KI - Künstliche Intelligenz (2018) 32:165–176
https://doi.org/10.1007/s13218-018-0548-6

TECHNICAL CONTRIBUTION

Industrial Applications of Answer Set Programming

Andreas Falkner[2] · Gerhard Friedrich[1] · Konstantin Schekotihin[1] · Richard Taupe[2] · Erich C. Teppan[1]

Received: 5 September 2017 / Accepted: 31 May 2018 / Published online: 13 June 2018

Abstract
Automated problem solving in combination with declarative specifications of search-problems have shown to substantially improve the implementation and maintenance costs as well as the man-machine interaction of deployed industrial applications. The knowledge representation and reasoning (KRR) framework of answer set programming (ASP) offers a rich representation language and high performance solvers. Therefore, ASP has become very attractive for the representation and solving of search-problems both for academia and industry. This article focuses on the latest industrial applications of ASP. We do not only present successful applications of ASP but also describe the development process and the design of ASP programs in an industrial context. Finally, we discuss current approaches to tackle the most significant application challenges such as grounding and runtime improvements by heuristics.

Keywords Answer set programming · Applications

1 Introduction

One of the most successful areas of artificial intelligence (AI) is the automatic generation of solutions for declaratively specified search-problems. Informally, given a search-problem, a programmer specifies the search space and the required properties such that for every problem instance (i.e. the inputs of a problem solver) the required properties are fulfilled by the solutions (i.e. the outputs of a problem solver).

As an example, consider the problem of configuring a safety installation. A programmer describes all technically feasible assemblies and the customer requirements an assembled system must satisfy. Ideally, it is not necessary to specify an algorithm to compute a solution. The automated problem solver generates the solutions just by exploiting the specification of the solution space and the required properties of the solutions. In addition, solutions may be ranked according to an optimization criterion.

The automatic generation of solutions to declaratively specified search-problems is extremely interesting for industry because it allows a substantial reduction of implementation and maintenance costs as well as the enhancement of user interactions, e.g. by the generation of explanations of inconsistent requirements. In [30], a reduction of the maintenance costs by more than 80% was reported.

In order to support a compact representation of search-problems, Answer Set Programming (ASP) offers many important language constructs such as recursive definition of predicates, default negation for dealing with the absence of information, disjunction, aggregation, weak constraints, and optimisation statements. All of them are employed in solving problems from classical areas of AI such as configuration, design, planning, scheduling, and diagnosis. Consequently, ASP has a very broad application range in research and practice.

In order to support the dissemination of ASP in practice it is important to present successful showcases as well as current engineering challenges. Regarding successful showcases, excellent reports have already been published, i.e. [11, 26, 45, 46, 51]. These reports are complemented by an

✉ Konstantin Schekotihin
konstantin.schekotihin@aau.at

Andreas Falkner
andreas.a.falkner@siemens.com

Gerhard Friedrich
Gerhard.Friedrich@aau.at

Richard Taupe
richard.taupe@siemens.com

Erich C. Teppan
Erich.Teppan@aau.at

[1] Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

[2] Siemens AG Österreich, Vienna, Austria

extensive listing of references of ASP applications and their areas.[1] The goal of this paper is to provide the latest status of industrial ASP applications and to present and discuss current engineering challenges and their solutions based on our long-standing experience in implementing solutions for industrial problems. First, we will give an overview of well-known industrial showcases of ASP and round this up by recent developments in Sect. 2. For ASP applications we describe the development process in Sect. 3 and the design in Sect. 4. Current challenges arising in applications are the grounding of large problem instances and, as many problems addressed by ASP are NP-hard, their efficient solving. We discuss state-of-the-art methods to meet these challenges in Sect. 5, which deals with grounding, and in Sect. 6, which is dedicated to heuristics.

2 General Overview of ASP Applications

Reports on ASP applications can be roughly divided into those where real-world data is employed by researchers to test the feasibility of applying ASP-based problem solving without commercial intention and those where companies or organisations are applying ASP to solve business cases. Subsequently we mainly concentrate on the latter, i.e. applications of ASP where companies are investing resources.

In the following, we will structure the description of applications along the classical application areas of AI, i.e. configuration, diagnosis and repair, planning, and classification. Afterwards, we will report on recent industrial applications of Datalog, which in its original form is a subset of ASP, and on further emerging industrial applications of ASP.

Configuration One of the most successful application areas of ASP is the configuration of systems. To our knowledge, [71] represents the first time that the benefits of ASP for solving configuration problems were discussed. Based on these results, configuration solutions were offered by Variantum Oy. Later, ASP was successfully applied to the configuration of Linux packages [39]. By means of this work the power of ASP was demonstrated in the Mancoosi solver competition of 2012 where the ASP-based configurator won all tracks.[2]

After promising tests of various ASP solvers for model-based diagnosis and repair of failed workflow instances [34] ASP was evaluated by Siemens within the RECONCILE[3] project. Thus, ASP was successfully applied to configure parts of a railway safety system of Siemens where specialised configurators failed to generate solutions for some hard real-world instances. This configuration problem became well known as the Partner Units Problem [74] since it is both a highly relevant real-world configuration problem and particularly hard to solve for general problem solvers [3]. A recent application of ASP in the area of configuration and design is the synthesis of optimal-sized concentrators [15].

If we regard a configuration task as the selection of entities such that constraints are satisfied and preferences are maximised then the application regarding automatic allotment of tourist packages (described in [51]) belongs to this class of tasks. In this application employed by the tour operator Top Class s.r.l., a set of tour packages and their pre-booked quantities are selected such that costs are minimised.

A further application where the output is a selection of entities is the assignment of employees to tasks such that these tasks can be accomplished and labour regulations are obeyed. In particular, ASP is applied to support team-building at the seaport of Gioia Tauro [66].

Repacking can be regarded as a classical reconfiguration task. ASP was successfully applied within a portfolio solver which solves the station repacking problem of reallocating the frequency bands of television broadcasters. By this technique a combination of a SAT solver and CLASP was able to significantly improve runtime and the number of solved instances [33].

Diagnosis and Repair In the area of diagnosis, an ASP-based system was employed to analyse failures of the automatic whitelisting systems of Google's Dynamic Remarketing Ads [12]. Diagnostic reasoning in form of a decision tree was implemented by ASP to diagnose whitelisting failures of advertisers, e.g. the reasons why an advertiser does not qualify for whitelisting. The ASP-based diagnosis system was implemented by a hybrid ASP system, which supports the integration of external procedures such as calls to a database.

Manually feeding tumor registries with data leads to different schemas for representing data as well as incomplete and erroneous information. With the help of an ASP-based system developed in cooperation with Pentaho Kettle, erroneous tuples are identified and corrected. In [51], it is reported that only 2% of the input tuples were detected as wrong and not repairable after cleaning. The input table was composed of 1,000,000 tuples collecting records from 155 municipals. The dictionary stored 15,000 tuples.

Planning In the area of planning an ASP-based system was introduced, which validates and generates plans for setting up the space shuttle for manoeuvres [62]. Usually such plans are prepared for standard situations. However, it was not possible to preplan all eventualities in case of failures. Consequently, an ASP-based system was implemented, which generates such plans to aid the human controllers. This work is based on the language P-log [11], which extends ASP by probabilistic arguments. Indeed, ASP serves as an excellent framework for realizing knowledge-based

[1] http://dropbox.com/s/pe261e4qi6bcyyh/aspAppTable.

[2] http://www.mancoosi.org/misc-2012/results/.

[3] http://isbi.aau.at/reconcile/.

planning including conformant (e.g. [24]) and conditional planners (e.g. [77, 80]). Because of its rich representation capabilities, ASP is applied in many applications of planning in robotics (e.g. [63]) and the combination of monitoring, diagnosis, and replanning (e.g. [28]). Furthermore, concepts of ASP are applied to implement a planner, which generates an executable workflow for producing desired outputs. In [61], a web service composition framework is described, which is employed for extracting and reusing phylogenetic trees. This framework comprises a planning module, a workflow configuration module, and an execution & monitoring module.

Classification Many recommender systems provide a classification of items, which best fit the customers' needs. Following this approach a tourist advisor was implemented in ASP and integrated in an e-tourism portal [49, 51]. In particular, given the preferences of customers, the properties of holiday packages, and background knowledge (e.g. beach holidays are not recommended in winter) the system selects those holiday packages which best fit the customers' needs.

A classification system for incoming calls to a contact centre of Telecom Italia is implemented by Exeura.[4] The purpose of the classification is to assign customers to a category and to exploit this category to route calling customers to the most appropriate service. The classification uses various background data related to customers in order to anticipate their needs. Since categories may be adapted frequently, the system allows the maintenance of the classification knowledge by call center operators. Operators are enabled to define categories by a decision graph, which is eventually translated to ASP rules.

Current question answering (QA) systems usually retrieve predefined answers which closely match the questions. Obvious problems are the completeness and the lack of explaining the reasons for an answer. [78] shows how ASP can be applied to improve QA for customers who intend to buy products for do-it-yourself projects. The basic question the system answers is 'Can I use an alternative tool/material instead of the suggested one under a certain circumstance?' Consequently, the main problem solving task is to classify new tools/materials as possible alternatives to tools/materials designated for a specific project a customer pursues. The basic solution idea is to specify the project-independent properties of tools/materials in a domain knowledge base and to describe project-specific properties of tools or materials in a project knowledge base. Recommendations for replacements are generated by exploiting the knowledge bases, the project rules, and the project constraints to identify the best matches. In addition, the system also addresses the knowledge acquisition problem by extracting parts of the domain knowledge base from documents using ASP.

Datalog Industrial success stories of Datalog are worth mentioning as well because of Datalog's close relation to ASP. In DIADEM [35], Datalog is employed to identify and extract information from web sites with high accuracy. This approach is commercially exploited by the spin-out Wrapidity, which was acquired by Meltwater. Furthermore, Datalog is successfully applied by LogicBlox [47] to implement enterprise software such as consumer demand prediction and supply chain optimisation.

Emerging Industrial Applications Beside the applications described above, where a business case exists and companies and organisations are investing in order to use ASP for their businesses, numerous applications were implemented showing the utility of ASP in real-world instances. In the following, we give examples for various application areas.

Bioinformatics In [58], ASP is applied to support hypotheses formulation in the context of signalling networks by an action language and abduction. In [36], ASP is employed to detect, explain, and repair errors in large-scale biological networks and datasets. Applications of ASP to phylogenetic systematics are described in [13]. In [22], the abstraction of the protein structure prediction problem is studied. [17] explores the use of ASP in genomics studies, such as haplotype inference and phylogenetic inference, in structural studies, such as RNA secondary structure prediction and protein structure prediction, as well as in systems biology. Similarly, haplotype inference by ASP is investigated in [29]. Moreover, [65] employs ASP for searching time-dependent relationships in the genomic and epigenomic analysis of cancer. [27] shows how ASP is used for generating explanations for complex biomedical queries related to drug discovery.

Scheduling The optimisation capabilities and the rich set of KR constructs allow a simple representation of scheduling problems, e.g. nurse scheduling [21].

Timetabling In [8] the successful application of ASP to the curriculum-based timetabling problem was demonstrated. The implementation based on CLINGO is able to compete with state-of-the-art special purpose solvers. This is underlined by discovering new bounds for many problem instances used in timetabling competitions. Moreover, the ASP approach allows for extensible and flexible problem formulations because of a first-order encoding.

Robotics ASP offers elegant solutions for representing actions and the associated representation problems such as the frame, ramification, and qualification problem. Consequently, there are numerous applications in the robotics domain, e.g. coordinating multiple robots tidying up a house as in [25].

Dynamic reconfiguration As described in [10], ASP is applied to configure caching strategies of routers in

[4] http://exeura.eu/en/solution/customer-profiling.

content-centric networking such that the overall network performance is optimised.

Information integration In [1], a classical application of ASP in the area of detecting inconsistent information is reported, which allows for the identification of inconsistent news information, e.g. whether or not different sources are talking about the same event. More generally, query answering in case of inconsistent data is an important application area of ASP [54].

Software engineering In software engineering ASP is successfully applied for the generation of test suites [7], in particular for constrained combinatorial testing. It turned out that the high-level implementation using ASP matched the performance of or even outperformed specialised tools.

*Design of embedded systems*The design of embedded systems is supported by systems synthesis where a task-level behavioural description is transformed into a structural representation. In order to accomplish this task ASP is combined with special problem solvers for background theories developed in the area of embedded systems. This allows the description, synthesis, and optimization of more sophisticated and complex systems [59, 60].

These applications show the broad and successful employment of ASP. ASP is of particular value for declaratively specified search-problems where an encoding of the problem specification by relations is most appropriate for programmers and domain experts. In the next sections we will describe important engineering topics for a successful application of ASP in practice.

3 Development Process of ASP Applications

Answer-set programs are often encoded according to a *guess-and-check* methodology, where part of the program encodes nondeterministic generation of solution candidates and another part rules out invalid ones by use of constraints.

Complementarily to that, preliminary work on so-called 'achievement-based' answer set programming is presented in [53]. Under this approach, a programmer constructs an answer-set program by appending rules in an order such that after every rule, the programmer can declare that a certain mathematical property 'has been achieved', which then holds until the program is completed. By accumulating rules and corresponding achievements, an answer-set program and a skeleton of a proof of its correctness grow hand in hand. This approach, however, may not be useful for programs with defaults. Also, to the best of our knowledge, it has so far only been applied to small example problems and tool support for automatic proofs does not yet exist.

Compared to academia, industrial applications show two main aspects: the domain of interest is typically a complex system of numerous interacting and interdependent components with a rich history, and there is a changing variety of regulations and methodologies which must be obeyed. For example, development must comply with project management (PM) standards such as ISO 21500:2012 or systems development approaches such as SDLC (Systems Development Life Cycle) [67] and must cover all the phases involved: initiation, system concept development, planning, requirements analysis, design, development, integration and testing, implementation, operations and maintenance, as well as disposal. For the last few decades, object-oriented (OO) analysis, design, and programming have become the main paradigm for software (SW) development [68]. In addition, over the last years, agile processes [32] have gained wide acceptance in industry due to their advantages: concentration on customer value, continuous deployment of working software, high responsiveness to changing requirements.

Taking all this into account, we propose the following steps to introduce ASP into industry:

Identify the needs Watch out for problems which are not (satisfactorily) solved with conventional methods and where the strengths of ASP can help. Define and document the application requirements properly, especially which parts of the whole application are relevant.

Design a valid specification of the problem Implement an ASP specification of the core problem. Validate this specification by 'small' test instances. The correct formulation of a problem is a time-consuming and error-prone task. ASP allows interactive problem refinement and tuning with stakeholders, thus reducing design costs [51].

Performance engineering Evaluate scalability to 'real-world' size. Try alternative ASP program implementations and evaluate their performance. See Sects. 5 and 6 for details on important performance aspects such as main memory consumption and runtime performance.

Integrate into the existing environment Design interfaces and implement a complete and efficient transformation from legacy input data to ASP and back for returning the solutions. Choose the best ASP program variant from the feasibility study and implement a clean ASP program which processes the transformed data. See Sect. 4 for an exemplary best-practice approach to this design phase. Define the necessary reasoning services, e.g. model checking, model finding, optimisation. If the usage scheme is highly interactive (in contrast to batch mode), consider possibilities for incremental solving. For many use cases, post-processing of the solutions (answer sets) will be necessary, e.g. extracting a Pareto front from all solutions, generating explanations, or giving recommendations to the user.

Testing and debugging Provide automated regression tests in order to ensure high quality. Debugging of ASP programs is difficult, especially if the program unexpectedly returns unsatisfiability for some input data. Knowledge-based diagnosis methods such as [70] are of great support.

Maintenance Structure the program well and use understandable wording, because industrial applications are long-lived and will be regularly changed and extended. The modularity of ASP fosters later adaptions.

In this development process, we regard knowledge base design and performance engineering as the steps most important and most different from conventional software engineering. Consequently, we elaborate on them in more detail in the next sections.

4 Design of ASP Applications

As an object-oriented (OO) approach is used to design software systems in many industrial domains, we suggest the OOASP approach which allows to analyse OO software models and their instances by means of ASP [31]. In particular, we consider models which can be described by a modelling language corresponding to a UML class diagram [68]. Such a language allows a software developer to specify an object model and additional constraints a valid instantiation must satisfy. OOASP has been implemented as a potential extension to any OO modelling environment and its practicability has been evaluated together with CSL (Configuration Specification Language) [19], which is a Siemens-internal tool for the design of product configurators based on the methodology of generative constraint satisfaction problems (GCSPs) [72].

An OOASP program comprises facts encoding the OO classes, attributes, associations and constraints. Integrity constraints encode model requirements to relations between objects of an instantiation and are derived from the given model automatically. Domain-specific constraints ensure satisfaction of requirements to a model instantiation. They could be derived from definitions given in a specification language, such as the Object Constraint Language (OCL).[5]

The OOASP framework supports the following reasoning tasks:

Validation (model checking) Verifies whether all integrity and domain-specific constraints hold in a given (complete or partial) instantiation.

Completion (model finding) Finds an extension of a given partial instantiation which satisfies all constraints or shows that such an extension does not exist (an empty instantiation can be seen as a special case).

Reconciliation (model repair) Finds a (preferred) set of changes required to transform a legacy instantiation of an outdated OO model to a valid instantiation of the new up-to-date model.

[5] http://www.omg.org/spec/OCL/2.4/.

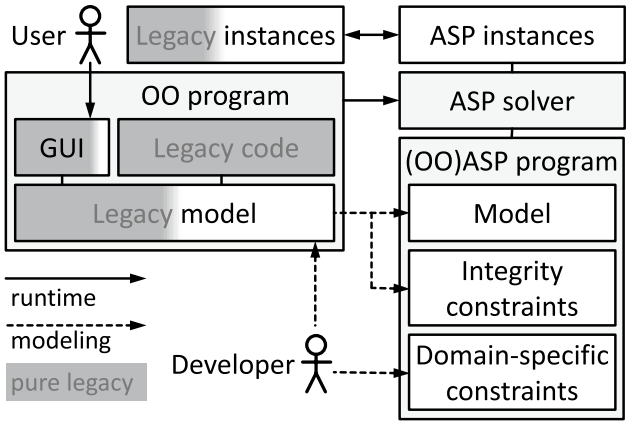

Fig. 1 Integration of (OO)ASP in an industrial application

A typical workflow of the integration of (OO)ASP into an OO industrial application is depicted in Fig. 1. The development starts with the creation or adaption of the OO model. For existing (legacy) applications, the relevant part of the model is selected and automatically transformed to an OOASP program comprising the data structure (model) and integrity constraints. The definition of domain-specific constraints is done in (OO)ASP and requires ASP knowledge (either by engaging a specialized developer or by training the available OO developer).

For example, the following domain-specific OOASP rule leads to the derivation of an `ooasp_cv` atom if the constraint that two modules of type `ModuleA` must not be placed next to each other in a hardware configuration problem is violated (for details refer to [31]):

```
ooasp_cv(IID,moduleANextToA(M1,M2,P1,P2)) :-
  ooasp_associated(IID,"Frame_modules",F,M1),
  ooasp_associated(IID,"Frame_modules",F,M2),
  ooasp_isa(IID,"ModuleA",M1),
  ooasp_isa(IID,"ModuleA",M2),
  ooasp_attribute_value(IID,"position",M1,P1),
  ooasp_attribute_value(IID,"position",M2,P2),
  M1!=M2, P2=P1+1.
```

This domain-specific constraint is defined using domain-independent predicates of the OOASP framework. For instance, `ooasp_associated` defines associations between objects, whereas `ooasp_isa` describes class membership.

During operations (or similarly during testing), the user inputs data via the (graphical) user interface (GUI) to achieve some desired results. In the corresponding OO program this leads to the creation or changing of instances, e.g. in a configurator some decision parameters are set. Implicitly, the application transforms those inputs to (OO)ASP, solves the corresponding ASP program, and transforms the solutions (answer sets) back to OO instances, e.g. derived

settings or recommendations for further decisions. If there are multiple alternatives (answer sets) then the user needs to select one in order to actually change the existing OO instances accordingly and start with the next interaction cycle.

After developing a valid specification of the problem in a design phase, it must be ensured in a feasibility evaluation and implementation phase that the answer-set program can handle real-world problem instances and finish such an interaction cycle in reasonable time. The next two sections describe important challenges and current solutions for the performance engineering of ASP applications.

5 Tackling the Grounding Challenge

Most ASP systems follow the so-called *ground-and-solve* paradigm and split the solving process into two steps: First, a *grounder* transforms the input program containing variables into a propositional encoding. Then, solutions for the resulting variable-free program are generated by a *solver*. However, grounding may increase the size of the input program substantially. Consequently, ASP programs must be implemented by considering this grounding step especially for large problem instances. A valid ASP program could serve as a basis for rewriting in order to reduce the size of the grounded program. However, it can be the case that no suitable ASP program is available for required sizes of problem instances such that the ground-and-solve paradigm is applicable. This leads to the so-called *grounding bottleneck*.

One example to demonstrate this is incremental scheduling with problem instances from the ASP competition [14].[6] For the instances incorporating 60 job operations the size of the grounding is more than 5 GB whereas 120 job operations require more than 50 GB. In industrial scheduling domains, the sizes of problem instances can be significantly higher. Scheduling instances of semiconductor manufacturers like Infineon incorporate >10,000 job operations for a weekly workload performed on >100 machines in the back-end (i.e. where chips are cut and packaged) and >100,000 jobs for a weekly workload performed on >1000 machines in the front-end (i.e. where the chips are actually produced). Thus, such instances are clearly out of reach for current ground-and-solve approaches. In particular, a superlinear increase of memory consumption in the size of the problem instances is a challenge for some application cases. Hence, various ASP-based approaches to tackle the grounding issue have emerged, which are described in the remainder of this section.

Constraint ASP Constraint answer set programming (CASP) [56] is a hybrid approach extending ASP by constraint programming (CP) features. Conceptually, it is very close to satisfiability modulo theory (SMT) approaches, which integrate first-order formulas with additional background theories such as real numbers or integers [69].

There are basically two approaches to combining ASP and CP. First, solvers like Clingcon [64] are based on the extension of the ASP input language in order to support the formulation of constraints. A different approach is followed by the solvers Ezcsp [4] and ASCASS [75] where ASP and CP are not integrated into one language. ASP rather acts as a specification language for constraint satisfaction problems (CSPs). The main idea is that answer sets constitute CSP encodings, which are used as input for a constraint solver. Thus, CSPs are expressed by means of special predicates for CSP variable definitions and different types of constraints.

A key success factor for CASP systems are global constraints provided by constraint solvers. The usage of appropriate global constraints can significantly reduce the size of a CSP representation. For certain classes of problems like industrial-size scheduling CASP has already been applied successfully [5]. Especially search problems with large variable domains often profit from the CASP representation due to the alleviation of the grounding bottleneck [52].

Multi-shot solving Multi-shot answer set solving [38] enables continuously changing logic programs, which can expand the domain of discourse step-by-step until a solution is found. Such a domain usually exists in configuration or planning problems, i.e. the maximum number of objects or steps. In contrast to the ground-and-solve approach, which requires definition of the maximum domain size beforehand, multi-shot solving allows for finding the domain size while searching for a solution. This results in ground programs that are as small as required to find a solution. Of course, if the minimal domain size is too large, multi-shot solving cannot help to overcome the grounding bottleneck.

Lazy grounding The main problem of CASP and multi-shot approaches is that they require custom encodings which mix standard ASP with other declarative and/or procedural languages. In turn, lazy grounding and solving approaches can be applied to arbitrary programs in standard ASP and avoid the grounding bottleneck by interleaving grounding and search. Known approaches to lazy grounding for ASP include ASPeRiX [50], GASP [16], Omiga [18], and, most recently, Alpha [79].

While lazy-grounding systems are able to limit their memory usage, their time consumption is not yet comparable to that of state-of-the-art solvers. One reason for this is that most of these systems do not exploit conflict-driven nogood learning (CDNL), which is a key success factor of

[6] Find instances, encodings and grounders/solvers at www.mat.unical.it/aspcomp2014. Our tests were done with gringo 4 and the 2014 version of the encoding.

state-of-the-art ASP solvers. ALPHA is the first lazy-grounding system to employ CDNL [79]. It consists of a grounder and a solver which, however, do not work in sequence (as in ground-and-solve), but interact cyclically.

6 Tackling the Heuristics Challenge

Solving real-world instances of industrial problems is hard, since a large number of these problems are intractable and no efficient algorithms are known. Nevertheless, recent breakthroughs in solving and optimization algorithms used in modern ASP solvers have allowed for their successful application to various industrial problems. For example, evaluation results for a configuration problem presented in [3] show that ASP was able to solve many test instances including real-world problems outperforming constraint and mixed integer programming. However, this study also shows that solving large industrial instances is not possible with any of the reasoning techniques applied. As the analysis indicates, the negative results are mostly due to a large number of indistinguishable alternative choices, i.e. non-deterministic decisions. Hence, the ability of a solver to make good decisions is crucial for its success.

One of the possibilities to improve the performance is, of course, to write *better encodings* of problems. Using the standard programming techniques ASP users can write simple, succinct and general encodings of various real-world problems in a short period of time. However, such encodings are often not optimal from the solver's point of view as they allow for symmetric solutions, lead to many unnecessary non-deterministic choices, provide no hint on preferred orders in which those choices have to be made, etc. For recent ASP competitions, such as [14, 42], highly sophisticated encodings have been developed by the best ASP programmers. Such encodings result in a significant improvement of solvers' performance in many situations, but they (a) require a highly experienced and competent ASP programmer to write such an encoding, and (b) are often efficient for subclasses of the problem only. Moreover, in the case of industrial problems, including the Combined Configuration Problem (CCP) [43] or Incremental Scheduling, timeouts were observed for a majority of considered instances although the best encodings available were used.

The next possibility for improving the performance is to tune the *solver configuration*. Modern solving algorithms, such as used in WASP [2] or CLASP [41], are highly parametrisable and provide numerous possibilities to influence the underlying search procedure. For instance, a user can select a *domain-independent heuristic*, which is used to make decisions each time a non-deterministic choice must be effected. Most of the modern solvers use look-back heuristics, such as VSIDS [57] or BERKMIN [44]. These heuristics analyse the conflicts, i.e. non-solutions of a given problem instance found during the search, and make decisions allowing the solver to concentrate on a conflict until it is completely learned. This new information helps the solver to make better decisions and avoid non-solutions in the future. In addition, by using a specific restart strategy a solver can reinitialize the search and use previously found conflicts to make better decisions from the start. Since the number of found conflicts can be large, a conflict deletion strategy can influence the solver's memory consumption. Also, a configuration might define how a solver should handle different types of rules, backtracking modes in case of a conflict, optimization strategies, etc. Given such a large number of possible configurations it is hard for an ASP user to select the best one. *Portfolio* solvers like ME-ASP [55] or CLASPFOLIO [48] aim to solve this problem by employing machine learning techniques to train a classifier that returns the best solver configuration for a given problem instance. Applications of portfolio solvers have enabled further improvement of performance, as demonstrated in ASP competitions [14, 42]. For instance, ME-ASP was an overall winner of the sixth competition. However, this improvement also did not result in the solution of large industrial cases. For instance, only the three smallest CCP instances were solved by ME-ASP.

Therefore, recent research efforts have focused on developing methods allowing for integration of *domain-specific heuristics* into ASP solvers. There are two approaches to introduce such heuristics: machine learning and manual encoding. An example of the first approach is given in [6], where the general idea is to *learn heuristics* from experience the solver obtains while searching for solutions of small instances. This method assumes that both small and large instances of the given problem are not random but contain some repetitive structures and the difference is only in the number of such structures. As a consequence, heuristics learned on the small instances can also be efficiently used for larger ones.

The manual definition of domain-specific heuristics can be either performed in a declarative or in a procedural way. The *declarative approach* allows a programmer to specify heuristics directly in the ASP program using specific extensions of the language. A representative declarative approach presented in [40] introduces the `_heuristic` predicate. Atoms over this predicate are treated by the solver HCLASP in a special way resulting in modification of values stored in VSIDS heuristic. Thus, a user can define initial weights, importance factors, decision levels and sign selection for atoms involved in non-deterministic decisions. HCLASP was successfully applied in [43] to find solutions for the CCP instances. The authors created two simple algorithms computing good initial heuristic weights for a given instance, encoded them using `_heuristic` predicate, and added them to the instance. This approach allowed to find solutions for 15 instances out

of 32, which is 12 instances more than ME-ASP. The heuristic predicate has been integrated into CLASP and subsequently replaced by the `#heuristic` directive [37].

Another approach to declarative specification of heuristics is based on the application of hybrid reasoning techniques in ASP, e.g. heuristic constraint answer set programming (HCASP) [75]. This has been investigated in our research project HINT,[7] resulting in the ASCASS solver already mentioned in Sect. 5. Since CP languages provide versatile possibilities for declaration of domain-specific heuristics, they can also be used in HCASP approaches. For instance, in EZCSP [4] or in ASCASS [75] it is possible to define an order on CP variables in which a CP solver should assign values to them. In addition, in HCASP systems one can use global constraints, like `alldifferent` or `knapsack`, allowing for application of heuristic propagation algorithms that might result in significant performance improvements. Finally, some HCASP systems provide expressive languages for definition of complex domain-specific heuristics. One of the examples is the ASCASS implementation of the QUICK-PUP heuristic [76]. This implementation scales well and can be applied to solve large PUP instances [73].

ASP solvers can also be extended with *procedural* definitions of domain-specific heuristics which are integrated in a way similar to the native ASP heuristics such as VSIDS. For instance, the CLINGO solver allows for definitions of custom propagators that are invoked just before the invocation of a native heuristic [37]. The propagator is an algorithm implemented in a procedural language, e.g. C++, that has access to various solver-internal structures. Given the current state of the solver an algorithm can output conflicts, i.e. partial assignments that are not in any solution. If new conflicts are saved, the native heuristic is not invoked and the solver tries to use the new information to find a solution. In this way a user can influence the search by resolving non-deterministic choices by means of conflicts. Another approach, implemented in HWASP [20], allows a user to define procedures replacing the native heuristics with custom ones. Using a specific data exchange interface the heuristic communicates with the solver and is invoked instead of the standard one if a non-deterministic choice has to be made. Experiments with such heuristics indicate that they allow ASP solvers to significantly improve their performance and to find solutions for very large instances of industrial problems. For instance, HWASP solver equipped with simple procedural domain-specific heuristics presented in [23, 43, 76] was able to find solutions for all instances of CCP and PUP problems. In case of CCP, HWASP required less than 200 seconds to return all solutions, including the ones of the largest 17 instances, which were previously unsolved.

[7] http://isbi.aau.at/hint/.

7 Conclusions

Declarative KRR frameworks have shown their great utility in practice, especially for search-problems such as design, configuration, planning, classification, and diagnosis.

ASP is one of the most attractive KRR frameworks because of its rich representation language. Therefore, ASP is used in academia and in industry for the encoding and solving of problems.

In this paper, we gave an overview of the best-known industrial showcases of ASP and described latest applications. This report was rounded up with additional emerging industrial applications of ASP and structured along the classical application areas of AI. This impressively shows the broad applicability of ASP.

Based on our industrial background, we outlined the development process and the latest engineering methods for successfully implementing ASP applications such as object-oriented design of ASP knowledge-bases and approaches to tackle memory and runtime challenges.

Although powerful solvers already allow many successful applications, we must be aware that ASP was designed for, and consequently is applied to, NP-hard problems. Therefore, further research in pushing the applicability frontier of ASP is most appreciable for expanding the usage of knowledge representation and reasoning in industry.

Acknowledgements Open access funding provided by University of Klagenfurt. Work has been conducted in the scope of the research projects *DynaCon (FFG-PNr.: 861263)* and *Productive4.0 (H2020-ECSEL-GANo.: 737459)*. DynaCon is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between 2017 and 2020 (see https://iktderzukunft.at/en/ for more information). We thank Agostino Dovier for valuable suggestions.

References

1. Albanese M, Broecheler M, Grant J, Martinez MV, Subrahmanian VS (2011) PLINI: a probabilistic logic program framework for inconsistent news information. In: Balduccini M, Son TC (eds) Logic programming, knowledge representation, and nonmonotonic reasoning—essays dedicated to Michael Gelfond on the occasion of his 65th birthday, Lecture notes in computer science, vol 6565. Springer, Berlin, Heidelberg, pp 347–376
2. Alviano M, Dodaro C, Leone N, Ricca F (2015) Advances in WASP. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic programming and nonmonotonic reasoning—13th international conference, LPNMR 2015, Lecture notes in computer science, vol 9345. Springer, Cham, pp 40–54

3. Aschinger M, Drescher C, Friedrich G, Gottlob G, Jeavons P, Ryabokon A, Thorstensen E (2011) Optimization methods for the partner units problem. In: Achterberg T, Beck JC (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems—8th international conference, CPAIOR 2011, Lecture notes in computer science, vol 6697. Springer, Berlin, Heidelberg, pp 4–19
4. Balduccini M (2009) Representing constraint satisfaction problems in answer set programming. ICLP09 workshop on answer set programming and other computing paradigms (ASPOCP09)
5. Balduccini M (2011) Industrial-size scheduling with ASP+CP. In: Delgrande JP, Faber W (eds) Logic programming and nonmonotonic reasoning—11th international conference, LPNMR 2011, Lecture notes in computer science, vol 6645. Springer, Berlin, Heidelberg, pp 284–296
6. Balduccini M (2011) Learning and using domain-specific heuristics in ASP solvers. AI Commun 24(2):147–164
7. Banbara M, Inoue K, Kaneyuki H, Okimoto T, Schaub T, Soh T, Tamura N (2017) catnap: generating test suites of constrained combinatorial testing with answer set programming. In: Balduccini M, Janhunen T (eds) Logic programming and nonmonotonic reasoning—14th international conference, LPNMR 2017, Lecture notes in computer science, vol 10377. Springer, Cham, pp 265–278
8. Banbara M, Inoue K, Kaufmann B, Okimoto T, Schaub T, Soh T, Tamura N, Wanko P (2018) teaspoon: solving the curriculum-based course timetabling problems with answer set programming. Ann Oper Res
9. Baral C, Gelfond M, Rushton JN (2009) Probabilistic reasoning with answer sets. TPLP 9(1):57–144
10. Beck H, Bierbaumer B, Dao-Tran M, Eiter T, Hellwagner H, Schekotihin K (2016) Rule-based stream reasoning for intelligent administration of content-centric networks. In: Michael L, Kakas AC (eds) Logics in artificial intelligence—15th European conference, JELIA 2016, Lecture notes in computer science, vol 10021. Springer International Publishing, pp 522–528
11. Brewka G, Eiter T, Truszczynski M (2011) Answer set programming at a glance. Commun ACM 54(12):92–103
12. Brik A, Remmel JB (2015) Diagnosing automatic whitelisting for dynamic remarketing ads using hybrid ASP. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic programming and nonmonotonic reasoning—13th international conference, LPNMR 2015, Lecture notes in computer science, vol 9345. Springer, Cham, pp 173–185
13. Brooks DR, Erdem E, Erdogan ST, Minett JW, Ringe D (2007) Inferring phylogenetic trees using answer set programming. J Autom Reason 39(4):471–511
14. Calimeri F, Gebser M, Maratea M, Ricca F (2016) Design and results of the fifth answer set programming competition. Artif Intell 231:151–181
15. Dahlem M, Jain T, Schneider K, Gillmann M (2017) Automatic synthesis of optimal-size concentrators by answer set programming. In: Balduccini M, Janhunen T (eds) Logic programming and nonmonotonic reasoning—14th international conference, LPNMR 2017, Lecture notes in computer science, vol 10377. Springer, Cham, pp 279–285
16. Dal Palù A, Dovier A, Pontelli E, Rossi G (2009) GASP: answer set programming with lazy grounding. Fundamenta Inform 96(3):297–322
17. Dal Palù A, Dovier A, Formisano A, Pontelli E (2014) Exploring life through logic programming: answer set programming in bioinformatics. Tech Rep TR-CS-NMSU-2014-10-24, New Mexico State University. https://www.cs.nmsu.edu/wp/wp-content/uploads/2014/10/TR-CS-NMSU-2014-10-24.pdf
18. Dao-Tran M, Eiter T, Fink M, Weidinger G, Weinzierl A (2012) Omiga: an open minded grounding on-the-fly answer set solver. In: del Cerro LF, Herzig A, Mengin J (eds) Logics in artificial intelligence—13th European conference, JELIA 2012, Lecture notes in computer science, vol 7519. Springer, Berlin, Heidelberg, pp 480–483
19. Dhungana D, Falkner AA, Haselböck A (2013) Generation of conjoint domain models for system-of-systems. In: Järvi J, Kästner C (eds) Generative programming: concepts and experiences, GPCE'13, ACM, Indianapolis, IN, USA—October 27–28, 2013, pp 159–168
20. Dodaro C, Gasteiger P, Leone N, Musitsch B, Ricca F, Schekotihin K (2016) Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). TPLP 16(5–6):653–669
21. Dodaro C, Maratea M (2017) Nurse scheduling via answer set programming. In: Balduccini M, Janhunen T (eds) Logic programming and nonmonotonic reasoning—14th international conference, LPNMR 2017, Lecture notes in computer science, vol 10377. Springer, Cham, pp 301–307
22. Dovier A, Formisano A, Pontelli E (2009) An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. J Exp Theor Artif Intell 21(2):79–121
23. Drescher C (2012) The partner units problem a constraint programming case study. In: IEEE 24th international conference on tools with artificial intelligence, ICTAI 2012, IEEE Computer Society, pp 170–177
24. Eiter T, Faber W, Leone N, Pfeifer G, Polleres A (2003) A logic programming approach to knowledge-state planning, II: the DLVK system. Artif Intell 144(1–2):157–211
25. Erdem E, Aker E, Patoglu V (2012) Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. Intell Ser Robot 5(4):275–291
26. Erdem E, Gelfond M, Leone N (2016) Applications of answer set programming. AI Magaz 37(3):53–68
27. Erdem E, Öztok U (2015) Generating explanations for biomedical queries. TPLP 15(1):35–78
28. Erdem E, Patoglu V, Saribatur ZG (2013) Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In: IEEE international conference on robotics and automation, ICRA 2015, IEEE, pp 2007–2013
29. Erdem E, Türe F (2008) Efficient haplotype inference with answer set programming. In: Fox D, Gomes CP (eds) Proceedings of the 23rd AAAI conference on artificial intelligence, AAAI 2008. AAAI Press, pp 436–441
30. Falkner AA, Friedrich G, Haselböck A, Schenner G, Schreiner H (2016) Twenty-five years of successful application of constraint technologies at Siemens. AI Magaz 37(4):67–80
31. Falkner AA, Ryabokon A, Schenner G, Shchekotykhin KM (2015) OOASP: connecting object-oriented and logic programming. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic programming and nonmonotonic reasoning—13th international conference, LPNMR 2015, Lecture notes in computer science, vol 9345. Springer, Cham, pp 332–345
32. Fowler M, Highsmith J (2001) The agile manifesto. Softw Dev 9(8):28–35
33. Fréchette A, Newman N, Leyton-Brown K (2016) Solving the station repacking problem. In: Schuurmans D, Wellman MP (eds) Proceedings of the thirtieth AAAI conference on artificial intelligence. AAAI Press, pp 702–709
34. Friedrich G, Fugini M, Mussi E, Pernici B, Tagni G (2010) Exception handling for repair in service-based processes. IEEE Trans Softw Eng 36(2):198–215
35. Furche T, Gottlob G, Grasso G, Gunes O, Guo X, Kravchenko A, Orsi G, Schallhart C, Sellers AJ, Wang C (2012) DIADEM: domain-centric, intelligent, automated data extraction methodology. In: Mille A, Gandon FL, Misselis J, Rabinovich M, Staab

S (eds) Proceedings of the 21st World Wide Web conference, WWW 2012. ACM, pp 267–270
36. Gebser M, Guziolowski C, Ivanchev M, Schaub T, Siegel A, Thiele S, Veber P (2010) Repair and prediction (under inconsistency) in large biological networks with answer set programming. In: Lin F, Sattler U, Truszczynski M (eds) Principles of knowledge representation and reasoning: proceedings of the twelfth international conference, KR 2010. AAAI Press
37. Gebser M, Kaminski R, Kaufmann B, Ostrowski M, Schaub T, Wanko P (2016) Theory solving made easy with clingo 5. In: ICLP (Technical Communications), OASICS, vol 52. Schloss Dagstuhl, pp 2:1–2:15
38. Gebser M, Kaminski R, Kaufmann B, Schaub T (2017) Multi-shot ASP solving with clingo. CoRR abs/1705.09811
39. Gebser M, Kaminski R, Schaub T (2011) aspcud: a linux package configuration tool based on answer set programming. In: Drescher C, Lynce I, Treinen R (eds) Proceedings second workshop on logics for component configuration, LoCoCo 2011, EPTCS, vol 65, pp 12–25
40. Gebser M, Kaufmann B, Romero J, Otero R, Schaub T, Wanko P (2013) Domain-specific heuristics in answer set programming. In: des Jardins M, Littman ML (eds) Proceedings of the twenty-seventh AAAI conference on artificial intelligence. AAAI Press
41. Gebser M, Kaufmann B, Schaub T (2012) Conflict-driven answer set solving: from theory to practice. Artif Intell 187:52–89
42. Gebser M, Maratea M, Ricca F (2015) The design of the sixth answer set programming competition—report. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic programming and nonmonotonic reasoning—13th international conference, LPNMR 2015, Lecture notes in computer science, vol 9345. Springer, Switzerland, pp 531–544
43. Gebser M, Ryabokon A, Schenner G (2015) Combining heuristics for configuration problems using answer set programming. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic programming and nonmonotonic reasoning—Proceedings of 13th international conference, LPNMR 2015, Lexington, KY, USA, September 27–30, 2015, Lecture notes in computer science, vol 9345. Springer, Cham, pp 384–397
44. Goldberg E, Novikov Y (2007) Berkmin: a fast and robust satsolver. Discrete Appl Math 155(12):1549–1561
45. Grasso G, Leone N, Manna M, Ricca F (2011) ASP at work: spin-off and applications of the DLV system. In: Balduccini M, Son TC (eds) Logic programming, knowledge representation, and nonmonotonic reasoning—essays dedicated to Michael Gelfond on the occasion of his 65th birthday, Lecture notes in computer science, vol 6565. Springer, Berlin, Heidelberg, pp 432–451
46. Grasso G, Leone N, Ricca F (2013) Answer set programming: language, applications and development tools. In: Faber W, Lembo D (eds) Web reasoning and rule systems—7th international conference, RR 2013, Lecture notes in computer science, vol 7994. Springer, Berlin, Heidelberg, pp 19–34
47. Green TJ, Aref M, Karvounarakis G (2012) Logicblox platform and language a tutorial. In: Barceló P, Pichler R (eds) Datalog in academia and industry—second international workshop, Datalog 2.0, Lecture notes in computer science, vol 7494. Springer, Berlin, Heidelberg, pp 1–8
48. Hoos H, Lindauer MT, Schaub T (2014) claspfolio 2: advances in algorithm selection for answer set programming. TPLP 14(4–5):569–585
49. Ielpa SM, Iiritano S, Leone N, Ricca F (2009) An asp-based system for e-tourism. In: Erdem E, Lin F, Schaub T (eds) Logic programming and nonmonotonic reasoning, Proceedings of 10th international conference, LPNMR 2009, Potsdam, Germany, September 14–18, 2009. Lecture notes in computer science, vol 5753. Springer, Berlin, Heidelberg, pp 368–381
50. Lefèvre C, Béatrix C, Stéphan I, Garcia L (2017) Asperix, a first-order forward chaining approach for answer set computing. TPLP 17(3):266–310
51. Leone N, Ricca F (2015) Answer set programming: a tour from the basics to advanced development tools and industrial applications. In: Faber W, Paschke A (eds) Reasoning web. Web logic rules—11th international summer school 2015, tutorial lectures, Lecture notes in computer science, vol 9203. Springer, Cham, pp 308–326
52. Lierler Y, Smith S, Truszczynski M, Westlund A (2012) Weighted-sequence problem: ASP vs CASP and declarative vs problem-oriented solving. In: Russo CV, Zhou N (eds) Practical aspects of declarative languages—14th international symposium, PADL 2012, Lecture notes in computer science, vol 7149. Springer, Berlin, Heidelberg, pp 63–77
53. Lifschitz V (2017) Achievements in answer set programming. TPLP 17(5–6):961–973
54. Manna M, Ricca F, Terracina G (2013) Consistent query answering via ASP from different perspectives: theory and practice. TPLP 13(2):227–252
55. Maratea M, Pulina L, Ricca F (2012) The multi-engine ASP solver me-asp. In: del Cerro LF, Herzig A, Mengin J (eds) Logics in artificial intelligence—13th European conference, JELIA 2012, Lecture notes in computer science, vol 7519. Springer, Berlin, Heidelberg, pp 484–487
56. Mellarkod VS, Gelfond M, Zhang Y (2008) Integrating answer set programming and constraint logic programming. Ann Math Artif Intell 53(1–4):251–287
57. Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th design automation conference, DAC 2001. ACM, pp 530–535
58. Nam TH, Baral C (2009) Hypothesizing about signaling networks. J Appl Logic 7(3):253–274
59. Neubauer K, Wanko P, Schaub T, Haubelt C (2017) Enhancing symbolic system synthesis through aspmt with partial assignment evaluation. In: Atienza D, Natale GD (eds) Design, automation and test in Europe conference and exhibition, DATE 2017. IEEE, pp 306–309
60. Neubauer K, Wanko P, Schaub T, Haubelt C (2018) Exact multi-objective design space exploration using aspmt. In: 2018 design, automation and test in Europe conference and exhibition, DATE 2018. IEEE, pp 257–260
61. Nguyen TH, Son TC, Pontelli E (2018) Automatic web services composition for phylotastic. In: Calimeri F, Hamlen KW, Leone N (eds) Practical aspects of declarative languages—20th international symposium, PADL 2018, Lecture notes in computer science, vol 10702. Springer, Cham, pp 186–202
62. Nogueira M, Balduccini M, Gelfond M, Watson R, Barry M (2001) An a-prolog decision support system for the space shuttle. In: Ramakrishnan IV (ed) Practical aspects of declarative languages, Proceedings of third international symposium, PADL 2001, Las Vegas, Nevada, March 11–12, 2001, Lecture notes in computer science, vol 1990. Springer, Berlin, Heidelberg, pp 169–183
63. Nouman A, Yalciner IF, Erdem E, Patoglu V (2016) Experimental evaluation of hybrid conditional planning for service robotics. In: Kulic D, Nakamura Y, Khatib O, Venture G (eds) International symposium on experimental robotics, ISER 2016, Springer

proceedings in advanced robotics, vol 1. Springer, Cham, pp 692–702
64. Ostrowski M, Schaub T (2012) ASP modulo CSP: the clingcon system. TPLP 12(4–5):485–503
65. Palù AD, Dovier A, Formisano A, Policriti A, Pontelli E (2016) Logic programming applied to genome evolution in cancer. In: Fiorentini C, Momigliano A (eds) Proceedings of the 31st Italian conference on computational logic, Milano, Italy, June 20–22, 2016, CEUR workshop proceedings, vol 1645. CEUR-WS.org, pp 148–157
66. Ricca F, Grasso G, Alviano M, Manna M, Lio V, Iiritano S, Leone N (2012) Team-building with answer set programming in the gioia-tauro seaport. TPLP 12(3):361–381
67. Roebuck K (2012) Systems development life cycle (SDLC). Emereo Publishing
68. Rumbaugh J, Jacobson I, Booch G (2005) The unified modeling language reference manual, 2nd edn. Addison-Wesley
69. Sebastiani R (2007) Lazy satisability modulo theories. JSAT 3(3–4):141–224
70. Shchekotykhin KM (2015) Interactive query-based debugging of ASP programs. In: B Bonet, S Koenig (eds) Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI Press, pp 1597–1603
71. Soininen T, Niemelä I (1999) Developing a declarative rule language for applications in product configuration. In: Gupta G (ed) Practical aspects of declarative languages, first international workshop, PADL'99, Lecture notes in computer science, vol 1551. Springer, Berlin, Heidelberg, pp 305–319
72. Stumptner M, Friedrich G, Haselböck A (1998) Generative constraint-based configuration of large technical systems. AI EDAM 12(4):307–320
73. Teppan EC (2016) Solving the partner units configuration problem with heuristic constraint answer set programming. In: Configuration workshop, pp 61–68
74. Teppan EC (2017) On the complexity of the partner units decision problem. Artif Intell 248:112–122
75. Teppan EC, Friedrich G (2016) Heuristic constraint answer set programming. In: GA Kaminka, M Fox, P Bouquet, E Hüllermeier, V Dignum, F Dignum, F van Harmelen (eds) ECAI 2016—22nd European conference on artificial intelligence, frontiers in artificial intelligence and applications, vol 285. IOS Press, pp 1692–1693
76. Teppan EC, Friedrich G, Falkner AA (2012) Quickpup: a heuristic backtracking algorithm for the partner units configuration problem. In: MPJ Fromherz, H Muñoz-Avila (eds) Proceedings of the twenty-fourth conference on innovative applications of artificial intelligence. AAAI
77. Tu PH, Son TC, Baral C (2007) Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. TPLP 7(4):377–450
78. Wang Y, Lee J, Kim DS (2017) A logic based approach to answering questions about alternatives in DIY domains. In: SP Singh, S Markovitch (eds) Proceedings of the thirty-first AAAI conference on artificial intelligence. AAAI Press, pp 4753–4759
79. Weinzierl A (2017) Blending lazy-grounding and CDNL search for answer-set solving. In: Balduccini M, Janhunen T (eds) Logic programming and nonmonotonic reasoning—14th international conference, LPNMR 2017, Lecture notes in computer science, vol 10377. Springer, Cham, pp 191–204
80. Yalciner IF, Nouman A, Patoglu V, Erdem E (2017) Hybrid conditional planning using answer set programming. TPLP 17(5–6):1027–1047

Andreas Falkner holds an MS and a Ph.D. degree in computer science from the Vienna University of Technology. Since 1992 he has been developing product configurators for complex technical systems in various domains at Siemens AG Österreich. At present, he is a senior research scientist at Siemens Corporate Technology, Research Group Configuration Technologies, and senior key expert for product configuration and mass customization.

Gerhard Friedrich is a full professor of computer science at the University Klagenfurt, Austria. He is the dean of the Faculty of Engineering and Technology and directs the research group on Intelligent Systems and Business Informatics. Before his academic career, he was the head of the Department for Configuration and Diagnosis Systems at Siemens AG Austria. His research interests include configuration, planning, diagnosis as well as knowledge representation, acquisition, maintenance and reasoning. Gerhard Friedrich received a PhD and a MS in Informatics from Vienna University of Technology, Austria. In 2012 he became a fellow of the European Association for Artificial Intelligence.

Konstantin Schekotihin is an associate professor of computer science at the University Klagenfurt, Austria. He received his MS in Informatics from NTU "Kharkow Polytechnic Institute", Ukraine and his PhD from University Klagenfurt. His research focus lies mainly on various aspects of knowledge representation and reasoning systems including knowledge acquisition and maintenance, reasoning techniques as well as different applications such as, e.g., configuration, planning, recommendation, etc.

Richard Taupe is a research scientist in the Configuration Technologies research group at Siemens' Corporate Technology. His research interests include answer set programming, constraint programming, and the application of such methods to industrial use cases. He received his MSc degree in computer science from the University of Klagenfurt, Austria, where he is currently pursuing a doctoral degree.

Erich Teppan is an associate professor at the University of Klagenfurt, Austria. He studied at the University of Klagenfurt and at the Westminster University (UK) and received his Engineer and Ph.D. degrees in computer science from the University of Klagenfurt. Teppan worked as a software engineer for various companies in Austria and Switzerland. His main research areas include human computer interaction, recommender systems, logic programming, heuristic search and artificial intelligence.