1

Extensions of Deductive Concept in Logic Programming and Some Applications

Ivana Berkovic, Biljana Radulovic and Petar Hotomski
University of Novi Sad, Technical faculty "Mihajlo Pupin" Zrenjanin
Serbia

1. Introduction

Automated reasoning systems are computer programs which have certain "intelligent" components and can be used as: shells of expert systems, dialog systems, human language translators, educational software etc. Inferring the conclusion in such systems is often based on resolution refutation method from a certain set of rules and facts. In order to infer a conclusion, these systems, apart from negating set query, use rules for deducting facts in axiom form and working fact base. Automated reasoning systems give answers to a set query which depend upon the knowledge base and fact base (Hotomski, 2004). In this sense, "automated reasoning is concerned with the discovery, formulation, and implementation of concepts and procedures that permit the computer to be used as a reasoning assistant" (Wos, 1985).
The developing of automated reasoning results into the developing of logic programming languages, especially PROLOG. In this paper the advantages and applications of changing one system for automated reasoning by the other are described. The determinate resolution system for automated theorem proving ATP (OL-resolution with marked literals) is especially put into the base of prolog-like language, as the surrogate for the concept of the negation as definite failure (SLDNF resolution) in PROLOG.

2. Ordered Linear Resolution as the Foundation of Automatic Theorem Proving

The most popular method for automatic theorem proving is the resolution method, which was discovered by J. A. Robinson in 1965 (Barr et al., 1982, Gevarter, 1985, Wos et al., 1992). Since 1965, many resolution forms and techniques have been developed because the pure resolution rule was unable to handle complex problems (Hotomski, 2004).
Here is used the general automatic method for determining if a theorem (conclusion) **A** follows from a given set of premises (axioms) **F**:

$$\mathbf{F} \vdash \mathbf{A}.$$

Each formula will be transformed to the *clauses form*. The *clauses* have the form:

$$\mathbf{L_1 \vee L_2 \vee \ ... \vee L_m}$$

where L_i are literals. The symbol for disjunction is: $\vee$.
The *literals* L_i have the form: $\mathbf{P(t_1, t_2, ... , t_n)}$ or $\mathbf{\neg P(t_1, t_2, ... , t_n)}$, where **P** is predicate symbol, ti is term, $\neg$ is negation. The literal $\mathbf{P(t_1, t_2, ... , t_n)}$ is called *positive* literal, the literal $\mathbf{\neg P(t_1, t_2, ... , t_n)}$ is called *negative* literal.
Resolution method is a syntactic method of deduction. *Reduction ad absurdum* is in the basis of resolution method:

$$\mathbf{F \vdash A \quad iff \quad F \cup \{ \neg A \} \vdash contradiction} .$$

Resolution rule will be applied on the set of clauses - axioms which was expanded by negating the desired conclusion in clause form.

Ordered Linear (OL) resolution rule with marked literals (Hotomski & Pevac, 1991, Hotomski, 2004, Kowalski & Kuchner, 1971) increases efficiency and doesn't disturb completeness of pure resolution rule.
The generating process of OL-resolvent from central clause (**d1**) and auxiliary clause (**d2**):

1. Redesignate variables (without common variables in the clauses).
2. Determine universal unificator Θ for last literal of **d1** and **k**-literal (**k=1,2,...**) of **d2** (if it exists for some k, else it is impossible to generate OL-resolvent for specification clauses).
3. Create resolvent with marked last literal in $\mathbf{d1\Theta}$ and add the rest of clause $\mathbf{d2\Theta}$ without **k**-literal ($\mathbf{d1\Theta}$ and $\mathbf{d2\Theta}$ are clauses, which were formed by universal unificator Θ applied on **d1** and **d2**, respectively).
4. Eliminate identical non-marked literals and tautology examination (tautologies are not memorized).
5. The Shortening Operation (delete all ending marked literals)
6. The Compressing Operation (delete the last non-marked literal, which is complemented in relation to negation, with some marked literal for unificator λ).
7. Repeat steps: 5 and 6 until the empty clause is got, or the Compressing Operation is not applied on the last non-marked literal.

The rule of OL-resolution with marked literals is separated in two parts: ***in-resolution*** and ***pre-resolution***. The steps: 1 - 5 are represented in-resolution. The steps: 6 - 7 are represented pre-resolution. Mid-resolvents are the products of in-resolution and without their memorizing, the completeness of the method can be lost. This modification of Ordered Linear resolution rule is served as the base for development of the system for automatic theorem proving ATP.

3. ATP System for Automatic Theorem Proving

ATP is a system for automatic theorem proving (Berkovic, 1994), which is implemented on personal computer by Pascal language. The rule of Ordered Linear Resolution with marked literals presents the system base. The system is developed at Technical Faculty "Mihajlo Pupin" in Zrenjanin. ATP is projected for scientific - researching, teaching and practical purpose. ATP system disposes three search strategies: breadth-first, depth-first and their combination. Numerously experiments with ATP system show that depth-first strategy is

the most efficient. In depth-first search, a new node is generated at the next level, from the one current, and the search is continuing deeper and deeper in this way until it is forced to backtracking.

The main characteristics of ATP system:

This system presents a complete logical deductive base: *the clauses-set is unsatisfied (contradictory) iff the empty clause is generated by finite use of the resolution rule*. So, the proof of conclusion **A** is completed ($\mathbf{F} \vdash \mathbf{A}$) when the empty clause is generated by the resolution from clauses-set $\mathbf{F} \cup \{\neg \mathbf{A}\}$.

Besides the theoretical completeness of the system, it has the satisfying practical efficiency limited by the space-time computer resources.

The first-order logic is the form of representation in ATP system (each formula is transformed into the clause form). This deductive base has no restriction in Horn clause (expansions concerning Horn clauses) and it allows the logical treatment of negation (escaping negation treatment as a definite failure).

Therefore, the system of automated reasoning ATP is put into the base for development of the descriptive language for logic programming LOGPRO. This logical complete deductive base is used for building a new descriptive logical programming language (Berkovic, 1997, Berkovic & Hotomski , 1997, Berkovic et al., 2003).

4. The Concept of LOGPRO – LOGic PROgramming Language Based on ATP System

Many logic programming languages have been implemented, but PROLOG is the most popular language and it is useful for solving many problems. PROLOG as a logic-oriented language (Bratko, 1986, Malpas, 1987, Pereira & Shieber, 1987) contains a resolution-based theorem-prover (PROLOG-system). The theorem-prover in PROLOG appears with the depth-first search approach. It uses the special resolution rule: ***SLDNF*** (Linear resolution with Selection function for Definite clauses and Negation as Failure).

4.1 Formalization on PROLOG

The first-order predicate logic is the form of representation in PROLOG. PROLOG-program is a set of sentences. Every sentence is finished by full stop. Program in PROLOG consists of axioms (rules, facts) and a theorem to be proved (goal). The axioms are restricted in ***Horn clauses*** form. ***Horn clauses*** (Hotomski & Pevac, 1991), are clauses with at most one positive literal.

The ***rules*** have the form: $\mathbf{G}$ **:-** $\mathbf{T_1, T_2, \ldots, T_n}$ **.**

where **G** is positive literal and $\mathbf{T_j}$ (**j=1,2,...,n**) are literals (positive or negative). The symbol for conjunction is: **,** . The element **G** is presented head of the rule. The elements $\mathbf{T_j}$ (**j=1,2,...,n**) are presented body of the rule. The separator **:-** corresponds to implication ($\Leftarrow$). The symbol for negation is: **not**.

The ***facts*** have the form: **G.**

where **G** is positive literal.

The ***goals*** (questions) have the form: **?-** $\mathbf{T_1, T_2, \ldots, T_n}$ **.**

where $\mathbf{T_i}$ (**i=1,2,...,n**) are literals.

Practically, programming in PROLOG is restrictive in a subset of first-order logic. Horn clauses are represented the first defect of PROLOG. The concept of negation as definite failure is represented the second defect of PROLOG.

4.1 Formalization on LOGPRO

An other approach to logic programming is implementation of a different deductive concept. The determinate system ATP for automated theorem proving is especially put into the base of prolog-like language LOGPRO, as the surrogate for the concept of negation as definite failure. This logical complete deductive base is used for building a new descriptive logic programming language.

The first-order logic is the form of representation in ATP system, too. But, this system has not restriction in Horn clauses. The program on logic language LOGPRO based on the ATP system is a set of sentences (clauses). There are three kinds of sentences: rules, facts and goals. Every sentence is finished by full stop.

The ***rules*** have the form: $\mathbf{G_1, G_2, \dots, G_m :- T_1, T_2, \dots, T_n .}$

where $\mathbf{G_i}$ ($\mathbf{i=1,2,\dots,m}$) and $\mathbf{T_j}$ ($\mathbf{j=1,2,\dots,n}$) are literals (positive or negative). The symbol for conjunction is: **,** . The elements $\mathbf{G_i}$ ($\mathbf{i=1,2,\dots,m}$) present head of the rule. The elements $\mathbf{T_j}$ ($\mathbf{j=1,2,\dots,n}$) present body of the rule. The separator **:-** corresponds to implication (⇐). The symbol for negation is: ~ .

The ***facts*** have the form: **G.**

where **G** is literal (positive or negative).

The ***goals*** (questions) have the form: $\mathbf{?- T_1, T_2, \dots, T_n .}$

where $\mathbf{T_i}$ ($\mathbf{i=1,2,\dots,n}$) are literals (positive or negative).

The rules and facts (axioms) are presented by auxiliary clauses. The goal (central clause) is negating the theorem to be proved. Symbol **?-** in goal is the substitution for negation. The execution procedure is ATP system based on OL-resolution with marked literals. This formulation enables eliminating the defects of PROLOG-system.

The logic programming languages PROLOG and LOGPRO are compared. PROLOG rules and facts do not allow the explicit statement of negative information. But, the declarative syntax of the logic programming language LOGPRO allows the expression of negative information in rules and facts. Also, it is possible to construct the rule with more than one element in the rule's head.

Example 1.

The problem of trying to formulate sentence:

"Alice likes whatever Queen dislikes, and dislikes whatever Queen likes." into PROLOG form (Subrahmanyam, 1985).

The representations:

```
likes(alice,X1) :- not likes(queen,X1).
not likes(alice,X1) :- likes(queen,X1).
```

are illegal in PROLOG because the second rule has a negation in head (it isn't Horn clause). It is possible to solve the problem by trick - using a modified predicate likes, and expressing the statement as:

```
likes(alice,X1,true) :- likes(queen,X1,false).
likes(alice,X1,false) :- likes(queen,X1,true).
```

The expansion concerning Horn clauses on the logic programming language based on ATP system has the possibilities to express the statement as:

likes(alice,X1) :- ~ likes(queen,X1).

~ likes(alice,X1) :- likes(queen,X1).

PROLOG-system has the negation defect (Brtka, 2001). This defect is corrected in ATP system. It can be illustrated by the following example.

Example 2.

Program in PROLOG:

vegetarian(tom).

vegetarian(ivan).

smoker(tom).

likes(ana,X1) :- not (smoker(X1)), vegetarian(X1).

PROLOG-system gives unconnected answers on the following questions:

?- likes(ana,X1).

no

?- likes(ana,ivan).

yes

If the last clause is now:

likes(ana,X1) :- vegetarian(X1), not (smoker(X1)).

PROLOG-system gives wrong answers on the following questions:

?- likes(ana,X1).

X1=ivan

?- likes(ana,ivan).

yes

These answers are incorrect because we have not data about Ivan and smoking. We don't know if Ivan is a smoker or not. The correct answer will be: "I don't know".

The program in LOGPRO:

vegetarian(tom).

vegetarian(ivan).

smoker(tom).

likes(ana,X1) :- ~ smoker(X1), vegetarian(X1).

ATP-system gives answers on the following questions:

?- likes(ana,X1).

Success=0

The proof isn't completed

?- likes(ana,ivan).

Success=0

The proof isn't completed

When the last clause is:

likes(ana,X1) :- vegetarian(X1), ~smoker(X1).

ATP system also gives the correct answers: **"Success=0, the proof isn't completed"**.

In fact, the system generates resolvents, but can not complete the proof with depth-first strategy. The treatment of negation as definite failure in this system is escaped. The concept of LOGPRO allows eliminating of endless branches, recursion using and works with

structures and lists, as well as PROLOG. It is presented in some concrete examples (Berkovic, 1997).

5. Applications of ATP System and LOGPRO

5.1. Time-Table and Scheduling System DEDUC

This system ATP is incorporated in the system for automatic creating of the combinatorial disposition DEDUC, where it has presented the satisfying practical efficiency.
DEDUC is a software package that integrates scientific results of Constraint Logic Programming and practical needs in generating combinatorial schedules. The work on the system started in 1991. After ten years the powerful system was developed.
System is based on synchronized work of two processes: data storing and optimization of gaps in the timetable. Theorem prover controls both processes to secure that initial requirements are not violated
System facilities and performances:

- Automatic and interactive generating of schedules for the initial data and conditions;
- Setting up and maintenance of the initial data and requirements by using the user friendly interface;
- Setting up various initial requirements like splitting classes into groups, connecting groups and classes, enabling multiple lessons; setting restrictions in using laboratories and rooms; setting teachers' requirements and the other pedagogic obligations and desirable demands;
- Generating schedules respecting school shifts;
- Screen browsing and printing the general, per-class and per-teacher schedules;
- Generating and archiving different versions of the schedules generated for the same initial data and conditions;
- Maintenance of the archived versions, data and conditions.

The practical efficiency and system limits can be observed for complicated examples as well as for the simple ones. Essentially, getting the acceptable version of the schedule depends on structure of the initial requirements and their connections with data, although the number of requirements has no influence.
Practical experiences in the real world assure that DEDUC system generates timetables with quality level higher than quality level of hand-made timetables. The time needed for generating a timetable varies from few minutes to several hours, and depends on amount of data, structure of conditions as well as on computer performances. More informations at http://deduce.tripod.com

5.2. A Technique for the Implicational Problem Resolving for Generalized Data Dependencies

A mechanism for generalized representation of various data dependency types, such as functional (*fd*), multivalued (*mvd*), join (*jd*), implied (*imd*) and inclusion (*id*) dependencies, has been developed in the relational data model theory. The initial supposition was that all the data dependencies (i.e. "rules" which hold among data) can be represented, in unified manner, by one, or more symbolic data templates, satisfying certain criteria, according to defined interpretation rules for such symbolic templates. On the basis of that supposition, the term of generalized data dependency has been introduced, so as to represent the other

dependency types in the same way. One of the important questions, arising when new data dependency type is introduced, is how it can be stated if a data dependency is a logical consequence of a given set of data dependencies. This problem in relational databases is known as the *implicational problem*.

At the beginning, the terms of: tableau, as a symbolic data template, generalized dependency (*gd*) and its interpretation are defined without explanations, because they are considered as already known. In (Lukovic et al., 1996, Lukovic et al., 1997) is presented a possible approach to resolving the implicational problem for *gds*, it is established at the same time a way of testing the implicational problem for all the specific data dependency types which can be formalized by means of *gds*. The proposed approach considers a usage of the ATP System.

To resolve the implicational problem for a given sets of *gds* Γ and arbitrary *gd* γ means to establish if $\Gamma \models \gamma$ holds. It is not practically possible to test he implicational problem $\Gamma \models \gamma$ by exact applying of definition of generalized dependencies by systematic generating of all the relations form *SAT (R)* and checking the implication $r \models \Gamma \Rightarrow r \models \gamma$, because *SAT(R)* is, in most cases, the set of high cardinality and it can be even infinite. Therefore, the other methods have to be applied so as to resolve the problem. According to the nature of *gds*, it is concluded that for the automations of the test $\Gamma \models \gamma$, the resolution procedure can be applied. Therefore, the set of Γ and the dependency γ will be represented by appropriate predicate formulas. According to the resolution theorem and theorem of generalized dependencies, the test of the condition $\Gamma \models \gamma$, where $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$, is performed by disproving procedure, on the basis of the set of initial assumptions $\mathbf{F}(\Gamma)$ and the negation of the conclusion $\neg F$. In that case, the theorem that should be proved by ATP System is of the form: $\mathbf{F}(\Gamma) \rightarrow F$.

To prove the theorem, the clauses should be built from the set of assumptions $\mathbf{F}(\Gamma)$ and negation $\neg F$. They represent the input for ATP. Beside that, two additional input parameters are: (i) maximal searching deep and (ii) maximal clause length. With respect to the resolution theorem, there are three possible outcomes from ATP. (a) *"positive"*; an empty clause has been reached, which means that the assertion *F* holds. According to theorem of generalized dependencies, $\Gamma \models \gamma$ holds, too; (b) *"negative"*: the empty clause has not been reached and there are no more possibilities for new clause generating. It means that the conclusion *F* cannot be derived from $\mathbf{F}(\Gamma)$. According to theorem of generalized dependencies, we conclude $\Gamma \models \gamma$ does not hold; (c) *"uncertain"*: the empty clause has not been obtained, whereas maximal searching deep and maximal clause length have been reached, or memory resources have been exhausted.

5.3. Intelligent Tutoring System Based on ATP System

The concept of LOGPRO can be an excellent base for an intelligent tutoring system iTUTOR (Brtka, 2001, Berkovic et al., 2003). Such deductive logic programming language can perform tasks that standard PROLOG system could not (Hotomski, 2004).

It is now possible to define predicate:

know(student_name, concept, degree).

where studentname is name of a student, concept is name of a concept that student should know and degree indicates grade of concept cognition.

Negation in the head of the rule can be implemented as:

~know(student_name, concept, degree):-

know(student_name, $concept_1$, $degree_1$),
know(student_name, $concept_2$, $degree_2$),
...
know(student_name, $concept_n$, $degree_n$),
$degree_1 < c_1$,
$degree_2 < c_2$,
...
$degree_n < c_n$.

which means that student does not know concept in certain degree if he does not know minor concepts $concept_1$, $concept_2$, ..., $concept_n$ in degree greater than or equal with c_1, c_2, ..., c_n where c_1, c_2, ..., c_n are predefined constants. Furthermore, one can calculate degree of concept cognition by adding a line at the end of previous rule:

degree is ($degree_1$+ $degree_2$+...+ $degree_n$)/n.

Rule:

~nextlesson(student_name, lesson_name):-

know(student_name, $concept_1$, $degree_1$),
know(student_name, $concept_2$, $degree_2$),
...
know(student_name, $concept_n$, $degree_n$),
$degree_1 > c_1$,
$degree_2 > c_2$,
...
$degree_n > c_n$.

indicates lessons that particular student should not learn because he knows all belonging concepts in degree greater than predefined constant.
Similar rule:

~nextlesson(student_name, lesson_name):-

~know(student_name, $concept_1$, $degree_1$),
know(student_name, $concept_2$, $degree_2$),
...
know(student_name, $concept_n$, $degree_n$),
$degree_1 > c_1$,
$degree_2 > c_2$,
...
$degree_n > c_n$.

where some predicates in the body of the rule are negated, indicates lessons that student should not learn because he does not know all belonging concepts in degree greater than predefined constant. Application of such logic programming language LOGPRO can overcome some problems during the process of student modelling in an intelligent tutoring system.

5.4. Baselog System as Deductive Databases

Baselog system concept and its program implementation enabled the integration of good properties of Datalog, Prolog and ATP system, and so is realized a more flexible system in reference to the work in the closed, respectively opened world. Specifically needs in the development for the work with databases ask just development and application of a such system, and it makes it more superior in reference to Datalog, Prolog and ATP system, considered separately (Radulovic, 1998, Radulovic & Hotomski, 2000).
Some automated reasoning systems can give incorrect answers if they work in closed world concept, or correct answers if they work in open world concept where the answer depends on fact base completeness. If fact, the base is incomplete, some of automated deduction systems can consult facts from various extension files. In order to enable work with greater data amount there arose a need to access certain databases which can even be distant and then take the piece of data which could be used for deducting a conclusion (Hotomski, 2004).

5.4.1. Close/Open World Assumption in Baselog

In (Ceri, 1989, Stonebraker et al., 1990, Ullman, 1994) is described Datalog, the logic programming language in the field of the database that is implemented in the post-relation software for database management system PROGRESS. Datalog works on the CWA-principle, respectively (by) adopting the Closed World Assumption. The CWA-principle declares (Radulovic, 1998, Radulovic & Hotomski, 2000) :

Definition 1. (The CWA – principle) If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
For knowledge databases is also characteristic the open world assumption. In the open world regime work classic systems for the automatic theorem proving, especially, ATP system (Berkovic, 1994, Berkovic, 1997). The knowledge bases contain limited knowledge segments from a certain field. They can be incomplete, i.e. they do not present total relevant knowledge. The applying of the closed world concept on such databases, can bring wrong answers to the asked questions. Because of that the pure concept of the closed world can not be applied for the databases used in the education computing software.
Universal resolution systems from theoretic aspect totally support the work with databases as well, but they show a practical deficiency. It can be seen in the fact that because of the endeavoring to get a semantically expected answer, it is necessary to give a complete space description where the solution is claimed.
In the database area it is, for example, exposed through the necessity of proposing following axiom:

$$t \neq t_1 \wedge \ t \neq t_2 \wedge \ldots \wedge t \neq t_n \Rightarrow \sim P(t)$$

where $t_1,\ldots,t_n$ are the relation database tuples, and P(t) means the tuple t belonging to the database relation (~ is negation).

As it can be seen, already for little number of tuples in database, this axiom has big length, so this theoretic possibility is left in practical applications. Both in Datalog and in Prolog it is made the attempt for solving this deficiency in specific ways. In Prolog it is the strategy of definite failure (Bratko, 1986, Berkovic, 1997) and in Datalog the CWA-principle

(Stonebraker et al., 1990). Meanwhile, no one of these solutions can satisfy education needs in fullness, for the following reasons.
In reference to possible user's questions, there are following options:

a) the answer to the question is deducible from the base,
b) the answer to the question is not deducible from the base,

where in b) we differ:

b1) the answer needs to be affirmative,
b2) the answer needs to be negative.

In a) when the answer is deducible from the base, it will be found and presented to a user either Prolog, Datalog or Logpro based on ATP-system.
Specificities are being reflected in b). According to the adopted the CWA-assumption in Datalog, respectively the definite failure concept in Prolog, there are possible incorrect or indefinite answers. So in b1) Datalog can generate the incorrect answer NO, while Prolog's answer "NO" can be interpreted as "*uncertain*". In b2) Datalog answer "NO" is correct, and Prolog answer "NO" can be interpreted as "NO". In both cases b1) and b2) Logpro based on ATP gives answer "*uncertain*".

We observe that in educative meaning Datalog according to the b1) does not satisfy, while Prolog and Logpro based on ATP give acceptable, but uncertain answers. In b2) Datalog gives correct and precise answer, while Prolog and Logpro based on ATP gives inadequately precise answers. From the educative aspect it is desirable to lessen the indefiniteness of the system answer and it is necessary to eliminate the non-allowed answers. Otherwise, there is need to keep the definiteness present in Datalog for b2) and eliminate non-allowed answer from b1). Implementing Baselog - system projected on the list of the CWA–predicate and the CWA–rule, a flexible concept has been realized. Such system all predicates which are in the CWA-list treats as Datalog, in closed-world, while all the other predicates treat in open world, i.e. works as ATP. With it, it is free of Prolog defects in reference to the negation treatment and the definite failure concept.

The basis for Baselog - system make following components (Radulovic & Hotomski, 2000, Radulovic, 1998):

- The CWA–predicate list, which is a part of the program,
- The CWA–rule,
- The CWA–controller by which is enlarged ATP resolution system.

The whole Baselog - system is the extension of the resolution method by the concepts of the opened and closed world. By the CWA–controller one provides dozing a degree of the world openness/closeness for the program predicates.
Every literal of the form $R(w_1,...,w_m)$ where R is predicate name mentioned in the CWA-predicate list, and $w_1,...,w_m$ are arguments, Baselog - system will treat in the closed system regime, while all the other predicates that are not in the CWA–predicate list, by the system will be treated in the open world regime. Here, the CWA–controller of Baselog -system uses the CWA–rule, formulated in the following way.

The CWA – RULE:
Let D is the clause of the forms $L_1 \vee L_2 \vee \ldots \vee L_p$ and let L_i, $1 \leq i \leq p$ is literal of the form $R(w_1,\ldots,w_m)$, where the predicate R is declared as the CWA–predicate. If $R(w_1,\ldots,w_m)$ can not be unified with no one base element, then $R(w_1,\ldots,w_m)$ will be deleted from clause D.
If exists unificator for L_i and some element from base, then clause D is not changed, there is no deleting.

The proof of the CWA–rule correctness is described in (Radulovic, 1998).

6. Conclusion

Completeness and universality of the resolution method, as the base of ATP system, enables it to be applied as the deductive base of prolog-like language. The relationship between programming language based on ATP system and programming language PROLOG are emphasized. The logic programming language based on ATP system enables eliminating the defects of PROLOG-system (the expansion concerning Horn clauses, escaping negation treatment as definite failure, eliminating of endless branches), keeping the main properties of prolog-language. In this paper are also described some applications of ATP system and LOGPRO such as: time-table and scheduling (DEDUC), a technique for the implicational problem resolving for generalized data dependencies, deductive databases (Baselog System) and intelligent tutoring system based on ATP system (iTUTOR). Also, ATP system is used for the development inference system and data mining (Brtka, 2008).

7. References

Barr, A.; Cohen, P.R. & Feigenbaum, E.A. (1982). *The Handbook of Artificial Intelligence*, Vol.I,II,III, Heuris Tech Press, W. Kaufmann, Inc., California

Berković, I. (1994). Variable Searching Strategies in the Educationally Oriented System for Automatic Theorem Proving, M.Sc. Thesis, Technical Faculty "Mihajlo Pupin", Zrenjanin, (in Serbian)

Berković, I. (1997). The Deductive Bases for the Development of the Descriptive Languages for the Logical Programming, Ph.D. Thesis, Technical Faculty "Mihajlo Pupin", Zrenjanin, (in Serbian)

Berković, I. & Hotomski, P. (1997). The concept of logic programming language based on resolution theorem prover, *Proceedings of VIII Conference on Logic and Computer Science LIRA,* pp. 111-120, Sept 1-4 1997, Novi Sad, Serbia

Berković, I.; Hotomski P. & Brtka V. (2003). The Concept of Logic Programming Language Based on the Resolution Theorem Prover and its Appliance to Intelligent Tutoring Systems; *IEEE Proceedings of 7th International Conference on Intelligent Engineering Systems*, pp 169 - 172; ISSN: 977.246.048.3/1562, Assiut – Luxor, Egypt, March 4 - 6. 2003.

Bratko, I. (1986). *PROLOG Programming for Artificial Intelligence*, Addison-Wesley Publ. Comp.

Brtka, V. (2001). Tutoring educational software, M.Sc. Thesis, Technical Faculty "Mihajlo Pupin", Zrenjanin, (in Serbian).

Brtka, V. (2008). Automated Synthesis of Rule Base in Inference Systems, Ph.D. Thesis, Technical Faculty "Mihajlo Pupin", Zrenjanin, (in Serbian)

Ceri, S.; Gottlob, G. & Tanza, L. (1989). What You Always Wanted to Know About Datalog (And Never Dared to Ask), *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 1 March 1989, pp. 146-167

Gevarter, W.B. (1985). *Intelligent Machines*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

Hotomski, P. & Pevac, I. (1991). *Mathematical and Programming Problems of Artificial Intelligence in the Field of Automatic Theorem Proving*, Naučna knjiga, Belgrade, (in Serbian)

Hotomski, P. (2004). *Systems of Artificial Intelligence*, University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, (in Serbian)

Kowalski, P. & Kuchner, D. (1971). Linear Resolution with Selection Function, *Artificial Intelligence*, Vol. 2, pp. 227-260

Luković, I.; Hotomski, P.; Radulović, B. & Berković, I. (1996). A Proof of Generalized Data Dependency Satisfaction by The Automatic Reasoning Method, *Proceedings of the Second Symposium on Comp, Sc. and Informatics YU INFO*, Brezovica, 02-05.-4.1996. (in Serbian)

Luković, I.; Hotomski, P.; Radulović, B. & Berković, I. (1997). A Technique for the Implicational Problem Resolving for Generalized Data Dependencies, *Proceedings of VIII Conf. on Logic and Computer Science LIRA '97*, pp. 111-119, Novi Sad, 01-04.09.1997,

Malpas. J. (1987). *PROLOG: A Relational Language and Its Applications*, Prentice-Hall International Inc.

Pereira, F.C.N. & Shieber, S.M. (1987). *PROLOG and Natural - Language Analysis*, CLSI, Leland Stanford Junior University

Radulović, B. & Hotomski, P. (2000). Projecting of Deductive Databases with CWA Management in Baselog System, *Novi Sad Journal of Mathematics*, pp 133-140. Vol 30, N2, 2000, Novi Sad, Serbia

Radulović, B. (1998). Database Projecting in the Field of Education Computer Software, Ph.D. Thesis, Technical Faculty "Mihajlo Pupin", Zrenjanin, (in serbian)

Radulović, B. & Hotomski, P. (1997). Database projecting in the Baselog-system, *Proceedings of VII Conf. "Informatics in eduation and new information technologies"*, pp. 71-77, Novi Sad, 1997, (in serbian)

Radulović, B.; Berković, I.; Hotomski, P. & Kazi, Z. (2008). The Development of Baselog System and Some Applications, *International Review on Computers and Software (I.RE.CO.S.)*, pp 390-395, Vol. 3 N. 4, July 2008, Print ISSN**:** 1828-6003

Stonebraker, M.; Rowe, L. A. & Hirohama, M. (1990). The Implementation of POSTGRES, *IEEE Transactions on Knowledge and Data Engineering*, pp. 125-142, Vol. 2, No. 1, March 1990

Subrahmanyam, P.A. (1985). The "Software Engineering" of Expert Systems: Is Prolog Appropriate?, *IEEE Transactions on Software Engineering*, pp. 1391-1400, Vol. SE-11, No. 11, november 1985.

Ullman, J. (1994). Assigning an Appropriate Meaning to Database Logic With Negation, www-db.stanford.edu/pub/papers/negation.ps

Wos, L. (1985). Automated Reasoning, *The American Mathematical Monthly,* pp. 85-93 Vol. 92, No. 2, february 1985.

Wos, L.; Overbeek, R.; Lusk, E. & Boyle J. (1992). *Automated Reasoning: Introduction and Applications*, Prentice-Hall

Engineering the Computer Science and IT

Edited by Safeeullah Soomro

ISBN 978-953-307-012-4
Hard cover, 506 pages
Publisher InTech
Published online 01, October, 2009
Published in print edition October, 2009

It has been many decades, since Computer Science has been able to achieve tremendous recognition and has been applied in various fields, mainly computer programming and software engineering. Many efforts have been taken to improve knowledge of researchers, educationists and others in the field of computer science and engineering. This book provides a further insight in this direction. It provides innovative ideas in the field of computer science and engineering with a view to face new challenges of the current and future centuries. This book comprises of 25 chapters focusing on the basic and applied research in the field of computer science and information technology. It increases knowledge in the topics such as web programming, logic programming, software debugging, real-time systems, statistical modeling, networking, program analysis, mathematical models and natural language processing.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ivana Berkovic, Biljana Radulovic and Petar Hotomski (2009). Extensions of Deductive Concept in Logic Programming and Some Applications, Engineering the Computer Science and IT, Safeeullah Soomro (Ed.), ISBN: 978-953-307-012-4, InTech, Available from: http://www.intechopen.com/books/engineering-the-computer-science-and-it/extensions-of-deductive-concept-in-logic-programming-and-some-applications

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821