Declarative problem solving attempts to tackle problems using a high-level representation of the expert knowledge on the problem at hand. In a logical setting such a declarative representation would employ an alphabet of constants, functions and predicate symbols that naturally represent objects, functions and relationships between these objects in the domain of discourse. In a truly declarative representation of the problem, the logic theory contains knowledge known to be true about the problem domain rather than information on how to solve tasks, i.e. it does not contain definitions of concepts describing problem solving methods. In such a setting, problem solving in many cases consists simply in filling in missing information. This could for example be finding the extension of a predicate or some logical relationship between existing predicates.

In such cases, therefore, problem solving with a declarative representation consists of extending the existing description (e.g., logical theory) of the problem to a new one such that the problem goal is satisfied in this extended description. This process is otherwise known by the name of abduction. In this logical setting, abduction as a problem solving method assumes that the general data structure for the solution to a problem (or solution carrier) is at the predicate level and hence a solution is described in the same terms and level as the problem itself.

In artificial intelligence a prototypical problem where abduction is used is that of diagnosis. Here abduction is employed to produce an explanation for the observed, often faulty, behavior of the system in terms of primitive causes according to some theory describing how primitive faults in a system propagate and lead to observable faults. Similarly, in the problem of knowledge assimilation a new piece of information is assimilated via abduction by finding a coherent reason for this information according to the current state of the knowledge. In planning, the task of reaching...
a goal state is solved through abduction by generating a set of actions that would bring about this desired state.

In all these cases the solution to the problem task is given by extending the current knowledge of the problem domain, contained in the given theory that declaratively describes the problem, with new knowledge. In contrast, deductive approaches to such problems require that reification of the predicates is performed. In these approaches the solution carrier is now a term in the language instead of a predicate. In general, knowledge representation in this reified language is considerably more complex and at a lower level and often requires encoding information pertaining to the specific problem task or the algorithms to solve this task.

Analyzing a little further the process of abduction as this is currently used in problems of artificial intelligence we see that typically this extra knowledge that is generated is confined to particular parts (predicates) of the theory. These predicates are often called abducible predicates and form the “holes” of information in the given theory T. In many cases this type of knowledge is described as instance or scenario knowledge to emphasise the fact that the abductive explanation represents one particular case in which the observation or goal under consideration would hold. This relative confinement of abductive hypotheses to scenario knowledge is one of the characteristics of abduction that distinguishes it from induction. Another important aspect of abductive reasoning concerns the multitude of possible explanations/solutions that can exist for the same observation/goal. As such, abduction is sometimes referred to as “inference to the best explanation” to emphasize the need for selecting amongst the possible solutions.

We have argued above that declarative problem solving (at least in a logical setting) brings in abduction naturally. Conversely, if we have a framework where we can support abduction directly, then this opens the way for high-level problem solving. Such a framework is abductive logic programming (ALP): the subject of this special issue. This forms an extension of logic programming where abduction is supported directly by providing a suitable high-level representation scheme and proof procedures to compute abduction. The framework is also sometimes called open logic programming (OLP) to emphasize its representational aspect that some of its predicates are open with missing information in the theory that describes our problem.

In the context of ALP, an expert represents his or her strong definitional knowledge, i.e. knowledge which fully determines one or a group of predicates in terms of other predicates, as a set of logic programming rules defining non-abducible predicates, and weaker assertional knowledge as integrity constraints. Often, the integrity constraints specify explicitly (some of) the requirements for the validity and quality of the solution. The use of ALP presents two further important advantages:

- Modular representation of properties of the problem, and consequently, flexibility under changes of the specification of the problem, whether this means changing the requirements of the problem or changing the background data of the problem. An important case of this is the need to adjust an old solution in a minimal way to the new requirements.

- The natural structure of the problem domain can be made explicit easily in the declarative setting of ALP; this extra structure can subsequently be exploited by the abductive solver and could have a significant effect on the computational behavior. As a consequence, ALP often allows us to formalize a wide variety of problems. A recent survey of the field has revealed the potential application of abduction in areas
such as databases updates, belief revision, planning, diagnosis, natural language processing, default reasoning, user modeling, legal reasoning, multi-agent systems, scheduling, recognition and software engineering.

But this wide range of potential applications poses an enormous challenge for developing systems that are at the same time (i) sufficiently flexible to be applied for this variety of problems and (ii) computationally effective. At this point of time, there has been relatively little work to show how abduction can be effective for solving practical problems of real-life scale.

Today, we have languages and frameworks (and some systems for them) like that of ALP that can support abduction. The main emphasis till now has been on setting up such frameworks and showing how they provide a general approach to declarative problem solving. But the field lacks coherent methodological guidelines and general-purpose, working systems, that could be employed for real-life applications. Apart from developing further more robust systems for abduction there is a need to concentrate efforts in general methodologies and programming principles that would enable us to address application problems in a more systematic and computationally effective way.

Abductive logic programming forms a start in the development of frameworks of high-level declarative logic with abduction. This volume presents a number of papers studying the foundations of such frameworks and shows potential applications of this paradigm of declarative problem solving.

One group of papers in the special issue is concerned with the logics and semantics of abduction. The paper by Poole is a part of his project to combine logic and decision/game theory into a coherent framework. The paper proposes the independent choice logic and demonstrates the strong relationships to abduction. The conditional logic programming language proposed by Gabbay, Giordano, Martelli, Olivetti and Sapi- no supports hypothetical and counter-factual reasoning and is based on an abductive semantics. The authors also present a goal-directed abductive proof procedure.

The contributions of Sakama and Inoue, and of You, Yuan and Goebel are investigations of the relationships between abductive logic programming and disjunctive logic programming. Sakama and Inoue propose reductions from abductive logic programming to disjunctive logic programming and vice versa under different semantics. The paper of You, Yuan and Goebel addresses the question whether the abductive procedure by Eshghi and Kowalski can be extended to reason on disjunctive programs. The authors develop the regular extension semantics for disjunctive logic programs and demonstrate how to extend the Eshghi and Kowalski procedure for sound and complete query answering with respect to this semantics.

In the second group of papers, the focus is on inferential aspects of abduction. Kakas, Michael and Mourlas propose an integration of abductive reasoning with constraint logic programming techniques and show the feasibility of this approach in a number of benchmarks, including planning and scheduling. Iwayama and Satoh first show that abductive logic programs under the extended stable semantics can be reduced to logic programs with integrity constraint under stable semantics and then focus on the computation of stable models of such logic programs. They propose a bottom up algorithm for computing stable models based on techniques from truth maintenance systems and refine the algorithm with top down expectation.

The paper of Shanahan studies the use of abduction for planning. The paper presents a simple abductive meta-interpreter suitable for partial order planning in event
calculus and shows that its execution effectively implements traditional planning algorithms for Strips-like languages.

This special issue could be created only with the help of the editor-in-chief, Maurice Bruynooghe, of the authors who contributed high quality papers and of the referees whose criticisms and suggestions were valuable input to improve the papers. We thank them all. In particular, we wish to thank Michael Gelfond who handled the refereeing of the paper that one of us coauthored.