PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 61 (75), 1997, 137–152

HOW TO GENERALIZE LOGIC PROGRAMMING TO ARBITRARY SET OF CLAUSES

Slaviša B. Prešić

Communicated by Žarko Mijajlović

Abstract. We state how one can extend Logic Programming to any set of clauses.

Basic part of Logic programming, particularly Prolog, in fact deals with the following two inference rules:

(1) $\mathcal{F}, p \vdash p$

(2) $\mathcal{F},\ p \vee \neg q_1 \vee \ldots \vee \neg q_k \vdash p \longleftarrow \mathcal{F} \vdash q_1, \ldots, q_k$

where $\mathcal{F}$ is a set of (positive) Horn formulas and p is any atom, i.e. a propositional letter. Indeed, the informal meaning of rule (1) is: *An atom p is a consequence of a set of clauses if p is an element of that set.* Similarly for rule (2) we have this meaning: *An atom p is a consequence of a set $\mathcal{F}$, $p \vee \neg q_1 \vee \ldots \vee \neg q_k$ (i.e. of the set $\mathcal{F}$, $q_1 \wedge \ldots \wedge q_k \Rightarrow p$), if $q_1, \ldots, q_k$ are consequences of the set $\mathcal{F}$.* In the sequel we use the following facts from mathematical logic (see [2]):

(3) *The notion of formal proof in case of propositional logic (assuming we have chosen some tautologies as axioms, and that* **modus ponens** *is the only inference rule).*

(4) **The Deduction theorem**[1)]: $\mathcal{F},\ A \vdash B \longleftrightarrow \mathcal{F} \vdash A \Rightarrow B$

where $\mathcal{F}$ is a set of propositional formulas, A, B are some such formulas.

(5) **Completeness Theorem**: *Any propositional formula is a logical theorem if and only if it is a tautology.*

Also we use the symbols $\bot$, $\top$ which can be introduced by the following definitions: $\bot$ *stands for* $a \wedge \neg a$; $\top$ *stands for* $a \vee \neg a$, where a is an atom (chosen arbitrarily).

AMS Subject Classification (1991): Primary 68N17

Supported by Ministry of Science and Technology of Serbia, grant number 04M03/C

1) In fact, only $\longrightarrow$-part is the deduction theorem. But, $\longleftarrow$-part is almost trivial.

Further, let $\mathcal{F}$ be any set of propositional formulas and ψ a formula or one of the symbols $\bot$, $\top$. Then **a sequent** is any expression of the form $\mathcal{F} \vdash \psi$, with the meaning: *ψ is a logical consequence of $\mathcal{F}$*

LEMMA 1. *Let $\mathcal{F}$ be any set of propositional formulas not containing the atom p, and let $\phi_1(p), \phi_2(p), \dots$ be propositional formulas containing p. Then we have the following equivalences*

$$(6)\qquad \begin{array}{rl} \text{(i)} & \mathcal{F}, \phi_1(p), \phi_2(p), \dots \vdash p \longleftrightarrow \mathcal{F}, \phi_1(\bot), \phi_2(\bot), \dots \vdash \bot \\ \text{(ii)} & \mathcal{F}, \phi_1(p), \phi_2(p), \dots \vdash \neg p \longleftrightarrow \mathcal{F}, \phi_1(\top), \phi_2(\top), \dots \vdash \bot \end{array}$$

Proof. First we prove the $\longrightarrow$ part of (i). Then, we have the following 'implication-chain':

$\mathcal{F}, \phi_1(p), \phi_2(p), \dots \vdash p$

$\longrightarrow$ For some formulas $f_1, \dots, f_r$ of $\mathcal{F}$ and some formulas $\phi_{i1}(p), \dots, \phi_{is}(p)$ we have: $f_1, \dots, f_r, \phi_{i1}(p), \dots, \phi_{is}(p), \dots \vdash p$ (Finiteness of the propositional proof)

$\longrightarrow$ $\vdash f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(p) \Rightarrow \cdots \Rightarrow \phi_{is}(p) \Rightarrow p$ (By (4))

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(p) \Rightarrow \cdots \Rightarrow \phi_{is}(p) \Rightarrow p$ is a tautology (By (5))

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(\bot) \Rightarrow \cdots \Rightarrow \phi_{is}(\bot) \Rightarrow \bot$ is a tautology

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(\bot) \Rightarrow \cdots \Rightarrow \phi_{is}(\bot) \Rightarrow \bot$ is a logical theorem (By (5))

$\longrightarrow$ Formula $f_1, \dots, f_r, \phi_{i1}(\bot), \dots, \phi_{is}(\bot) \vdash \bot$ holds. (By (4))

$\longrightarrow$ $\mathcal{F}, \phi_1(\bot), \phi_2(\bot), \dots \vdash \bot$

which completes the proof. A proof of $\longleftarrow$ part of (i) reads:

$\mathcal{F}, \phi_1(\bot), \phi_2(\bot), \dots \vdash \bot$

$\longrightarrow$ For some formulas $f_1, \dots, f_r$ of $\mathcal{F}$ and some formulas $\phi_{i1}(\bot), \dots, \phi_{is}(\bot)$ have: $f_1, \dots, f_r, \phi_{i1}(\bot), \dots, \phi_{is}(\bot), \dots \vdash \bot$ (Finiteness of every formal proof)

$\longrightarrow$ $\vdash f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(\bot) \Rightarrow \cdots \Rightarrow \phi_{is}(\bot) \Rightarrow \bot$ (By (4))

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(\bot) \Rightarrow \cdots \Rightarrow \phi_{is}(\bot) \Rightarrow \bot$ is a tautology (By (5))

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(p) \Rightarrow \cdots \Rightarrow \phi_{is}(p) \Rightarrow p$ is a tautology

$\longrightarrow$ Formula $f_1 \Rightarrow \cdots \Rightarrow f_r \Rightarrow \phi_{i1}(p) \Rightarrow \cdots \Rightarrow \phi_{is}(p) \Rightarrow p$ is a logical theorem (By (5))

$\longrightarrow$ Formula $f_1, \dots, f_r, \phi_{i1}(p), \dots, \phi_{is}(p) \vdash p$ holds. (By (4))

$\longrightarrow$ $\mathcal{F}, \phi_1(p), \phi_2(p), \dots \vdash p$

which completes the proof of (i). We omit a proof of (ii) because (ii) can be proved in a similar way as (i).

Notice that Lemma 1 can be expressed by the following words: *A literal*[2] *ψ is a logical consequence of the given set if and only if the corresponding set is inconsistent.* Now we prove the following lemma.

LEMMA 2. *The equivalence*

(7) $\mathcal{F},\ p_1 \vee\ \ldots \vee p_k \vdash\ \bot \longleftrightarrow\ \mathcal{F} \vdash \neg p_1,\ \ldots\ , \mathcal{F} \vdash \neg p_k$ *(p_i is any literal)*

is true.

Proof. We have the following 'equivalence chain':

$\mathcal{F}, p_1 \vee \ldots \vee p_k \vdash \bot$

$\longleftrightarrow \mathcal{F} \vdash (p_1 \vee \ldots \vee p_k \Longrightarrow \bot)$ (By (4))

$\longleftrightarrow \mathcal{F} \vdash (\neg p_1 \wedge \ldots \wedge \neg p_k)$ (Using a well known tautology)

$\longleftrightarrow \mathcal{F} \vdash \neg p_1, \ldots , \mathcal{F} \vdash \neg p_k$

which completes the proof.

Besides (6) and (7) we emphasize the following obvious equivalences

(8) $\vdash \top\ \longleftrightarrow\ \mathcal{F}, \bot \vdash \bot$

(9) $\mathcal{F}, \top \vdash A\ \longleftrightarrow \mathcal{F} \vdash\ A$ (A is a literal or the symbol $\bot$)

Suppose now that $\mathcal{F}$ is a given set of clauses and ψ is a literal or $\bot$. Is it possible that by using the equivalences (6), (7), (8), (9) one can establish whether or not ψ is a logical consequence of $\mathcal{F}$? In order to give the answer we introduce the following inference rules[3]

(R1) $\mathcal{F}, \bot \vdash \bot \longleftarrow \vdash \top$

(R2) $\mathcal{F}, \phi_1(p), \phi_2(p), \ldots \vdash p\ \longleftarrow \mathcal{F}, \phi_1(\bot), \phi_2(\bot), \ldots \vdash\ \bot$

$\mathcal{F}, \phi_1(p), \phi_2(p), \ldots \vdash \neg p\ \longleftarrow \mathcal{F}, \phi_1(\top), \phi_2(\top), \ldots \vdash\ \bot$

($\phi_i(p)$ is any clause containing p)

(R3) $\mathcal{F},\ p_1 \vee \ldots \vee p_k \vdash\ \bot \longleftarrow\ \mathcal{F} \vdash \neg p_1, \ldots,\ \mathcal{F} \vdash \neg p_k$

(where p_i is any literal; while $\neg p_i$ is its opposite literal)

(R4) $\mathcal{F}, \top \vdash\ A \longleftarrow \mathcal{F} \vdash\ A$ (A is a literal or the symbol $\bot$)

We emphasize that in the sequel for the set $\mathcal{F}$ we suppose that it does not contain a clause of the form $\ldots q \vee \neg q \ldots$,where q is any atom. Namely, such a formula is equivalent to $\top$, consequently it should be omitted[4]. Similarly, if it happens that by applying rule (R2) some clause becomes equivalent to $\top$, then we also omit it. Roughly speaking rules (R1), (R2), (R3), (R4) are used as follows:

We start with a question (a sequent) of the form $\mathcal{F} \vdash \psi$ and apply several times rules (R2), (R3), (R4). If at some step we can apply rule (R1) the procedure halts with the conclusion that ψ is a logical consequence of $\mathcal{F}$. However, if at some

[2] A literal is an atom or the negation of an atom

[3] We point out that the set $\mathcal{F}$ may be also an empty set. Also, in rule (R3) k may be 1.

[4] This is compatible with rule (R4)

step we obtain the sequent $\vdash \bot$ (then $\mathcal{F}$ is an empty set) the procedure halts with conclusion that ψ is not a logical consequence of $\mathcal{F}$.

Example 1. Let p, q, r, s, t be atoms. Answer the following questions:

1) $p \vdash p$? 2) $p,\ q \vdash p$? 3) $\vdash p$? 4) $q \vdash p$?

5) $\neg q \vee p,\ q \vee p \vdash p$? 6) $p, \neg p \vee q \vee \neg r,\ p \vee \neg q \vee s,\ p \vee s \vee \neg t \vdash \bot$?

Answers. 1) Applying (R2) we obtain the sequent $\bot \vdash \bot$ and by (R1) we get the sequent $\vdash \top$ so the answer is: *Yes.*

2) Applying (R2) we obtain a new question i.e. the sequent $\bot, q \vdash \bot$, and now applying (R1) we obtain the sequent $\vdash \top$ so the answer is: *Yes.*

3) Applying (R2) we obtain the sequent $\vdash \bot$ so the answer is: *No.*

4) By (R2) we obtain the sequent $q \vdash \bot$ and after that by (R3) we obtain the sequent $\vdash \neg q$. By (R2) we obtain the sequent $\vdash \bot$ such that the answer is: *No.*

5) By (R2) we obtain the sequent $\neg q, q \vdash \bot$. Now by (R3) applied to the literal $\neg q$ we obtain the sequent $q \vdash q$, further by (R2) we obtain the sequent $\bot \vdash \bot$ and finally by (R1) we obtain the sequent $\vdash \top$ so the answer is: *Yes.*

6) Now by (R3) applied to the clause p we obtain the sequent

$$\neg p \vee q \vee \neg r,\ p \vee \neg q \vee s,\ p \vee s \vee \neg t \vdash \neg p$$

By (R2) (and (R4) applied twice) we obtain the sequent $q \vee \neg r \vdash \bot$. At this step applying (R3) we obtain two new sequents, i.e. questions $\vdash \neg q$? and $\vdash r$? The answer to the first question is *No*, so the final answer is also: *No.*

Concerning rules (R1)–(R4) we have the following lemma.

LEMMA 3. (Soundness of rules (R1)–(R4)). *Let $\mathcal{F}$ be any set of clauses. Suppose that we start with a sequent $\mathcal{F} \vdash \psi$, where ψ is a literal or the symbol $\bot$. Then, if by use of rules* (R1)–(R4) *we obtain the sequent $\vdash \top$ or the sequent $\vdash \bot$, then ψ is / is not a logical consequence of the set $\mathcal{F}$ respectively.*

Proof follows immediately from the fact that rules (R1)–(R4) are based on logical equivalences (6)–(9).

Let $\mathcal{F} \vdash \psi$ be any sequent. By *Val($\mathcal{F} \vdash \psi$)* we denote its *truth value*, defined by: If ψ is a logical consequence of the set $\mathcal{F}$, then *Val($\mathcal{F} \vdash \psi$)* is *true*, otherwise *Val($\mathcal{F} \vdash \psi$)* is *false.*

According to this definition and to rules (R1)–(R4), i.e. to equivalences (6)–(9) we have the following equalities

$$\text{(10)}\quad \begin{array}{ll} Val(\vdash \top) = true & Val(\mathcal{F}, \bot \vdash \bot) = true \\ Val(\vdash \bot) = false & Val(\mathcal{F}, \top \vdash\ psi) = Val(\mathcal{F} \vdash \psi) \end{array}$$

$$Val(\mathcal{F}, \phi_1(p), \phi_2(p), \ldots \vdash p) = Val(\mathcal{F}, \phi_1(\bot), \phi_2(\bot), \ldots \vdash\ \bot)$$

$$Val(\mathcal{F}, \phi_1(p), \phi_2(p), \ldots \vdash \neg p) = Val(\mathcal{F}, \phi_1(\top), \phi_2(\top), ... \vdash\ \bot)$$

($\phi_i(p)$ is any clause containing p)

$$Val(\mathcal{F},\ p_1 \vee ... \vee p_k \vdash \bot = Val(\mathcal{F} \vdash \neg p_1) \textbf{ and } \dots \textbf{ and } Val(\mathcal{F} \vdash \neg p_k)$$

(where p_i is any literal, i.e. an atom or the negation of an atom)

Suppose that $\mathcal{F}$ is a finite set. Then, in fact, these equalities define the function *Val* recursively on cardinality of the set $\mathcal{F}$. Consequently these equalities suggest how to calculate $Val(\mathcal{F} \vdash \psi)$. In other words we have the following assertion:

(11) *If $\mathcal{F}$ is a finite set, then one can effectively calculate $Val(\vdash\psi)$, i.e. to establish whether or not ψ is a logical consequence of the set $\mathcal{F}$.*

Next we will prove the following basic theorem.

THEOREM 1. (Completeness) *Let $\mathcal{F}$ be a set of clauses and ψ a literal or the symbol $\bot$. Then: ψ is a logical consequence of the set $\mathcal{F}$ if and only if starting with $\mathcal{F} \vdash \psi$ and applying the rules* (R1)–(R4) *a finite number of times one can obtain the sequent $\vdash\ \top$.*

Proof. The *if* part follows immediately from Lemma 3. To prove *only-if* part suppose now that ψ is a logical consequence of the set $\mathcal{F}$. Then ψ is a logical consequence of some *finite* subset $\mathcal{A}$ of the set $\mathcal{F}$ (since every formal proof is finite). Next, by (11) we conclude that starting by the sequent $\mathcal{A} \vdash \psi$ and applying the rules (R1)–(R4) a finite number of times one can obtain the sequent $\vdash\ \top$. Consequently, also starting with the sequent $\mathcal{F} \vdash \psi$ and applying the rules (R1)–(R4) a finite number of times one can obtain the sequent $\vdash\ \top$. The proof is complete.

Remark 1. Theorem 1 can be extended to the case when ψ is some compound disjunction, as $\neg p \vee q \vee \neg r$. Namely, denoting $\mathcal{F}$ by $\mathcal{F}(p,q,r)$ we have the following equivalence: $\mathcal{F}(p,q,r) \vdash \neg p \vee q \vee \neg r \ \longleftrightarrow\ \mathcal{F}(\top,\bot,\top) \vdash \bot$.

Now we are going step-by-step to describe a procedure, denoted by *PL*, by which one can search for the answer to a question of the form $\mathcal{F} \vdash \psi$. Rules (R1)–(R4) are basic part of the procedure *PL* (some other rules will be introduced in the sequel). In some cases, for instance if $\mathcal{F}$ is a finite set, the procedure *PL* will be a genuine algorithm. We emphasize that in general this procedure will be somewhat similar to Prolog algorithm. Notice that the name of the procedure comes from the initials of the words: 'Prolog Like'. In the sequel we will give details of the procedure *PL*.

First, if $\mathcal{F}$ is not a finite set and ψ is not a logical consequence of $\mathcal{F}$ how can we infer the sequent $\vdash \bot$? Obviously, this can be very difficult. Sometimes we can use the following lemma.

LEMMA 4. ($\bot$-answer) *Let $\mathcal{F}$ be a consistent set of clauses, which does not contain a literal ψ, but may contain its negation $\neg\psi$. Then ψ is not a logical consequence of $\mathcal{F}$.*

Proof. By assumption $\mathcal{F}$ has no clause of the form $\psi \vee ...$, but may have a clause of the form $\neg\psi \vee ...$. Using this fact and the assumption $\mathcal{F}$ is consistent we conclude that $\mathcal{F}$ has at least one model in which the literal ψ has value *false*. Thus, ψ cannot be a logical consequence of $\mathcal{F}$. The proof is complete.

As we shall see now this lemma is used in Prolog algorithm. For instance, let us calculate Val of the following clause

(*) $a \vee \neg b \vdash c$

Applying (R2), i.e. equivalence (6), we obtain the clause $a \vee \neg b \vdash \perp$.

Was this step a necessary one? As we know, 'seeing by Prolog eyes' we can put the following question *Does c follow from the Prolog clause a : − b ?* and the answer is *No.* The Prolog reason for that is: *This c can not be unified with 'the head' of that clause.*

As a matter of fact, the genuine reason is implied by Lemma 4, since any set of positive Horn formulas is consistent. So, by Lemma 4 the $Val(*)$ is *false.*

Based on this remark and Lemma 4 we introduce the additional rule

(R5) $\vdash \perp \longleftarrow \mathcal{F} \vdash \psi$, *($\mathcal{F}$ is consistent, ψ is any literal not occurring in $\mathcal{F}$)*

As we have already said, the rule (R5) can be used if $\mathcal{F}$ is a set of positive Horn formulas.

Second, let us pay attention to rule (R3). To justify its presence in the procedure *PL* we will prove the following lemma.

Lemma 5. (Relevancy property) *Let in the sequent*

$$\mathcal{F},\ p \vee f_1,\ p \vee f_2, \ldots, \neg p \vee ff_1, \neg p \vee ff_2, \ldots \vdash p$$

p be a literal and f_i, ff_j, $\mathcal{F}$ do not contain p. Then the following implication holds

(*) $\mathcal{F}, p \vee f_1,\ p \vee f_2, \ldots, \neg p \vee ff_1,\ \neg p \vee ff_2, \ldots \vdash p \longrightarrow \mathcal{F}, p \vee f_1,\ p \vee f_2, \ldots \vdash p$

Proof. The implication (*) follows from the fact that according to (6) we have the following equivalences

$$\mathcal{F}, p \vee f_1,\ p \vee f_2, \ldots, \neg p \vee ff_1, \neg p \vee ff_2, \ldots \vdash p \longleftrightarrow \mathcal{F}, f_1, f_2, \ldots \vdash \perp$$

$$\mathcal{F}, p \vee f_1,\ p \vee f_2, \ldots, . \vdash p \longleftrightarrow \mathcal{F}, f_1, f_2, \ldots \vdash \perp$$

The proof is complete.

Further, notice that rule (R3) is not deterministic. Namely, suppose that we have a sequent $\mathcal{F}, p \vee f_1, p \vee f_2, ... \neg p \vee ff_1, \neg p \vee ff_2, \ldots \vdash p$ where p, and $\neg p$ do not occur in $\mathcal{F}$ and that

(δ) $p \vee f_1, p \vee f_2, \ldots$

are all clauses containing p and suppose that they are arranged in the same order as in (δ). After applying rule (R2) we get the following new sequent $\mathcal{F}, f_1, f_2, \ldots \vdash \perp$. Now we should apply rule (R3). The clauses f_1, $f_2, \ldots$ will be called *relevant.* We introduce the following convention in the procedure *PL*:

(12) *Rule* (R3) *will be first applied to the relevant clause*[5] f_1*, after that, if needed, to the relevant clause* f_2*, and so on.*

[5]But, if such a clause is $\perp$, then we apply rule (R1)

Example 2. Calculate the given sequent, written in Prolog style.[6)]

(Δ) $a :- b, c.\ b :- c.\ a :- d.\ d :- e.\ e. \vdash a$

Answer. Using rules (R1)–(R4), i.e. equivalences (6)–(9) we have the following equivalence chain[7)]

$a :- b, c.\ b :- c.\ a :- d.\ d :- e.\ e. \vdash a$

$\longleftrightarrow\ a \vee \neg b \vee \neg c,\ b \vee \neg c,\ a \vee \neg d,\ d \vee \neg e,\ e \vdash\ a$

(First we have translated Prolog clauses to the ordinary clauses)

(P) $\longleftrightarrow\ \underline{\neg b \vee \neg c, \neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash \bot$ (Using (6). We have underlined the relevant clauses and put them at the very beginning)

$\longleftrightarrow$ Cl1 and Cl2, where Cl1, Cl2 are the abbreviations for the following clauses: $\underline{\neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash\ b$; $\underline{\neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash\ c$ respectively. Further, for the clause Cl1 we have

Cl1 $\longleftrightarrow\ \underline{\neg c},\ \underline{\neg d},\ d \vee \neg e,\ e\ \vdash\ \bot$ (Using (6)). Notice that we put the new relevant clause $\neg c$ in front of the old relevant clause.

(P') $\longleftrightarrow\ \underline{\neg d},\ d \vee \neg e,\ e\ \vdash\ c$ (Using (7))

(P1) $\longleftrightarrow\ \underline{\neg d},\ d \vee \neg e,\ e\ \vdash\ \bot$ (Using (6))

$\longleftrightarrow\ d \vee \neg e,\ e\ \vdash\ d$ (Using (7))

$\longleftrightarrow\ \underline{\neg e},\ e\ \vdash\ \bot$ (Using (6))

$\longleftrightarrow\ e\ \vdash e$ (Using (7))

$\longleftrightarrow\ \bot\ \vdash\ \bot$ (Using (6))

(P1') $\longleftrightarrow\ \top$ (Using (8))

So, $Val(Cl1)$ is *true.* Consequently we should calculate $Val(Cl2)$ too. We have:

Cl2 $\longleftrightarrow\ \underline{\neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash\ c$

(P2) $\longleftrightarrow\ \underline{\neg d},\ d \vee \neg e,\ e \vdash \bot$

$\longleftrightarrow\ d \vee \neg e,\ e \vdash d$

$\longleftrightarrow\ \underline{\neg e},\ e \vdash \bot$

$\longleftrightarrow\ e \vdash e$

$\longleftrightarrow\ \bot \vdash \bot$

(P2') $\longleftrightarrow\ \vdash \top$

The $Val(Cl2)$ is also *true*, such that $Val(\Delta)$ is *true.*

Notice that despite of the fact that we managed to calculate $Val(\Delta)$ by using the procedure *PL* we had to repeat some steps (see parts from (P1) to (P1') and

[6)]Notice that the labels (P), (P'), (P1), (P1'), (P2), (P2') occurred in this example below are used to mark some steps of 'calculation'.

[7)]We use the equivalence $\longleftrightarrow$ symbol instead of the implication symbol $\longleftarrow$ on purpose. Notice that in the sequel we shall do it regularly.

from (P2) to (P2')). However, if we had first used the second relevant clause, i.e. the clause $\neg d$, then we would have the following very short calculation:

$$\begin{aligned}
&a \vee \neg b \vee \neg c,\ b \vee \neg c,\ a \vee \neg d,\ d \vee \neg e,\ e \vdash a \\
&\quad\longleftrightarrow\ \underline{\neg b \vee \neg c, \neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash \perp \\
&\quad\longleftrightarrow\ \underline{\neg b \vee \neg c},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash d \\
&\quad\longleftrightarrow\ \underline{\neg e, \neg b \vee \neg c},\ b \vee \neg c,\ e \vdash \perp \\
&\quad\longleftrightarrow\ \underline{\neg b \vee \neg c},\ b \vee \neg c,\ e \vdash e \\
&\quad\longleftrightarrow\ \underline{\neg b \vee \neg c},\ b \vee \neg c,\ \perp \vdash \perp \\
&\quad\longleftrightarrow\ \vdash \top
\end{aligned}$$

Let us again pay attention to steps (P1)–(P1') and (P2)–(P2'). In step (P1) we have the sequent $\neg d,\ d \vee \neg e,\ e \vdash\ \perp$, which is by (6) equivalent to $a \vee \neg d,\ d \vee \neg e,\ e \vdash\ a$. The use of this sequent practically means the following. We have failed to prove a by the clauses $a : -b, c.$ and $b : -c.$ and consequently we want to apply the new a-clause (i.e. a clause of the form $a\ :\ -\ldots$). In other words we are going to apply the new relevant clause, i.e. the clause $\neg d$. Accordingly instead of making steps (P1)–(P1') and (P2)–(P2') it would be better to do the following in the spirit of general Prolog algorithm:

(13) *First we should find the place*[8] *at which the previous relevant clause*[9] *was activated. Second, this clause should be omitted, third from this place we should continue the procedure PL by employing the new relevant clause.*

The place in question is P', i.e. the place in which we have introduced the abbreviations Cl1, Cl2. According to (13) and assuming that we have done the previous calculation up to the place P', from this place we continue the calculation as follows:

(P') $\longleftrightarrow \underline{\neg d},\ d \vee \neg e,\ e \vdash\ c$

In fact at this place we leave the sequent $\underline{\neg d},\ d \vee \neg e,\ e \vdash\ c$, and consequently we also leave the whole **and-expression** C1 **and** C2 (that is very important to emphasize).

$\longleftrightarrow b \vee \neg c,\ d \vee \neg e,\ e \vdash\ d$

Execution of the second require in (13) can be imagined in the following way. Suppose that we use **a stack**[10] whose 'members' are the sequents which are 'links' in the equivalence chain. Then in such a stack we should go to its member which is $\underline{\neg b \vee \neg c, \neg d},\ b \vee \neg c,\ d \vee \neg e,\ e \vdash \perp$ and replace it by the following one $\underline{\neg d},\ \overline{b \vee \neg c,\ d \vee} \neg e,\ e \vdash \perp$

$\longleftrightarrow\ \underline{\neg e},\ b \vee \neg c,\ e\ \vdash\ \perp$

$\longleftrightarrow\ b \vee \neg c,\ e\ \vdash\ e$

[8] Notice, that in general this is not a trivial problem. In Example 3 we will explain an idea by which one can solve the problem.

[9] Here this is the clause $\neg b \vee \neg c$

[10] Such an idea is typical for Computer Science.

$\longleftrightarrow\ b \vee \neg c,\ \bot\ \vdash\ \bot$

$\longleftrightarrow\ \vdash\ \top$

But, how can we know at which step we may apply (13)? In order to give an answer, suppose that in certain step we have a sequent $\Phi \vdash \psi$, where Φ is a set of some clauses and ψ is a literal. Then we introduce the following definition

(14) *The sequent* $\Phi \vdash$ *psi is* **replaceable** *if* ψ *does not occur*[11] *in* Φ*, and also at least one clause of* Φ *is a relevant clause.*

The sequents $a,\ b \vee \neg c \vdash \neg d$; $a,\ b \vee d,\ c \vee d \vdash \neg d$ and $a,\ b \vee \neg d \vdash d$ are examples of replaceable sequents, supposing that the left hand side of them contains at least one relevant clause. Further, if we, for instance, have to calculate *Val* of the sequent $a,\ b \vee c \vdash d$, then this sequent is not a replaceable sequent. But if the same sequent appears during a calculation of some other sequent and a or $b \vee c$ became a relevant clause, then the sequent $a,\ b \vee c$ is a replaceable sequent.

In connection with (14) we have the following criterion for applying (13):

(15) *In the procedure PL we apply* (13) *just in case when in certain step we have a replaceable sequent*

However, in the opposite case, i.e. when we do not have a replaceable clause, then we first apply equivalence (6) and after that we perform the following:

(16) *We put each new relevant clause in front of the old relevant clauses*

For instance, suppose that we want to find *Val* of the sequent

(*) $$a : -b.\ b : -c.\ c.\ a : -d. \vdash a$$

i.e., of the sequent $a \vee \neg b,\ b \vee \neg c,\ c,\ a \vee \neg d \vdash a$. Then we have the following calculation (i.e. equivalence chain)

(*) $\longleftrightarrow\ \underline{\neg b, \neg d},\ b \vee \neg c,\ c \vdash \bot$

$\longleftrightarrow\ \underline{\neg d},\ b \vee \neg c,\ c \vdash b$

$\longleftrightarrow\ \underline{\neg c, \neg d},\ c \vdash \bot$

We apply (16). The reason, stated in Prolog style, reads: *Until now in the attempt to prove a we followed this way. To prove a we want first to prove b. But to prove b we should prove c. Just for that reason the relevant clause* $\neg c$ *should be put at the very beginning, i.e., before the clause* $\neg d$*.*

$\longleftrightarrow\ \underline{\neg d},\ c \vdash c$

$\longleftrightarrow\ \underline{\neg d},\ \bot \vdash \bot$

$\longleftrightarrow\ \vdash \top$

So, the final result is: $Val(*)$ is *true.*

In the following Example we will mainly explain more details how to use rule (13), i.e. how to find the corresponding place. For this purpose we shall use two things: labels like [1], [2], ... to mark some clauses.

[11] However, $\neg\psi$ may occur in Φ

Example 3. Calculate the given sequent, where some of them are written in Prolog style.

1) $p :-q.\ p :-r.\ p :-s.\ s. \vdash\ p$

2) $p :-a, b.\ a :-c.\ a :-d.\ d.\ p :-r, d.\ r :-d.\ \vdash\ p$

3) $p :-q.\ p :-r.\ q :-a.\ r :-b.\ a \vee\ b \vdash p$

4) $p :-q, r.\ q :-a.\ q :-b, r.\ r :-s.\ s.\ p :-t.\ t. \vdash p$

Solution. 1) We have the following equivalence chain

1) $\longleftrightarrow\ p \vee \neg q,\ p \vee \neg r,\ p \vee \neg s,\ s \vdash p$

$\longleftrightarrow\ [1]\neg q,\ [1]\neg r,\ [1]\neg s,\ s \vdash \bot$

(We have three relevant clauses: $\neg q, \neg r$ and $\neg s$. In front of them we put [1]. This serves 'as an address' by which one memorizes the place at which these clauses are involved. In general, for such a place we will say that it is *origin-place* of the related address. Notice also that we have not underlined the relevant clauses, for they are enough marked by the related addresses. In the sequel we will do in the same way.)

$\longleftrightarrow\ [1]\neg r,\ [1]\neg s,\ s \vdash\ [1]\ q$

(Notice, that because q originates from the relevant clause $[1]\neg q$ we put the address in front of it. Further, we have a replaceable sequent; so we apply (13) going to the origin-place of the address [1]. This is just the previous 'link' in the equivalence chain.)

$\longleftrightarrow\ [1]\neg s,\ s \vdash\ [1]\ r$

(Again we have a replaceable sequent; so we apply (13) going again to the origin-place of the address [1].)

$\longleftrightarrow\ s \vdash\ [1]\ s$

$\longleftrightarrow\ \bot \vdash\ \bot$

$\longleftrightarrow\ \vdash\ \top$

So, we have: $Val(1))\ =\ $ *true.*

2) We have the following equivalence chain

2) $\longleftrightarrow\ p \vee \neg a \vee \neg b,\ a \vee \neg c,\ a \vee \neg d,\ d,\ p \vee \neg r \vee \neg d,\ r \vee \neg d \vdash\ p$

$\longleftrightarrow [1]\neg a \vee \neg b,\ [1]\neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ \bot$

(We have two relevant clauses: $\neg a \vee \neg b$ and $\neg r \vee \neg d$. In front of them we put the address [1].)

$\longleftrightarrow\ [1]\neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ a$

and $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

(Notice, that because a and b originate from $[1]\neg a \vee \neg b$ we put the address [1] in front of both of them.)

$\longleftrightarrow\ [2]\neg c,\ [2]\neg d,\ [1]\neg r \vee \neg d,\ d,\ r \vee \neg d \vdash\ \bot$

(We have two new relevant clauses, for which we use a new address [2].)

and $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

$\longleftrightarrow\ [2]\neg d,\ [1]\neg r \vee \neg d,\ d,\ r \vee \neg d \vdash\ [2]\ c$

(Here c is replaceable. We go to the origin-place of the address [2] and apply (13).)

and $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

$\longleftrightarrow\ [1]\neg r \vee \neg d,\ d,\ r \vee \neg d \vdash\ [2]\ d$

and $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

$\longleftrightarrow\ \bot \vdash\ \bot$ **and** $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

$\longleftrightarrow\ \vdash\ \top$ **and** $[1]\ \neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

$\longleftrightarrow\ [1]\neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ b$

(Here b is replaceable. We go to the origin place of the address [1], i.e., to the sequent $[1]\neg a \vee \neg b,\ [1]\neg r \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ \bot$ and after omitting the clause $[1]\neg a \vee \neg b$ we apply (6) to the relevant clause $[1]\neg r \vee \neg d$.)

$\longleftrightarrow\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ r$

and $a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ [3]\neg d,\ a \vee \neg c,\ a \vee \neg d,\ d\ \vdash\ \bot$

(We have a new relevant clause $\neg d$. We gave to it the address [3].)

and $a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ a \vee \neg c,\ a \vee \neg d,\ d\ \vdash\ [3]\ d$

and $a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ a \vee \neg c,\ \bot\ \vdash\ \bot$ **and** $a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ \vdash\ \top$ **and** $a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ a \vee \neg c,\ a \vee \neg d,\ d,\ r \vee \neg d \vdash\ [1]\ d$

$\longleftrightarrow\ a \vee \neg c,\ \bot \vdash\ \bot$

$\longleftrightarrow\ \vdash\ \top$

So, the final result is $Val(2)) =$ *true.*

3) We have the following equivalences

3) $\longleftrightarrow\ p \vee \neg q,\ p \vee \neg r,\ q \vee \neg a,\ r \vee \neg b,\ a \vee b \vdash p$

$\longleftrightarrow\ [1]\neg q,\ [1]\neg r,\ q \vee \neg a,\ r \vee \neg b,\ a \vee b \vdash \bot$

(We have two relevant clauses, in front of them we put [1].)

$\longleftrightarrow\ [1]\neg r,\ q \vee \neg a,\ r \vee \neg b,\ a \vee b \vdash\ [1]\ q$

$\longleftrightarrow\ [2]\neg a,\ [1]\neg r,\ \ r \vee \neg b,\ a \vee b \vdash\ \bot$

(We have a new relevant clause. We gave to it the address [2] and also put it in front of the old relevant clauses)[12)]

$\longleftrightarrow\ [1]\neg r,\ r \vee \neg b,\ a \vee b \vdash\ [2]\ a$

$\longleftrightarrow\ [3]\ b,\ [1]\neg r,\ r \vee \neg b \vdash\ \bot$

[12)]The reason is obvious: to prove q we employ the clause $q : -a$, i.e. $q \vee \neg a$.

(Again a new relevant clause. New address is [3].)

$\longleftrightarrow$ $[1]\neg r,\ r \vee \neg b \vdash\ [3]\neg b$

$\longleftrightarrow$ $[4]\ r,\ [1]\neg r, \vdash\ \bot$

(Notice that the clause r (with the new address) is the first one.)

$\longleftrightarrow$ $[1]\neg r, \vdash\ [4]\neg r$

$\longleftrightarrow$ $\bot \vdash\ \bot$

$\longleftrightarrow \vdash\ \top$

So, the answer is: $Val(2))$ is *true.*

4) We have the following equivalence chain

4) $\longleftrightarrow$ $p \vee \neg q \vee \neg r,\ q \vee \neg a,\ q \vee \neg b \vee \neg r,\ r \vee \neg s,\ s,\ p \vee \neg t,\ t \vdash p$

$\longleftrightarrow$ $[1]\neg q \vee \neg r,\ [1]\neg t,\ q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash \bot$

$\longleftrightarrow$ $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ q$

and $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow$ $[2]\neg a,\ [2]\neg b \vee \neg r,\ [1]\neg t, r \vee \neg s, s,\ t \vdash\ \bot$

(Notice that now in front of $\neg t$ we put two new relevant clauses.)

and $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow$ $[2]\neg b \vee \neg r,\ [1]\neg t, r \vee \neg s, s,\ t \vdash\ [2]\ a$

(This a is replaceable. We should go to origin place of the address [2], and apply (13). Notice that after omitting the clause $\neg a$ at the origin place of [2] the following and-expression

$[2]\neg b \vee \neg r,\ [1]\neg t, r \vee \neg s, s,\ t \vdash\ \bot$

and $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

holds.)

and $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [2]\ r$

$\longleftrightarrow$ $[1]\neg t, r \vee \neg s, s,\ t \vdash\ [2]\ b$ **and** $[1]\neg t, r \vee \neg s, s,\ t \vdash\ [2]\ r$

and $\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ r$

(Consider the first clause $[1]t, r \vee \neg s, s, t \vdash\ [2]b$. This b is replaceable. Therefore we go to the origin place of [2] at which the expression

$[2]\neg b \vee \neg r,\ [1]\neg t, r \vee \neg s, s,\ t \vdash\ \bot$

and $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

stands. Now we first omit the clause $[2]\neg b \vee \neg r$ and apply (7) to the relevant clause $[1]\neg t$. So, we obtain the next 'link':)

$\longleftrightarrow$ $r \vee \neg s, s,\ t \vdash\ [1]\ t$ **and** $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow$ $r \vee \neg s, s,\ \bot \vdash\ \bot$ **and** $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow \vdash \top$ **and** $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow$ $[1]\neg t, q \vee \neg a, q \vee \neg b \vee \neg r, r \vee \neg s, s,\ t \vdash\ [1]\ r$

$\longleftrightarrow$ $[3]\neg s,\ [1]\neg t, q \vee \neg a, s,\ t \vdash\ \bot$

$\longleftrightarrow$ $[1]\neg t, q \vee \neg a, s,\ t \vdash [3]\ s$

$\longleftrightarrow$ $[1]\neg t, q \vee \neg a, \bot,\ t \vdash\ \bot$

$\longleftrightarrow$ $\vdash\ \top$

So the result is: $Val(4)) = \mathit{true}$.

Let us now consider the following Prolog program

(17) $\beta(f(X)) : -\alpha(X).$
$\alpha(a).$

(a and f are symbols representing a constant, a function respectively, while α and β represent relations) and put the question $? - \beta(Y)$. The answer is *yes.* Using a and f we can build the corresponding *Herbrand universe* [1], whose elements are *ground* terms, as

$a,\ f(a),\ f(f(a)),\ f(f(f(a))), \ldots$

Denote by Prop((17)) ('Propositional set of (17)') the set of all clauses of the form (17) where X is replaced by various elements of the Herbrand universe. This set is a set of propositional clauses, whose atoms have the following forms: $\alpha(t1), \beta(t2)$ where $t1,\ t2$ can be any elements of Herbrand universe. Prolog algorithm in fact deals with such a set Prop((17)). In connection with this we point out that the meaning of the question $? - \beta(Y)$ is the following:

Can one using Prolog algorithm infer a consequence of the form $\beta(Y)$, where Y may be some element of the Herbrand universe. Notice that Y has a status of an *unknown*, while X is a variable (whose value comes from the Herbrand universe)

As it is well known, in search of such a Y one uses a *unification* algorithm. Similarly, in the procedure *PL* we also use such an algorithm. Here we explain how by *PL* one can treat the above Prolog question. Namely, we have the following equivalence chain

$\beta(f(X)) : -\alpha(X), \alpha(a) \vdash \beta(Y)$

(It is supposed that X 'runs' over the Herbrand universe, i.e. that the left hand side of this sequent is Prop((17))).

$\longleftrightarrow$ $\beta(f(X)) \vee \neg\alpha(X), \alpha(a) \vdash \beta(Y_0)$

(X runs over Herbrand universe. We replaced Y by Y_0 in order to emphasize that Y is not a variable, but an unknown.)

$\longleftrightarrow$ $[1]\neg\alpha(X_0),\ \beta(f(X)) \vee \neg\alpha(X), \alpha(a) \vdash\ \bot)$

(Y_0 obtains the value $f(X_0)$, where X_0 is a new unknown, X runs over Herbrand universe. Notice that we have rewritten the clauses $\beta(f(X)) \vee \neg\alpha(X), \alpha(a)$ (see Remark 2 below))

$\longleftrightarrow$ $\beta(f(X)) \vee \neg\alpha(X), \alpha(a) \vdash\ [1]\alpha(X_0)$

(X_0 obtains the value a, consequently the value of Y_0 is $f(a)$, X runs over Herbrand universe)

$\longleftrightarrow\ \beta(f(X)) \vee \neg\alpha(X),\ \bot \vdash\ \bot$

$\longleftrightarrow\ \vdash\ \top$

Thus, the answer is *true*, and the desired value of Y, i.e. of Y_0 is $f(a)$.

Remark 2. In Lemma 1 we have the equivalence (6) (i):

$\mathcal{F}, \phi_1(p), \phi_2(p), \ldots \vdash p\ \longleftrightarrow \mathcal{F}, \phi_1(\bot), \phi_2(\bot), ... \vdash\ \bot$

It is interesting that besides this equivalence we also have another one

$\mathcal{F}, \phi_1(\bot), \phi_2(\bot), \ldots \vdash \bot\ \longleftrightarrow \mathcal{F}, \phi_1(p), \phi_2(p), \ldots\ \phi_{r1}(\bot), \phi_{r2}(\bot), \ldots \vdash \bot$
where $r1, r2, \ldots \in \{1, 2, \ldots\}$

which can be easily proved.

Now we are going to generalize the idea applied in the above example. So, let Ψ be a set of clauses ([1]) and ϕ a clause. How by use of PL one can:

(18) Find $Val(\Psi \vdash \phi)$

We allow that ϕ contains some *unknowns* denoted for instance by $X_0, Y_0, \ldots$. But what to do if ϕ contains some variables? One such example is the following: $a(X) \vdash\ a(X)$ where X is a variable. In this particular example we can solve problem (18) as follows

$a(X) \vdash a(X)$

$\longleftrightarrow\ a(X) \vdash a(c)$

(Here c is a new constant symbol in respect with $a(X) \vdash a(X)$, which means that c does not occur in the sequent $a(X) \vdash a(X)$. See Remark 3 below. Notice that now Herbrand universe is the singleton $\{c\}$.)

$\longleftrightarrow\ \bot, a(X) \vdash \bot$

(The substitution $X --> c$ unifies the atoms $a(c)$ and $a(x)$)

Remark 3. In general, if c is a new constant with respect to Ψ and ϕ, then the following logical equivalence $\Psi \vdash \phi(X) \longleftrightarrow\ \Psi \vdash \phi(c)$ holds.

Having in mind this Remark any problem of the type (18) can be reduced to some problem of the type (18), but where ϕ does not contain any variable. Further, according to Remark 1 the obtained problem can be reduced to a problem of type (18) in which ϕ is $\bot$. So finally we conclude that in the study of problems of type (18) we may suppose that ϕ is a literal or the symbol $\bot$.

What is the genuine reason for the use of the notion of an 'unknown'? To see that consider this example:

Assume: $a(1), a(2)$ hold, and then put the question: ? $(\exists X)\ a(X)$ (In words: Is there an X such that $a(X)$ is true). As a matter of fact in this question we encounter the *unknown* X_0, whose value must be an element of the Herbrand universe, i.e. of the set $\{1,2\}$. In other words the question reduces to a problem of type (18), with: Ψ is $a(1),\ a(2)$, and ϕ is $a(X_0)$.

As it is well known in Prolog algorithm one of the most important components is *the backtracking procedure*. A similar fact holds for the procedure *PL*. Why does Prolog use the backtracking idea? The essential reason is the following one:

> It may happen that at some step certain unknown, say X, gets a value which is not a good one. Consequently, roughly said, we should go back to the place, where this unknown got this value, and try to find a new value for it. After that we continue the Prolog algorithm.

In the procedure *PL* we proceed just in such a way. Here is an example:

Example 4. Find the truth value of the given sequent

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ c(X_0)$

Solution. In order to make the use of a backtracking idea possible, we will mark the places in which unknowns get their values. Accordingly we have the following equivalence chain:

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ c(X_0)$

(**The first value-place for** X_0. Here we will connect $c(X_0)$ with the clause $c(X) \vee \neg a(X) \vee b(X)$, i.e., to unify $c(X_0)$ with $c(X)$. Another possibility is to connect $c(X_0)$ with $c(77)$, which means that X_0 will get the value 77.)

$\longleftrightarrow\ [1]\neg a(X_0) \vee b(X_0),\ a(1),\ a(2),\ b(1), \neg b(2),$
$c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ \perp$

$\longleftrightarrow\ a(1),\ a(2),\ b(1), \neg b(2),\ \ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1]\ a(X_0)$

(**The second value-place for** X_0. Now X_0 will get the value 1)

and

$a(1),\ a(2),\ b(1), \neg b(2),\ \ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1]\neg b(X_0)$

$\longleftrightarrow\ \perp,\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ \perp)$

and

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1]\neg b(X_0)$

$\longleftrightarrow \vdash\ \top$ **and**

$a(1),\ a(2),\ b(1), \neg b(2),\ \ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1]\neg b(X_0)$

$\longleftrightarrow\ a(1),\ a(2),\ b(1), \neg b(2),\ \ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1]\neg b(X_0)$

$\longleftrightarrow\ a(1),\ a(2), \neg b(2),\ \ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ \perp$

(It is not difficult to see that the last sequent is *false*. Maybe the value 1 for X_0 is not an appropriate value? Therefore we will go to the last value-place of X_0, i.e. to the second value-place, in order to give X_0 some new value. To be clearer we will first rewrite the 'link' in that place.)

$\longleftrightarrow\ a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ [1]\ a(X_0)$

and

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1] \neg b(X_0)$

(Now X_0 will take the value 2).

$\longleftrightarrow\ a(1),\ \bot,\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ [1]\ \bot$

and

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1] \neg b(X_0)$

$\longleftrightarrow \vdash\ \top$ **and**

$a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77) \vdash\ [1] \neg b(X_0)$

$\longleftrightarrow\ a(1),\ a(2),\ b(1), \neg b(2),\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ [1] \neg b(X_0)$

(Now X_0 has the value 2)

$\longleftrightarrow\ a(1),\ a(2),\ b(1),\ \bot,\ c(X) \vee \neg a(X) \vee b(X),\ c(77)\ \vdash\ \bot$

$\longleftrightarrow \vdash\ \top$

So, finally we have result *true* and the unknown X_0 has the value 2. Suppose the contrary, i.e. that the result was *false*. In such a case in the second value-place of X_0 there is no new value for X_0, and consequently we would go back to the first value-place of X_0 and then X_0 would get the value 77, etc.

References

1. J.W. Lloyd, *Foundations of Logic Programming,* Springer-Verlag, 1984
2. E. Mendelson, *Introduction to Mathematical Logic,* Van Nostrand, Princeton, 1979.
3. S.B. Prešić, *Generalizing logic programming to arbitrary set of clauses*, Sci. Rev. Belgrade 19/20 (1996), 75–81

Matematički fakultet
Studentski trg 16
11001 Beograd, p.p. 550
Yugoslavia

(Received 08 09 1996)
(Revised 27 12 96)