Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 246 (2009) 183–198
www.elsevier.com/locate/entcs

Extending Constructive Logic Negation with Types

Susana Munoz-Hernandez[1] [,2]

Babel Group,
Facultad de Informática, Universidad Politécnica de Madrid.
Campus de Montegancedo. Boadilla del Monte. Madrid-28660, Spain

Juan José Moreno-Navarro [3]

IMDEA-Software & Universidad Politécnica de Madrid.
Campus de Montegancedo. Boadilla del Monte. Madrid-28660, Spain

Abstract

Negation has traditionally been a difficult issue in Logic Programming. Most of Prolog programmers have been restricted to use just a weak negation technique, like negation as failure.
Many alternative semantics were proposed for achieving constructive negation in the last 20 years, but no implementation was provided so far because of its exponential complexity and the difficulty for developing it. First effective implementations of constructive negation into standard Prolog compilers are available just recently, around 2003, provided by our previous works.
In this paper we present an extension of our implementations by introducing types in programs, thus improving usability as well as efficiency in some cases of our implementations of constructive negation. This can make constructive negation an interesting approach for its use in data bases querying, web search, filtered search, ontologies querying, coding rules, business rules, etc.
Thanks to the use of types, our constructive negation can provide concrete values as results, instead of constraints (as in our previous works). We provide details about the semantics and the implementation in our approaches of classical, finite constructive, and intensional negation. The paper also includes some practical examples additionally allowing for providing measurements of computational behavior.

Keywords: Logic Programming Implementation, Negation, Types, Constraint Logic Programming, Constructive Negation, Non-monotonic Reasoning.

1 Introduction

The beginning of logic is tied with that of scientific thinking. Its application for modeling human reasoning is clear as a programming language. But one of the main elements of logic, that is negation, is hardly represented in Logic Programming.

[1] This work is partially supported by the project DESAFIOS - TIN 2006-15660-C02-02 from the Spanish Ministry of Education and Science and project PROMESAS - S-0505/TIC/0407 from Comunidad de Madrid.

[2] Email: susana@fi.upm.es

[3] Email: jjmoreno@fi.upm.es

doi:10.1016/j.entcs.2009.07.022

1.1 Logic Programming and Negation

Negation is probably the most significant aspect of logic that was not included from the outset. Dealing with negation involves significant additional complexity. Nevertheless, the use of negation is very natural and plays an important role in many knowledge representation and reasoning systems, like web semantics, natural language processing, constraints management in databases, program composition, manipulation and transformation, coding rule checking, business rules, default reasoning, negative queries (search of false information), etc.

There are many ways of understanding and incorporating negation into Logic Programming, the problems really start at the semantic level, where the different proposals differ not only in the semantics but also as to expressiveness. Unfortunately, current Prolog [4] compilers support a very limited number of negation techniques: negation as failure under Fitting/Kunen semantics [9] (sound only under some circumstances usually not checked by compilers) which is a built-in in most Prolog compilers (Quintus, SICStus, Ciao, BinProlog, etc.), and the "delay technique" (applying negation as failure only *when* the variables of the negated goal become ground, which is sound but incomplete due to the possibility of floundering) which is present in Nu-Prolog, Gödel, and Prolog systems that implement delays (most of the above).

Among all proposals, constructive negation [5,21] (that we will call *classical* constructive negation) is probably one of the most promising because it has been proved to be sound and complete, and its semantics is fully compatible with the Prolog one. A previous paper [4] provided a simpler variant for negating goals that have a finite number of solutions (that we will call *finite* constructive negation). Another interesting approach, different to these ones, is the transformation proposed by Barbuti et all [2] that we will call *intensional* constructive negation. In the paper we will use these three approaches.

Attending to what we have expounded in this section, it is clear the interest for achieving a sound and complete implementation for these techniques. 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. We are trying to fill a long time open gap in this area (remember that the original papers are from late 80s) facing the problem of providing a correct, effective and complete implementation, integrated into a standard Prolog compiler.

It was just during the last years [15,14] when we have provided effective implementations of some constructive negation techniques [5]. Here in this paper we improve the expressiveness and usability of our implementations of classical, finite and intensional constructive negation by including types.

[4] We understand Prolog as depth-first, left to right implementation of SLD resolution for Horn clause programs, ignoring, in principle, side effects, cuts, etc.

[5] More details about differences in between the constructive negation techniques can be found at [15,14]