Intelligenza Artificiale 7 (2013) 113–124
DOI 10.3233/IA-130053
IOS Press

Automated reasoning

Marco Gavanelli[a,*] and Toni Mancini[b]
[a]*EnDiF Engineering Department, Ferrara University, Italy*
[b]*Computer Science Department, Sapienza University, Rome, Italy*

Abstract. Knowledge representation and automated reasoning are two of the pillars of Artificial Intelligence but, differently from other pillars, they are strictly intertwined. Depending on how knowledge is represented, different types of reasoning can be applied and, on the other hand, new developments in the automated reasoning column fosters new ideas on the knowledge representation side. The Italian community has been always very involved in these fascinating themes, and this is witnessed by the lively group of knowledge representation and automated reasoning (Rappresentazione della Conoscenza e Ragionamento Automatico, RCRA) of AI*IA.

In this paper we survey the developments on automated reasoning in the last 25 years, with particular emphasis on the research of the Italian community and of the RCRA group. The focus will be mainly on the algorithmic side, while a companion paper focuses more on the knowledge representation side, and on the vast area of semantic technologies.

Keywords: Automated reasoning, logic programming, constraints, planning and scheduling, propositional logic

1. Introduction

For thousands of years, scientists and philosophers have tried to understand and formalize how humans reason, which types of inferences are correct, which are not; whether the reasoning methods can be automatized, what can be proven and what cannot. The advent of powerful electronic computers spurred an endless amount of applications for formal methods, and the automated reasoning became soon one of the main areas of what will be called *Artificial Intelligence* (AI).

Under the vast umbrella of *automated reasoning*, there are actually a number of disciplines, ranging from reasoning in logics in the broader sense, to a number of specializations, e.g., logic programming, constraint reasoning, and various forms of non monotonic reasoning.

Clearly, the type of reasoning, its possible outcomes and complexity depend strongly on how knowledge is represented; a companion paper [141] will delve into the subject of knowledge representation (KR), complexity analysis of reasoning tasks in the various KR frameworks, and on the vast area of semantic technologies.

This paper does not have the ambition to present all the results developed in this wide and fascinating area, and it will necessarily have to cut many important branches. Our aim is to provide some hints about the research developed in the last 25 years by the Italian community, focusing in particular on the activities of the RCRA interest group (Rappresentazione della Conoscenza e Ragionamento Automatico, i.e., Knowledge Representation and Automated Reasoning, http://rcra.aixia.it) of the AI*IA, the Italian Association for Artificial Intelligence.

We apologize in advance with the readers and the authors for the many missing works.

In the rest of the paper, we will discuss some of the main advancements in the areas of Propositional Logic (Section 2), Logic Programming (Section 3), Constraint Reasoning (Section 4), Quantified Reasoning (Section 5) and Planning and Scheduling (Section 6).

*Corresponding author: Marco Gavanelli, EnDiF Engineering Department, Ferrara University, Italy. E-mail: marco.gavanelli@unife.it.

2. Propositional logic

Despite its simplicity, propositional logic has been successfully used as the target framework to encode reasoning tasks from higher-level formalisms.

At the core of such reasoning activities we often find the Propositional Satisfiability (SAT) problem, namely the issue of checking whether a propositional formula in conjunctive normal form (CNF) is satisfiable or not [18]. Despite the fact that SAT is a NP-complete problem, the last decade witnessed the development of algorithms (both complete, as DPLL and CDCL, and incomplete, as those based on local search) [18] and solver implementations reaching impressive performance. This is one of the main motivations for such an extensive use of SAT in AI reasoning tasks.

The Italian community has extensively worked on these topics. In [40] the authors present a compiler that translates a combinatorial problem specification into a SAT test. Problems are specified in a logic-based language called NP-Spec [35], which allows the user to define complex problems in a highly declarative way, and whose expressive power is such as to capture all problems which belong to the complexity class NP.

Example 1. Consider the well-known *graph coloring* problem: given a graph and a number $k > 0$ of colors, the problem amounts to assign to each node of the graph one of the k colors, in such a way that nodes linked by an edge are assigned different colors. The problem is NP-complete for $k \geq 3$.

An NP-Spec program for this problem is as follows:

```
DATABASE
        k = ...; // no. of colors
        n = ...; // no. of graph nodes
        edge = ...; // set of graph edges
SPECIFICATION
        Partition({1..n},coloring,k).
        fail <- edge(X,Y), coloring(X,C),
                  coloring(Y,C).
```

An NP-Spec program has two sections: the `DATABASE` section (which can be given as a separate file) defines a particular instance of the problem (i.e., a graph as for Example 1), while the `SPECIFICATION` section defines the search space (the set of partitions of the n nodes into k sets in Example 1) and the problem constraints. Constraints are specified as a stratified datalog program. The NP-Spec semantics is based on the notion of model minimality. In Example 1, a non-deterministically guessed element of the search space, i.e., a partition `coloring` of the n nodes into k sets (one for each color), is a solution to the problem if and only if *no* edge (X, Y) exists in the graph such that both X and Y are in the same set of `coloring`, i.e., literal `fail` is *not* inferred.

In [39] the authors solve typical Constraint Programming (CP) benchmark problems by compiling them into SAT instances and by using state-of-the-art solvers, showing how this approach can be very appealing. Different encodings can be used to convert a same problem into SAT, including the direct-encoding, the log-encoding, the support encoding and the log-support encoding [86].

SAT solvers are the main workhorse in the area of formal verification of transition systems [19, 20] called Bounded Model Checking (BMC). Differently from symbolic model checking [33, 50], in BMC Binary Decision Diagrams (BDD) are replaced by SAT decision procedures. This avoids the memory problems due to the BDD blow up and often speeds up the verification. BMC has been successfully applied to a large number of applications, from the verification of security protocols [7] to the validation of complex CPU design [56] and now is largely used in industry as a solid alternative to symbolic model checking. Extensions of propositional logic have also been exploited to develop decision procedures for various modal logics [96, 100, 140], and generalizations of SAT decision procedures have been proposed to deal with qualitative preferences [69].

In the last decade much work has been done in extending propositional logic as to decide the satisfiability of boolean combinations of propositional atoms and atoms in a given decidable first-order theory (or a combination of theories). Such a framework is known as Satisfiability Modulo Theories (SMT) [18]. Efficient and scalable SMT solvers have been released in the last years. They extend SAT algorithms like DPLL and CDCL with embedded decision procedures for several theories, like the theory of equality and uninterpreted functions, the theory of difference constraints over the rationals or the integers, the quantifier-free fragment of linear arithmetic over the rationals or the integers, the theory of arrays, and the theory of bit-vectors.

The Italian community has carried out the design and development of state-of-the-art SMT solvers (e.g., [52] and its previous releases), the design of decision procedures for new theories (e.g., the theory of costs [51]), the successful application of SMT solvers in several verification (e.g, [5, 8, 53, 135]) and reasoning (e.g., [6, 24]) tasks.

3. Logic programming

Logic programming is a declarative paradigm, in which the programmer states some logic formulae that are true, and the solver derives the conclusions that logically follow, usually driven by a query. A logic program is usually a set of clauses, in the form of implications

$$a \leftarrow l_1, l_2, \dots, l_n$$

in which a is an atom, and $l_1, \dots, l_n$ are literals (atoms or negated atoms). In case the literals $l_1, \dots l_n$ are true, the atom a is also considered true.

The Italian community has been very active in the field of Logic Programming; due to the limited space, we cannot delve in all the fields and this survey is necessarily limited to some of the results.

The concept of modularity in Logic Programming has been approached from two orthogonal lines of research (see [32] for a survey). One approach conceives modularity as a *meta-linguistic* concept [29, 30, 115, 129]: modules are viewed as independent subprograms in which the modular construct should be independent from the adopted logic language. This is essential to compose modules written in different languages.

The second approach is based on extending the syntax of programs with new logical connectives [11, 95, 111]. Many works in this area use implication goals of the form $D \supset G$ in the body of clauses; if in the context of a program $\mathcal{P}$, the goal $D \supset G$ is evaluated, this is interpreted operationally as a request to load the set of clauses D, add them to $\mathcal{P}$, attempt to prove G, and finally discard the new clauses after the derivation for G succeeds or fails.

3.1. *Answer set programming*

Various semantics have been proposed for logic programming; one of the most successful is the Stable Model Semantics [92]. Differently from other semantics, stable models are typically computed bottom-up, from known facts to goals, instead of from goals to facts. This requires a *grounding* phase, in which, starting from a program $\mathcal{P}$ one generates a ground program equivalent to $\mathcal{P}$. Afterwards, a search phase is started, which tries to find an *answer set*, i.e., a model of the ground program.

The first Answer Set Programming (ASP) solver was Smodels [127], then others followed. One of the most successful solvers is DLV [112], that also features disjunctive rules, aggregates [77], ontological reasoning [42], weak constraints [31], and there exist parallel implementations [131]. Reductions of ASP to SAT have also been investigated (see, e.g., [97]). ASP has an impressive number of practical applications, including knowledge representation, bioinformatics [72], hydroinformatics [89], team building [138].

3.2. *Abductive logic programming*

Logic programming is based on the deduction inference: starting from a known fact a and a rule $b \leftarrow a$, one can deduce the truth of b. While this ensures correctness of the derived facts, humans adopt other rules in very practical scenarios. The philosopher Peirce classifies the reasoning processes into deductive, inductive and abductive.

Inductive reasoning is the process of synthesizing new rules from known facts, possibly from positive and negative examples; this is surveyed in a companion paper in this same issue [82].

Abductive reasoning, instead, is the type of reasoning typically associated to diagnosis: given an observable b, and given that we know that $b \leftarrow a$, one way to explain the truth of b is to hypothesize the truth of a.

Abductive Logic Programming (ALP) [106, 108] extends logic programming to accommodate abductive reasoning. In ALP, there are some syntactically distinguished predicates that have no definition, and cannot be proven: an abductive proof-procedure will *assume* their possible truth, and provide the abduced literal in the answer.

Definition 1. An Abductive Logic Program (ALP) is a triple $T = \langle \mathcal{P}, \mathcal{A}, \mathcal{IC} \rangle$, where

- $\mathcal{P}$ is a logic program,
- $\mathcal{A}$ is a set of predicates, called abducible, that are not defined in $\mathcal{P}$,
- $\mathcal{IC}$ is a set of formulae, often in the form of implications, called Integrity Constraints.

Given an abductive logic program T and a formula G, the goal of abduction is to find a set $\Delta \subseteq \mathcal{A}$ of ground atoms such that

$$\mathcal{P} \cup \Delta \models G$$

and

$$\mathcal{P} \cup \Delta \models \mathcal{IC}.$$

Example 2. For example, an abductive program could be:

```
headache :- flu.
```

where `flu` is declared as an abducible predicate. Given the query `:- headache`, an abductive proof-procedure will provide as answer

```
yes, flu
```

meaning that the derivation succeeds, provided that the truth of the atom `flu` can be assumed.

A number of abductive proof-procedures have been proposed; some have also been integrated with constraint reasoning [3, 4, 107, 116], some are implemented on top of Answer Set Programming [132]. Abductive constraint programming languages have been used for a variety of applications, including agents, planning, web service composition [1, 2], web sites verification [117] and two-player games [87].

4. Constraint reasoning

Many problems in AI can be modeled through constraints. A Constraint Satisfaction Problem (CSP) is defined through a set of decision variables that range on some given domains, and a set of constraints (relations between the variables); a solution is sought that satisfies all constraints.

Definition 2. A Constraint Satisfaction Problem (CSP) is a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ such that $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains, and $\mathcal{C} = \{c_1, \ldots, c_k\}$ is a set of relations linking one or more variables taken from set $\mathcal{X}$.

A solution is an assignment $X_1 \mapsto x_1, \ldots, X_n \mapsto x_n$ of values $x_i \in D_i$ to the variables satisfying all the constraints, i.e., for each constraint $c(X_a, X_b, \ldots)$, the relation $c(x_a, x_b, \ldots)$ is true.

Theoretically, every problem in the NP class can be defined as a CSP. For example, one can model scheduling problems [128], resource assignment problems, crew rostering problems, flow problems, routing problems, just to name a few. Also, by suitably restricting the structure of the problem, interesting tractable subclasses can be defined (see, e.g., [101, 102]).

As in many real-life problems there can be more than one solution satisfying all the constraints, some solutions could be preferable to others. In this case, one typically defines an objective function, which should be minimized or maximized. The resulting problem is called a Constraint Optimization Problem (COP). In multi-criteria optimization [85], more than one objective function is considered.

Constraint satisfaction and optimization has been addressed through many different techniques. In a first classification, we distinguish complete algorithms from heuristic ones. Complete algorithms use systematic search techniques (usually based on tree search) that ensure that a solution will be definitely found, if it exists; in case there is no solution, they are able to prove it. For optimization problems, the optimal solution will be provided, together with a proof of optimality. Crucial for these algorithms is the ability to avoid searching all the search space, by *pruning* some branches of the search tree. This is usually done by using *constraint propagation* techniques to remove a-priori inconsistent values from the domains of the variables, taking advantage of local consistency notions and other structural properties of the CSP at hand (see, e.g, [25, 27]). One of the first, and most used local consistency notions is *arc-consistency* [114].

Definition 3. A constraint $c(X, Y)$ is arc-consistent if and only if:

– $\forall x \in D_X, \exists y \in D_Y$ such that $c(x, y)$

and, vice-versa:

– $\forall y \in D_Y, \exists x \in D_X$ such that $c(x, y)$.

Example 3. Consider a scheduling problem in which a set of activities $A_1, \ldots, A_n$ must be scheduled on one machine. Each activity A_i has a duration d_i, cannot be executed before some earliest start time e_i, and cannot terminate later than some latest end time l_i. The machine can execute only one activity at a time.

This problem can be modeled as a CSP in which the variables are the start times (one for each activity) $S_1, \ldots, S_n$; the domain of $S_i = [e_i, l_i - d_i]$, and, for each pair of activities (A_i, A_j), we have a constraint stating that the two activities cannot overlap:

$$S_i + d_i \leq S_j \vee S_j + d_j \leq S_i.$$

In may cases, a problem can be modeled in different ways as a CSP. Usually, constraints that involve many variables (often called *global constraints*) are able to perform clever pruning by using specialized algorithms.

Example 4. Consider a CSP consisting of 3 variables A, B and C, that range on domains all equal to $\{0, 1\}$; the set of constraints requires that all the variables should take different values, i.e., $\mathcal{C} = \{A \neq B, B \neq C, A \neq C\}$. Arc-consistency propagation does not remove any value from the domains, but, by reasoning on the cardinalities, one can easily see that there is no solution: we have three variables that should take different values taken from a domain of cardinality two. The specialized constraint `alldifferent` [137] imposes that a set of variables take mutually different values, and is able to perform stronger propagation, by relying on graph algorithms.

In the scheduling problem, the global constraint `cumulative` [12] takes as arguments three equally-long lists of values:

– S: a list of start times
– D: a list of durations
– R a list of resource consumptions

plus a resource limit L; *cumulative*(S, D, R, L) enforces that in any time, the resource usage of the tasks does not exceed the limit. This constraint is able to perform a very strong propagation.

However, in very large or difficult problems, the search space is so large that even with the help of pruning, exploring all the search space cannot be done in reasonable time, so one has to stop the search after some timeout, and take the best solution found so far even if it is not optimal. In such cases, heuristic algorithms can be more effective. These algorithms use techniques ranging from local search [67] to Genetic Algorithms, to Ant Colony Optimization [130], just to name a few. These algorithms propose efficient techniques to drive the search toward promising solutions, and can typically find very good solutions in a short time. On the other hand, they give up the idea of systematic search, so they cannot find a provably optimal solution.

The solving algorithms have been integrated into programming languages; the first languages to enjoy a constraint solver were logic languages. In fact, constraint solvers integrate well into the logic paradigm, that features logic variables interpreted as unknowns, and backtracking search. This spurred the classes of Constraint Logic Programming languages [105], that is parametric with respect to the so-called *sort*. There exist now various languages in the CLP framework, each on a different domain: CLP(FD), on the sort of Finite Domains (i.e., decision variables can range on domains which are finite), CLP(R), on the reals, CLP(SET) [75], in which variables range on sets, etc. There are also works about the integration of different solvers [61, 88]. A recent survey of the works in CLP, focused on the Italian community, can be found in [91].

Beside the pre-defined sorts, it is also possible to define new sorts and solvers. One example of a language able to define new solvers in a declarative way is Constraint Handling Rules (CHR).

Example 5. For example, in CHR one can define the less or equal constraint simply by stating the usual properties as follows:

$$\begin{aligned}
&\textit{reflexivity} \quad @ \quad A \leq A \Leftrightarrow \textit{true}\\
&\textit{antisymmetry} @ A \leq B, B \leq A \Leftrightarrow A = B.\\
&\textit{transitivity} \quad @ A \leq B, B \leq C \Rightarrow A \leq C.
\end{aligned}$$

and the solver will take care to perform the propagation; e.g., if the user imposes the constraints $X \leq Y, Y \leq X$, the solver applies the second rule, removes the two constraints and imposes the unification $X = Y$. Differently from the first two rules, the transitivity rule contains an implication symbol, and this is interpreted operationally as adding the new constraint $A \leq C$ without removing the two constraints $A \leq B$ and $B \leq C$.

For CHR, a number of features have been proven [68], amongst which confluence, compositionality [65], and decidability [84]. The number of applications developed in CHR is impressive (see, e.g., the web page[1] *"The first fifty applications using CHR"*, amongst which we find many works of Italian researchers [21, 71, 136]). There are also a number of extensions of CHR, for example to allow for a probabilistic weighting of the rules [83].

Constraint solvers have also been developed as libraries of object oriented languages, typically C++ or Java. Initially, libraries were based on constraint propagation and tree search, like ILOG Solver or GeCode; more recently, libraries based on local search were also proposed (e.g., [67]) as well as solvers like the one in [37, 121] which also offers a declarative language to model both a combinatorial problem and the local search strategy to be used.

The performance of constraint and ASP solvers has also been compared experimentally [73, 120].

[1] http://www.cs.kuleuven.be/~dtai/projects/CHR/chr-appls.html

Other ways to improve the solution process are to change the problem formulation. For example, some approaches include rewriting (through folding and unfolding steps) a constraint logic program [79], to make it more efficient for a specific instance or a query. Another idea is to remove some symmetrical parts of the search space; this can be done either by rewriting the constraint program, in a new model that does not contain symmetries, or by adding the so-called *symmetry breaking constraints*.

Example 6. Suppose that in a scheduling problem, we have n activities that have to be scheduled on a single machine. Suppose that, amongst the n activities, 3 of them are identical (identical duration, identical resource usage, identical requirements in terms of preceding activities, etc.).

In this case, if we are given a solution, we can always compute another equivalent solution by permuting these 3 activities. So, if we remove this symmetry, we could reduce the search space of a factor of 3!. In order to remove this symmetry, we can impose an order amongst these 3 activities, for example with

$$S_1 + d_1 \leq S_2, S_2 + d_2 \leq S_3.$$

Completeness is retained, in the sense that each solution removed by these constraints is symmetric to some other solution that is not removed; so, for example, if we have the objective to minimize the makespan, we will still be able to find an optimal solution.

Finding symmetries can be difficult if done by hand, but there are works in which the symmetries are found automatically [118].

By regarding constraint specifications as logical formulas, one can also perform automated reasoning activities on them, in order to recognize notable structural properties and exploit them by reformulation (see, e.g., [36]) or by synthesizing an optimized search strategy [119]. Such reasoning tasks can be conveniently carried out by exploiting automated theorem proving technology [38].

It is worth noting that in Operations Research a variety of techniques have been designed to solve constraint optimization problems, and they are very effective in specific problem types, although constraint programming approaches are more effective on others. For this reason, a number of researchers started hybridizing constraint programming with operations research methods, for example by exploiting the information provided by linear solvers, like reduced costs [81], or by decomposing the problems with Bender's decomposition [17, 104], local branching [110], or column generation [103].

Finally, various methods exist to integrate local search with CLP, e.g., [80].

Amongst the applications of constraint reasoning there are visual search [58], sport scheduling [139], protein folding [60, 72], hydroinformatics problems [46], optimal placement of biomass power plants [45], and computational sustainability problems [90], as well as static analysis, abstract interpretation [10, 66] and security verification [14, 57].

4.1. Extensions

A number of extensions have been proposed for the classical constraint satisfaction problem model. The Interactive CSP model [59] overcomes the limit of having domains defined at the beginning of the search, and tries to obtain domain values during search. In fact, in some applications the domains are unknown at the beginning, and domain values are computed through an expensive process, thus minimizing the number of extracted values is crucial.

In some cases, satisfying all the constraints is impossible, so some of them are declared as *soft* constraints. While *hard* (usual) constraints must be satisfied in any solutions, soft constraints can be violated, but with a penalty. A number of frameworks have been proposed to address this very practical need, including fuzzy reasoning, probabilistic extensions, the Semiring-based CSP [22].

Other references can be found in [91].

5. Quantified reasoning

Many reasoning activities of practical interest in AI, as contingent planning, adversarial game playing, control design, and model checking, must deal with various forms of uncertainty, as adversary players, exogenous actions, or uncontrollable events.

To handle these problems, suitable formalisms have been proposed. Two such formalisms have raised much interest in the last decade: QBF and QCSP.

Definition 4. A Quantified Boolean Formula (QBF) has the form:

$$Q_1 X_1 \; Q_2 X_2 \; \cdots Q_n X_n \;\; \varphi(X_1, \ldots, X_n) \qquad (1)$$

where $\varphi(X_1, \ldots, X_n)$ is a propositional formula involving the propositional variables $X_1, \ldots, X_n$ and every

Q_i $(1 \leq i \leq n)$ is either an existential ($\exists$) or a universal quantifier ($\forall$).

The expression $\exists X_i \psi$ means that there exists a truth assignment to X_i such that ψ is true. Analogously, $\forall X_i \psi$ means that for each truth assignment to X_i, ψ is true.

Quantifier alternation is best understood using an "adversarial" or "game-theoretic" viewpoint, where two players interact. One of them is allowed to choose the values for the existential variables, and its aim is to ultimately make the formula true, while the other assigns the universal variables and aims at falsifying it.

Example 7. The following formula is a QBF:

$$\exists X\, \forall Y\, \exists Z\ \ (X \vee \neg Y) \wedge (\neg Z \vee \neg X \vee Y) \qquad (2)$$

The formula is true if and only if the existential player has a choice to set the truth value of X such that, for all values that the universal players can give to Y, the existential player can always choose a value for Z in order to satisfy $(X \vee \neg Y) \wedge (\neg Z \vee \neg X \vee Y)$.

Formula (2) is actually true. This is because, if the existential player assigns X to true, then (s)he can assign Z to false for any value that can the universal player give to Y.

Equivalently, we can say that a QBF is true if and only if the existential player has a *winning strategy*. A winning strategy for the existential player is a function *for each* existential variable X_i that specifies which value to pick for X_i to satisfy the quantifier-free formula, depending on the values assigned to the universal variables that precede it.

To a QBF we can associate an *and-or* tree, where *and* (respectively, *or*) nodes are associated to universal (respectively, existential) variables. Figure 1 shows the *and-or* tree T associated to the QBF (2) of Example 7. A winning strategy can be seen as a subtree W of T, inductively defined as follows:

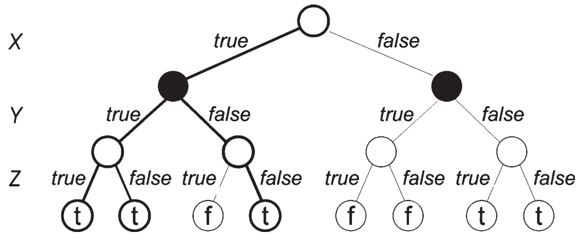

Fig. 1. The and-or tree associated to QBF (2): *and* and *or* nodes are denoted, respectively, in black and white background. Paths from the root to leaves denoted with "t" (*true*) identify models of the quantifier-free propositional formula.

1. W contains and is rooted at the root note of T,
2. for each *and* (respectively *or*) node of T that belongs to W, all (respectively, at least one of) its children in T belong also to W, and
3. all leaves of W (which are also leaves of T) are labelled with *true*.

As QBF can be seen as a generalization of SAT, a Quantified Constraint Satisfaction Problem (QCSP) can be regarded as a generalization of CSP (Definition 2) where variables can be existentially or universally quantified.

Example 8. The following formula

$$\exists X_1 \in [2,3] \forall X_2 \in [3,4] \exists X_3 \in [3,6]\ X_1 + X_2 \leq X_3$$

defines a QCSP. The problem is true if and only if there exists a value in [2, 3] for X_1 such that, for all values in [3, 4] that can be assigned to X_2, there exists a value in [3, 6] for X_3 for which $X_1 + X_2 \leq X_3$.

Dealing with quantifier alternations makes reasoning on these frameworks more difficult than solving a SAT or CSP instance (solving a QBF or a QCSP is PSPACE-complete). Notwithstanding this complexity, much work has been done in implementing efficient algorithms for these problems. Today, current solvers can tackle real problems of considerable size.

The Italian community has been involved in this fascinating area since its beginning, by proposing solvers for QBF (as the early one in [34] up to parallel [113] and multi-engine solvers [133]) and QCSP [15, 16], as well as search algorithms and preprocessing and pruning techniques [26, 28, 41, 99, 134]. QBF formulas with a $\exists\forall$ quantifier prefix (called 2QBF) can be also uniformly encoded in ASP. Experiments show [123] that current ASP solvers (and, in particular, DLV, see Section 3.1) are competitive on such formulas.

The effectiveness of such approaches has been assessed in various application areas, as the verification of circuits design [122, 124]. Also, libraries of benchmark problems have been published and comparative evaluations of the available QBF solvers have been performed on them [125, 126] to monitor the advancements of the state-of-the-art solvers.

6. Planning and scheduling

Planning considers courses of actions that should be performed to obtain a goal, starting from a given state of the world; executing an action changes the state of the world, and, in particular, can make executable actions that were not applicable and vice-versa.

In scheduling, instead, a set of actions (often named *tasks*) is given in input, but not all actions can be executed at the same time: some of them require other actions to be executed before, some require some limited resources, some can be executed only in given time-windows, etc. The objective is deciding when each action should be executed, satisfying all the constraints. There is often an objective function to be optimized, that could be minimizing the whole duration (makespan) of the schedule, minimizing some cost (that, for example, can depend on the usage of resources), reducing the delays of some of the activities, etc.

In short, *"scheduling is primarily concerned with figuring out when to carry out actions while planning is concerned with what actions need to be carried out. In practice, this distinction often blurs and many real-world problems involve figuring out both what and when"* [63], and indeed in various applications planning and scheduling have been integrated [48].

Various techniques have been used to solve planning problems, like forward or backward chaining, using graph representations [23], using tactics [142], reduction to SAT [44, 78, 94, 98, 109], Local Search [93], reduction to LTL [43] or to model checking [54, 64]. Other planning algorithms are based on CLP(FD) [13, 74].

The applications of planning are countless. Probably, the most intuitive application of planning is deciding the movements of robots. Indeed, in robotics there are a number of problems that make planning even more difficult, like considering sensing actions [62] or the need to take into consideration the actions of humans [55].

Beside these intuitive applications, planning has been used to train decision makers to teach them how to take joint decisions under stress [47], and to decide the first attack on a forest fire [9].

Scheduling has been addressed, in the CSP world, with two main approaches [49]. The first is assigning to each task a decision variable representing its start time, and taking decisions by assigning them precise time instants [128].

The second is associating decision variables with precedence relations between activities. An effective procedure to find feasible schedules in the presence of limited resources is to identify minimal sets of activities causing a resource over-usage in case of overlapping execution, and then introduce during search precedence constraints so that the set of conflicting activities cannot be scheduled at the same time [49].

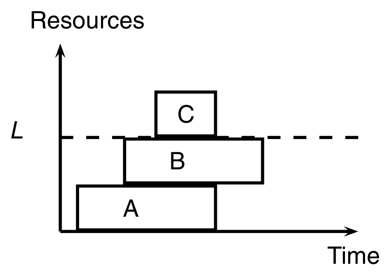

Fig. 2. Example of a scheduling problem.

Example 9. For example, suppose to have three activities A, B and C; each activity consumes one resource unit, and we have limit L of two resources.

One way to solve the problem is to solve the (simpler) problem that disregards the resource limit. We could get a solution like the one depicted in Fig. 2, that clearly does not satisfy the constraint on the resource limit.

In this case, one can select two of the activities in the conflict (say, A and B), and impose that these two activities do not overlap in time: either A precedes B, or B precedes A.

7. Conclusions

In this paper, we surveyed the research that the Italian community, and the RCRA interest group in particular, carried out on topics related to Automated Reasoning in the last 25 years. The RCRA interest group witnessed much of these research efforts, as it organizes yearly workshops since 1993. From 2007, the RCRA workshop series has an international call for papers and attracts researchers from all over the world. It has been co-located with important AI conferences, such as ICLP, CP-AI-OR, IJCAI, AI*IA, and edits special issues in international journals.

We are well aware that we left behind many important works published by Italian colleagues. We apologize with the authors. Unfortunately, this was necessary (although painful) for space reasons.

References

[1] M. Alberti, M. Cattafi, M. Gavanelli, E. Lamma, F. Chesani, M. Montali, P. Mello and P. Torroni, Integrating abductive logic programming and description logics in a dynamic contracting architecture. In *IEEE Int Conf on Web Services*, 2009.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello and M. Montali, An abductive framework for a-priori verification of web services. In *PPDP*, 2006.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello and P. Torroni, Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logics* **9**(4) (2008).

[4] M. Alberti, M. Gavanelli and E. Lamma, The CHR-based implementation of the SCIFF abductive system. *Fundam Inform* **124**(4) (2013), 365–381.

[5] P. Arcaini, A. Gargantini and E. Riccobene, Optimizing the automatic test generation by SAT and SMT solving for boolean expressions. In P. Alexander, C.S. Pasareanu, and J.G. Hosking, editors, *ASE*, IEEE, 2011, pp. 388–391.

[6] A. Armando, M.P. Bonacina, S. Ranise and S. Schulz, New results on rewrite-based satisfiability procedures. *ACM Trans Comput Log* **10**(1), (2009).

[7] A. Armando and L. Compagna, Sat-based model-checking for security protocols analysis. *International Journal of Information Security* **7**(1) (2008), 3–32.

[8] A. Armando, J. Mantovani and L. Platania, Bounded model checking of software using SMT solvers instead of sat solvers. *STTT* **11**(1) (2009), 69–83.

[9] P. Avesani, A. Perini and F. Ricci, Interactive case-based planning for forest fire management. *Appl Intell* **13** (2000).

[10] R. Bagnara, R. Gori, P.M. Hill and E. Zaffanella, Finite-tree analysis for constraint logic-based languages, In *SAS* (2001).

[11] M. Baldoni, L. Giordano and A. Martelli, A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning. *J Log Comput* **8**(5) (1998), 597–635.

[12] P. Baptiste and C. Le Pape, A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *IJCAI* (1995), 600–606.

[13] R. Barruffi and M. Milano, Interactive constraint satisfaction techniques for information gathering in planning, In *ECAI* (1998).

[14] G. Bella and S. Bistarelli, Soft constraint programming to analysing security protocols. *TPLP* **4**(5-6) (2004), 545–572.

[15] M. Benedetti, A. Lallouet and J. Vautard, Reusing CSP propagators for QCSPs. In F. Azevedo, P. Barahona, F. Fages, and F. Rossi, editors, *CSCLP*, volume 4651 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 63–77.

[16] M. Benedetti, A. Lallouet and J. Vautard, QCSP made practical by virtue of restricted quantification. In M.M. Veloso, editor, *IJCAI* (2007), pp. 38–43.

[17] L. Benini, M. Lombardi, M. Mantovani, M. Milano and M. Ruggiero, Multi-stage Benders decomposition for optimizing multicore architectures. In *CPAIOR* (2008).

[18] A. Biere, *Handbook of satisfiability*, volume 185, IOS Press, 2009.

[19] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman and Y. Zhu, Bounded model checking. *Advances in Computers* **58** (2003), 117–148.

[20] A. Biere, A. Cimatti, E.M. Clarke and Y. Zhu, Symbolic model checking without BDDs. In *Proc of TACAS 1999*, volume 1579 of LNCS, Springer, 1999.

[21] S. Bistarelli, T.W. Frühwirth and M. Marte, Soft constraint propagation and solving in CHRs. In *SAC ACM*, (2002).

[22] S. Bistarelli, U. Montanari and F. Rossi, Semiring based constraint solving and optimization. *Journal of the ACM* **44**(2) (1997), 201–236.

[23] A. Blum and M.L. Furst, Fast planning through planning graph analysis. *Artif Intell* **90**(1-2) (1997), 281–300.

[24] M.P. Bonacina, C. Lynch and L.M. de Moura, On deciding satisfiability by theorem proving with speculative inferences. *J Autom Reasoning* **47**(2) (2011), 161–189.

[25] L. Bordeaux, M. Cadoli and T. Mancini, Exploiting fixable, substitutable and determined values in constraint satisfaction problems. In *LPAR*, volume 2923 of LNCS, Springer, 2004, pp. 270–284.

[26] L. Bordeaux, M. Cadoli and T. Mancini, CSP properties for quantified constraints: Definitions and complexity. In M.M. Veloso and S. Kambhampati, editors, *AAAI*, pp. 360–365. AAAI Press/The MIT Press, 2005.

[27] L. Bordeaux, M. Cadoli and T. Mancini, A unifying framework for structural properties of CSPs: Definitions, complexity, tractability. *J Artif Intell Res (JAIR)* **32** (2008), 607–629.

[28] L. Bordeaux, M. Cadoli and T. Mancini, Generalizing consistency and other constraint properties to quantified constraints. *ACM Trans Comput Log* **10**(3) (2009).

[29] A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi and M.C. Meo, Differential logic programming. In *POPL* (1993).

[30] A. Brogi, E. Lamma and P. Mello, Compositional modeltheoretic semantics for logic programs. *New Generation Comput* **11**(1), (1992).

[31] F. Buccafurri, N. Leone and P. Rullo, Strong and weak constraints in disjunctive datalog. In J. Dix, U. Furbach, and A. Nerode, editors, *LPNMR*, volume 1265 of LNCS, Springer, 1997, pp. 2–17.

[32] M. Bugliesi, E. Lamma and P. Mello, Modularity in logic programming. *J Log Program* **19/20** (1994), 443–502.

[33] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, Symbolic model checking: 10^{20} states and beyond. *Information and Computation* **98**(2), (1992).

[34] M. Cadoli, A. Giovanardi and M. Schaerf, An algorithm to evaluate quantified boolean formulae. In J. Mostow and C. Rich, editors, *AAAI/IAAI*, (1998), pp. 262–267. AAAI Press/The MIT Press.

[35] M. Cadoli, G. Ianni, L. Palopoli, A. Schaerf and D. Vasile, NP-SPEC: An executable specification language for solving all problems in NP. *Computer Languages* **26**(2-4) (2000), 165–195.

[36] M. Cadoli and T. Mancini, Automated reformulation of specifications by safe delay of constraints. *Artificial Intelligence* **170**(8-9) (2006), 779–801.

[37] M. Cadoli and T. Mancini, Combining relational algebra, SQL, constraint modelling, and local search. *TPLP* **7**(1-2) (2007), 37–65.

[38] M. Cadoli and T. Mancini, Using a theorem prover for reasoning on constraint problems. *Applied Artificial Intelligence* **21**(4/5) (2007), 383–404.

[39] M. Cadoli, T. Mancini and F. Patrizi, SAT as an effective solving technology for constraint problems. In *Proc of IS-MIS 2006*, volume 4203 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 540–549.

[40] M. Cadoli and A. Schaerf, Compiling problem specifications into SAT. *Artificial Intelligence* **162**(1-2) (2005), 89–120.

[41] M. Cadoli, M. Schaerf, A. Giovanardi and M. Giovanardi, An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *J Autom Reasoning* **28**(2) (2002).

[42] F. Calimeri, S. Galizia, M. Ruffolo and P. Rullo, Enhancing disjunctive logic programming for ontology specification. In F. Buccafurri, editor, *APPIA-GULP-PRODE*, 2003.

[43] D. Calvanese, G. De Giacomo and M. Vardi, Reasoning about actions and planning in LTL action theories. In D. Fensel, F. Giunchiglia, D.L. McGuinness, and M.-A. Williams, editors, *KR*, Morgan Kaufmann, 2002.

[44] C. Castellini, E. Giunchiglia and A. Tacchella, SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artif Intell* **147**(1-2) (2003).

[45] M. Cattafi, M. Gavanelli, M. Milano and P. Cagnoli, Sustainable biomass power plant location in the Italian Emilia-Romagna region. *ACM Transactions on Intelligent Systems and Technology* **2**(4) (2011) 33 1–33.

[46] M. Cattafi, M. Gavanelli, M. Nonato, S. Alvisi and M. Franchini, Optimal placement of valves in a water distribution network with CLP(FD). *Theory and Practice of Logic Programming* **11**(4-5) (2011), 731–747.

[47] A. Cesta, G. Cortellessa, R. De Benedictis and K. Strickland, Using planning to train crisis decision makers. In R. Pirrone and F. Sorbello, editors, *AI*IA*, LNCS, 2011.

[48] A. Cesta, R. De Benedictis, A. Orlandini, R. Rasconi, L. Carotenuto and A. Ceriello, Integrating planning and scheduling in the ISS fluid science laboratory domain. In *IEA/AIE*, volume 7906 of LNCS, Springer, 2013.

[49] A. Cesta, A. Oddi and S. Smith. A constraint-based method for project scheduling with time windows. *J Heuristics* **8**(1) (2002), 109–136.

[50] A. Cimatti, E.M. Clarke, F. Giunchiglia and M. Roveri, NuSMV: A new symbolic model checker. *STTT* **2**(4) (2000) 410–425.

[51] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani and C. Stenico, Satisfiability modulo the theory of costs: Foundations and applications. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of LNCS, Springer, 2010.

[52] A. Cimatti, A. Griggio, B. Schaafsma and R. Sebastiani, The MathSAT5 SMT solver. In N. Piterman and S. Smolka, editors, *TACAS*, volume 7795 of LNCS, 2013.

[53] A. Cimatti, S. Mover and S. Tonetta, SMT-based scenario verification for hybrid systems. *Formal Methods in System Design* **42**(1) (2013), 46–66.

[54] A. Cimatti, M. Pistore, M. Roveri and P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* **147**(1) (2003), 35–84.

[55] M. Cirillo, L. Karlsson and A. Saffiotti, Human-aware task planning: An application to mobile robots. *ACM TIST* **1**(2) (2010), 15.

[56] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella and M.Y. Vardi, Benefits of bounded model checking at an industrial setting. In *Proc of CAV 2001*, volume 2102 of Lecture Notes in Computer Science, Springer, 2001.

[57] R. Corin and S. Etalle, An improved constraint-based system for the verification of security protocols. In *SAS*, 2002.

[58] R. Cucchiara, M. Gavanelli, E. Lamma, P. Mello, M. Milano and M. Piccardi, Extending CLP(FD) with interactive data acquisition for 3D visual object recognition. In *Proc PACLP 1999* (1999), 137–155.

[59] R. Cucchiara, M. Gavanelli, E. Lamma, P. Mello, M. Milano and M. Piccardi, From eager to lazy constrained data acquisition: A general framework. *New Generation Computing* **19**(4) (2001), 339–367.

[60] A. Dal Palú, A. Dovier, F. Fogolari and E. Pontelli, CLP-based protein fragment assembly. *TPLP* **10**(4-6) (2010).

[61] A. Dal Palú, A. Dovier, E. Pontelli and G. Rossi, Integrating finite domain constraints and CLP with sets. In *PPDP* (2003).

[62] G. De Giacomo, L. Iocchi, D. Nardi and R. Rosati, Planning with sensing for a mobile robot. In *ECP* (1997).

[63] T. Dean and S. Kambhampati, Planning and scheduling. In A. Tucker, editor, *The Computer Science and Engineering Handbook*, CRC Press, 1997, pp. 614–636.

[64] G. Della Penna, D. Magazzeni, F. Mercorio and B. Intrigila. UPMurphi: A tool for universal planning on PDDL+problems. In A. Gerevini, A. E. Howe, A. Cesta, and I. Refanidis, editors, *ICAPS*, AAAI, 2009.

[65] G. Delzanno, M. Gabbrielli and M. Meo, A compositional semantics for CHR. In *PPDP '05*, ACM, 2005.

[66] G. Delzanno, R. Giacobazzi and F. Ranzato, Static analysis, abstract interpretation and verification in (constraint logic) programming. In A. Dovier and E. Pontelli, editors, *25 Years GULP*, volume 6125 of Lecture Notes in Computer Science, Springer, 2010, pp. 136–158.

[67] L. Di Gaspero and A. Schaerf, Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Softw Pract Exper* **33**(8) (2003), 733–765.

[68] C. Di Giusto, M. Gabbrielli and M.C. Meo, On the expressive power of multiple heads in CHR. *ACM Trans Comput Log* **13**(1) (2012), 6.

[69] E. Di Rosa, E. Giunchiglia and M. Maratea, Solving satisfiability problems with preferences. *Constraints* **15**(4) (2010).

[70] J. Dix, U. Furbach and A. Nerode, editors. *LPNMR'97*, volume 1265 of LNCS. Springer, 1997.

[71] G. Dondossola and E. Ratto, GRF temporal reasoning language. Technical report, CISE, Milano, 1993.

[72] A. Dovier, Recent constraint/logic programming based advances in the solution of the protein folding problem. *Intelligenza Artificiale* **5**(1) (2011), 113–117.

[73] A. Dovier, A. Formisano and E. Pontelli, A comparison of CLP(FD) and ASP solutions to NP-complete problems. In M. Gabbrielli and G. Gupta, editors, *ICLP*, volume 3668 of Lecture Notes in Computer Science. Springer, 2005.

[74] A. Dovier, A. Formisano and E. Pontelli, An investigation of multi-agent planning in CLP. *Fundam Inform* **105**(1-2) (2010).

[75] A. Dovier, C. Piazza and G. Rossi, A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. *ACM Trans Comput Log* **9**(3) (2008).

[76] A. Dovier and E. Pontelli, editors. *A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP*, volume 6125 of Lecture Notes in Computer Science. Springer, 2010.

[77] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi and G. Ielpa, Design and implementation of aggregate functions in the DLV system. *TPLP* **8**(5-6) (2008), 545–580.

[78] P. Ferraris and E. Giunchiglia, Planning as satisfiability in nondeterministic domains. In *AAAI/IAAI*, 2000.

[79] F. Fioravanti, A. Pettorossi, M. Proietti and V. Senni, Program transformation for development, verification, and synthesis of programs. *Intelligenza Artificiale* **5**(1) (2011).

[80] F. Focacci, F. Laburthe and A. Lodi, Local search and constraint programming: LS and CP illustrated on a transportation problem. In M. Milano, editor, *Constraint and Integer Programming. Towards a Unified Methodology*, 2003.

[81] F. Focacci, M. Milano and A. Lodi, Soving TSP with time windows with constraints. In *ICLP* (1999).

[82] N. Di Mauro, P. Frasconi, F. Angiulli, D. Bacciu, M. de Gemmis, F. Esposito, N. Fanizzi, S. Ferilli, M. Gori, F. A. Lisi, P. Lops, D. Malerba, A. Micheli, M. Pelillo , F. Ricci, F. Riguzzi, L. Saitta and G. Semeraro, Italian Machine Learning and Data Mining research: The last years. *Intelligenza Artificiale* **7**(2) (2013), 77–89.

[83] T. Frühwirth, A. Di Pierro and H. Wiklicky, An implementation of probabilistic constraint handling rules. In M. Comini and M. Falaschi, editors, *WFLP*, 2002.

[84] M. Gabbrielli, J. Mauro, M. Meo and J. Sneyers, Decidability properties for fragments of CHR. *TPLP* **10**(4-6) (2010).

[85] M. Gavanelli, An algorithm for multi-criteria optimization in csps. In *ECAI 2002* (2002), 136–140.

[86] M. Gavanelli, The log-support encoding of CSP into SAT. In C. Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pp. 815–822. Springer, 2007.

[87] M. Gavanelli, M. Alberti and E. Lamma, Integration of abductive reasoning and constraint optimization in SCIFF. In *ICLP 2009*, Springer-Verlag, 2009.

[88] M. Gavanelli, E. Lamma, P. Mello and M. Milano, Dealing with incomplete knowledge on CLP(FD) variable domains. *ACM Transactions on Programming Languages and Systems*, **27**(2) (2005), 236–263.

[89] M. Gavanelli, M. Nonato and A. Peano, An ASP approach for the valves positioning optimization in a water distribution system. *Journal of Logic and Computation* (2014).

[90] M. Gavanelli, F. Riguzzi, M. Milano and P. Cagnoli, Logicbased decision support for strategic environmental assessment. *Theory and Practice of Logic Programming* **10** (4-6) (2010), 643–658.

[91] M. Gavanelli and F. Rossi, Constraint logic programming. In A. Dovier and E. Pontelli, editors, *25 Years GULP*, volume 6125 of Lecture Notes in Computer Science, Springer, 2010, pp. 64–86.

[92] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming. In *ICLP/SLP* (1988), 1070–1080.

[93] A. Gerevini, A. Saetti and I. Serina, Planning through stochastic local search and temporal action graphs in lpg. *J Artif Intell Res (JAIR)* **20** (2003), 239–290.

[94] A. Gerevini and I. Serina, Planning as propositional CSP: From Walksat to local search techniques for action graphs. *Constraints* **8**(4) (2003), 389–413.

[95] L. Giordano, A.Martelli and G. Rossi, Extending horn clause logic with implication goals. *Theor Comput Sci* **95**(1) (1992), 43–74.

[96] E. Giunchiglia, F. Giunchiglia, R. Sebastiani and A. Tacchella, SAT vs. translation based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non-Classical Logics* **10**(2) (2000), 145–172.

[97] E. Giunchiglia, Y. Lierler and M. Maratea, Answer set programming based on propositional satisfiability. *J Autom Reasoning* **36**(4) (2006), 345–377.

[98] E. Giunchiglia and M. Maratea, Introducing preferences in planning as satisfiability. *J Log Comput* **21**(2) (2011), 205–229.

[99] E. Giunchiglia, M. Narizzano and A. Tacchella, Backjumping for quantified boolean logic satisfiability. *Artif Intell* **145**(1-2) (2003), 99–120.

[100] E. Giunchiglia, A. Tacchella and F. Giunchiglia, SAT-based decision procedures for classical modal logics. *Journal of Automated Reasoning* **28**(2) (2002) 143–171.

[101] G. Gottlob, G. Greco and T. Mancini, Conditional constraint satisfaction: Logical foundations and complexity. In M.M. Veloso, editor, *IJCAI* (2007), 88–93.

[102] G. Gottlob, N. Leone and F. Scarcello, A comparison of structural CSP decomposition methods. *Artif Intell* **124**(2) (2000), 243–282.

[103] S. Gualandi and F. Malucelli, Constraint programming-based column generation. *4OR: A Quarterly Journal of Operations Research* **7**(2) (2009), 113–137.

[104] J. Hooker, *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. John Wiley & Sons, 2000.

[105] J. Jaffar and M.J. Maher, Constraint logic programming: A survey. *J. Log. Program.* **19/20** (1994), 503–581.

[106] A. Kakas, R. Kowalski and F. Toni, Abductive Logic Programming. *Journal of Logic and Computation* **2**(6) (1993).

[107] A. Kakas, A. Michael and C. Mourlas, ACLP: Abductive constraint logic programming. *J. Log. Program* **44** (2000).

[108] A.C. Kakas and P. Mancarella, On the relation between Truth Maintenance and Abduction. In *PRICAI* (1990).

[109] H.A. Kautz and B. Selman, Planning as satisfiability. In *ECAI* (1992), pp. 359–363.

[110] Z. Kızıltan, A. Lodi, M. Milano and F. Parisini, Bounding, filtering and diversification in CP-based local branching. *J. Heuristics* **18**(3) (2012), 353–374.

[111] E. Lamma, P. Mello and A. Natali, An extended Warren abstract machine for the execution of structured logic programs. *J Log Program* **14**(3-4) (1992), 187–222.

[112] N. Leone, G. Pfeifer, W. Faber, F. Calimeri, T. Dell'Armi, T. Eiter, G. Gottlob, G. Ianni, G. Ielpa, C. Koch, S. Perri and A. Polleres, The dlv system. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *JELIA*, volume 2424 of Lecture Notes in Computer Science, Springer, 2002, pp. 537–540.

[113] M.D.T. Lewis, T. Schubert, B. Becker, P. Marin, M. Narizzano and E. Giunchiglia, Parallel QBF solving with advanced knowledge sharing. *Fundam Inform* **107**(2-3) (2011), 139–166.

[114] A. Mackworth, Consistency in networks of relations. *Artif Intell* **8**(1) (1977).

[115] P. Mancarella and D. Pedreschi, An algebra of logic programs. In *Proc 5th ICLP*, The MIT Press, 1988.

[116] P. Mancarella, G. Terreni, F. Sadri, F. Toni and U. Endriss, The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *TPLP* **9**(6) (2009), 691–750.

[117] P. Mancarella, G. Terreni and F. Toni, Web sites verification: An abductive logic programming tool. In V. Dahl and I. Niemelä, editors, *ICLP*, volume 4670 of LNCS, 2007.

[118] T. Mancini and M. Cadoli, Detecting and breaking symmetries by reasoning on problem specifications. In J.-D. Zucker and L. Saitta, editors, *SARA 2005*, (2005), pp. 165–181.

[119] T. Mancini and M. Cadoli, Exploiting functional dependencies in declarative problem specifications. *Artificial Intelligence* **171**(16–17) (2007), 985–1010.

[120] T. Mancini, M. Cadoli, D. Micaletto and F. Patrizi, Evaluating ASP and commercial solvers on the CSPLib. *Constraints* **13**(4) (2008).

[121] T. Mancini, P. Flener and J. Pearson, Combinatorial problem solving over relational databases: View synthesis through constraint-based local search. In S. Ossowski and P. Lecca, editors, *SAC*, ACM, 2012. pp. 80–87.

[122] H. Mangassarian, A. G. Veneris and M. Benedetti, Robust QBF encodings for sequential circuits with applications to verification, debug, and test. *IEEE Trans Computers* **59**(7) (2010), 981–994.

[123] M. Maratea, F. Ricca, W. Faber and N. Leone, Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *J Algorithms* **63**(1-3) (2008), 70–89.

[124] P. Marin, C. Miller, M.D.T. Lewis and B. Becker, Verification of partial designs using incremental QBF solving. In W. Rosenstiel and L. Thiele, editors, *DATE*, IEEE, 2012, pp. 623–628.

[125] M. Narizzano, C. Peschiera, L. Pulina and A. Tacchella, Evaluating and certifying QBFs: A comparison of state-of-the-art tools. *AI Commun* **22**(4) (2009), 191–210.

[126] M. Narizzano, L. Pulina and A. Tacchella, Report of the third QBF solvers evaluation. *JSAT* **2**(1-4) (2006), 145–164.

[127] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach and A. Nerode, editors, *LPNMR*, volume 1265 of LNCS, Springer, 1997, pp. 421–430.

[128] W. Nuijten and C. Le Pape, Constraint-based job shop scheduling with ILOG-SCHEDULER. *J Heuristics* **3**(4) (1998).

[129] R. O'Keefe, Towards an algebra for constructing logic programs. In *IEEE Symposium on Logic Programming*, 1985.

[130] S. Oliveira, M. Hussin, T. Stützle, A. Roli and M. Dorigo, A detailed analysis of the population-based ant colony optimization algorithm for the TSP and the QAP. In N. Krasnogor and P.L. Lanzi, editors, *GECCO*, ACM, 2011, pp. 13–14.

[131] S. Perri, F. Ricca and M. Sirianni, Parallel instantiation of ASP programs: Techniques and experiments. *TPLP* **13**(2) (2013).

[132] S. Perri, F. Scarcello and N. Leone, Abductive logic programs with penalization: Semantics, complexity and implementation. *TPLP* **5**(1-2) (2005), 123–159.

[133] L. Pulina and A. Tacchella, A self-adaptive multiengine solver for quantified boolean formulas. *Constraints* **14**(1) (2009), 80–116.

[134] L. Pulina and A. Tacchella, A structural approach to reasoning with quantified boolean formulas. In C. Boutilier, editor, *IJCAI* (2009), pp. 596–602.

[135] L. Pulina and A. Tacchella, Challenging SMT solvers to verify neural networks. *AI Commun* **25**(2) (2012), 117–135.

[136] A. Raffaetá and T. Frühwirth, Spatio-temporal annotated constraint logic programming. In *PADL* (2001).

[137] J.-C. Régin, A filtering algorithm for constraints of difference in csps. In B. Hayes-Roth and R. E. Korf, editors, *AAAI*, 1994.

[138] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano and N. Leone, Team-building with answer set programming in the Gioia-Tauro seaport. *TPLP* **12**(3) (2012), 361–381.

[139] A. Schaerf, Scheduling sport tournaments using constraint logic programming. *Constraints* **4**(1) (1999), 43–65.

[140] R. Sebastiani and M. Vescovi, Automated reasoning in modal and description logics via SAT encoding: the case study of K(m)/ALC-satisfiability. *J Artificial Intelligence Research (JAIR)* **35** (2009), 343–389.

[141] G. Semeraro, P. Basile, R. Basili, M. de Gemmis, C. Ghidini, M. Lenzerini, P. Lops, A. Moschitti, C. Musto, F. Narducci, A. Pipitone, R. Pirrone, P. Poccianti and L. Serafini. Semantic technologies for industry: From knowledge modeling and integration to intelligent applications. *Intelligenza Artificiale* **7**(2) (2013), 125–137.

[142] P. Traverso, A. Cimatti, L. Spalazzi, A. Armando and E. Giunchiglia, MRG: Building planners for real-world complex applications. *Applied Artificial Intelligence* **8**(3) (1994).

[143] M. Veloso, editor. *Proc. IJCAI 2007*, 2007.