CH-Prolog: A Proof Procedure for Positive Disjunctive Logic Programming

Wenjin Lu
Department of Computer Science, University of Koblenz-Landau
Rheinau 1, D-56075 Koblenz, Germany
luc@uni-koblenz.de

Abstract

The success of Prolog motivates people to use full first-order logic instead of only Horn clauses as the basis of logic programming. One of the main work in this extending is to seek proof procedure for new logic programming. Positive disjunctive logic programming extends Horn clause programming by allowing more than one atoms to occur in the head of a program clause. In this paper we propose a new proof procedure for disjunctive logic programming which is based on novel program transformation. With this transformation, the new proof procedure shares many important properties enjoyed by SLD-resolution. The soundness and completeness of the proof procedure with respect to computing answers are given.

Key Words: Disjunctive Logic Programming, SLD-resolution, Proof Procedure.

1. Introduction

The success of Prolog motivates people to use full first-order logic instead of only Horn clauses as the basis of logic programming (Loveland 1987). Positive disjunctive logic programming is one of the efforts in this consideration, it extends Horn clause programming by allowing more than one atom to occur in the head of a program clause. The declarative semantics of such programs is defined by the set of all logic consequences of the program and negative information can be concluded by GCWA (Minker 1982).

In the seeking of proof procedures for positive disjunctive logic programming, one would like to preserve as many important properties enjoyed by SLD-resolution as possible. SLD-resolution is goal oriented, allows a declarative and procedural reading capability, has a linear input format, uses a positive implication format, and needs no contrapositive occurrences of the implications. Moreover, the process is intuitionistically sound, which insures a certain constructive nature to the inference structure. These properties, in fact, have become a standard in measuring the quality of proof procedures for positive disjunctive logic programming. Unfortunately, it is not easy to have such a proof procedure that satisfies all the above-mentioned properties.

In this paper we propose a new proof procedure for disjunctive logic programming which is based on the program transformation introduced in (Lu & Furbach 1998). In (Lu & Furbach 1998), a novel view on propositional disjunctive logic programming has been proposed which says, from computational point of view, a disjunctive logic program is equivalent to a Horn program together with a control program. In this paper we extend the CH-transformation introduced in (Lu & Furbach 1998) to first order disjunctive logic program. The new proof procedure then is based on the extended CH-transformation. It works just like the SLD-resolution except an extra checking. As a result it preserves many important properties enjoyed by SLD-resolution. In contrast with existing procedures in (Baumgartner, Furbach, & Stolzenburg 1997; Loveland 1987; 1991), it dispenses with the ancestor cancellation completely and therefore speeds up the inner-loop. In addition, its conceptual simplicity makes it easy to be implemented. A possible implementation would consist of the SLD-resolution together with a special-propose theorem prover (for subsumption test).

In the rest of the paper we are not going to present the concepts concerning first order logic, logic programming and automatic theorem proving. We refer reader to see the standard books (Chang & Lee 1973; Loveland 1978; Lloyd 1984; Lobo, Minker, & Rajasekar 1992) on these subjects. Some notions and results are presented at the place where they are needed.

The rest of the paper is organized as follows, after extending the CH-transformation to first order logic programs in section 2, we introduce the new proof procedure in section 3. Section 4 discusses the soundness and completeness of the procedure. We conclude the paper in section 5 with some comments on existing work.

2. CH-Transformation

This section extends the CH-transformation of propositional disjunctive programs introduced in (Lu & Furbach 1998) and its properties to first order case.

Definition 1 (CH-transformation) *Let P be a disjunctive program. For each clause $C \in P$ of the form*

$$a_1 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

the CH-transformation of C, denoted by $CH(C)$, is a set of clauses defined by:

$$CH(C) = \begin{cases} \{C\}: & n = 1, \\ C_{ch}: & n > 1, \end{cases}$$

where C_{ch} is the set of the following clauses:

$$\begin{array}{l} A_1(V_C) \vee \ldots \vee A_n(V_C) \\ a_1 \leftarrow b_1, \ldots, b_m, A_1(V_C) \\ \quad \vdots \\ a_n \leftarrow b_1, \ldots, b_m, A_n(V_C) \end{array}$$

and A_1, ..., A_n are new predicates not occurring in P, which are referred as control predicates, V_C is the tuple of all variables appearing in C. The atoms formed by control predicates are called control atoms and the clause

$$A_1(V_C) \vee A_2(V_C) \vee ... \vee A_n(V_C) \qquad (1)$$

is called a control clause. The CH-transformation of P, denoted by $CH(P)$, is defined by

$$CH(P) = \bigcup_{C \in P} CH(C) .$$

By the definition, a clause in $CH(P)$ is either a Horn clause or of the form (1) (control clause). Therefore $CH(P)$ can be written as follows:

$$CH(P) = P_H + P_C$$

where P_C is the program consisting of all control clauses in $CH(P)$ and P_H is the Horn program consisting of all Horn clauses in $CH(P)$. P_C is called the control program. The following facts are trivial.

- The control clauses in P_C consist of only control atoms. No control atom occurs more than once in P_C.
- Each Horn clause in P_H contains at most one control atom. No control atom occurs twice in P_H.
- Each control atom occurs exactly twice in $CH(P)$, once in P_C and once in P_H. It forms a one-to-one mapping.

The following theorem and its corollary are very important for developing a proof procedure for disjunctive programs in the next section. Its proof can be found in (Lu 1998).

Theorem 2 *Let P be a disjunctive program and $CH(P) = P_H + P_C$ be the CH-transformation of P. Then for any formula α which contains no control atoms, we have $P \models \alpha$ iff $CH(P) \models \alpha$.*

Corollary 3 (Model Preservation) *Let P be a disjunctive program and $CH(P) = P_H + P_C$ is the CH-transformation of P. Then M is a model of P iff there is a model M_C of P_C such that $M \cup M_C$ is a model of $CH(P)$.*

3. CH-Prolog

In this section we provide the proof procedure *CH-Prolog* for disjunctive logic programs. The prefix "CH" on the one hand hints that the procedure is based on the CH-transformation and on the other indicates that CH-Prolog is a Prolog with a "CHecking".

Prolog deals with Horn programs. The basic proof procedure for Prolog is the SLD-resolution which uses SLD derivation as an inference mechanism. An SLD-tree for a Horn program P and a goal G is a tree in which each branch is a SLD derivation of $P \cup \{G\}$. A branch corresponding to a successful derivation (ending with goal $\Box$) is called a success branch, a branch corresponding to an infinite derivation is called an infinite branch and a branch corresponding to a failed derivation is called a failure branch.

It has been proved that SLD-resolution is sound and complete wrt. Horn programs. Roughly speaking, $P \cup \{G\}$ is unsatisfiable iff there is a success branch in an SLD-tree for P with goal G.

Given a disjunctive program P, let $CH(P) = P_H + P_C$ be the CH-transformation of P. Recall that P_H is a Horn program and P_C is a set of disjunctive facts consisting of only control atoms (formed by new predicates not appearing in P). Each clause in P_H contains at most one control atom and control atoms never occur in the head of any Horn clause in the Horn program P_H. Then the new proof procedure is motivated by the following observation. Given a goal $\leftarrow Q$, assume that $P \cup \{\neg Q\}$ is unsatisfiable. Then either

- $P_H \cup \{\neg Q\}$ is unsatisfiable, in this case, by completeness and soundness of SLD-resolution, there must be a successful branch in the SLD-tree of P_H and goal $\leftarrow Q$, or
- there must be a failure branch ending with a subgoal consisting of only control atoms. Let $\leftarrow C_1, \ldots, \leftarrow C_n$ be all the leaf nodes such that C_i $(1 \leq i \leq$

n) is a conjunction of only control atoms, then $\{\forall(\neg C_1) \wedge \ldots \wedge \forall(\neg C_n)\} \cup P_C$ must be unsatisfiable, that is, $\exists(C_1 \vee \ldots \vee C_n)$ is a logic consequence of P_C. Note that $C_1 \vee \ldots \vee C_n$ is a disjunctive normal formula (DNF) and P_C can be understood as a conjunctive normal formula (CNF), furthermore, they both contain no negative literals, therefore, after converting $C_1 \vee \ldots \vee C_n$ into a conjunctive normal formula, its unsatisfiability can be checked by subsumption.

With this observation a new calculus is introduced by simulating SLD-tree building for P_H and $\leftarrow Q$. To introduce the proof procedure formally, we need some notions.

Definition 4 (Restricted SLD-tree) *Let P be a disjunctive program and $CH(P) = P_H + P_C$ be the CH-transformation of P. Let G be a goal. Assume R to be a computation rule that only selects non control atoms. A restricted SLD-tree for $P \cup \{G\}$ is the SLD-tree for P_H and goal G via R. A restricted success branch in a restricted SLD-tree is the branch ending either with an empty clause $\Box$ or with a goal consisting of only control atoms.*

By the above observation, only the ending subgoals of the restricted successful branches need to be kept for checking. Therefore in a CH-Prolog deduction, each line typically has the form

$$G\#C_1\#\ldots\#C_n,$$

where G is the current selected goal in an SLD-tree and C_i's are conjunction of control atoms in the ending subgoal of the restricted successful branches. No variables are shared among $G, C_1, \ldots, C_n$.

As in nH-Prolog, a CH-Prolog derivation is composed of blocks, each block basically simulates a branch in SLD-tree.

Definition 5 (SLDCH-derivation) *Let P be a disjunctive program and $CH(P) = P_H + P_C$. Let G be a goal. Assume R to be a computation rule that only selects non control atoms. An SLDCH-derivation is a sequence $L_0, L_1, \ldots, L_n \ldots$, where*

1. *L_0 is the given goal G and*

2. *If $L_i = G_i\#C_1\#\ldots\#C_k$ and G_i contains a non control atom, then $L_{i+1} = G_{i+1}\#C_1\#\ldots\#C_k$ G_{i+1} is derived from G_i and a variant of some clause $C_{i+1} \in P_H$ using a most general unifier θ via R.*

3. *If $L_i = G_i\#C_1\#\ldots\#C_k$ and G_i consists of only control atoms, then*

   Checking: *if there exists a substitution θ such that $(C_1 \vee \ldots \vee C_k \vee C_{k+1})\theta$ is a logic consequence of P_C, where $G_i = \leftarrow C_{k+1}$, then $L_{i+1} = \Box$. Otherwise*
   Restart: $L_{i+1} = G\#C_1\#\ldots\#C_k\#C_{k+1}$.

A block is a subsequence that begins with a restart (start) line and ends before the next restart line (if any). An SLDCH-derivation is called an SLDCH-refutation if it contains a empty clause $\Box$.

Although an SLDCH-derivation is defined in a similar way as near-Horn Prolog, the differences are apparent. In an nH-Prolog derivation a typical line has the following form:

$$G\#a[D],$$

where G is a subgoal, a is an atom called active head, $[D]$ denotes a list of deferred heads. When a program clause C of the form

$$a_1 \vee \ldots \vee a_m \leftarrow b_1, \ldots, b_n$$

is called because one of the atoms in the head, say a_i, unifies with the calling goal in G, the calling head goal is replaced by the body $b_1, \ldots, b_n$ (suitably instantiated) as for Prolog, but also the remaining head atoms $\{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m\}$ are entered leftmost in the deferred head list. Variables may be shared among the G, the active head and the deferred head list, and thus instantiation of a variable in one portion of line may affect other elements in the line. In addition, nH-Prolog needs cancellation operation: that is, if the calling goal unifies with an active head, the calling goal is cancelled.

CH-Prolog differs from the near-Horn Prolog mainly in the following two aspects:

1. no variable is shared among G and C_i's in a line $G\#C_1\#\ldots\#C_n$,

2. ancestor cancellation is completely avoided.

Roughly speaking, except the checking rule, SLDCH-derivation works exactly same as SLD-derivation. Therefore in CH-Prolog, high inner-loop speed can be expected.

Example 6 (SLDCH-refutation) *This example is taken from (Loveland 1991), there it is used to demonstrate that the Naive nH-Prolog is not complete, that is, q is a logic consequence but it does not exists a Naive nH-Prolog refutation for $P \cup \{\neg q\}$.*

$$\begin{array}{lll} P: & q \leftarrow a, b & \quad G: \ \leftarrow q \\ & a \leftarrow c & \\ & a \leftarrow d & \\ & c \vee d & \\ & b \leftarrow e & \\ & b \leftarrow f & \\ & e \vee f & \end{array}$$

then the CH-transformation of P is as follows.

$$CH(P) = P_H + P_C \ ,$$

where

$$\begin{array}{llll} P_H: & q \leftarrow a, b & P_C: & C_1 \vee C_2 \\ & a \leftarrow c & & C_3 \vee C_4 \\ & a \leftarrow d & & \\ & c \leftarrow C_1 & & \\ & d \leftarrow C_2 & & \\ & b \leftarrow e & & \\ & b \leftarrow f & & \\ & e \leftarrow C_3 & & \\ & f \leftarrow C_4 \ . & & \end{array}$$

The following is an SLDCH-refutation of $P \cup \{G\}$.

start:	**restart:**
$\leftarrow q$	$\leftarrow q\#C_1, C_3\#C_1, C_4$
$\leftarrow a, b$	$\leftarrow a, b\#C_1, C_3\#C_1, C_4$
$\leftarrow c, b$	$\leftarrow d, b\#C_1, C_3\#C_1, C_4$
$\leftarrow C_1, b$	$\leftarrow C_2, b\#C_1, C_3\#C_1, C_4$
$\leftarrow C_1, e$	$\leftarrow C_2, e\#C_1, C_3\#C_1, C_4$
$\leftarrow C_1, C_3$	$\leftarrow C_2, C_3\#C_1, C_3\#C_1, C_4$

restart:	**restart:**
$\leftarrow q\#C_1, C_3$	$\leftarrow q\#C_1, C_3\#C_1, C_4\#C_2, C_3$
$\leftarrow a, b\#C_1, C_3$	$\leftarrow a, b\#C_1, C_3\#C_1, C_4\#C_2, C_3$
$\leftarrow c, b\#C_1, C_3$	$\leftarrow d, b\#C_1, C_3\#C_1, C_4\#C_2, C_3$
$\leftarrow C_1, b\#C_1, C_3$	$\leftarrow C_2, b\#C_1, C_3\#C_1, C_4\#C_2, C_3$
$\leftarrow C_1, f\#C_1, C_3$	$\leftarrow C_2, f\#C_1, C_3\#C_1, C_4\#C_2, C_3$
$\leftarrow C_1, C_4\#C_1, C_3$	$\leftarrow C_2, C_4\#C_1, C_3\#C_1, C_4\#C_2, C_3$
	$\square$

4. Soundness and Completeness

In this section we discuss soundness and completeness of CH-Prolog. Since CH-Prolog is developed as an interpreter for disjunctive programs, as argued in (Baumgartner, Furbach, & Stolzenburg 1997), it is more desirable to have soundness and completeness with respect to answers. Therefore we first begin with the definition of a correct answer and computed answer.

Definition 7 (Answers, Correct Answer) *Let P be a disjunctive program. If $\leftarrow Q$ is a query, and $\theta_1, \ldots, \theta_m$ are substitutions for the variables from Q, then $Q\theta_1 \vee \ldots \vee Q\theta_m$ is an answer. An answer $Q\theta_1 \vee \ldots \vee Q\theta_m$ is a correct answer if $P \models \forall(Q\theta_1 \vee \ldots \vee Q\theta_m)$.*

Definition 8 (Computed Answer) *Let P be a disjunctive program and $CH(P) = P_H + P_C$. Let $\leftarrow Q$ be a goal and used as the top clause in an SLDCH-refutation. Let the refutation contain m blocks $B_1, \ldots, B_m$ and B_i $(1 \leq i \leq m)$ end with the subgoal $\leftarrow C_i$. Assume that $\leftarrow Q$ is called in each B_i $(1 \leq i \leq m)$ with renaming substitution ρ_i. Let θ_i $(1 \leq i \leq m)$ be the composition of substitutions computed in B_i, and substitution σ be a most general substitution such that $C_1\sigma \vee \ldots \vee C_m\sigma$ is a logic consequence of P_C. Let σ_i $(1 \leq i \leq m)$ be the substitution obtained by restricting σ to the variables in $\rho_i\theta_i$. Then an SLDCH-computed answer is given as*

$$Q\rho_1\theta_1\sigma_1 \vee \ldots \vee Q\rho_m\theta_m\sigma_m .$$

Example 9 (Computed Answer) *Let P be the following program:*

$$\begin{array}{llll} P: & P(a,y) \leftarrow R(a,y) & G: & \leftarrow P(x,y) \\ & P(x,b) \leftarrow S(x,b) & & \\ & R(x,y) \vee S(x,y) & & \end{array}$$

where G is a goal. The CH-transformation $CH(P) = P_H + P_C$ of P is as follows:

$$\begin{array}{llll} P_H: & P(a,y) \leftarrow R(a,y) & P_C: & C_1(x,y) \vee C_2(x,y) \\ & P(x,b) \leftarrow S(x,b) & & \\ & R(x,y) \leftarrow C_1(x,y) & & \\ & S(x,y) \leftarrow C_2(x,y) & & \end{array}$$

The following is an SLDCH-refutation of $P \cup \{G\}$:

(1)	$\leftarrow P(x,y)$	
(2)	$\leftarrow S(x,b)$	$\{y \leftarrow b\}$
(3)	$\leftarrow C_2(x,b)$	
(4)	$\leftarrow P(u,v)\#C_2(x,b)$	$\{restart\}$
(5)	$\leftarrow R(a,v)\#C_2(x,b)$	$\{u \leftarrow a\}$
(6)	$\leftarrow C_1(a,v)\#C_2(x,b)$	
(7)	$\square$	$\{x \leftarrow a, v \leftarrow b\}\{checking\}$

This refutation has two blocks B_1 and B_2. B_1 consists of the first three lines, and B_2 consists of the lines (4), (5), (6), in which line (4) is a restart line with goal $\leftarrow P(x,y)\rho$, where $\rho = \{x \leftarrow u, y \leftarrow v\}$ is a renaming substitution. Line (7) is a checking. The substitution computed in B_1 is $\theta_1 = \{y \leftarrow b\}$, and the substitution computed in B_2 is $\theta_2 = \{u \leftarrow a\}$. The substitution computed in the checking step is $\sigma = \{x \leftarrow a, v \leftarrow b\}$. Therefore, the computed answer is

$$P(x,y)\theta_1\sigma_1 \vee P(x,y)\rho\theta_2\sigma_2 = P(a,b)$$

where $\sigma_1 = \{x \leftarrow a\}$ and $\sigma_2 = \{v \leftarrow b\}$ are the substitutions obtained by restricting σ to the variables in θ_1 and $\rho\theta_2$, respectively.

The following theorem shows that the SLDCH-computed answers are correct answers.

Theorem 10 (Soundness) *Let P be a disjunctive program and let $\leftarrow Q$ be a goal. If there is an SLDCH-refutation with computed answer $Q\eta_1 \vee \ldots \vee Q\eta_m$, then*

$$P \models \forall(Q\eta_1 \vee \ldots \vee Q\eta_m) \ .$$

Proof: See (Lu 1998).

Next we turn to the completeness of CH-prolog. As noted in (Baumgartner, Furbach, & Stolzenburg 1997), the answer completeness theorem stated in the following way has a very subtle difference from the result presented in (Lobo, Minker, & Rajasekar 1992). The approach of (Lobo, Minker, & Rajasekar 1992) can not handle the case correctly if there are variable interdependencies in disjunctive answers.

Theorem 11 (Answer-Completeness of SLDCH) *Let P be a disjunctive program, $\leftarrow Q$ be a query and $Q\theta_1 \vee \ldots \vee Q\theta_l$ be a correct answer for P. Then there exists an SLDCH-refutation with computed answer $Q\sigma_1 \vee \ldots \vee Q\sigma_m$ such that $Q\sigma_1 \vee \ldots \vee Q\sigma_m$ entails $Q\theta_1 \vee \ldots \vee Q\theta_l$, i.e.,*

$$\exists \delta\ \forall i \in \{1,\ldots,m\}\ \exists j \in \{1,\ldots,l\}\ \ Q\sigma_i\delta = Q\theta_j\ .$$

Proof: See (Lu 1998).

5. Conclusions

In this paper we present a new proof procedure for disjunctive logic programming. Compared with existing proof procedures, the proposed CH-Prolog shares more important properties enjoyed by Prolog. Its conceptual simplicity makes it easy to be implemented.

Several proof procedures for disjunctive logic programming have been proposed in literature. In (Lobo, Minker, & Rajasekar 1992), SLI-resolution is used as a calculus for positive disjunctive logic programming, but it completely ignores the consideration on the contrapositives. Being aware of this deficiency of SLI, (Baumgartner, Furbach, & Stolzenburg 1997) developed a proof procedure for positive disjunctive logic programming based on model elimination (Loveland 1968). In (Baumgartner, Furbach, & Stolzenburg 1997), a family of restart variants of model elimination and a mechanism for computing answers were introduced. As indicated in (Loveland & Reed 1992), model elimination does have two main disadvantages. first, the operation of ancestor cancellation implies a sacrifice in inner-loop speed. As the depth of refutation increases, the number of ancestor goals which must be checked for cancellation increases proportionally. Second, the use of contrapositives destroys the procedural reading of clauses. Restart model elimination may be seen as addressing the second disadvantage of model elimination by employing the restart operation. Near-Horn Prolog (Loveland 1987; 1991) seems to be an attractive proof procedure for positive disjunctive logic programming, it shares many properties with SLD-resolution, except that the linearity and the procedure reading properties are local only. The degree of locality depends on the number of uses of non-Horn clauses in the computation. However, as a member of the ancestry family of procedures (Loveland & Reed 1992), the disadvantage of ancestor cancellation is not completely addressed, that is, ancestor cancellation operation still remains (in a limited form) in near-Horn Prolog. The main difference between CH-Prolog and Near-Horn Prolog is that while the former doing checking (subsumption) in the last step of a derivation, the later does it (for ancestor cancellation) in every step of a derivation. Therefore, CH-Prolog share higher inner-loop speed than Near-Horn Prolog.

Many interesting topics remain to be done. Next we are going to present a real implementation and comparing it with existing procedures.

References

Baumgartner, P.; Furbach, U.; and Stolzenburg, F. 1997. Computing Answers with Model Elimination. *Artificial Intelligence* 90(1 2):135 176.

Chang, C., and Lee, R. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press.

Lloyd, J. 1984. *Foundations of Logic Programming.* Springer.

Lobo, J.; Minker, J.; and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming.* MIT-Press.

Loveland, D. W., and Reed, D. W. 1992. Near-Horn Prolog and the Ancestry Family of Procedures. Technical Report CS-1992-20, Department of Computer Science, Duke University, Durham, North Carolina.

Loveland, D. 1968. Mechanical Theorem Proving by Model Elimination. *JACM* 15(2).

Loveland, D. 1978. *Automated Theorem Proving - A Logical Basis.* North Holland.

Loveland, D. 1987. Near-Horn Prolog. In Lassez, J.-L., ed., *Proc. of the 4th Int. Conf. on Logic Programming*, 456 469. The MIT Press.

Loveland. D. 1991. Near-Horn Prolog and Beyond. *Journal of Automated Reasoning* 7:1–26.

Lu, W., and Furbach, U. 1998. Disjuctive logic program = Horn program + Control program. In *Proceedings of JELIA'98*, number 1489 in Lecture Notes in Artificial Intelligence, 33 46. Springer-Verlag.

Lu, W. 1998. *Nonmonotonic Reasoning based on Minimal Models and Its Implementation.* Ph.D. diss., Dept. of Computer Science, Uinversity of Koblenz-Landau.

Minker, J. 1982. On indefinite databases and the closed world assumption. In *Proceedings of the 6th Conference on Automated Deduction, New York*, 292–308. Berlin: Springer.