AI-Based Tool for Curriculum-Based Course Timetabling of University of Potsdam

Christian Dohrmann, Ulrike Lucke, Torsten Schaub, and Sebastian Schellhorn

University of Potsdam, Germany
{christian.dohrmann, ulrike.lucke, torsten.schaub, sebastian.schellhorn}@uni-potsdam.de

Abstract

Education is a fundamental part of universities and comes together with the time consuming non-trivial task of finding suited timetables for teaching. The corresponding problem is called Curriculum-Based Course Timetabling and considers possible constraints when looking for a conflict free timetable. We addressed this problem by following a knowledge representation and reasoning approach, describing constraints and possible assignments of courses to time slots. Possible assignments are collected by a prototype webinterface and corresponding constraints are respected by finding automatically a timetable using an AI. We discuss resolved redundancies, saved time and possible future benefits, when comparing traditional and new approach regarding needs of University of Potsdam.

1 Motivation

Education is a fundamental part of universities, usually addressed by offering a variety of courses may containing lectures, exercises, seminars, projects, etc. within a defined study program. In order to obtain optimal timetables, all offered courses must be assigned to a time slot and room, respecting some constraints. A time slot is a combination of a day and a time. The corresponding problem is called Curriculum-Based Course Timetabling (CB-CTT; [4]) and considers possible constraints when looking for a conflict free timetable. A solution to a CB-CTT problem is an assignment of courses to time slots and rooms, such that each corresponding hard constraint is satisfied. An optimal solution is a solution with minimal penalty regarding soft constraints. Traditionally, such CB-CTT problems are solved by humans and involving a lot of time and agreements. Especially, when several people are involved to develop timetables, redundancies and conflicts among resulting plans may caused. In addition, obtained timetables are often not proved to be optimal and possibly violate soft or hard constraints. In our work, we model corresponding soft and hard constraints in context of University of Potsdam (UP), to get conflict free and optimal timetables derived by an AI. Thanks to its simple modeling language and high performance solving capabilities, Answer Set Programming (ASP; [3]) is a well suited approach of knowledge representation and reasoning to model CB-CTT problems, as [2] showed. We developed a prototype webinterface allowing planners and lecturers to enter availabilities of teaching times and suited rooms regarding courses, to find during a couple of seconds automatically a timetable provably respecting needed constraints among curricula. By comparing the new and traditional approach, we illustrate resolved redundancies, saved time and possible future benefits.

2 Approach

In the following, we describe the traditional and new approach of finding a timetable and link it to the course catalog at UP. To this end, let us first describe the structure and relation among

curricula, modules, examination tasks and courses. A cohort of students is given by a corresponding program and semester wrt the curriculum. Each curriculum is defined by referring to modules needed to be accomplished. Modules define the number and type of courses and course components as well as their secondary and primary examination tasks. For simplicity, in the following we refer courses and course components by courses and make explicit when needed. Courses are linked to, possibly several, secondary and sometimes primary examination tasks of a set of modules. Since modules are associated to possibly several curricula, the linkage of examination tasks to courses induces involved curricula and cohorts possibly participating to a course. Whenever a module occurs in several curricula, possibly of different institutes or faculties, then the goal to find a conflict free timetable becomes more complicated. In the traditional case, lectures with a big number of participants are planned first, due to a limited number of large lecture halls and imports/exports of courses among curricula. The resulting timetable regarding big lectures, needs a lot of communication and agreements among several planners to ensure a conflict free timetable for involved cohorts and lecturers. Especially, in case of degree programmes to become a teacher of allowed combinations of topics, UP developed a time frame model to prevent conflicts among courses of most common combination of topics[1]. On top of scheduled big lectures, remaining courses are planned by planners of each institute to achieve suitable timetables. A timetable should avoid conflicts regarding multiple usage of rooms, lecturers and cohorts as well as respecting particular demands of courses, like technical equipment. A timetable becomes more optimal, if it respects for instance traveling time of consecutive courses, teaching load per day and gaps among courses of a day aiming on same cohort or lecturer, respectively. As soon as a solution is found, each planner has to copy it course by course to universities course catalog, link each course to examination tasks of particular modules and book a room for the assigned time slot.
Our approach aims on time consuming communication among planners and lecturers as well as redundancies when a solution was found but has to get copied to the course catalog, linked to modules and booked to rooms. To this end, we developed a prototype webinterface allowing planners and lecturers to create courses and course components together with its module linkages, prioritized available time slots as well as rooms and relations among courses. Four pairwise relations among courses are provided:

- two course components are not allowed to take place in parallel, e.g. obligatory lectures of same cohort
- two course components have to take place in parallel (same time slot, but different rooms), e.g. two groups of exercises of the same course
- two course components have to take place simultaneous (same time slot and room), e.g. hosting two seminars sharing major content in same room
- two course components have to take place consecutive, e.g. having an exercise directly after the corresponding lecture

For each course component, the webinterface allows to state the expected number of participants as well as the need of particular equipment, e.g. large board. See figure 1 illustrating parts of the prototype webinterface. The corresponding planner checks the consistence and correctness of the input data for the webinterface. The input data together with a logic representation of necessary constraints is given to an AI, that searches for an optimal timetable. The solution is a visualization of the resulting timetable and double checked by planners. Note that the new

[1] https://www.uni-potsdam.de/en/studium/studying/organizing-your-studies/potsdams-time-frame-model

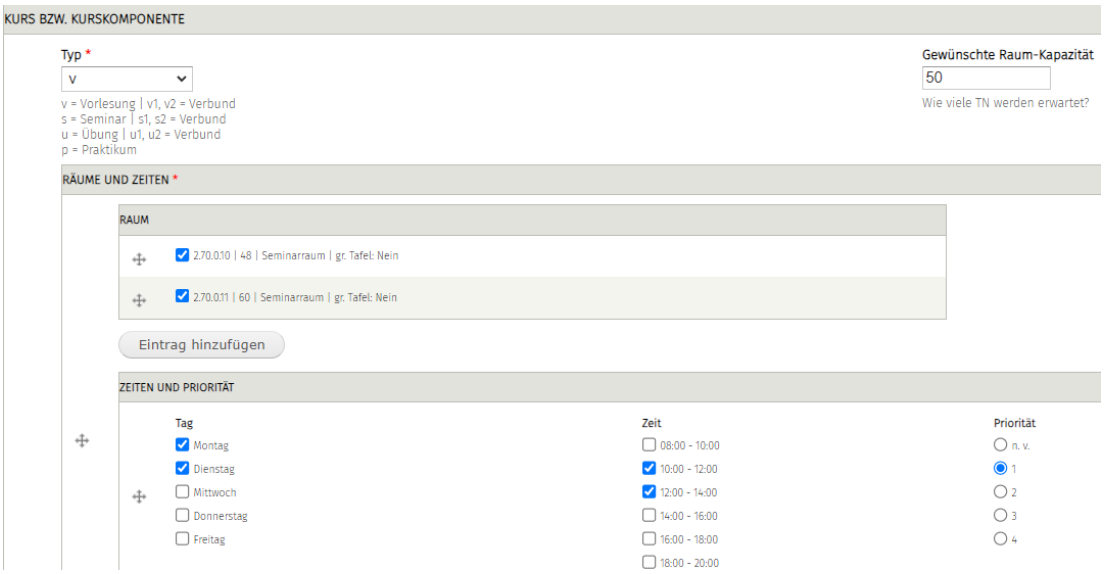

Figure 1: Webinterface inducing availabilities by the cross product of days, times and rooms for a lecture.

approach follows the traditional idea of splitting the process into two steps, by first finding a timetable regarding big lectures and then completing it to a timetable for each institute. The resulting timetable of each institute has to (automatically) transferred to universities course catalog respecting linked modules, time slots and rooms. This last step of integrating the obtained timetable automatically to the course catalog is not yet part of the prototype but will omit a lot of redundancies.

3 ASP Paradigm

ASP [3, 6] is an approach to declarative problem solving and belongs to the area of knowledge representation and reasoning. The roots of ASP go back to logic programming, nonmonotonic reasoning and constraint satisfaction [15]. The idea of ASP is to describe the problem by using a formal representation, rather than telling a computer how to solve the problem. Figure 3 illustrates this approach [13]. An easy and human readable modeling language [7] is used to obtain a logic program modeling the original problem. By using problem solvers like *clingo* [12], a solution, i.e. stable model [14], of a problem is found and has to be interpreted by the user. Due to its simple but rich high-level modeling language as well as its elaboration-tolerance and its high-performance solving capacities, ASP becomes a well suited approach to model real world problems. The range of applications to model and solve combinatorial search problems is among planning, scheduling, configuration, probabilistic reasoning, diagnosis and repair, classification, query answering, explanation generation, multi-agent systems, natural language processing, computational biology, music composition, model checking and robotics [8]. In Terms of industrial applications, ASP has been used for solving (automated product) configuration [18, 16], planning [17, 19], scheduling [1], timetabling [2] and other problems [10]. Especially, for industrial applications, tackled by a declarative approach like ASP, significant improvements wrt implementation, maintenance costs and human-machine interaction were

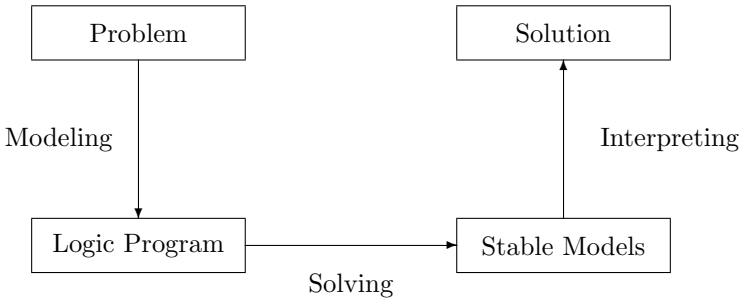

Figure 2: Declarative problem solving.

shown. For instance, Siemens reported to save more than 80% of maintenance costs by using a knowledge representation and reasoning approach [9].

3.1 Modeling with ASP

ASP provides a powerful high-level modeling language, which allows recursive definitions, default negation for dealing with the absence of information, disjunctions, aggregates, weight constraints, optimization statements and external atoms. In the following, we give a short introduction to basics of the modeling language of ASP [7] and the stable models semantics of logic programs [14].

Terms are either constants, variables, arithmetic terms or functional terms. Constants are denoted by strings starting with lowercase letter, quoted strings or integers. Variables are denoted by strings starting with uppercase letter. A predicate atom, is of form $p(t_1, \ldots, t_n)$, where p is a predicate name, $t_1, \ldots, t_n$ are terms and $n \geq 0$ is the arity of the predicate atom. Predicate atoms of arity 0 are represented by its name without parentheses. In the following, predicate atoms are called atoms for short and are denoted by a. An atom is called ground and can be understood as propositional, if all its variables are substituted by constants. A literal is an atom a or its default negation $not\ a$. Default negation, also known as negation as failure, refers to the absence of information, whereas "classical" negation induces the presence of negated information. A rule r is of form

$$a_0 \leftarrow a_1, \ldots, a_n, not\ a_{n+1}, \ldots, not\ a_m$$

where $0 \leq n \leq m$ and each a_i is an atom for $1 \leq i \leq m$. Let $head(r) = a_0$ and $body(r) = \{a_1, \ldots, a_n, not\ a_{n+1}, \ldots, not\ a_m\}$ the head and the body of a rule r, respectively. For a set of literals X, let $X^+ = \{a \mid a \in X\}$ and $X^- = \{a \mid not\ a \in X\}$. Intuitively, the $head(r)$ of a rule r must hold, if the $body(r)$ holds. The body $body(r)$ of rule r holds, if atoms of $body(r)^+$ are provably true and atoms of $body(r)^-$ are possibly false. A rule r is called fact, whenever $body(r) = \emptyset$.[2] A rule r is called (integrity) constraint, whenever $head(r) \cap body(r)^- \neq \emptyset$.Integrity constraints are used to eliminate unintended solution candidates, whenever its body is satisfied. As an example, consider rule

```
conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))) :-
                        book(42, ("AI", v)), book(43, ("AI", u)),
```

[2]Usually, we drop $\leftarrow$ in this case.

```
consecutive(("AI", v), ("AI", u)),
availability_day(42, fr),
not availability_day(43, fr).
```

deriving a conflict among two course components of "AI" lecture and exercise that shall take place consecutive, whenever they are scheduled on different days.[3] A logic program is a finite set of rules. A set of ground atoms X is a model of a logic program P, if for every $r \in P$ holds $head(r) \in X$ whenever $body(r)^+ \subseteq X$ and $body(r)^- \cap X = \emptyset$. The stable model of a program P is defined relative to a set of atoms X, by the so called reduct P^X, defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P, body(r)^- \cap X = \emptyset\}$$

X is a stable model of a program P, if X is the $\subseteq$-minimal model of P^X. As an example consider the following program P

```
1  room("2.70.0.10").
2  room("2.70.0.11").
3  room_capacity("2.70.0.10", 48).
4  room_capacity("2.70.0.11", 60).
5  course(("AI", v)).
6  course(("AI", u)).
7  course_size(("AI", v), 50).
8  course_size(("AI", u), 50).
9  consecutive(("AI", v), ("AI", u)).
10 availability_course(42, ("AI", v)).
11 availability_room(42, "2.70.0.11").
12 availability_day(42, fr).
13 availability_time(42, 10).
14 availability_course(43, ("AI", u)).
15 availability_room(43, "2.70.0.10").
16 availability_day(43, mo).
17 availability_time(43, 12).
18 availability_course(44, ("AI", u)).
19 availability_room(44, "2.70.0.11").
20 availability_day(44, fr).
21 availability_time(44, 12).
22 book(42, ("AI", v)) :- availability_course(42, ("AI", v)).
23 book(43, ("AI", u)) :- availability_course(43, ("AI", u)),
24                        not book(44, ("AI", u)).
25 book(44, ("AI", u)) :- availability_course(44, ("AI", u)),
26                        not book(43, ("AI", u)).
27 conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))) :-
28                              book(42, ("AI", v)), book(43, ("AI", u)),
29                              consecutive(("AI", v), ("AI", u)),
30                              availability_day(42, fr),
31                              not availability_day(43, fr).
32 conflict :- conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))),
33           not conflict.
```

where lines 1-21 are facts setting rooms, room capacities, course components with expected number of participants, stating "AI" lecture and exercise as consecutive as well as declaring possible time slots and rooms for each course component. Line 22 sets "AI" lecture to time

[3] In the syntax of an ASP encoding '$\leftarrow$' is represented by ': −' and each rule terminates by a period '.'.

slot with identifier 42, since there is no other option. Lines 23-26 state that either time slot with identifier 43 or 44 has to be taken for the "AI" exercise. Lines 27-31 derive a conflict predicate, whenever "AI" lecture and exercise are not at same day and thus not consecutive. Lines 32-33 form an integrity constraint ruling out models containing conflict predicate derived by lines 27-31. Since atoms of lines 1-21 are facts, they belong to any model of P. Let X be the set containing atoms of lines 1-21. Note that X is not a model of P, since the body of rule in line 22 is satisfied, but not its head. Sets

$$X \cup \{\texttt{book(42, ("AI", v)), book(43, ("AI", u)),}\\ \texttt{conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))), conflict}\}$$

and

$$X \cup \{\texttt{book(42, ("AI", v)), book(44, ("AI", u))}\}$$

are models of P, but the latter one is the only stable model of P.
The basic approach of modeling a problem by an ASP encoding follows a generate-and-test methodology, also known as guess-and-check, inspired by intuitions on *NP* problems. In case of CB-CTT, one could think of generating any timetable as a model and ruling out certain models by posing particular (hard) constraints.

3.2 Modeling CB-CTT of UP

Since a course may consists of several course components of possibly different types, e.g. a lecture and an exercise, we identify a course component by a unique tuple of a name and its type, e.g. lecture *("Artificial Intelligence", v)* and exercise *("Artificial Intelligence", u)* of Artificial Intelligence course. We modeled the following hard (H0-8) and soft (S0-13) constraints.

H0. Components: No component of a course should take place in parallel to its corresponding lecture, except if explicitly stated.

H1. Courses: One availability for each course component must be assigned to a time slot and room.

H2. Cohort: Obligatory lectures of the same cohort must be all scheduled in different time slots, except explicitly stated to take place in parallel.

H3. RoomOccupancy: Two course components cannot take place in the same room and time slot, except explicitly stated.

H4. Lecturer: Course components sharing a lecturer cannot be scheduled in parallel, except explicitly stated.

H5. NotParallel: Avoiding two course components to take place in parallel, whenever explicitly stated.

H6. Parallel: Two course components have to take place in parallel, whenever explicitly stated.

H7. Simultaneous: Two course components have to take place simultaneous, whenever explicitly stated.

H8. Consecutive: Two course components have to be scheduled consecutive, whenever explicitly stated.

S0. Availability: For each course component, try to serve prioritized availability on time slot and room. The penalty points reflecting corresponding priorities.

S1. RoomCapacity: For each course component, penalty points for the number of students that are expect to attend the course minus the number of seats in the corresponding room are imposed on each violation.

S2. Cohort: Course components addressed to the same cohort should be scheduled in different time slots, except explicitly stated to be in parallel. Each violation counts as a penalty point.

S3. Gaps: For a cohort as well as lecturer, corresponding course components should be scheduled in time slots as close as possible. The penalty regarding two courses sharing a cohort or lecturer, and a day is given by subtracting the earlier time from the later time.

S4. RoomStability: Two course components stated to be consecutive should be booked in the same room. The penalty points reflecting each violation.

S5. MaxLoad: For a cohort as well as lecturer the number of corresponding course components per day should be lower or equal to a given maximum. The penalty points reflecting the number of courses above the maximum.

S6. TravelTime: For a cohort as well as a lecturer, traveling time between rooms with two adjacent course components should be as small as possible. The penalty is reflected by the traveling time itself.

S7-13. RoomSuitability: Some course components prefer particular equipment like a large board, projector, computer, microphone, camera, media table or whiteboard. Each violation counts as a penalty point.

Note that some constraints are similar to those presented in [2], others are new or modified. Due to lack of space, we drop the corresponding logic program here[4]. Optimization tries to minimize penalty points regarding above soft constraints. The system of [2], reads instances of a standard input format [5], translates them into ASP facts and assigns potentially any course to any time slot. In contrast, we used a direct modeling dedicated to constraints and needs of the UP, which partially is not covered by the standard input format. As a design decision, we reduced the search space to collected availabilities only, instead of checking for all possible time slots and rooms for each course component.

4 Results and Impact

In [2] was shown, that treating CB-CTT problems by an ASP based approach outperforms state of the art approaches and makes it well suited to tackle CB-CTT problems. Our real world approach is still at a prototype stage and is lacking access to some data needed to apply all presented and modeled constraints. For winter 2023/24, an optimal and conflict free plan was found for about 160 courses aimed at big lecture halls of the Faculty of Science, all 90 courses of the Institute of Mathematics and 100 courses of the Institute of Computer Science, involving constraints H1, H3, H5, S0, S1 and S7 of above ASP based modeling. Analogously, for summer 2024, an optimal and conflict free plan was found involving about 200 courses and constraints H0, H1, H3, H5, H6, H7, H8, S0, S1, S4 and S7. Solver *clingo* was integrated to the web interface and needed about 170 seconds for winter and less than one second for summer, respectively, to solve the problem regarding the faculty. Finding plans for the institutes took less than a second, respectively. A major benefit of the presented AI-based approach is its fairness, transparency and elaboration tolerance. Planners and lecturers are able to set a prioritized selection of availabilities and preferences to upcoming teaching periods, resulting in a provable conflict free timetable. Getting a conflict free timetable for a faculty in a couple of seconds, respecting dependencies among several institutes at the same time, resolves a lot of communication overhead and conflicts of the earlier approach. Thus, our approach gives planners already the opportunity to address their additional available time to other tasks. As

[4]The particular ASP encoding representing above constraints can be found at GitHub: https://github.com/schellhorn/CBCTT-UP.

soon as the integration of the resulting timetable to the universities course catalog is done automatically, then even more redundancies are resolved and a lot more capacities become available. Due to our current prototype state, we did not yet compare to other approaches[11] regarding performance.

5 Future Work

For the future, we plan to involve an interface, allowing us to access data like previous course catalogues and curricula as well as their modules. Furthermore, we plan to develop an interface, allowing us to push final timetables to universities course catalog automatically. Another, not trivial task, is to provide opportunities to the user to analyze and solve conflicts, whenever needed. In a next step, we are going to extend the number of users and involve them in a feedback loop, to improve the usage of the system and check for additional needs. In a long run, one could think about to add functionalities like drag and drop to take smaller changes by hand to a calculated timetable into account, preserving consistence regarding hard and soft constraints. When establishing the mentioned workflow to get conflict free timetables, one could measure the average amount of time needed per planner using the traditional and new approach, respectively. Moreover, one could compare the quality of resulting timetables regarding desired hard and soft constraints. Comparing quality and used time of both approaches may highlight the usage and impact of our new approach using AI of knowledge representation and reasoning. The data about availabilities of different types of course components to several time slots, provides insights to the needs of room infrastructure. One could think about, to possibly exploit entered availabilities and number of participants to derive location, size and quantity of needed lecture halls and seminar rooms, when plan to build new ones. Finally, our approach provides a lot potential to get rid of redundancies, save money and lift administrative planning tasks to the next level.

References

[1] M. Alviano, C. Dodaro, and M. Maratea. An advanced answer set programming encoding for nurse scheduling. In F. Esposito, R. Basili, S. Ferilli, and F. Lisi, editors, *Proceedings of the Sixteenth International Conference of the Italian Association for Artificial Intelligence*, volume 10640, pages 468–482, 2017.

[2] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, and P. Wanko. teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 275(1):3–37, 2019.

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[4] A. Bettinelli, V. Cacchiani, R. Roberti, and Paolo Toth. An overview of curriculum-based course timetabling. *TOP*, 23(2):313–349, 2015.

[5] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf. Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, 194(1):59–70, 2012.

[6] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[7] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, and T. Schaub. ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020.

[8] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Mag.*, 37(3):53–68, 2016.

[9] A. Falkner, G. Friedrich, A. Haselböck, G. Schenner, and H. Schreiner. Twenty-five years of successful application of constraint technologies at siemens. 37(4):67–80, 2016.

[10] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. Teppan. Industrial applications of answer set programming. *Künstliche Intelligenz*, 32(2-3):165–176, 2018.

[11] Thomas Feutrier. *Landscape Analysis and Solver Reconfiguration for the Curriculum-Based Course Timetabling*. Theses, Université de Lille, December 2023.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko. Theory solving made easy with clingo 5. In M. Carro and A. King, editors, *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, volume 52 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[15] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[16] K. Herud, J. Baumeister, O. Sabuncu, and T. Schaub. Conflict handling in product configuration using answer set programming. In *Proceedings of the Fifteenth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'22)*, volume 3193 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[17] Y. Jiang, S. Zhang, P. Khandelwal, and P. Stone. Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems. *Frontiers of Information Technology and Electronic Engineering*, 20(3):363–373, 2019.

[18] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In Gopal Gupta, editor, *Practical Aspects of Declarative Languages, First International Workshop, PADL '99, San Antonio, Texas, USA, January 18-19, 1999, Proceedings*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 1999.

[19] Tran Cao Son, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: A survey. *Theory Pract. Log. Program.*, 23(1):226–298, 2023.