Generalizing Allowedness While Retaining Completeness of SLDNF-Resolution

Hendrik Decker*
ECRC
Arabellastr. 17
D-8000 München 81, Germany

Lawrence Cavedon
Australian AI Institute
1 Grattan St.
Carlton, 3053, Australia

Abstract We propose various generalizations of the usual definition of allowedness used to prove the completeness of SLDNF-resolution. In particular, we define the property of recursively covered programs and goals. We show that, for programs and goals that are call-consistent, even and recursively covered, SLDNF-resolution computes a complete set of ground answers. We then propose further generalized conditions that ensure that SLDNF-resolution is flounder-free. Moreover, this allows us to define a class of programs that subsumes all three major syntactic classes of programs and goals for which SLDNF-resolution is known to be complete; i.e., programs and goals that are either definite, or hierarchical and weakly allowed, or call-consistent, strict and allowed. We conjecture that our generalizations preserve the completeness of SLDNF-resolution. We also investigate the possibility of weakening the other syntactic conditions, i.e., even and call-consistent, while retaining completeness.

Introduction: The problem, its background and a summary of results

SLDNF-resolution [L1] is known to be incomplete in general, i.e., there may be correct ground answers to queries (goals) that are not subsumed by the set of computed answers. The problem is to identify large classes of logic programs and goals for which SLDNF-resolution is complete.

Completeness results have been proved for classes of programs and goals that are defined by certain syntactic properties. For example, both SLD-resolution and the negation-as-failure rule are well-known to be complete for definite programs and goals [JLL] [L1]. Completeness has also been shown for programs and goals that are hierarchical and (weakly) allowed [Cl] [Sh1] [L1]. [CL] proves completeness for programs and goals that are allowed, strict and stratified [ABW]. More recently, completeness has been shown, in [Ku2], for programs and goals that are allowed, strict and call-consistent [Sa] (call-consistent is named semi-strict in [Ku2]).

All of the properties mentioned above are recursively decidable by static analysis. There are other properties that are sufficient for the completeness of SLDNF-resolution (see, e.g., [BM1] [Ca2]), but which are not recursively decidable. In this paper, we focus attention on syntactic properties that are known to be decidable, in particular, allowedness and generalizations thereof.

With regard to completeness, allowedness has been referred to as “the most restrictive condition”, “the most desirable ...” and “the most difficult to weaken” [CL]. In this paper, we introduce the property of recursively covered, which generalizes allowedness in the following way. Allowedness restrictions are no longer imposed on all variables, but only on those on which variables in the top-level goal depend (in a manner defined below).

*supported by ESPRIT BRA 3012
Further generalizations of the restrictions on variables are proposed in the long version of the paper [DC].

In section 1, we define syntactic properties that are used to prove known results about the completeness of SLDNF-resolution. In particular, we define the class of even programs and goals. This class was first introduced, informally, in [Ku2]. It generalizes the class of strict programs and goals. The main contribution of this paper is a completeness result for programs and goals that are call-consistent, even and recursively covered (section 2). This result generalizes the completeness theorem for programs and goals that are call-consistent, even and allowed [Ku2]. In section 3, we propose generalizations of other syntactic conditions; i.e., definite, as well as hierarchical and weakly allowed. These generalizations characterize classes of programs and goals that are flounder-free (i.e., no attempt to compute any derivation rooted at the goal ever reaches a non-empty goal consisting of only non-ground negative literals [LI]). The widest class that we show to be flounder-free consists of programs and goals that are generally covered. We conjecture that the joint conditions of call-consistent, even and generally covered preserve the completeness of SLDNF-resolution. Moreover, this class includes all syntactic classes of programs and goals for which SLDNF-resolution is known to be complete. Extensions of allowedness in order to capture language constructs the interpretation of which usually is “built-in” (e.g., equality, comparison operators) are discussed in section 4. In section 5, we investigate the possibility of weakening the definitions of even and call-consistent while retaining completeness.

There are approaches to obtaining completeness results for the evaluation of logic programs and goals other than generalizing syntactic properties. Some such approaches involve extending, or considering alternatives to, the usual declarative or procedural semantics. In this paper, we adhere to the most widely accepted declarative semantics of a logic program $P$, given by the logical consequences of its completion $\text{comp}(P)$ [Cl] [LI]. Except where explicitly stated otherwise, we assume the usual 2-valued logic. The procedural semantics of a logic program is taken to be SLDNF-resolution.

Discussion of various issues regarding completeness can be found in [Sh3] [Ca1] [PP] [Sh4].

1 Definitions of basic concepts

We assume the reader is familiar with the foundations of logic programming [LI] to the extent that the definitions below need not be explained beyond given examples and comments.

1.1 Dependencies

1.1.1 Definition (clause, goal, program clause, fact, logic program)

A logic programming clause (clause, for short) is an expression of the form $A \leftarrow W$, where $A$ is an atom and $W$ is a conjunction of literals. As usual, $A \leftarrow W$ stands for the universal closure of $A \lor \neg W$. We call $A$ the head and $W$ the body of $A \leftarrow W$. Either $A$ or $W$ may be absent. When $A$ is absent, $\leftarrow W$ is called a goal. Otherwise, $A \leftarrow W$ is called a program clause. A program clause with an empty body is called a fact. A (logic) program is a finite set of program clauses.