Reasoning about University Study Regulations Module-based Student Constraints and Preferences in Answer Set Programming

Henry Otunuya[0000−0002−1012−7887] and Sebastian Schellhorn[0000−0003−4928−2515]

University of Potsdam, Germany
henry.chuks.otunuya@uni-potsdam.de
sebastian.schellhorn@uni-potsdam.de

Abstract. University study regulations are legal documents which usually induce an enormous amount of valid study plans from which students may choose. And with several other works dealing with courses and/or examinations and not necessarily the regulations behind them, this work is part of an attempt to shift perspectives. Indeed, the combinatorial solution space induced by university study regulations is not at all reasonable for students to explore manually, let alone check for the satisfaction of certain desirable characteristics. Hence, building upon our previous work, and as part of the effort in the digitalization of higher educational institutions, we hereby present four hard and soft constraints which students are able to apply to find study plans which suits their needs.We motivate and formalize the four constraints to reflect needs on the number of ECTS per semester, occurrences of modules in a specific semester range, the number of modules for a particular semester and the number of total semesters needed to finish studies. We use Answer Set Programming (ASP) to model these constraints. An ASP-driven user interface prototypes our implementation allowing for user interactions with the constraints and solution space exploration according to individual demands.

Keywords: Study regulations · Answer set programming · Study plan.

1 Introduction

With university study regulations[1], higher educational institutions are able to offer and regulate the modules and examinations required by students to earn a degree. In such offers, university study regulations are mostly open when it comes to what modules students can take and in what semester.

[1] Also known as *university study and examinations regulations*, in universities like in TU Berlin [16] and Heinrich-Heine-Universität Düsseldorf [7].

This openness creates both opportunities and challenges [9, 8]. The opportunities come from the freedom to choose whatever module a student finds interesting, when to take such module and what other module to combine with it. On the otherhand there are challenges, some of which include how to keep track that the general study regulations constraints are satisfied - such as total ECTS credit, dependency between selected modules and credit requirements before thesis. Other challenges deal with how to ensure that a given credits bound for certain semesters are accomplished, that certain modules are part (or not part) of a study plan or that a certain optimization is performed. Admittedly, the prominence of the different challenges become more evident with increasing number of modules.

Furthermore, some of those challenges were pointed out by students interviewed within the *CAVAS+* project[2] [14], which this work is part of, further drawing our attention to them within the university environment.

In [6], we presented a set-based mathematical model of a study regulations problem from the University of Potsdam in Germany. There, we formally defined some study regulations constraints using the M.Sc. Cognitive Systems study program as our test-bed. With that we were able to ensure that some general rules in study regulations are checked in an automated fashion. Answer Set Programming (ASP) [10], a declarative combinatorial problem solving approach, was used to encode the problem. This work simply builds on that foundational paper.

From literature, it was observed that many works deal with the scheduling of courses, and not the undelying regulations behind the offered courses [11, 15, 12]. We argue that it is crucial to have a connection to actual university study regulations as this aligns such offerings with the legal goals of the higher educational institution involved.

This work and our foundational paper is part of an attempt to fill that gap. Our goal is to present some student-related hard and soft constraints which can be introduced into the solution space of study plans, thereby addressing some challenges. We add four constraints to reflect needs on the number of ECTS per semester, occurrences of modules in a specific semester range, the number of modules for a particular semester and the number of total semesters needed to finish studies. An ASP-driven user interface prototypes our implementation allowing for user interactions with the constraints and solution space exploration according to individual demands.

The paper is structured as follows - in Section 2 we present the hard constraints followed by the associated soft constraints in a mathematically formal way, with all the soft constraints sharing a common optimization function. An introduction to ASP, the ASP encodings of the constraints and our user interface are discussed in Section 3. In Section 4 we provide some related works, and then conclude with some future works in Section 5.

[2] `https://www.uni-potsdam.de/en/cavas-plus-en/index`(last accessed: June 2026).

2 Formalization of hard constraints and preferences

With hard constraints students are able to specify the required characterics of any potential study plan. These are constraints which must be satisfied. On the other hand, when hard constraints are not satisfiable at all, there is the possibility of switching to soft constraints, which encode the preferences. The first and the last constraints presented here are relatable to questions asked by interviewed students within the *CAVAS+* project [14], while the other two were inspired from the desire of students to have individualized study plans reported in literature [9, 13].

Since students are possibly not always interested in solutions which strictly satisfy given constraints and sometimes it may not even be possible to find a solution which satisfies given hard constraints or their combinations, preferences come into play.

To this end, we define our objective function $dtb : \mathbb{N}^3 \to \mathbb{N}$ which is the distance from the value x to some lower bound l and upper bound u, written as $dtb(x, l, u)$, as:

$$dtb(x,l,u) = \begin{cases} l - x & \text{if} \quad x < l \\ x - u & \text{if} \quad x > u \\ 0 & \text{otherwise} \end{cases}$$

With *dtb* we are able to get a value close to l and u, and then optimize such values.

We define our soft constraints as so-called *preference relations*. A preference relation is a preorder over possible study plans. That is, $\preceq \subseteq (S_i)_{i=1}^n \times (S'_i)_{i=1}^k$ for all $\preceq \in \{\preceq_{sc}, \preceq_{ms}, \preceq_{sm}, \preceq_{sb}\}$[3], where $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$ are possible study plans from some *basic study regulation problem* $\mathcal{B}$[4].

For any two possible study plans $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$, $(S_i)_{i=1}^n$ is preferred or equal to $(S'_i)_{i=1}^k$ with respect to function *dtb* on some bound, iff $(S_i)_{i=1}^n \preceq (S'_i)_{i=1}^k$, for some given preference relation $\preceq \in \{\preceq_{sc}, \preceq_{ms}, \preceq_{sm}, \preceq_{sb}\}$.

With each of the preference relations, except $\preceq_{sb}$, the result of the summed up *dtb* values in the first study plan $(S_i)_{i=1}^n$ must be less than or equal to the result of the summed up *dtb* values in the second study plan $(S'_i)_{i=1}^k$. For $\preceq_{sb}$, no aggregation is performed, only a simple check.

In the following subsections, we present hard constraints and preferences which relate to modules in our study regulations.

[3] These are our preference relations, defined in the following subsections.

[4] Notations not defined here, such as $\mathcal{B}, (S_i)_{i=1}^n$, come for our foundational work. For quick reference we put them all in Appendix B.

2.1 Semester-to-ECTS credits bound

Each module has an ECTS value representing the workload needed in accomplishing the module. Functions $scl, scu : \{1, 2, \ldots, n\} \to \mathbb{N}$ assign semesters of length $n \in \mathbb{N}$ to lower and upper bounds credits respectively[5].

In order to bound the ECTS workload in each semester, we employ the above functions. Constraint $R_{sc} \subseteq 2^{(2^M)^n}$ ensures that the semesters are within their ECTS bounds:

$$R_{sc} = \{\{(S_i)_{i=1}^n \subseteq M^n \mid scl(i) \leq \textstyle\sum_{m \in S_i} c(m) \leq scu(i) \text{ for all } 1 \leq i \leq n\}\}$$

For example, let us consider a hypothetical student Paul, who wants to take between 30 to 42 ECTS credits in the second semester, nothing in the third and exactly 30 ECTS credits in the fourth. That is:

$$scl = \{2 \mapsto 30, 3 \mapsto 0, 4 \mapsto 30\}$$
$$scu = \{2 \mapsto 42, 3 \mapsto 0, 4 \mapsto 30\}$$

The preceding formalization is encapsulated by the tuple (B, scl, scu, R_{sc}), which represents a *study regulation problem with semester-to-ECTS credits bound*. For the preference, we introduce a shared framework which will be used for all the preferences presented in this paper.

Preference on semester-to-ECTS credits bound Given our *scl* and *scu* bound, we define the preference relation $\preceq_{sc}$ on semester-to-ECTS credits bound wrt possible study plans $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$ by $(S_i)_{i=1}^n \preceq_{sc} (S'_i)_{i=1}^k$ iff:

$$\sum_{i=1}^{n}(dtb(\sum_{m \in S_i} c(m), scl(i), scu(i))) \leq \sum_{i=1}^{k}(dtb(\sum_{m \in S'_i} c(m), scl(i), scu(i)))$$

The $\preceq_{sc}$ preference relation prefers study plans with total ECTS credit of all the given semester mappings closest to all associated scl, scu range values.

2.2 Module-to-semester bound

Furthermore, a student may want to assign modules to specific semesters. In order to require modules $M_I \subseteq M$ to be available within a certain semester bound, functions $msl, msu : M_I \to \{1, 2, \ldots, n\}$ assign each $m \in M_I$ to a lower bound and to an upper bound of semester of length $n \in \mathbb{N}$, respectively[6]. We

[5] If no bounds are given for any semester (respectively undefined), then *scl* and *scu* take 0 and ∞ by default, respectively.

[6] If no bounds are given for any module (respectively undefined), then *mcl* and *mcu* take 1 and n by default, respectively.

introduce constraint $R_{ms} \subseteq 2^{(2^M)^n}$ to ensure that modules in M_I always belong to the resulting study plans and stay within their assigned bounds:

$$R_{ms} = \{\{(S_i)_{i=1}^n \subseteq M^n \mid M_I \subseteq \overline{(S_i)_{i=1}^n}\},$$
$$\{(S_i)_{i=1}^n \subseteq M^n \mid m \in M_I \cap S_i, msl(m) \leq i \leq msu(m) \text{ for all } 1 \leq i \leq n\}\}$$

With this Paul wants to state that module $am_{1,1}$ must be accomplished in the first three semesters and module pm_2 can occur only in the second semester. That is:

$$M_I = \{am_{1,1}, pm_2\}$$
$$msl = \{am_{1,1} \mapsto 1, pm_2 \mapsto 2\}$$
$$msu = \{am_{1,1} \mapsto 3, pm_2 \mapsto 2\}$$

The preceding formalization is encapsulated by the tuple $(B, M_I, msl, msu, R_{ms})$, which represents a *study regulation problem with module-to-semester bound*.

Preference on module-to-semester bound Given our msl and msu bound, we define the preference relation $\preceq_{ms}$ on module-to-semester bound wrt possible study plans $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$ by $(S_i)_{i=1}^n \preceq_{ms} (S'_i)_{i=1}^k$ iff:

$$\sum_{\substack{i=1 \\ m \in M_I \cap S_i}}^{n} (dtb(i, msl(m), msu(m))) \leq \sum_{\substack{i=1 \\ m \in M_I \cap S'_i}}^{k} (dtb(i, msl(m), msu(m)))$$

The $\preceq_{ms}$ preference relation prefers study plans with modules of all the given modules mappings closest to all associated msl, msu range values.

2.3 Semester-to-number of modules bound

Sometimes students may want to ensure that a certain number of modules are available in each semester. We generalize this constraint with the number of modules bound. This will allow a student to set a lower and an upper bound. The lower and upper bounds for a module are assigned by functions $sml, smu : \{1, 2, \ldots, n\} \to \mathbb{N}$ respectively[7], for semesters of length $n \in \mathbb{N}$.

Constraint $R_{sm} \subseteq 2^{(2^M)^n}$ ensures that for each semester with a set bound, such bound is respected:

$$R_{sm} = \{\{(S_i)_{i=1}^n \subseteq M^n \mid sml(i) \leq |S_i| \leq smu(i) \text{ for all } 1 \leq i \leq n\}\}$$

For example, Paul wants to take at least five modules in the first semester and limits his second semester by at most four modules. That is:

$$sml = \{1 \mapsto 5\}$$
$$smu = \{2 \mapsto 4\}$$

[7] If no bounds are given for any semester (respectively undefined), then sml and smu take 0 and ∞ by default, respectively.

The preceding formalization is encapsulated by the tuple (B, sml, smu, R_{sm}), which represents a *study regulation problem with semester-to-number of modules bound*.

Preference on semester-to-number of modules bound Given our sml and smu bound, we define the preference relation $\preceq_{sm}$ on semester-to-number of modules bound wrt possible study plans $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$ by $(S_i)_{i=1}^n \preceq_{sm} (S'_i)_{i=1}^k$ iff:

$$\sum_{i=1}^{n}(dtb(|S_i|, sml(i), smu(i))) \leq \sum_{i=1}^{k}(dtb(|S'_i|, sml(i), smu(i)))$$

The $\preceq_{sm}$ preference relation prefers study plans with semesters of all the given semester mappings closest to all associated sml, smu range values.

2.4 Number of semesters bound

Another need of students often is, to define the number of semesters they want to study in. Formally, $sl, su \in \mathbb{N}$ respectively holds the lower and the upper bound for any study plan. Constraint $R_s \subseteq 2^{(2^M)^n}$ ensures that all resulting study plans are within the given semester bounds:

$$R_s = \{\{(S_i)_{i=1}^n \subseteq M^n \mid s_l \leq n \leq s_u\}\}$$

For example, Paul wants to finish is study within 4 or 5 semesters. That is:

$$sl = 4$$
$$su = 5$$

The preceding formalization is encapsulated by the tuple (B, sl, su, R_s), which represents a *study regulation problem with number of semesters bound*.

Preference on number of semesters bound Given our sl and su bound, we define the preference relation $\preceq_{sb}$ on number of semesters bound wrt possible study plans $(S_i)_{i=1}^n$ and $(S'_i)_{i=1}^k$ by $(S_i)_{i=1}^n \preceq_{sb} (S'_i)_{i=1}^k$ iff:

$$dtb(n, sl, su) \leq dtb(k, sl, su)$$

The $\preceq_{sb}$ preference relation prefers study plans with semester length closest to sl, su range values.

Each combination of hard constraints in the preceding subsections interact with the general constraints in $\mathcal{B}$ in a similar fashion. For example, if we consider sl, su and R_s in addition to $\mathcal{B}$, the resulting study plan $(S_i)_{i=1}^n$ respects $(S_i)_{i=1}^n \in \bigcap_{r \in R_s \cup R_g \cup R_t} r$.

In the case of Paul and applying all his hard constraints,

$$(S_i)_{i=1}^4 = (\{bm_1, bm_3, am_{1,1}, am_{1,2}, am_{3,1}, pm_1\}, \{bm_2, im, am_{3,2}, pm_2\}, \{\}, \{msc\})$$

would be a possible solution, with total ECTS credits of 48, 42, 0 and 30 for semesters 1 to 4, respectively.

Each preference means something different and together allows a student to have some desired study plan(s).

3 ASP Encoding and User Interface

ASP is a logic-based paradigm suitable for solving combinatorial problems. An ASP program is a set of rules of the form

$$\mathtt{a}_0 \ \mathtt{:-}\ \mathtt{a}_1\mathtt{,...,a}_m\mathtt{,not\ a}_{m+1}\mathtt{,...,not\ a}_n$$

where each $\mathtt{a}_i$ is an atom of form $\mathtt{p(t}_1\mathtt{,...,t}_k\mathtt{)}$ and all $\mathtt{t}_i$s are terms, composed of function symbols and variables. For $0 \leq m \leq n$, atom $\mathtt{a}_0$ is often called the head atom, while $\mathtt{a}_1$ to $\mathtt{a}_m$ and $\mathtt{not\ a}_{m+1}$ to $\mathtt{not\ a}_n$ are also referred to as positive and negative body literals, respectively. An expression is said to be ground, if it contains no variables.

A rule is called a fact if $m = n = 0$, and an integrity constraint if the head, $\mathtt{a}_0$, is empty. Intuitively, the rule says that if $\mathtt{a}_1$ to $\mathtt{a}_m$ are true and $\mathtt{a}_{m+1}$ to $\mathtt{a}_n$ are false, then $\mathtt{a}_0$ must be true.

Semantically, an ASP program induces a set of stable models, being distinguished models of the program determined by the stable models semantics [5].

For an in-depth coverage on ASP, we direct the interested reader to [10, 4].

3.1 Encoding

Here, we talk about the ASP encoding of the constraints presented in the previous section - for brevity we only focus on the first hard and soft constraints. The other encodings are in Appendix A. Note that the ASP encoding which generates the study plans was already presented in our previous paper, on top of which the following encoding operates. Additionally, as in that paper, our encoding was solved with the ASP system *clingo*[8].

For the hard constraints, we re-use the `map(F,E,V)` predicate which defines functions. That is, every function defined above is represented by `map(F,E,V)` which says that function `F` applied to element `E` gives value `V`. For example, with the *scl* and *scu* functions of the *semester-to-ECTS credits bound*, we have them as `map(scl,I,L)` and `map(scu,I,U)` respectively, with `I`, `L` and `U` being variables.

The complete encoding for the semester-to-ECTS credits bound constraint is given in Listing 1.1.

As a recap, `in(_,s(I))` holds modules which satisfy both the `map(scl,I,L)` and `map(scu,I,U)` generated from user interaction on the user interface, lines 2-4 show the hard constraints. There we say, if there is *semester-to-ECTS credits bound* constraint set for semester `I`, then the sum of the credits in that semester must be within the `L,U` bound.

[8] Version 5.4.0 - `https://potassco.org/clingo/` (last accessed: June 2026)

```
1  % hard constraint
2  :- in(_,s(I)), map(scl,I,L), map(scu,I,U),
3      not L #sum{ V,M : in(M,s(I)), map(c,M,V) } U,
4      not checked_pref(map(scl,I,L)), not checked_pref(map(scu,I,U)).

6  % soft constraint
7  map(dtb,I,D) :- in(_,s(I)), map(scl,I,L), map(scu,I,U),
8                  X = #sum{ V,M : in(M,s(I)), map(c,M,V) }, X < L,
9                  D = L - X.
10 map(dtb,I,D) :- in(_,s(I)), map(scl,I,L), map(scu,I,U),
11                 X = #sum{ V,M : in(M,s(I)), map(c,M,V) }, X > U,
12                 D = X - U.

14 #minimize{ D,I: map(dtb,I,D), map(scl,I,L), checked_pref(map(scl,I,L)),
15                                map(scu,I,U), checked_pref(map(scu,I,U)) }.
```

Listing 1.1. Hard and soft constraint of *semester-to-ECTS credits bound* constraint in `semester_to_ects.lp`.

Atom `map(dtb,I,D)` holds the *dtb* function value `D` for semester `I`. This predicate is used in lines 14-15 to encode the optimization of the associated constraint.

The `checked_pref/1` predicate (with arity 1) is connected to the user interface and will be explained in the next subsection. The other constraints in Appendix A have a similar structure with different ways for generating the *dtb*-related atoms. The encodings of the other constraints can be found in their associated files in the repository[9].

3.2 User Interface

In this section, we present our ASP-driven user interface made with the *clinguin* system[10]. We only discuss the newly introduced ui-related atoms here and for the basis refer the interested reader to the foundational paper [6].

Additionally, for an in-depth look into the *clinguin* system the interested reader is referred to [2]. Our implementation has a so-called *custom backend*[11] which defines Python functions for handling the adding and removing, the conversion into preferences and the off/on switching of constraints[12].

[9] `https://github.com/krr-up/study-regulations-modules-constraints-encodings/tree/main/encodings` (last accessed: June 2026)

[10] Version 2.8.4 - `https://clinguin.readthedocs.io/en/latest/` (last accessed: June 2026)

[11] `https://github.com/krr-up/study-regulations-modules-constraints-encodings/blob/main/encodings/user_interface/backend.py` (last accessed: June 2026)

[12] Functions `add_constraint`, `remove_constraint` and `make_preference` are implemented in `backend.py`.

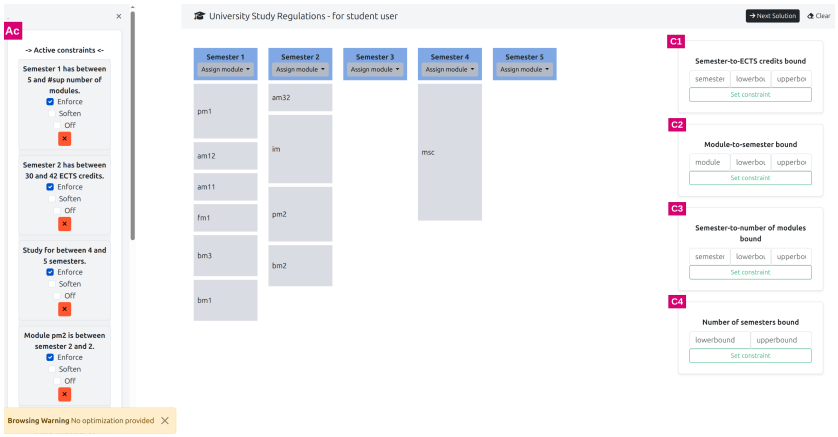

Fig. 1. User interface with constraints

The `checked_pref/1` predicate is generated when the user checks the *Soften* checkbox and the corresponding constraint passed as the value, and removed when the checkbox is unchecked.

Figure 1 shows the screenshot our *clinguin* implementation[13]. Labels **C1** (for *semester-to-ECTS credits bound* constraint), **C2** (for *module-to-semester bound* constraint), **C3** (for *semester-to-number of modules bound* constraint), **C4** (for *number of semesters bound* constraint), and **Ac** were added to the screenshot to aid the following explanation.

With study plans shown in the middle of the user interface, labels **C1** to **C4** show HTML cards for the four constraints. **C1** to **C3** have three input fields and **C4** two fields. With **C1**, a student is able to enter a semester with the lowerbound and upperbound ECTS credits values. Clicking on *Set constraint* introduces and activates a constraint and limits the solution space accordingly.

With **C2**, a student is able to enter a module and set the lower and upper bound of its semester values. With **C3**, a student is able to enter a semester with the lower and upper bound number of modules for that semester. And lastly with **C4**, the lower and upper bound semester values can be set.

The **Ac** sidebar shows the activated constraints. Each of the activated constraints share the same options to refine them. First, the constraint is reflected by an English sentence, followed by the *Enforce*, *Soften* and *Off* checkboxes. The first two checkboxes behave like HTML radio-buttons, that is, a constraint is either in *Enforce* or in *Soften*, but not in both. For instance, the first activated constraint belongs to **C3** and meets Paul's desire to accomplish 5 to #sup[14] number of modules in the first semester.

[13] Note that we used five semesters simply for the sake of demonstrating the working of our preferences, as explained below.

[14] #sup in *clingo* is supremum which *stands for* ∞.

Per default, *Enforce* is checked when a constraint is activated. *Soften*, when checked, turns the associated hard constraint into a soft constraint. A soft constraint minimizes the *dtb* function value shared among the constraints **C1** to **C4**. This objective function essentially holds the deviation from a given lower and upper bound.

Checkbox *Off*, when checked, deactivates the associated constraint. Thus, a user is able to toggle a particular constraint on and off.

The button with ×, removes a constraint and relaxes the solution space. Similarly, the *Clear all constraints* button removes all active constraints from the user interface and the underlying encoding.

The full user interface encoding can be found in the *user_interface* directory of our repository[15].

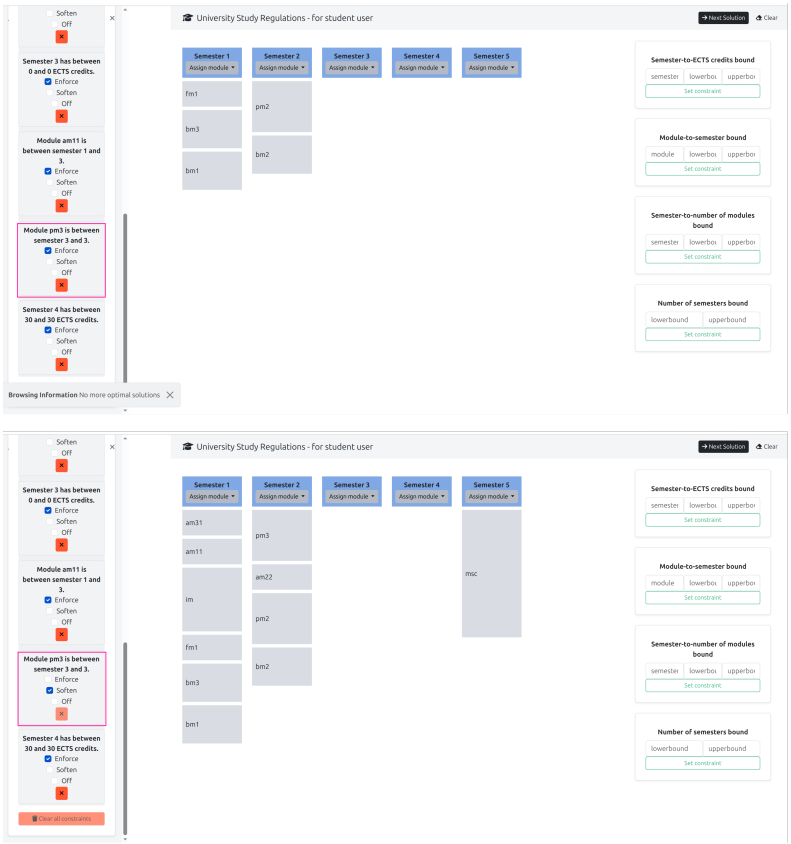

Fig. 2. User interface with violating constraint

[15] `https://github.com/krr-up/study-regulations-modules-constraints-encodings/blob/main/encodings/user_interface` (last accessed: June 2026)

For the sake of illustration, if Paul was to require that the module pm_3 be placed in the third semester, there would be no study plan since it violates his above constraint which says "no module should be in the third semester".

In Figure 2, we see that this is addressed by an info message at the bottom of the **Ac** sidebar which says "Browsing Information No more optimal solutions". However, softening the constraint gives a solution where module pm_3 is close to the third semester.

We observed a solving latency of less than one second with the adding and reasoning with the different constraints[16]. Due to this and since our problem tackles an NP-hard problem with a reasonable small number of modules and semesters, we are not yet concerned with performance, but would evaluate that as part of a usability evaluation in the future.

4 Related Works

Most of the works we encountered in literature are not about university study regulations, rather they deal with actual courses or process-mined event logs which operate on some possible university regulations [3, 17, 1].

However, [9] presented a university study planning tool for students which was based on examination regulations. They provided the user interface for study planning, but no modeling or reasoning engine for the study plans given. [11] presented a university course recommendation framework named *Hybrid Course Recommendation*. It relied on the following scores from a student - *interest-based* score, *timing-based* score and *grade-based* score for making student recommendations using historic data. Though, the solver used was not mentioned in the paper.

Along the same line, [12] presented an interactive university course recommendation system named *CourseQ*. The authors employed, among other things, topic-modeling to recommend courses related to different topic areas using the *Latent Dirichlet Allocation (LDA)* generative probabilistic model. They also reasoned with student interests.

This work is distinguished from those in that it containts student-related module constraints mathematically formalized, to the best of our knowledge, for the first time. This is also the first time *clinguin* is used to implement the presented university study regulations constraint concepts, having the ability to switch the constraints on and off (and back), and from hard to soft (and back).

5 Conclusion and Future Works

We have introduced four student-related hard and soft constraints formalizations, modeling and implementation in Answer Set Programming (ASP). Our

[16] The execution was carried out on a Linux machine equipped with Intel® Core™ i7-6700 × 8 processor and 32 GiB RAM.

approach does not only capture the core of the problem which are the constraints, but also the design of a user interface, thereby showing the versatility of our ASP approach.

The constraints are fed in as ASP facts which are then reasoned within the encodings, and displayed in the user interface. The *clinguin* user interface, which serves as a prototype, allow a user to interact with the constraints switching between hard and soft constraints and resulting in personalized study plans.

For future work, we would like to explore the possibility of reasoning with other kinds of constraints like *liked/disliked modules* and *module keywords*, *shortest study plan* and *dependent modules distance*. Also, we would like to explore how to give *explanations* when no study plan is found. And lastly, perform some usability evaluation with students and explore reasoning from the perspective of *study regulations designers*, for the analysis of study regulations and generation of exemplary study plans.

Acknowledgements. This work was partially funded by the German Federal Ministry of Research, Technology and Space under contract number 16DHBKI024. We also thank Torsten Schaub for supervising the initial developement of this work, Javier Romero for guiding with the preference formalization at the initial stage and Mariia Merzliakova whose *clinguin* study regulations ground work within *CAVAS+* was adopted and extended.

Disclosure of Interests. The authors declare no competing interests.

References

1. C. Behuet et al. “A Rule Language and Feature Model for Educational Timetabling”. In: *14th International Conference on the Practice and Theory of Automated Timetabling*. 2024.
2. A. Beiser, S. Hahn, and T. Schaub. “ASP-driven User-interaction with Clinguin”. In: 2024. URL: https://dx.doi.org/10.4204/EPTCS.416.19.
3. A. Bogarin, R. Cerezo, and C. Romero. “A survey on educational process mining”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.1 (2018), e1230.
4. M. Gebser et al. *Answer set solving in practice*. Cham: Springer International Publishing, 2013. ISBN: 978-3-031-01561-8. URL: https://doi.org/10.1007/978-3-031-01561-8_1.
5. M. Gelfond and V. Lifschitz. “Logic Programs with Classical Negation”. In: pp. 579–597.
6. S. Hahn et al. “Reasoning About Study Regulations in Answer Set Programming”. In: *Theory and Practice of Logic Programming* 24.4 (2024), pp. 790–804. URL: https://doi.org/10.48550/arXiv.2408.04528.
7. HHU. *Studien- und Prüfungsordnungen*. https://www.hhu.de/en/studies/organising-your-studies/examinations/study-and-exam-regulations-german-only. [Accessed 14-12-2025].

8. T. Hirmer, J. Etschmann, and A. Henrich. “Requirements and prototypical implementation of a study planning assistant in CS programs”. In: *2022 International Symposium on Educational Technology (ISET)*. IEEE. 2022, pp. 281–285.
9. S. Judel et al. “Supporting individualized study paths using an interactive study planning tool”. In: Gesellschaft für Informatik eV, 2023.
10. V. Lifschitz. *Answer set programming*. Vol. 3. Springer Heidelberg, 2019. ISBN: 978-3-030-24657-0. URL: https://doi.org/10.1007/978-3-030-24658-7.
11. B. Ma, Y. Taniguchi, and S. Konomi. “Course Recommendation for University Environments.” In: *International educational data mining society* (2020).
12. B. Ma et al. “Exploration and Explanation: An Interactive Course Recommendation System for University Environments.” In: *IUI Workshops*. 2021.
13. M. Ochs et al. “Design-focused development of a course recommender system for digital study planning”. In: *European Conference on Advances in Databases and Information Systems*. Springer. 2023, pp. 575–582.
14. H. Otunuya, S. Lindow, and M. Merzliakova. “Reasoning over Student Study Plans with Answer Set Programming”. In: *INFORMATIK 2025*. Bonn: Gesellschaft für Informatik e.V., 2025, p. 1143. DOI: 10.18420/inf2025_97.
15. A.L. Richards and R. Tsay. “An Optimal Slack-Based Course Scheduling Algorithm for Personalised Study Plans”. In: *Proceedings of the 2020 9th International Conference on Educational and Information Technology*. ICEIT 2020. Oxford, United Kingdom: Association for Computing Machinery, 2020, pp. 1–7. ISBN: 9781450375085. DOI: 10.1145/3383923.3383925. URL: https://doi.org/10.1145/3383923.3383925.
16. TU Berlin. *Study and examination regulations for all degree programs*. https://www.tu.berlin/en/pruefungen/examinations/study-and-examination-regulations/. [Accessed 14-12-2025].
17. M. Wagner et al. “A Combined Approach of Process Mining and Rule-based AI for Study Planning and Monitoring in Higher Education”. In: *Proceedings of the International Conference on Process Mining (ICPM’22): Process Mining Workshops*. Springer-Verlag, 2023, pp. 513–525.

Appendix

Appendix A

```
% -- mi set
in(M,mi) :- map(msl,M,_).
in(M,mi) :- map(msu,M,_).

% -- hard constraint
:- in(M,mi), not in(M,s(_)).

:- in(M,s(I)), map(msl,M,L), I < L, not checked_pref(map(msl,M,L)).
:- in(M,s(I)), map(msu,M,U), I > U, not checked_pref(map(msl,M,U)).

% -- soft : distance to bound L and U (dtb) -
map(dtb,M,D) :- in(M,s(X)), in(M,mi), map(msl,M,L), map(msu,M,U), X < L,
                D = L - X.
map(dtb,M,D) :- in(M,s(X)), in(M,mi), map(msl,M,L), map(msu,M,U), X > U,
                D = X - U.

#minimize{ D,M: map(dtb,M,D), map(msl,M,L), checked_pref(map(msl,M,L)),
                               map(msl,M,U), checked_pref(map(msl,M,U)) }.
```

Listing 1.2. Hard and soft constraint of *module-to-semester bound* constraint in `module_to_semester.lp`.

```
% hard constraint -
% if semester I is in a solution, then the total count of the modules in
% I must be between L and U.
:- in(_,s(I)), map(sml,I,L), map(smu,I,U), not L #count{M : in(M,s(I)) } U.

% soft constraint : distance to L and U bound (dtb) -
map(dtb,I,D) :- in(_,s(I)), map(sml,I,L), X = #count{ M: in(M,s(I)) }, X < L,
                D = L - X, not checked_pref(map(sml,I,L)).
map(dtb,I,D) :- in(_,s(I)), map(smu,I,U), X = #count{ M: in(M,s(I)) }, X > U,
                D = X - U, not checked_pref(map(smu,I,U)).

#minimize{ D,I: map(dtb,I,D), map(sml,I,L), checked_pref(map(sml,I,L)),
                              map(smu,I,U), checked_pref(map(smu,I,U)) }.
```

Listing 1.3. Hard and soft constraint of *semester-to-number of modules bound* constraint in `semester_to_number_of_modules.lp`.

```
% hard constraint
:- map(sl,L), n < L, not checked_pref(map(sl,L)).
:- map(su,U), n > U, not checked_pref(map(su,U)).

% soft constraint : distance to bound L and U (dtb) -
map(dtb,D) :- map(sl,L), map(su,U), n < L, D = L - n.
map(dtb,D) :- map(sl,L), map(su,U), n > U, D = n - U.

#minimize{ D: map(dtb,D), map(sl,L), checked_pref(map(sl,L)),
                          map(su,U), checked_pref(map(su,U)) }.
```

Listing 1.4. Hard and soft constraint of *number of semesters bound* constraint in `number_of_semesters.lp`.

Appendix B

A basic study regulation problem $\mathcal{B}$ is a tuple $(M, G, c, s, l, u, R_g, R_t)$, where

1. M is a set of modules,
2. $G \subseteq 2^M$ distinguishes certain groups of modules,
3. $c : M \to \mathbb{N}$ gives the credits of each module,
4. $s : M \to \{w, s, e\}$ assigns a regular semester to a module,
5. $l : G \to \mathbb{N}$ returns the lower-bound of the credits of a module group,
6. $u : G \to \mathbb{N}$ returns the upper-bound of the credits of a module group,
7. $R_g \subseteq 2^{(2^M)^n}$ is a set of global constraints expressing study regulations, and
8. $R_t \subseteq 2^{(2^M)^n}$ is a set of temporal constraints expressing study regulations.

Functions c and s give the credit points of a module and its semesters turnus, respectively. The elements l and u are partial functions delineating the number of credits obtained per module group; a specific number of credits is captured by an equal lower and upper bound.

The regulation constraints in R_g and R_t are represented extensionally: each constraint r in $R_g \cup R_t$ is represented by the sequences of sets of modules $r \subseteq (2^M)^n$ satisfying it. While the sets of constraints R_g and R_t share the same mathematical structure, they differ in purpose and are therefore separated for clarity's sake. R_t expresses temporal constraints over the sequence of semesters, while R_g does not make use of this sequential structure, and rather expresses global constraints over the entire set of modules.

A study plan for a basic study regulation is a finite sequence of sets of modules satisfying all regulation constraints. More precisely, a sequence $(S_i)_{i=1}^n$ of modules of length n such that $S_i \subseteq M$ for $1 \leq i \leq n$ is a *study plan* for $(M, G, c, s, l, u, R_g, R_t)$ if $(S_i)_{i=1}^n \in \bigcap_{r \in R_g \cup R_t} r$.