15 COOPERATION OF ABDUCTION AND INDUCTION IN LOGIC PROGRAMMING

Evelina Lamma, Paola Mello, Fabrizio Riguzzi, Floriana Esposito, Stefano Ferilli, and Giovanni Semeraro

15.1 INTRODUCTION

This chapter proposes an approach for the cooperation of abduction and induction in the context of Logic Programming. We do not take a stance on the debate on the nature of abduction and induction (see Flach and Kakas, this volume), rather we assume the definitions that are given in Abductive Logic Programming (ALP) and Inductive Logic Programming (ILP).

We present an algorithm where abduction helps induction by generating atomic hypotheses that can be used as new training examples or for completing an incomplete background knowledge. Induction helps abduction by generalizing abductive explanations.

A number of approaches for the cooperation of abduction and induction are presented in this volume (e.g., by Abe, Sakama, Inoue and Haneda, Mooney). Even if these approaches have been developed independently, they show remarkable similarities, leading one to think that there is a "natural way" for the integration of the two inference processes, as it has been pointed out in the introductory chapter by Flach and Kakas.

The algorithm solves a new learning problem where background and target theory are abductive theories, and abductive derivability is used as the example coverage relation. The algorithm is an extension of a basic top-down algorithm adopted in ILP (Bergadano and Gunetti, 1996) where the proof procedure defined in (Kakas and

Mancarella, 1990c) for abductive logic programs is used for testing the coverage of examples in substitution of the deductive proof procedure of Logic Programming.

The algorithm has been implemented in a system called LAP (Lamma et al., 1997) by using Sicstus Prolog 3#5. The code of the system and some of the examples shown in the chapter are available at http://www-lia.deis.unibo.it/Software/LAP/.

We also discuss how to learn abductive theories: we show that, in case of complete knowledge, the rule part of an abductive theory can be also learned without abduction. Abduction is not essential to this task, but it is essential in case of absence of information, i.e. when the background theory is abductive.

The chapter is organized as follows: in Section 15.2 we recall the main concepts of Abductive Logic Programming, Inductive Logic Programming, and the definition of the abductive learning framework. Section 15.3 presents the learning algorithm. In Section 15.4 we apply the algorithm to the problem of learning from incomplete knowledge, learning theories for abductive diagnosis and learning exceptions to rules. Our approach to the integration of abduction and induction is discussed in detail and is compared with works by other authors in Section 15.5. Section 15.6 concludes and presents directions for future work.

15.2 ABDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

In this section we recall the definitions of abduction and induction in Logic Programming given by Flach and Kakas in the introductory chapter and we add the satisfaction of integrity constraints to abduction and the avoidance of negative examples to induction.

15.2.1 Abductive Logic Programming

In a Logic Programming setting, an abductive theory (Kakas et al., 1997) is a triple \( \langle P, A, IC \rangle \) where:

- \( P \) is a normal logic program;
- \( A \) is a set of abducible predicates;
- \( IC \) is a set of integrity constraints in the form of denials, i.e.:
  \[ \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_{m+n}. \]

Given an abductive theory \( T = \langle P, A, IC \rangle \) and a formula \( G \), an abductive explanation \( \Delta \) for \( G \) is a set of ground atoms of predicates in \( A \) such that \( P \cup \Delta \models G \) (\( \Delta \) explains \( G \)) and \( P \cup \Delta \models IC \) (\( \Delta \) is consistent). When there exists an abductive explanation for \( G \) in \( T \), we say that \( T \) abductively entails \( G \) and we write \( T \models_A G \).

Negation As Failure (Clark, 1978) is replaced, in ALP, by Negation by Default (Eshghi and Kowalski, 1989) and is obtained by transforming the program into its positive version: for each predicate symbol \( p/\text{arity} \) in the program, a new predicate symbol