# A Real Implementation for Constructive Negation

Susana Muñoz  Juan José Moreno-Navarro

LSIIS, Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n Boadilla del Monte
28660 Madrid, Spain **
{ susana | jjmoreno}@fi.upm.es

**Keywords** Constructive Negation, Negation in Logic Programming, Constraint Logic Programming, Implementations of Logic Programming.

Logic Programming has been advocated as a language for system specification, especially for logical behaviours, rules and knowledge. However, modeling problems involving negation, which is quite natural in many cases, is somewhat restricted if Prolog is used as the specification/implementation language. These constraints are not related to theory viewpoint, where users can find many different models with their respective semantics; they concern practical implementation issues. The negation capabilities supported by current Prolog systems are rather limited, and a correct and complete implementation there is not available. Of all the proposals, constructive negation [1, 2] is probably the most promising because it has been proven to be sound and complete [4], and its semantics is fully compatible with Prolog's.

Intuitively, the constructive negation of a goal, $cneg(G)$, is the negation of the frontier $Frontier(G) \equiv C_1 \vee ... \vee C_N$ (formal definition in [4]) of the goal $G$. After running some preliminary experiments with the constructive negation technique following Chan's description, we realized that the algorithm needed some additional explanations and modifications.

Our goal is to give an algorithmic description of constructive negation, i.e. explicitly stating the details and discussing the pragmatic ideas needed to provide a real implementation. Early results for a concrete implementation extending the Ciao Prolog compiler are also presented.

Constructive negation was, in fact, announced in early versions of the Eclipse Prolog compiler, but was removed from the latest releases. The reasons seem to be related to some technical problems with the use of coroutining (risk of floundering) and the management of constrained solutions. It is our belief that these problems cannot be easily and efficiently overcome. Therefore, we decided to design an implementation from scratch. One of our additional requirements is that we want to use a standard Prolog implementation (to be able to reuse thousands of existing Prolog lines and maintain their efficiency), so we will avoid implementation-level manipulations.

We provide an additional step of **simplification** during the generation of frontier terms. We should take into account terms with universally quantified variables (that were not taken into account in [1, 2]) because without simplifying them it is impossible

** This work was partly supported by the Spanish MCYT project TIC2000-1632.

to obtain results. We also provide a variant in the **negation of terms with free variables** that entails universal quantifications. There is a detail that was not considered in former approaches and that is necessary to get a sound implementation: the existence of universally quantified variables by the iterative application of the method.

An instrumental step for managing negation is to be able to handle disequalities between terms with a "constructive" behaviour. Moreover, when an equation $\exists\, \overline{Y}.\ X = t(\overline{Y})$ is negated, the free variables in the equation must be universally quantified, unless affected by a more external quantification, i.e. $\forall\, \overline{Y}.\ X \neq t(\overline{Y})$ is the correct negation. As we explained in [3], the inclusion of disequalities and constrained answers has a very low cost.

Our constructive negation algorithm and the implementation techniques admit some additional optimizations that can improve the runtime behaviour of the system.

- **Compact representation of the information**. The advantage is twofold. On the one hand constraints contain more information and failing branches can be detected earlier (i.e. the search space could be smaller). On the other hand, if we ask for all solutions instead of using backtracking, we are cutting the search tree by offering all the solutions in a single answer.
- **Pruning subgoals**. The frontiers generation search tree can be cut with a double action over the ground subgoals: removing the subgoals whose failure we are able to detect early on, and simplifying the subgoals that can be reduced to true.
- **Constraint simplification**. During the whole process for negating a goal,the frontier variables are constrained. In cases where the constraints are satisfiable, they can be eliminated and where the constraints can be reduced to fail, the evaluation can be stopped with result *true*.

Having given a detailed specification of algorithm in a detailed way we proceed to provide a real, complete and consistent implementation. The results that we have reported are very encouraging, because we have proved that it is possible to extend Prolog with a constructive negation module relatively inexpensively and we have provided experimental results. Nevertheless, we are working to improve the efficiency of the implementation. This include a more accurate selection of the frontier based on the demanded form. Other future work is to incorporate our algorithm at the WAM machine level. We are testing the implementation and trying to improve the code, and our intention is to include it in the next version of Ciao Prolog [1].

## References


1. D. Chan. Constructive negation based on the complete database. In *ICLP'88*, pages 111–125. The MIT Press, 1988.
2. D. Chan. An extension of constructive negation and its application in coroutining. In *Proc. NACLP'89*, pages 477–493. The MIT Press, 1989.
3. S. Muñoz and J. J. Moreno-Navarro. How to incorporate negation in a prolog compiler. In E. Pontelli and V. Santos Costa, editors, *PADL'2000*, volume 1753 of *LNCS*, pages 124–140, Boston, MA (USA), 2000. Springer-Verlag.
4. P. Stuckey. Negation and constraint logic programming. In *Information and Computation*, volume 118(1), pages 12–33, 1995.


[1] http://www.clip.dia.fi.upm.es/Software