nested implications of atoms. Free variables in these procedure definitions are interpreted as universally quantified. *Implication and universal quantifying are sufficient for the purposes of ordinary Prolog.*

This natural equivalence of $A_n \wedge \cdots \wedge A_1 \rightarrow A_0$ to nested implications of atoms holds in both, classical and intuitionistic logic, while its equivalence to $\neg A_n \vee \cdots \vee \neg A_1 \vee A_0$ holds only in classical logic. This is another reason for preferring the implication and the universal quantifier as *primitive connectives* for expressing program clauses. Nevertheless, SLD-resolution takes the universal quantifier, the disjunction and the negation as primitive connectives. The reason of this can be found in the origins of SLD-resolution, introduced by Kowalski [10] and called so in Apt [1]. SLD-Resolution is the restriction of linear resolution to horn clauses considered as disjunctions. Linear resolution, independently proposed by Loveland [11] and [12], is a skillful way of searching refutations with J. A. Robinson's *refutation procedure* [18] for clauses, namely, for universal quantified disjunctions of atomic formulas or negations of atomic formulas.

In classical logic, a formula ξ is derivable from a set Σ if and only if $\Sigma \cup \{\neg\xi\}$ has no model, and every set of formulas can be transformed to a set of clauses, so that no model satisfies the former if and only if no model satisfies the latter. Robinson proves in an elegant way that no model satisfies a set of clauses if and only if a contradiction may be found with his resolution. This is the basis of theorem proving with Robinson's resolution. These arguments hold only in classical logic and are in the framework of *model theory*: *for intuitionistic logic, we work in the framework of proof theory.* It is interesting to note that Robinson's resolution is correct for intuitionistic logic (although not for Johansson's minimal logic). Our real problem with resolution is that *implication cannot be expressed with clauses in intuitionistic logic.*

3 Variables, Unknowns and Arbitrary Constants

Following the considerations in the above section, we allow in our formulas only the universal quantifier "$\forall$" and the implication "$\rightarrow$" as logical symbols. Hence, the horn clause $A_n \wedge \cdots \wedge A_1 \rightarrow A_0$ is considered here to be an abbreviation of $(A_n \rightarrow (A_{n-1} \rightarrow (\cdots \rightarrow (A_1 \rightarrow A_0)\cdots)))$.

There is a big difference in the meaning of the symbol x in the equality $(x+1)^2 = x^2+2x+1$ and in the equation $x^2 = 2$. In the first, x is a **variable**, the equality holds *for all* possible values of x. In the second the x is an **unknown** to be determined by solving the equation. It is convenient to avoid possible confusions by taking different kinds of symbols for unknowns and for variables. Solving the unknown x in the equation $x^2 = a^2$ yields $x = a$ and $x = -a$, this a in the equation neither represent an unknown nor a variable, it represents an arbitrary number, but fixed, such an a is called an **arbitrary constant**. In our formal language, we have, in addition to our symbols for constants, functions and relations, an infinite set of **symbols for variables** v_i, an infinite set of **symbols for unknowns** x_i, and an infinite set of **symbols for arbitrary constants** q_i. Our **terms** don't contain variables, our **formulas** don't contain variables not bound by a quantifier, but **open terms** and **open formulas** may contain such symbols.

The symbols called "variables" in Prolog are *unknowns if they appear in a goal, variables if they appear in the program clauses.* Till now it is custom to use the same kind of symbols for variables and for unknowns in Prolog, but we take here different kinds of symbols. We can **close** all formulas of a program by adding quantifiers in front of them binding their variables, this would not alter the intended meaning of the program. Each time we need to unify the head of a program clause with a goal, we can delete the quantifiers and substitute the variables by **new unknowns**, this is equivalent to the necessary renaming in an application of SLD-resolution, unification of the head of the clause and the goal yields terms for the unknowns appearing in them, the unknowns appearing in the body are then substituted by the calculated terms for obtaining the new goals. An application of SLD-resolution brings new goals with possibly new unknowns and leaves the formulas of the program unaltered. An important last remark is that *unification is exclusively used for finding terms for the unknowns.*

4 SLD-Resolution as Reduction of Sequents

In the programs and goals we don't restrict ourselves to horn clauses, in both we allow arbitrary formulas constructed with the symbols for implication and universal quantification. Since we only deal with a logical extension of SLD-resolution and not the *control* for the search of derivations, we consider a program to be a set of formulas instead of a list of formulas. A goal is a single formula. Unknowns and arbitrary constants may be present in programs and goals. A pair consisting of a program Σ and a goal ξ is denoted by $\Sigma \vdash \xi$ and is called a **sequent** in this article.

Sequents $\Sigma \vdash A$ in ordinary Prolog contain only horn clauses without unknowns in their programs Σ and their goals A are atoms. An application of SLD-resolution with the horn clause $V(A_n \wedge \cdots \wedge A_1 \rightarrow A_0)$ of Σ, where V is a block of quantifiers, leads to the new goals $A''_n, \ldots, A''_1$. These new goals are obtained as explained in the previous section, by deleting the quantifiers V, by substituting its variables by new unknowns for obtaining $A'_n \wedge \cdots \wedge A'_1 \rightarrow A'_0$, by unifying the head A'_0 with the goal A, and by substituting the calculated terms for the unknowns in the A'_k for obtaining the A''_k. We say that the

application of SLD-resolution **SLD-reduces** the sequent $\Sigma \vdash A$ to the possibly empty list of sequents $\Sigma \vdash A''_n$, ..., $\Sigma \vdash A''_1$, where the goals A''_i may contain new unknowns, and also that this application provides terms for the unknowns in A **for which this reduction is valid**. These terms provided by SLD-Resolution may contain unknowns, even some of the new unknowns added in the process. With mj-resolution something similar is done for *reducing a sequent to a list of sequents*, but in the case that the original sequent is of the particular form treated by SLD-resolution, mj-resolution behaves exactly as SLD-resolution. This is why it is an extension of SLD-resolution.

5 mj-Resolution

We understand a sequent $\Sigma \vdash \xi$ without unknowns as the statement of the problem of verifying the intuitionistic derivability of ξ from Σ. In the case that $\Sigma \vdash \xi$ contains unknowns, we understand it as the more general problem of searching for terms for these unknowns, such that, after substituting the unknowns with these terms, ξ is derivable from Σ in intuitionistic logic. — Hence, $\{A, A \to B\} \vdash B$ represents the problem of verifying if B is derivable from A and $A \to B$ in intuitionistic logic (this is the case), $\{(A(x) \to A(y)) \to A(y)\} \vdash A(x)$ represents the problem of searching x and y such that $A(x)$ is derivable from $(A(x) \to A(y)) \to A(y)$ in intuitionistic logic (this is true if and only if x and y are substituted by the same term), $\vdash (\forall v((S(v) \to \forall v S(v)) \to B)) \to B$ represents the problem of verifying the intuitionistic derivability of $(\forall v((S(v) \to \forall v S(v)) \to B)) \to B$ (it is not derivable).

mj-Resolution gives *rules to reduce* the solvability of the problem denoted by $\Sigma \vdash \xi$ to the solvability of a list of problems of the same kind. A reduction rule may provide **solving terms** for the unknowns of the original problem (perhaps depending on unknowns, even on new auxiliary unknowns) or may impose **restrictions** on the unknowns in the new list of sequents under which the reduction is valid. If the list of new sequents is empty, the original problem is *trivially solved.*

5.1 The Three Kinds of mj-Resolution Rules

There are three kinds of mj-resolution rules for reducing a sequent: **d-reductions**, **m-reductions** and **g-reductions**. Only g-reductions impose restrictions, only m-reductions provide solving terms.

d-Reductions are for reducing sequents of the form $\Sigma \vdash \eta \to \xi$ to $\Sigma \cup \{\eta\} \vdash \xi$. The correctness of this rule follows from the deduction theorem.

g-Reductions are for reducing sequents of the form $\Sigma \vdash \forall v \xi(v)$ to $\Sigma \vdash \xi(q)$, where q is a new arbitrary constant not appearing in $\Sigma \cup \{\forall v \xi(v)\}$. The restriction imposed by this rule is that q must not appear in the terms to be found for the unknowns in $\Sigma \vdash \xi(q)$. The correctness of the rule follows from the theorem of generalization of constants.

m-Reductions are used to reduce sequents of the form $\Sigma \vdash A$, whose goal A is atomic. m-Reduction is very similar to SLD-resolution. For performing an m-reduction of $\Sigma \vdash A$ it is necessary to *select* a formula ξ in the program Σ whose *head matches* the goal A, different selections of a matching formula may lead to different reductions. The general definition of m-reduction requires first to define **principal quantifiers**, **head** and **body** of an arbitrary formula. Before introducing the general definition of this more complicated kind of reduction, we want to motivate it first by introducing some particular cases in

which the *selected* formula has a special form and by presenting some examples that use these particular cases.

A problem denoted by a sequent is positively solved by mj-resolution if reductions can be applied recursively — by reducing the original sequent, or sequents that were obtained by reduction of other sequents — until no sequent remains. A track of the reductions may be written down as a *search tree* during this process. This is also done with SLD-resolution, but since during the process the program doesn't change, only goals are reduced to other goals with SLD-resolution. It is important to note that all symbols, with exception of variables, denote the same objects in all sequents arisen during this process of recursive reduction. Each unknown is to be substituted by the same term in all sequents. If an unknown x is substituted by a term t, then the unknowns of t **inherit** the restrictions of x, this means: if there is a restriction that x must not be substituted by terms containing an arbitrary constant q, then the restrictions that the unknowns appearing in t must not be substituted by terms containing q is imposed.

5.2 m-Reductions Closing a Branch of the Search Tree

In accordance with [13], we call a formula ξ of the form VB, in which V is a possibly empty block of quantifiers and B an atomic open formula (perhaps containing unknowns), a **unit clause**. The **head** of such a unit clause is B, its **body** is empty, its **principal quantifiers** are the ones in the block V.

Let $\Sigma \vdash A$ be a sequent whose goal A is atomic. Let VB be a unit clause from Σ. We say that the head of VB **matches** the atomic goal A, if deleting the principal quantifiers V of VB and substituting its variables by new unknowns yields an atom B' that unifies with the goal A in such a way, that the calculated terms don't contain forbidden arbitrary constants for the unknowns to which they are assigned (such restrictions may have arisen in previous g-reductions). We can select such a unit clause VB for reducing the problem $\Sigma \vdash A$ to the empty list, the **solving terms** are given by the unification of B' with A. This reduction is called a **trivial m-reduction** selecting the unit clause VB. It corresponds to the unification with a unit clause in SLD-resolution. Its correctness is obvious.

A d-reduction of $\vdash \forall v A(v,x) \rightarrow A(c_1,c_2)$ leads to $\forall v A(v,x) \vdash A(c_1,c_2)$. After substituting the variable v in $A(v,x)$ by a new unknown x_1 one gets the atomic formula $A(x_1,x)$ that unifies with the goal $A(c_1,c_2)$. A trivial m-reduction leads to the empty list and to the solution $x = c_2$ for the original problem (the unification $x_1 = c_1$ for the auxiliary unknown x_1 is not relevant to the original problem). Hence, we conclude that the formula $\forall v A(v,x) \rightarrow A(c_1,c_2)$ is derivable in intuitionistic logic after substituting the unknown x by c_2.

A d-reduction of $\vdash A(x) \rightarrow \forall v(B \rightarrow A(v))$ leads to $A(x) \vdash \forall v(B \rightarrow A(v))$. A g-reduction leads to $A(x) \vdash B \rightarrow A(q)$ and imposes the restriction that the new arbitrary constant q doesn't appear in the term to be assigned to the unknown x. We have now two unit clauses in the program, $A(x)$ and B, the first unifies with the goal $A(q)$, but by assigning to x the forbidden arbitrary constant q, the second doesn't unify with the goal: the head of none of these unit clauses matches the goal $A(q)$. There is no possible selection of a formula in the program whose head matches the goal for an m-reduction. There are no alternative reductions by mj-resolution. Due to the *completeness of mj-resolution*, we can conclude that there is no x such that $A(x) \rightarrow \forall v(B \rightarrow A(v))$ is derivable in intuitionistic logic.

5.3 m-Reduction without Unification

We call a formula ξ of the form $(\xi_n \to (\xi_{n-1} \to (\cdots \to (\xi_1 \to B)\cdots)))$, where B is an atomic formula, a **formula without principal quantifiers**. This formula can be abbreviated with $\xi_n \wedge \cdots \wedge \xi_1 \to A$, the **body** of ξ is the list $\xi_n, \ldots, \xi_1$ of its subformulas, its **head** is the atomic subformula B.

Let $\Sigma \vdash A$ be a sequent whose goal A is atomic. Let ξ from Σ be a formula without principal quantifiers whose head B coincides with A. We say that the head of ξ matches the goal A, and that an m-reduction of $\Sigma \vdash A$ selecting this formula ξ leads to the list of sequents $\Sigma \vdash \xi_n, \ldots, \Sigma \vdash \xi_1$, where the ξ_k are the formulas in the body of ξ. This m-reduction does not alter any unknown, it is trivial if $n = 0$. Obviously, solving the problems denoted by $\Sigma \vdash \xi_n, \ldots, \Sigma \vdash \xi_1$ yields a solution of $\Sigma \vdash A$ by using modus ponens, this proves the correctness of the rule.

A d-reduction of $((\forall v(A \to B(v))) \to B(x)) \to A \vdash B(c) \to A$ leads to $((\forall v(A \to B(v))) \to B(x)) \to A, B(c) \vdash A$. The head of the first formula in the program coincides with the goal, an m-reduction selecting this formula leads to $((\forall v(A \to B(v))) \to B(x)) \to A, B(c) \vdash (\forall v(A \to B(v))) \to B(x)$. A d-reduction leads to $((\forall v(A \to B(v))) \to B(x)) \to A, B(c), (\forall v(A \to B(v))) \vdash B(x)$. Finally, a trivial m-reduction selecting $B(c)$ leads to the empty list and to the assignment of the constant c to the unknown x. Hence, the original goal is derivable in intuitionistic logic if $x = c$.

We can abbreviate the propositional formula $(A \to (B \to C))$ by $A \wedge B \to C$ and the propositional formula $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$ by $(A \to B) \wedge (A \wedge B \to C) \wedge A \to C$. Three consecutive d-reductions of the sequent $\vdash (A \to B) \wedge (A \wedge B \to C) \wedge A \to C$ lead to the sequent $(A \to B), (A \wedge B \to C), A \vdash C$. The head of the formula $A \wedge B \to C$ in the program coincides with the goal C, its body consists of A and B. An m-reduction selecting this formula leads to the two sequents $(A \to B), (A \wedge B \to C), A \vdash A$ and $(A \to B), (A \wedge B \to C), A \vdash B$. The second one can be reduced to the first one by an m-reduction selecting the formula $A \to B$. The first can be reduced to the empty list by a trivial m-reduction. Hence, the propositional formula $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$ is derivable in intuitionistic logic.

5.4 m-Reduction Selecting Generalized Horn Clauses

We call a formula ξ of the form $V(\xi_n \to (\xi_{n-1} \to (\cdots \to (\xi_1 \to B)\cdots)))$, where V is a possibly empty block of quantifiers and B an atomic open formula (perhaps containing unknowns), a **generalized horn clause**. This formula can be abbreviated with $V(\xi_n \wedge \cdots \wedge \xi_1 \to B)$. The **body** of ξ is the list of open subformulas $\xi_n, \ldots, \xi_1$, the **head** of ξ is the atomic open formula B, its **principal quantifiers** are the ones in the block V. This formula is a horn clause if and only if the open formulas ξ_i are positive literals, namely, if and only if they don't contain implications and universal quantifiers. This formula is a unit clause if and only if $n = 0$, namely, if and only if its body is empty. This formula is one without principal quantifiers, as defined above, if and only if V is empty. — By deleting the principal quantifiers of a generalized horn clause ξ and substituting its variables by terms, for example by new unknowns, one gets a formula without principal quantifiers.

Let $\Sigma \vdash A$ be a sequent whose goal A is atomic. Let ξ in the program Σ be a generalized horn clause. We say that the head of ξ **matches** the atomic goal A of $\Sigma \vdash A$, if deleting its principal quantifiers and substituting its variables by new unknowns yields a formula ξ' without principal quantifiers whose head B' unifies with the goal A in such a way, that the calculated terms don't contain forbidden arbitrary constants for the unknowns

to which they are assigned. The m-reduction of $\Sigma \vdash A$ selecting this formula ξ leads to $\Sigma'' \vdash \xi''_n, \ldots, \Sigma'' \vdash \xi''_1$, where Σ'' and ξ''_k are obtained by substituting the unknowns of Σ and the unknowns of the formulae ξ'_k in the body of ξ' by the solving terms calculated in the unification of the head B' of ξ' with the atomic goal A. These calculated terms are called the **solving terms** of the m-reduction. This particular case mj-resolution is more general than SLD-resolution. The proof of its correctness is similar to the proof for SLD-resolution, it follows easily from *dictum de omni* and *modus ponens*.

A d-reduction of $\vdash (\forall v(C \rightarrow A(v))) \rightarrow (C \rightarrow \forall v A(v))$ leads to $\forall v(C \rightarrow A(v)) \vdash C \rightarrow \forall v A(v)$. A second d-reduction leads to $\forall v(C \rightarrow A(v)), C \vdash \forall v A(v)$. A g-reduction leads to $\forall v(C \rightarrow A(v)), C \vdash A(q)$. The first formula is a horn clause, after substituting the variable v in its head by a new unknown x one gets the formula $A(x)$ that unifies with the goal $A(q)$. An m-reduction selecting this clause yields $\forall v(C \rightarrow A(v)), C \vdash C$. The latter sequent is reduced to the empty list by a trivial m-reduction.

The formula $\xi = \forall v((S(v) \rightarrow \forall v S(v)) \rightarrow \Box)$ is a generalized horn clause, its head is $\Box$, its body is $S(v) \rightarrow \forall v S(v)$. We try now to reduce $\vdash \xi \rightarrow \Box$. A d-reduction leads to $\xi \vdash \Box$. Deleting the principal quantifier of ξ and substituting its variable v by the new unknown x yield $(S(x) \rightarrow \forall v S(v)) \rightarrow \Box$, the head of this formula coincides with the atomic goal $\Box$. Hence, we can select ξ for reducing the original sequent to $\xi \vdash S(x) \rightarrow \forall v S(v)$ by m-reduction. A d-reduction leads to $\xi, S(x) \vdash \forall v S(v)$. A g-reduction leads to $\xi, S(x) \vdash S(q)$ and imposes the restriction that the new arbitrary constant q must not appear in the term for the unknown x. Now, we have two formulas in the program, the first has $\Box$ as head and doesn't match the goal $S(q)$, the second is the atomic formula $S(x)$, it unifies with the goal $S(q)$, but the unifier $x = q$ is forbidden by a previous g-reduction. There are no other reduction alternatives. Due to the completeness of mj-resolution, we can conclude that $\xi \rightarrow \Box$ is not derivable in intuitionistic logic.

5.5 The General Definition of m-Reduction

The formula $A \rightarrow \forall v B(v)$ is equivalent to the generalized horn clause $\forall v(A \rightarrow B(v))$. The formula $\forall v(A(v) \rightarrow \forall v B(v))$ is equivalent to the generalized horn clause $\forall v \forall w(A(v) \rightarrow B(w))$. The formula $C \rightarrow (\forall v(A(v) \rightarrow \forall v B(v)))$ is equivalent to the generalized horn clause $\forall v \forall w(C \wedge A(v) \rightarrow B(w))$. Every formula can be transformed to a generalized horn clause only by moving quantifiers to the front, perhaps after renaming variables. By induction one can prove that every formula ξ has the form $V_n(\xi_n \rightarrow V_{n-1}(\xi_{n-1} \rightarrow (\cdots \rightarrow V_1(\xi_1 \rightarrow V_0 B) \cdots)))$, where B is an atomic open formula. The V_k can be moved to the front for obtaining an equivalent generalized clause ξ' of the form $V'_n V'_{n-1} \cdots V'_1 V'_0(\xi'_n \wedge \cdots \wedge \xi'_1 \rightarrow B')$, where the V'_k, the ξ'_k, and B' are the V_k, the ξ_k, and B after necessary renamings. The **principal quantifiers** of ξ are the V_k, the **head** of ξ is the atomic open formula B, the **body** of ξ is the list consisting of the open formulas $\xi_n, \ldots, \xi_1$. This general definition is consistent with the previous definitions for particular cases.

We can delete a principal quantifier of a formula ξ and substitute the variable it binds by a term for obtaining a formula η that is implied by ξ. We can do this repeatedly with all principal quantifiers, until we get a formula without principal quantifiers that is implied by ξ. For example, deleting the principal quantifier of $A \rightarrow \forall v B(v)$ and substituting its variable by an unknown x yields $A \rightarrow B(x)$. Deleting the principal quantifiers of $\forall v(A(v) \rightarrow \forall v B(v))$ and substituting their variables by the unknowns x_1 and x_2 yields $A(x_1) \rightarrow B(x_2)$. Deleting the principal quantifiers of $C \rightarrow (\forall v(A(v) \rightarrow \forall v B(v)))$ and substituting its variables by the unknowns x_1 and x_2 yields $C \wedge A(x_1) \rightarrow B(x_2)$. Exactly

the same result can be obtained by first moving the principal quantifiers to the front, for obtaining an equivalent generalized horn clause, and then by deleting these principal quantifiers at the front and substituting their variables by terms.

Let $\Sigma \vdash A$ be a sequent whose goal A is atomic. The concept of a formula ξ from Σ that **matches** the atomic goal A and the concept of an m-reduction of $\Sigma \vdash A$ selecting this formula ξ is defined with exactly the same words as it was done for the special case that ξ be a generalized horn clause. The only possible difference is that in this case the principal quantifiers of ξ don't need to be at the front of ξ, so that the process of deleting them and substituting their variables could demand a little more work. This general definition is consistent with all particular cases presented above.

Let ξ be given by $(A \to B) \to (\forall v((\forall w(A \to C(v,w))) \to B))$. The only principal quantifier of ξ is the one with the variable v, the head of ξ is B, the formulas of the body are $A \to B$ and $\forall w(A \to C(v,w))$. As a last example, we consider the sequent $\xi \vdash \xi$. If ξ were atomic, this sequent could be reduced by a trivial m-reduction, but strict mj-resolution doesn't allow it in this case: we are forced to perform an m-, d- or g-reduction according to the form of the goal. A d-reduction adds the formula $A \to B$ to the program, the new goal is $\forall v((\forall w(A \to C(v,w))) \to B)$. A g-reduction deletes the quantifier $\forall v$ and substitutes v by the new arbitrary constant q, the goal becomes $(\forall w(A \to C(q,w))) \to B$. A d-reduction adds the formula $\forall w(A \to C(q,w))$ to the program, the new goal is B. Now, the head of the original formula ξ and the head of the first formula added to the program, namely $A \to B$, match the goal B. We select ξ for an m-reduction, the variable v of the principal quantifier is substituted by the new unknown x. This m-reduction leads to two sequents, the goals of these sequents are $\gamma_1 = (A \to B)$ and $\gamma_2 = (\forall w(A \to C(x,w)))$, the programs of both sequents are the same and contain $\gamma_1' = (A \to B)$ and $\gamma_2' = (\forall w(A \to C(q,w)))$. We reduce the sequent, whose goal is γ_1, first by a d-reduction that adds A to the program and changes the goal to B, and then by an m-reduction selecting γ_1' that changes the goal to A; since A was added to the program, a trivial m-reduction closes this branch of the process. We reduce the sequent whose goal is γ_2 first by a g-reduction that changes the goal to $A \to C(x,q_2)$, here the new arbitrary constant q_2 is different from the q present in the program, q_2 shouldn't appear in the term to be assigned to x. A d-reduction adds A to the program and changes the goal to $C(x,q_2)$. γ_2' is the only formula whose head matches the goal, deleting its principal quantifier and substituting its variable w by the new unknown x_2 leads to the formula $A \to C(q,x_2)$, the equation $C(q,x_2) = C(x,q_2)$ has the allowed unifier $x = q, x_2 = q_2$. An m-reduction selecting γ_1 leads to a sequent having A in the program and whose goal is A. The procedure ends positively, verifying that ξ is derivable from ξ in intuitionistic logic.

6 Correctness and Completeness of mj-Resolution

If the problem $\Sigma \vdash \xi$ is reduced to the problems $\Sigma_n \vdash \xi_n, \ldots, \Sigma_1 \vdash \xi_1$ by mj-resolution, then we have a rule to build a solution of $\Sigma \vdash \xi$ with a solution of all $\Sigma_n \vdash \xi_n, \ldots, \Sigma_1 \vdash \xi_1$, independent of how the last solution looks like. This means that **mj-resolution is correct.**

In the following theorem, we state the completeness of mj-resolution. It can be proved by usual means of formal logic. A proof may be found in [16].

Theorem 1 (Completeness of mj-Resolution) *The problem denoted by a sequent is solvable if and only if the sequent can be recursively reduced by mj-resolution until no*

sequent remains.

We now state a theorem that allows to use mj-resolution as a complete and correct proof procedure for classical logic. This theorem may also be proved by usual means of formal logic, as it is done in [16]. Let $\Box$ be a predicate symbol of arity 0 for denoting *contradiction.* The negation of α may be paraphrased as usual with $\alpha \rightarrow \Box$. This allows to paraphrase all usual logical symbols of classical logic. For each predicate symbol R of arity n let w_R denote the formula $\forall\bar{v}(((R(\bar{v}) \rightarrow \Box) \rightarrow \Box) \rightarrow R(\bar{v}))$, where $\bar{v}$ is the n-tuple of the first n variables of the language. Let $\mathcal{W}$ denote the set containing all w_R with exception of $w_\Box$.

Theorem 2 *The problem of finding terms for the unknowns in $\Sigma \vdash \xi$ such that ξ be derivable from Σ in classical logic is equivalent to the intuitionistic problem $\Sigma \cup \mathcal{W} \vdash \xi$ of the kind treated in this article.*

Acknowledgements

I am grateful to Professor Dr. Otto Mayer for his continuous support without which this work could not have been done, to Anna Perekhod and Torkel Franzén for valuable comments related to this article, and to my teacher of Prolog, Gérard Battani.

References

[1] K. R. Apt and M. H. van Emden, "Contributions to the Theory of Logic Programming", J. ACM 29, 3 (July 1982), 841-862.

[2] M. Aronsson, "GCLA, The Design, Use, and Implementation of a Program Development System", A dissertation submitted to Stockholm University, October 1993.

[3] G. Battani and H. Meloni, "Interpreteur du Language de Programmation PROLOG", Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1973.

[4] A. H. Colmerauer, P. Kanoui, P. Roussel and R. Pasero, "Un Systeme de Communication Homme-Machine en Francais", Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille, 1973.

[5] K. L. Clark, "Negation as Failure", in Logic and Databases, H. Gallaire and J. Minker(eds), Plenum Press, New York, 1978, 293-322.

[6] D. M. Gabbay — U. Reyle, "Computation with run time skolemization (N-Prolog part 3)", Journal of Applied Non-Classical Logics, Vol. 3, N 1/1993, pages 93–128

[7] Lars Hallnäs und Peter Schroeder-Heister, "A Proof-Theoretic Approach to Logic Programming" J. Logic Computat., Vol 1 No. 2, 1990, p. 261-283 and Vol 1 No. 5, 1991, p. 635-660.

[8] James Harland, "A Proof-Theoretic Analysis of Goal Directed Provability", Journal of Logic and Computation, 4:1:69-88, Jan. 1994.

[9] Hassan Aït-Kaci "Warren's Abstract Machine, A Tutorial Reconstruction" The MIT Press, Cambridge, Massachusetts, London, 1991.

[10] R. A. Kowalski, "Predicate Logic as a Programming Language", Information Processing 74, Stockholm, North Holland, 1974, 569-574.

[11] D. W. Loveland, "A linear format for resolution", Proc. IRIA Symp. Authomatic Demonstration, Versailles, France, 1968, Springer-Verlag, New York, pp. 147-162.

[12] D. Luckham, "Refinement theorems in resolution theory", Proc. IRIA Symp. Automatic Demonstration, Versailles, France, 1968, Springer-Verlag, New-York, pp. 193–190.

[13] J. W. Lloyd, "Foundations of Logic Programming", Second, Extended Edition, Springer-Verlag, Berlin, 1987.

[14] D. Miller, "A Logical Analysis of Modules in Logic Programming", J. Logic Programming 1989:79–108

[15] D. Miller, G. Nadathur, F. Pfenning and A. Scedrov, "Uniform Proofs as a Foundation of Logic Programming", Anals of Pure and Applied Logic 51:125–157,1991.

[16] Rodrigo Readi Nasser, "A Reduction Oriented Calculus", submitted for publication (see also Internal Report 293/97 of 21.10.1997, C. S., University of Kaiserslautern).

[17] David W. Reed and Donald W. Loveland, "A Comparison of Three Prolog Extensions", J. Logic Programming 1992:12:25–50.

[18] J. A. Robinson, "A machine oriented logic based on the resolution principle", J. ACM 12, 1965, No. 1, 23–41.

[19] L. Thorne McCarthy, "Clausal Intuitionistic Logic", J. Logic Programming 1988:5:1–31,93–132.

[20] D. H. D. Warren, "An abstract prolog instruction set", Technical Note 309, SRI International, Menlo Park, CA, October 1983.