

CDL Assistant

Helping students with exam planning

Project for the Fundamentals of Artificial Intelligence course
University of Bari Aldo Moro, Academic Year 2020/2021

Giuseppe Colavito
g.colavito2@studenti.uniba.it
736047

March 28, 2022

Contents

1	Introduction	3
2	Requirements	3
2.1	Prerequisites	3
2.2	Goals	4
2.2.1	Cdl goal	5
2.2.2	Covers goal	5
2.2.3	Cfu goal	6
2.2.4	Suggested Prerequisites	6
2.3	Other minor functionalities	6
3	Conceptualization	6
3.1	Entities and Relationships	7
3.2	Functions	9
3.2.1	Prerequisites	9
3.2.2	Goals	10
4	Inference	11
4.1	Rules structure	11
4.2	Backward	12
4.3	Forward	12
5	Knowledge Base	14
5.1	Structure	14
5.2	Prerequisites	16
5.3	Goals	23
5.3.1	Goal Cdl	23
5.3.2	Goal Covers	25
5.3.3	Goal Cfu	30
5.3.4	Adjust ordering	34
6	Linguistic Module	36
6.1	Intent Handling	36
7	User Interface	40
8	Conclusions and Future Developments	46

1 Introduction

CDL Assistant is an expert system based on symbolic reasoning which aims to help *Informatica*'s students in planning their exams. CDL stands for *Corso di Laurea*, which is the italian for degree course. In a degree course, the various exams can have prerequisites, and it is important to respect them:

- It is hard to understand topics from a teaching which as a prerequisite, without having done prerequisite exams before,
- if you do an exam without doing prerequisite exams, the exam is not valid and you will have to do it again.

CDL Assistant aims to help students in different ways. Students enrolled in a degree course can have different purposes:

- graduate, which means doing all the exams they haven't already done
- learn some topics of their interest, which means doing exams covering that topics,
- obtain a number of credits (*cfu*).

It can also show some informations about teachers, books and topics covered by exams. Cdl Assistant can be used in a command line interface, and answers to some predefined questions in natural language.

2 Requirements

In this section, we identify the main goals of the project, which are an answer to the different purposes a student can have. To do so, we need to understand what exactly is a prerequisite for a cdl exam.

2.1 Prerequisites

Given two exams, T_p and T_t , we can say that T_p is a prerequisite teaching for T_t , if it is not possible to do the T_t exam without doing T_p before. This is a simple example of prerequisite. But it is not the only kind of prerequisite. Some prerequisite might involve more exams. For example, let's take a set of teachings of the first year:

$$T_1^1, T_1^2, \dots, T_1^n$$

Now, let's take a teaching of the third year T_3^x . A possible prerequisite could be that is not possible to do a third year exam T_3^x without doing $T_1^1, T_1^2, \dots, T_1^m$ before, with $m \leq n$. Some prerequisites are based on the *ssd* (Settore Scientifico Disciplinare) code of teachings. There are two more kinds of prerequisites, which emerged reading the Manifesto, which are a *Math prerequisite* and an *Year prerequisite*. So it is important

to have clear naming for all the prerequisites to understand well all the logic behind the system. From now on, we will call:

- **Standard Prerequisites**, prerequisites such as the ones with ssd code above, or defined directly between two teachings.
- **Math Prerequisite**, the prerequisite for which it is necessary to do a given number of math exams in order to be able to do other exams of a given year;
- **Year Prerequisite**, the prerequisite for which it is necessary to do a given number of exams of a given year in order to be able to do other exams of another year;
- **Suggested Prerequisites**, the ones recommended by the teacher, these are the only ones which are not mandatory.

For the *Informatica* cdl, the standard prerequisites are:

*The "Analisi Matematica" exam is prerequisite for "Calcolo Numerico".
The exams "Programmazione" and "Architettura degli Elaboratori e Sistemi Operativi"
are prerequisites for exams of the second year which have ssd code INF/01 and
ING-INF/05.*

*The "Laboratorio di Programmazione" exam is prerequisite for the exams of the second
year second semester which have ssd code INF/01 and ING-INF/05.*

The math prerequisite says:

*If a student does not pass the admission test, in order for him to do a second year
exam, he should pass an exam in the math field*

While the year prerequisite says:

*It is not possible to do a third year exam without having passed six exams taught in the
first year of the course*

*It is not possible to do a second year exam without having passed two first year exams
with ssd code INF/01 or ING-INF/05*

2.2 Goals

At the basis of each goal, there is a common task: finding an ordering of teachings which respects prerequisites, so a student can do exams in that ordering without problems. We identify three goals:

- Cdl goal,
- Covers goal,
- Cfu goal

In addition to the mandatory prerequisites, some teachers also add some suggested prerequisites, which are not mandatory but it is good to follow them to be able to do the exam knowing all the background topics. This leads to another goal, which is to refine an ordering in order to make it compliant with suggested prerequisites.

2.2.1 Cdl goal

The system should be able to find an ordering of teachings which respects prerequisites given:

- cdl name,
- already done exams,
- enrollment year,
- enrollment semester,
- admission test result (passed / not passed).

This means that we need the ordering to contain all the exams of a cdl, in order to be able to graduate following the ordering. The enrollment year and semester are useful because this way a student is able receive an ordering of exams he can actually do (a student cannot do exams with year and semester higher than the ones he is enrolled in). The admission test result is important for the prerequisites of *Informatica* cdl, we will see it in a while.

2.2.2 Covers goal

The system should be able to find an ordering of teaching to learn some topics, still respecting the prerequisites, given:

- the topics to learn,
- cdl name,
- already done exams,
- enrollment year,
- enrollment semester,
- admission test result (passed / not passed).

Following the ordering, a student learns all the topics he wants to, if they are taught in the cdl.

2.2.3 Cfu goal

The system should be able to find an ordering of teaching to obtain a number of cfu, still respecting the prerequisites, given:

- the number of cfu,
- cdl name,
- already done exams,
- enrollment year,
- enrollment semester,
- admission test result (passed / not passed).

Following the ordering, a student should be able to obtain the number of cfu he needs, if they are less than or equal the maximum number of cfu.

2.2.4 Suggested Prerequisites

The system should be able to find an ordering of teaching which respects both the mandatory prerequisites and the suggested prerequisites, given:

- an ordering which respects mandatory prerequisites,
- cdl name.

2.3 Other minor functionalities

The system should be also able to:

- show informations about teachers of a given teaching,
- show topics covered by a given teaching,
- show teachings for a given cdl,
- show suggested books for a given teaching.

3 Conceptualization

In this section we formalize the knowledge about degree courses, identifying the entities of interest to make the system work properly and to satisfy the requirements.

3.1 Entities and Relationships

The **teaching** entity is crucial for our purpose. For a teaching, we need to know:

- name
- number of cfu
- ssd code
- if it is optional

It is important to have the number of cfu since we have a goal based on that. We need the ssd code since some prerequisites are based on that. And we need to know if a teaching is optional, in order to take in consideration only some of them when trying to achieve the cdl goal.

Of course a teaching belongs to a cdl, so we identify the entity **cdl**. For a cdl, we need to know:

- name
- class code
- number of years it takes to attend all the teachings belonging to it
- total number of credits achievable with exams

The class code is useful in case some cdls have the same name. The number of years and the total number of credits are useful to be sure that a goal is reachable (integrity checks).

In this way, we are able to introduce a relationship between a **teaching** and his **cdl**, **taught in**:

- name of teaching
- name of cdl
- year in which the teaching is taught
- semester in which the teaching is taught

A teaching must have a **teacher**:

- teacher name
- teacher email
- teacher phone number

Those information are useful for a student who wants to get in contact with a teacher. We than define a relationship between **teaching** and **teacher**, **taught by**:

- name of the teaching
- name of the teacher
- course of the teaching taught by the teaching

Since there can be different courses of the same teaching, taught by different teachers, we must consider that.

Also a teaching covers some **topics**. For topics, we don't have information other than their name. So we define a relationship between a **teaching** and a **topic**, the **covers** relationship:

- name of the teaching
- name of the topic

We also need the **book** entity:

- book name
- isbn-13
- first author

Isbn and author are useful to be able to find the book, since some books might have the same name.

We then introduce the **suggested for** relationship, between **book** and **teaching**:

- name of the book
- name of the teaching

In order to find an ordering of teachings which respects prerequisites, we need to have some relationships between exams. One of them is the **is prerequisite** relationship:

- prerequisite teaching name
- teaching name

The prerequisite teaching is prerequisite to the other teaching, so it has to be done before it.

We introduce another relationship between teachings, **is prerequisite ssd**:

- prerequisite teaching name
- ssd code
- year
- semester

- name of cdl

So if there are some teachings taught in the given year and semester, with the given ssd code, the prerequisite teaching has to be done before all the exams with that ssd code, taught in that year and semester. From this relation, it is possible to infer some **is prerequisite** relationships between teachings, we'll see the functions to do so in a while.

There are also suggested prerequisites to take in consideration, so we need the **is suggested prerequisite** relationship:

- prerequisite teaching name
- teaching name

The prerequisite teaching is suggested prerequisite to the other teaching, so it recommended to do the prerequisite teaching before it.

3.2 Functions

Here we list all the function which are useful to satisfy requirements.

3.2.1 Prerequisites

is_prerequisite(+PrerequisiteTeaching, +Teaching, +CdlName)

Which returns true if the teaching is taught in an year, semester and has an ssd code such that the prerequisite teaching is prerequisite for the teaching, based on the **is prerequisite ssd** relationship.

respects_prerequisites(+PartialOrdering, +DoneExams, +CdlName)

Which returns true if the partial ordering respects prerequisites given the fact that there are some already done exams for a cdl. Of course this function should take into account all the knowledge about the cdl prerequisites, and should behave differently based on the cdl and its kind of prerequisites. This applies for every function about prerequisites which has the name of the cdl as input, even if in our case, only the implementation for the Informatica degree course is available.

third_year_prerequisite(+DoneExams, +CdlName)

Which returns true if the third year prerequisite is satisfied. (do six exams of the first year).

second_year_prerequisite(+DoneExams, +CdlName)

Which returns true if the second year prerequisite is satisfied. (do two exams of the first year with ssd code INF/01 or ING-INF/05).

math_prerequisite(+PartialOrdering, +DoneExams, +CdlName)

Which returns true if the math year prerequisite is satisfied. (do the admission test or a math exam, so ssd code must start with MAT).

get_standard_prerequisites(+Exam, +DoneExams, CdlName, -StandardPrerequisites)

Given an exam and the exams that have been already done, this should return the set of standard prerequisites exams which are still not satisfied, if there are some.

get_math_prerequisites(+DoneExams, +Year, +CdlName, -MathPrerequisites)

Given the exams that have been already done and the year in which the to-do exam is taught, this should return the set of math exams that can help satisfying the math prerequisite, if it is not satisfied.

get_year_prerequisites(+DoneExams, +Year, +CdlName, -YearPrerequisites)

Given the exams that have been already done and the year in which the to-do exam is taught, this should return the set of exams that can help satisfying the year prerequisite, if it is not satisfied.

pick_prerequisites(+StandardPrerequisites, +MathPrerequisites, +YearPrerequisites, -Prerequisites)

Given the set of exams that can help satisfying each kind of prerequisites, it returns a set of the prerequisite teachings which can help satisfying as many different kinds of prerequisites as possible.

Suppose that every kind of prerequisite is still not satisfied. If doing an exam can help satisfying all the different kind of mandatory prerequisites, namely Standard, Math and Year, than it should be in the output. An exam that can help satisfying only two kinds of mandatory prerequisites should not be in the output if there are some exams which can help satisfying more kind of prerequisites. If all prerequisites are satisfied, it should return an empty set of exams.

set_of_suggested_prerequisites(+Exam, -SuggestedPrerequisites)

Which returns the set of suggested prerequisite for an exam.

3.2.2 Goals

goal_cdl(+CdlName, +DoneExams, +SubYear, +SubSemester, -Ordering)

Which returns an ordering of all exams of a cdl which respects prerequisites, given that there are some already done exams, that the student is enrolled in an year and a semester.

goal_covers(+CdlName, +DoneExams, +Topics, +SubYear, +SubSemester,

-Ordering)

Which returns an ordering of all teachings of a cdl covering a list of topics, given that there are some already done exams, that the student is enrolled in an year and a semester. The ordering must still respect the prerequisites.

goal_cfu(+CdlName, +DoneExams, +SubYear, +SubSemester, +Required-Cfu, -Ordering)

Which returns an ordering of all teachings of a cdl which respects prerequisites and that lets you achieve the required number of cfu, given that there are some already done exams, that the student is enrolled in an year and a semester.

adjust_ordering(+Ordering, +CdlName, -AdjustedOrdering)

Which returns an ordering which respects suggested prerequisites, given an ordering which satisfies mandatory prerequisites and the cdl name

4 Inference

We decided to code the information about the domain in Prolog language. In order to be able to use both backward and forward chaining, an inferential engine based on Prolog has been built.

4.1 Rules structure

To do so, a new rule structure has been defined, together with new operators to make the code more readable.

Operators:

```
:- op(1100, xfx, if).  
:- op(1000, xfy, or).  
:- op(900, xfy, and).
```

Rule structure:

```
% this is a fact  
rule(Fact if true).  
% this is a rule  
rule(Consequence if Condition).  
% rules with and/or operators  
rule(Consequence if Condition1 and Condition2).  
rule(Consequence if Condition1 or Condition2).
```

Of course the operators can be combined in order to have more complex conditions.

4.2 Backward

Now that we have this new formalism of rules, we built some backward chaining rules in Prolog, which are able to reach the selected goals working from the consequence to the conditions:

```
backward(true) :- !.                                     % Goal reached
backward(Fact and Facts) :- !,                           % Conjunction of
    ↪ facts
    backward(Fact),
    backward(Facts).
backward(Fact or Facts) :- !,                             % Disjunction of
    ↪ facts
    ((backward(Fact),!);
    backward(Facts)).
backward(callp(X)) :- !,                                  % Callable fact
    call(X).
backward(Fact) :- !,                                     % Single fact
    rule(Fact if Condition),                             % Match rule
    backward(Condition).                                 % Check condition
```

4.3 Forward

Also a forward chaining rule is implemented, which will work from conditions to consequence, finding every fact that it is possible to infer from the initial knowledge base. Note that a mechanism to control the number of iterations has been added. This is necessary if the set of facts and rules has not a finite fixed point. In our case, having rules involving lists, makes the knowledge base have no finite fixed point: the more you go on iterating, the more facts you will be able to prove. With rules about lists, the facts that can be inferred from the knowledge base are growing exponentially, and not taking care of this makes the forward chaining algorithm loop and crash.

```
forward(Steps, FinalBase):-
    forward_iteration(Steps, nil, [true], FinalBase).
    % Current base is [true]

forward_iteration(Steps, PreviousBase, CurrentBase, FinalBase) :-
    (    ( Steps = 0
        % Reached max iterations
        ; PreviousBase = CurrentBase )
        % Reached a fixed point
    -> FinalBase = CurrentBase
    ;    setof(Fact, derived(Fact, CurrentBase), NewFacts),
```

```

    % Derive new facts
    ord_union(NewFacts, CurrentBase, NewBase),
    % Add new facts into current base
    succ(Steps0, Steps),
    forward_iteration(Steps0, CurrentBase, NewBase, FinalBase) ).
    % Repeat with new base

derived(Fact, Base) :-
    rule(Fact if Condition),
    % Match a rule
    satisfy(Base, Condition).
    % Verify if condition is satisfied given base

satisfy(Base, G and Gs) :-
    % Condition is a conjunction
    !,
    member(G, Base),
    satisfy(Base, Gs).
satisfy(Base, G or Gs) :-
    % Condition is a disjunction
    !,
    ( member(G, Base)
    ; satisfy(Base, Gs) ).
satisfy(_, callp(X)) :-
    % Condition is a callable fact
    !,
    call(X).
satisfy(Base, Condition) :-
    % Condition is an atomic proposition
    member(Condition, Base).

```

The forward chaining algorithm has been tested on a small knowledge base with a finite fixed point, and also on another one without a finite fixed point, and it is fully working. We tested it also with our knowledge base, and decided not to use it. With rules about lists, every combination of possible lists is tried, even lists which will never be useful in real cases (such as lists with repetitions). Another problem is about memory: without checks in every utility rule, in a few iteration the number of inferred facts gets too big to be handled. Instead of adding checks in every rule, making all the code look longer, harder to read and redundant, we preferred to use just the backward chaining algorithm.

5 Knowledge Base

For this project, the focus is on the *Dipartimento di Informatica* degree courses. The system is designed in a way such that adding new cdl's is not problematic, but of course rules about the specific prerequisites of the new cdl's must be added in order to make that work properly. Since it is not possible to retrieve the data automatically, at the moment the system supports only one cdl, *Informatica Triennale*. We made this choice (taking just one degree course) because the data about teachings is not well structured. At first sight, it may seem that the data is structured, but every teacher uses his own convention in writing prerequisites, covered topics and so on about teachings: some are more schematic, others are more discursive. So, the data acquisition process cannot be automatic and is time consuming. With an automatic procedure to extract data, it could have been easier to fetch all the data about more courses and then focus just on coding the prerequisites specified in the Manifesto. We preferred to add more goals and functionalities in a way to show a more complete proof of concept, instead of adding more courses and having a more useless system, but with lots of courses.

5.1 Structure

cdl.pl

Contains information about cdl. Only the Informatica cdl is available. As in the conceptualization, the information about a cdl are his name, his code, the number of years and the total credits achievable with exams.

```
rule(cdl('informatica', 'l-31', 3, 162) if true).
```

teacher.pl

Contains information about teachers, which are name, email and telephone number.

```
rule(teacher('giuseppe pirlo', 'giuseppe.pirlo@uniba.it',  
            '+390805443295') if true).
```

book.pl

Contains information about books and for which teaching are they suggested for. The stored information about book are name, isbn-13 and first author.

```
rule(book('sistemi operativi concetti ed esempi',  
         '978-8891904553', 'silberschatz') if true).  
  
rule(suggested_for('sistemi operativi concetti ed esempi',  
                  'architettura degli elaboratori e sistemi operativi')  
     if true).
```

teaching.pl

Contains information about teachings. For the teaching we have name, cfu, ssd code and optional (true/false). Here are also stored some relations about teachings: taught by, mapping a teaching to his teacher, and taught in, mapping a teaching to the cdl it belongs to.

```
rule(teaching('calcolo numerico', 6, 'mat/08', false) if true).
```

```
rule(taught_by('calcolo numerico', 'a', 'felice iavernaro') if true).
```

```
rule(taught_by('calcolo numerico', 'b', 'alessandro pugliese') if true).
```

```
rule(taught_in('calcolo numerico', 'informatica', 2, 1) if true).
```

covers.pl

Contains information about topics covered by teachings.

```
rule(covers('matematica discreta', 'teoria degli insiemi') if true).
```

```
rule(covers('programmazione', 'algoritmi fondamentali') if true).
```

inference.pl

Contains backward and forward chaining algorithms as in section [Inference](#) section.

utils.pl

Contains some utility rules which are not specific for the domain but are more general, so they are written in Prolog without wrapping them in the new rule structure, and used with:

```
callp(Rule(Param1, Param2, ..., ParamN)).
```

prerequisites.pl

Contains all information about prerequisites, we will go in further details about this in the next section, [Prerequisites](#).

```
rule(is_prerequisite_ssd('programmazione', 'inf/01', 2, 1,  
    ↪ 'informatica') if true).
```

```
rule(is_prerequisite_ssd('programmazione', 'inf/01', 2, 2,  
    ↪ 'informatica') if true).
```

```
rule(is_prerequisite_ssd('laboratorio di informatica',  
    'ing-inf/05', 2, 2, 'informatica') if true).
```

```

rule(is_prerequisite('analisi matematica','calcolo numerico',
                    'informatica') if true).

rule(is_suggested_prerequisite('matematica discreta', 'basi di dati',
                              'informatica') if true).

```

integrity_checks.pl

Contains some checks to do while trying to reach a goal.

```

rule(valid_teaching_list(TeachingList, Cdl) if
    % all elements are teachings
    is_teaching_list(TeachingList) and
    % teachings belong to cdl
    belongs_to_cdl(TeachingList, Cdl) and
    % there are no repetitions
    callp(is_set(TeachingList))
).

```

goal.pl

Contains rules to reach goals, which will be the main focus of section [Goals](#).

5.2 Prerequisites

As in previous sections, we identified four kinds of prerequisites. Let's now talk about the three which are mandatory.

For the standard prerequisite, we have to pass all the exams which are directly a prerequisite for the to-do exam, or via the `ssd` code prerequisite rule:

```

rule(is_prerequisite(PExam, TExam, 'informatica') if
    teaching(PExam, _, _, _) and
    teaching(TExam, _, Category, _) and
    taught_in(TExam, 'informatica', Year, Semester) and
    taught_in(PExam, 'informatica', _, _) and
    is_prerequisite_ssd(PExam, Category, Year, Semester,
                      'informatica') and
    callp(not(PExam = TExam))
).

rule(get_standard_prerequisites(Exam, DoneExams, 'informatica',
                              Prerequisites) if
    % get the prerequisites for Exam
    callp(find_all(TeachingName,
                  is_prerequisite(TeachingName, Exam, 'informatica'),

```



```

        PrerequisiteTeachingNames)) and
% some prerequisites could be already done,
% remove them from the prerequisite list
callp(subtract(PrerequisiteTeachingNames, DoneExams,
        ToFilterPrerequisites)) and
% remove duplicates
callp(list_to_set(ToFilterPrerequisites, Prerequisites))
).

```

In this case, the get rule is used both to check if the prerequisites are satisfied (if the prerequisite list is empty, prerequisites are satisfied) and to retrieve the to-do prerequisite exams for an exam if there are still some.

For the Year prerequisites, we have the third year prerequisite, doing six exams of the first year to be able to attend a third year exam:

```

% third year prerequisite : do 6 exams of the first year
rule(third_year_prerequisite(DoneExams, 'informatica') if
    third_year_prerequisite_no(DoneExams, 'informatica', N) and
    callp(N >= 6)
).

```

```

rule(third_year_prerequisite_no([], 'informatica', 0) if true).
rule(third_year_prerequisite_no([DoneExam|DoneExams], 'informatica',
                                N) if
    teaching(DoneExam, _, _, _) and
    (
        (
            third_year_prerequisite_no(DoneExams, 'informatica',
                                        Ne) and
            taught_in(DoneExam, 'informatica', 1, _) and
            callp(N is Ne + 1)
        )or
        (
            third_year_prerequisite_no(DoneExams, 'informatica', N)
        )
    )
).

```

And the second year prerequisite, doing two exams in the categories INF/01 of ING-INF/05

```

% second year prerequisite : do 2 exams of the categories
% [inf/01, ing-inf/05]

```

```

rule(second_year_prerequisite(DoneExams, 'informatica') if
    second_year_prerequisite_no(DoneExams, 'informatica', N) and
    callp(N >= 2)
).
rule(second_year_prerequisite_no([], 'informatica', 0) if true).
rule(second_year_prerequisite_no([DoneExam|DoneExams], 'informatica',
    N) if
    (
        (
            second_year_prerequisite_no(DoneExams, 'informatica', Ne)
            ↪ and
            teaching(DoneExam, _, Category, _) and
            taught_in(DoneExam, 'informatica', 1, _) and
            (
                callp(Category = 'inf/01') or
                callp(Category = 'ing-inf/05')
            ) and
            callp(N is Ne + 1)
        )
        or
        (
            second_year_prerequisite_no(DoneExams, 'informatica', N)
        )
    )
).

```

For the Math prerequisites, we have to pass a math exam or the admission test. Note that the admission test is treated as an exam, so it is important to add to the done exams the *'admission test'* if it has been passed.

```

% math prerequisite : do a math exam or do admission test
rule(math_prerequisite([DoneExam|DoneExams], 'informatica') if
    teaching(DoneExam, _, Category, _) and
    (
        callp(DoneExam = 'test di ammissione')
        or
        (
            taught_in(DoneExam, 'informatica', 1, _) and
            callp(sub_string(Category, _, _, _, 'mat/'))
        )
    )
    or
    math_prerequisite(DoneExams, 'informatica')

```

```
)
).
```

But just checking the prerequisites is not enough. We also need something to know which exams we should do to satisfy a prerequisite. For the standard prerequisite we already have something like this. For the other kinds of prerequisite, we build a rule to get exams which will help satisfying the prerequisite.

Math:

```
rule(get_math_prerequisites(_, Year, 'informatica',
                             Prerequisites) if
    (
        callp(Year = 1) or
        callp(Year = 3)
    )and
    callp(Prerequisites = []))
).
```

```
rule(get_math_prerequisites(DoneExams, Year, 'informatica',
                             Prerequisites) if
    callp(Year = 2) and
    (
        ( % if math prerequisite is satisfied, no math prerequisites
          math_prerequisite(DoneExams, 'informatica') and
          callp(MathPrerequisites = []))
        )
    or
    (
        % if not take all exams that can help satisfy prerequisite
        callp(find_all(Teaching,
            (
                teaching(Teaching, _, Category, _) and
                taught_in(Teaching, 'informatica', 1, _) and
                callp(sub_string(Category, _, _, _, 'mat/'))
            ), MathPrerequisites))
        )
    )and
    callp(subtract(MathPrerequisites, DoneExams, Prerequisites))
).
```

Year:

```
rule(get_year_prerequisites(_, Year, 'informatica',
```

```

YearPrerequisites) if
callp(Year = 1) and
callp(YearPrerequisites = [])
).

rule(get_year_prerequisites(DoneExams, Year, 'informatica',
Prerequisites) if
callp(Year = 2) and
(
    (
        % if second year prerequisite is satisfied
        second_year_prerequisite(DoneExams, 'informatica') and
        callp(SYPrerequisites = [])
    )
    or
    (
        % if not take all exams that can help satisfy prerequisite
        callp(find_all(Teaching,
            (
                teaching(Teaching, _, Category, _) and
                taught_in(Teaching, 'informatica', 1, _) and
                (
                    callp(Category = 'inf/01') or
                    (
                        callp(\+(Category = 'inf/01')) and
                        callp(Category = 'ing-inf/05')
                    )
                )
            ), SYPrerequisites))
    )
)and
callp(subtract(SYPrerequisites, DoneExams, Prerequisites))
).

rule(get_year_prerequisites(DoneExams, Year, 'informatica',
Prerequisites) if
callp(Year = 3) and
(
    (
        % if third year prerequisite is satisfied
        third_year_prerequisite(DoneExams, 'informatica') and

```

```

        callp(TYPrerequisites = [])
    )
    or
    ( % if not take all exams that can help satisfy prerequisite
      callp(find_all(
          Teaching,
          taught_in(Teaching, 'informatica', 1, _),
          TYPrerequisites))
    )
)and
callp(subtract(TYPrerequisites, DoneExams, Prerequisites))
).
```

The rationale is the same of when checking for the prerequisite, but this time we retrieve the exams with the characteristics we need to satisfy the prerequisites. In the case of third year prerequisites, we need six first year exams, but the list Prerequisites which will make the rule true is a list with all the first year exams. This because all the first year exams can help to satisfy the prerequisite. Of course we will pick the exams from the list one by one and check again if the prerequisite is satisfied every time.

Now we have all the building blocks to build a rule which will select among the prerequisite exams the ones which are involved in the higher number of prerequisites, in order to optimize the ordering.

```

rule(pick_prerequisites(STPrerequisites, MathPrerequisites,
    YearPrerequisites, Prerequisites) if
    callp(intersection(STPrerequisites, MathPrerequisites,
        ST_M_Intersection)) and
    callp(intersection(ST_M_Intersection, YearPrerequisites,
        Intersection)) and
    (
        ( % if there are exams that can satisfy all prerequisites
          % prioritize them
          callp(\+ Intersection = []) and
          callp(Prerequisites = Intersection)
        )
    or
    ( % if there are exams that can satisfy two kind
      % of prerequisites, prioritize them
      callp(intersection(YearPrerequisites, MathPrerequisites,
          Y_M_Intersection)) and
      callp(intersection(STPrerequisites, YearPrerequisites,
          ST_YIntersection)) and
    )
).
```

```

callp(union(Y_M_Intersection, ST_YIntersection,
            PartialUnion)) and
callp(union(PartialUnion, ST_M_Intersection,
            Union)) and
% union contains exam that can help satisfying
% two kind of prerequisites
(
    (
        callp(\+ Union = []) and
        callp(Prerequisites = Union)
    )
or
    ( % else take the exams that can help satisfying
      % one kind of prerequisites
      % or no prerequisites
      callp(append(STPrerequisites, MathPrerequisites,
                  PartialPrerequisites)) and
      callp(append(PartialPrerequisites,
                  YearPrerequisites,
                  Prerequisites))
    )
)
)
).

```

This works by performing multiple intersections and unions in order to understand if there are some exams which take part to more than one kind of prerequisite. If all the sets are disjoint, then the rule will be true with the union of all prerequisites.

The higher level rule, which will be used in the goal rules, is this one:

```

rule(set_of_prioritized_prerequisites(Exam, DoneExams, 'informatica',
                                     Prerequisites) if
    taught_in(Exam, 'informatica', Year, _) and
    get_standard_prerequisites(Exam, DoneExams, 'informatica',
                              STPrerequisites) and
    get_math_prerequisites(DoneExams, Year, 'informatica',
                           MathPrerequisites) and
    get_year_prerequisites(DoneExams, Year, 'informatica',
                           YearPrerequisites) and
    pick_prerequisites(STPrerequisites, MathPrerequisites,
                       YearPrerequisites, Prerequisites)

```

).

We can build a rule upon this one, to understand if prerequisites are satisfied for all exams:

```
rule(respects_prerequisites([OrderedExam], DoneExams, 'informatica') if
    set_of_prioritized_prerequisites(OrderedExam, DoneExams,
                                     'informatica', [])
).

rule(respects_prerequisites([OrderedExam|OrderedExams], DoneExams,
    'informatica') if
    set_of_prioritized_prerequisites(OrderedExam, DoneExams,
                                     'informatica', []) and
    respects_prerequisites(OrderedExams, [OrderedExam|DoneExams],
    'informatica')
).
```

Note that in the done exams list is not important the order of the exams but only the presence.

5.3 Goals

After talking about prerequisite and how to pick which exam to prioritize, we can talk about goal rules.

5.3.1 Goal Cdl

Remember that our cdl goal is to do all the exams in a cdl, with an ordering which respects mandatory prerequisites.

```
rule(goal_cdl(Cdl, DoneExams, SubYear, SubSemester, Ordering) if
    % check integrity of the parameters
    cdl(Cdl, _, NoYears, _) and
    callp(SubYear =< NoYears) and
    callp(SubSemester =< 2) and
    % check that all the teachings are valid and belong to cdl
    valid_teaching_list(DoneExams, Cdl, SubYear, SubSemester) and
    % retrieve all the teachings of a cdl
    cdl_teachings(Cdl, SubYear, SubSemester, Teachings) and
    % remove teachings which are already done
    callp(subtract(Teachings, DoneExams, ToDoExams)) and
    % if necessary, add the optional exams
    add_optional_exams(ToDoExams, Cdl, NoYears, SubYear,
        SubSemester, ToShuffleExams) and
```

```

    % random permute the exams
    callp(random_permutation(ToShuffleExams, ShuffledExams)) and
    % find an ordering which respects prerequisites
    ordering(ShuffledExams, DoneExams, Cdl, Ordering)
).

```

Note that the optional exams are added only if the subscription year is the maximum number of years and the subscription semester is the last. We random permute the exams to have a bit of variability in the output, since in the knowledge base they are ordered by year and semester, and without this steps we would obtain every time the trivial ordering.

One can decide if he wants to specify the subscription year and semester, with this two rules we ensure that if that parameters are not specified, we use default values, which are the maximum number of years and the last semester.

```

rule(goal_cdl(Cdl, DoneExams, Ordering) if
    cdl(Cdl, _, NoYears, _) and
    goal_cdl(Cdl, DoneExams, NoYears, 2, Ordering)
).

```

```

rule(goal_cdl(Cdl, DoneExams, SubYear, Ordering) if
    cdl(Cdl, _, _, _) and
    goal_cdl(Cdl, DoneExams, SubYear, 2, Ordering)
).

```

Of course the core of this goal rule is the ordering rule. The logic behind this rule is the following: if there are some prerequisite exams, randomly pick one of them. The picked exams is set as the first to-do exam.

If there are not prerequisite exams to-do, the exams is put in the done exams and in the ordering.

In both cases, the rule gets than called recursively. The base step is the one in which there are no to-do exams, and the rule is true if the ordering is empty.

```

rule(ordering([], _, _, []) if true).

rule(ordering([ToDoExam|ToDoExams], DoneExams, Cdl,
    Ordering) if
    % check prerequisites
    set_of_prioritized_prerequisites(ToDoExam, DoneExams, Cdl,
        Prerequisites) and
    (
        (
            callp(Prerequisites = []) and

```



```

        % the exam should be put in the ordering
        callp(Ordering = [ToDoExam|RecOrdering]) and
        % recursively call the ordering rule
        ordering(ToDoExams, [ToDoExam|DoneExams], Cdl, RecOrdering)
    ) or
    (
        % randomly pick a prerequisite exam
        pick_an_exam(Prerequisites, Exam) and
        % remove exam from todo list
        callp(subtract(ToDoExams, [Exam],
                        ToDoExamsWithoutNewExam)) and
        % recursively call the ordering rule
        ordering([Exam, ToDoExam|ToDoExamsWithoutNewExam],
                DoneExams, Cdl, Ordering)
    )
).

```

Using the *set_of_prioritized_prerequisites* rule, we are sure to pick the exams which are the most convenient to do to get rid of all the different kind of prerequisites as soon as possible.

5.3.2 Goal Covers

The ordering rule used in this goal is different from the one used for the previous goal. We expect this goal to be focused on a smaller set of exams than the whole set of cdl exams. But we also want the ordering to have a number of exams which is optimal. Since the ordering rule works one exam per time, it tries to satisfy all the prerequisites for the first exam it encounters and it does not take into account all the other to-do exams. In this way, it could happen that in the final ordering, the number of to-do exams is not optimal.

So if there are two third year exams, which also have some standard prerequisites, it could happen that in the ordering there are seven first year exams, instead of six, which is the optimal number.

We found a solution to this problem, but it is better to explain it after seeing the code.

```

rule(goal_covers(Cdl, DoneExams, Topics, Ordering) if
    cdl(Cdl, _, NoYears, _) and
    goal_covers(Cdl, DoneExams, Topics, NoYears, 2, Ordering)
).
rule(goal_covers(Cdl, DoneExams, Topics, SubYear, Ordering) if
    goal_covers(Cdl, DoneExams, Topics, SubYear, 2, Ordering)

```

```

).
rule(goal_covers(Cdl, DoneExams, Topics, SubYear, SubSemester,
                Ordering) if
    % check integrity of the parameters
    cdl(Cdl, _, NoYears, _) and
    callp(SubYear <= NoYears) and
    callp(SubSemester <= 2) and
    valid_teaching_list(DoneExams, Cdl, SubYear, SubSemester) and
    % find all the teachings in the selected year and below
    covers_teachings(Cdl, Topics, SubYear, SubSemester, Teachings) and
    % remove teachings which are already done
    callp(subtract(Teachings, DoneExams, ToDoExams)) and
    % random permute the exams
    callp(random_permutation(ToDoExams, ShuffledExams)) and
    % remove duplicates if some
    callp(list_to_set(ShuffledExams, NewToDoExams)) and
    % find an ordering which respects prerequisites
    covers_ordering(NewToDoExams, DoneExams, Cdl, Ordering)
).

```

These rules rely on the *covers_ordering* rule, which wraps all the logic. In the *goal_covers* rule, we only retrieve the teachings covering the selected topics and call the ordering rule. But let's see what the *covers_ordering* rule does.

```

rule(covers_ordering(ToDoExams, DoneExams, Cdl, Ordering) if
    union_prerequisites(ToDoExams, DoneExams, Cdl, Prerequisites) and
    covers_ordering_(Prerequisites, ToDoExams, DoneExams, Cdl, Ordering)
).

```

```

rule(covers_ordering_(Prerequisites, ToDoExams, DoneExams, Cdl,
                    Ordering) if
    split_exams(ToDoExams, DoneExams, Cdl, NoPrereqExams, _) and
    % all exams have prerequisites to do
    callp(NoPrereqExams = []) and
    % pick the exam which has the most standard prerequisites
    pick_multiple_priority_exam(Prerequisites, DoneExams, Cdl,
                                ToDoPrerequisite) and
    % do exam if possible
    set_of_prioritized_prerequisites(ToDoPrerequisite, DoneExams, Cdl,
                                    PrioritizedPrerequisites) and
    (
        (

```

```

    callp(PrioritizedPrerequisites = []) and
    % update prerequisites
    union_prerequisites(ToDoExams,
                        [ToDoPrerequisite | DoneExams],
                        Cdl, NewPrerequisites) and
    % call recursively
    callp(Ordering = [ToDoPrerequisite | PartialOrdering]) and
    covers_ordering_(NewPrerequisites, ToDoExams,
                    [ToDoPrerequisite | DoneExams],
                    Cdl, PartialOrdering)
) or
(
    covers_ordering_(PrioritizedPrerequisites, ToDoExams,
                    DoneExams, Cdl, Ordering)
)
).
rule(covers_ordering_(Prerequisites, ToDoExams, DoneExams, Cdl,
                    Ordering) if
    split_exams(ToDoExams, DoneExams, Cdl, NoPrereqExams, _) and
    % if some have no prerequisite, set as done
    callp(\+ NoPrereqExams = []) and
    callp(subtract(ToDoExams, NoPrereqExams, NewToDoExams)) and
    callp(append(DoneExams, NoPrereqExams, NewDoneExams)) and
    covers_ordering_(Prerequisites, NewToDoExams, NewDoneExams,
                    Cdl, PartialOrdering) and
    callp(append(NoPrereqExams, PartialOrdering, Ordering))
).
rule(covers_ordering_(_, [], _, _, []) if true).

```

As we can see, this goal rule is designed to do the exams which cover the selected topics as soon as the prerequisites are satisfied. At every iteration, if an exam has its prerequisites satisfied, it is set as done. If there are not exams which have their prerequisites satisfied, then we try to do some prerequisite exams. Note that the prerequisite of the to-do exams are treated as an union. So the goal is not to do a single exam. At each step, we update the prerequisite of each to do exam and merge them in an unique set. How do we choose the prerequisite exam to do? We pick the exam which helps satisfying the more prerequisite for every to-do exam. This happens using the rule *pick_multiple_priority_exam*. This way we are sure to have the optimal number of exams in the ordering.

% this aims to pick the exam which satisfy more prerequisites

```

rule(pick_multiple_priority_exam(SetExams, CompareExams, DoneExams,
                                'informatica', Exam) if
    count_multiple_prerequisites(SetExams, CompareExams, DoneExams,
                                'informatica', Lens) and
    callp(min_1(CompareExams, Lens, Exam, _))
).

rule(count_multiple_prerequisites(SetExams, [CompareExam|CompareExams],
                                DoneExams, 'informatica',
                                [Count|Counts]) if
    count_prerequisites_list(SetExams, [CompareExam|DoneExams],
                              'informatica', Len) and
    callp(sum_list(Len, Count)) and
    count_multiple_prerequisites(SetExams, CompareExams, DoneExams,
                              'informatica', Counts)
).

rule(count_multiple_prerequisites(_, [], _, _, []) if true).

rule(count_prerequisites_list([Exam|Exams], DoneExams, 'informatica',
                              [Len|Lens]) if
    count_prerequisites(Exam, DoneExams, 'informatica', Len) and
    count_prerequisites_list(Exams, DoneExams, 'informatica', Lens)
).

rule(count_prerequisites_list([], _, _, []) if true).

rule(count_prerequisites(Exam, DoneExams, 'informatica', Len) if
    taught_in(Exam, 'informatica', Year, _) and
    get_standard_prerequisites(Exam, DoneExams, 'informatica',
                               STPrerequisites) and
    get_math_prerequisites(DoneExams, Year, 'informatica',
                           MathPrerequisites) and
    get_year_prerequisites(DoneExams, Year, 'informatica',
                           YearPrerequisites) and
    callp(length(STPrerequisites, L1)) and
    callp(length(MathPrerequisites, L2)) and
    callp(length(YearPrerequisites, L3)) and
    callp(Len is L1 + L2 + L3)
).

min_1([Y],[X],Y,X) :- !.

```

```

min_l([_|Ys], [X|Xs], My, Mx):-
    min_l(Ys, Xs, My, Mx),
    Mx < X, !.
min_l([Y|Ys], [X|Xs], Y, X):-
    min_l(Ys, Xs, _, Mx),
    X =< Mx, !.

```

To better understand the logic, see Figure 1. For each to-do exam, we evaluate how

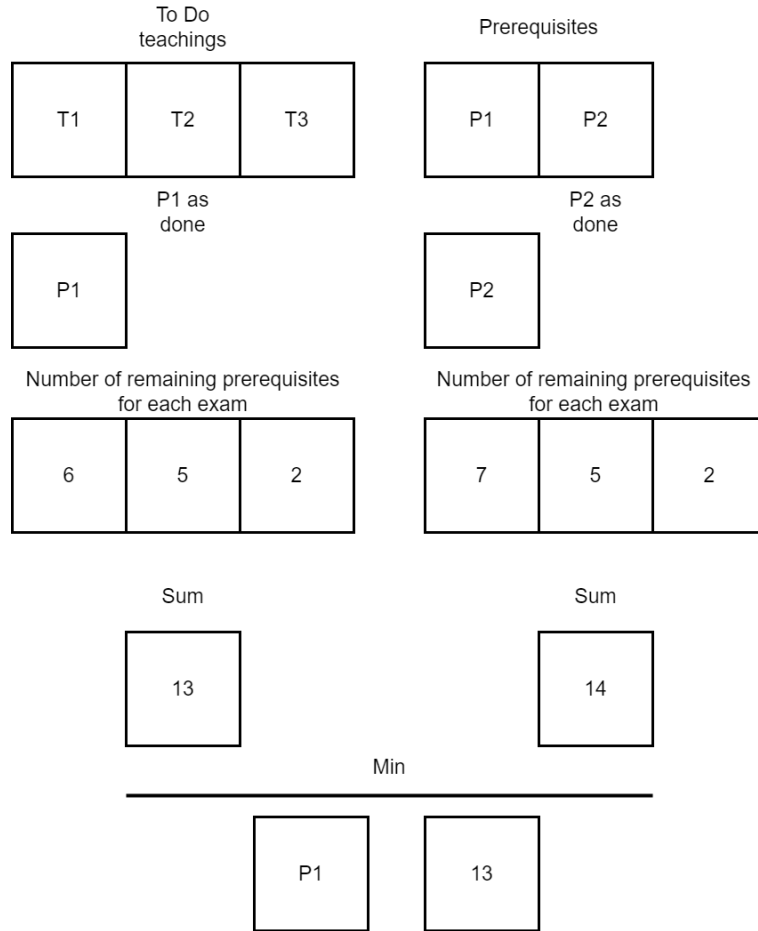


Figure 1: How to select the prerequisite exam to do in the covers goal

many to-do prerequisites remain if setting one of the prerequisites as done. For example, we set *P1* as done, evaluate the remaining prerequisites for each exam and sum them to have a unique number for each prerequisite. We do the same for *P2*, and then pick the exam with the lower sum, meaning that it satisfies more prerequisites for the union of the to-do exams.

The *min_l* rule is a general rule about list, so it is written in Prolog without the new rule formalism. It aims to find the minimum in a list which is mapped with another lists.

It is used to retrieve the exam which lowers the number of remaining prerequisites. We use two lists to make a mapping, where the position matters.

5.3.3 Goal Cfu

We initially tried to achieve our cfu goal sampling an exam based on its number of cfu: higher number of cfu of an exam, higher the probability of that exam to be sampled. This way we aimed to prioritize exams with an higher number of cfu, to achieve the highest number of cfu as fast as possible, having also a bit of variability in the ordering. But this way, it is easy to exceed the number of cfu and get an ordering which has more cfu than requested. Assuming that our user is a lazy student, and he wants to achieve just the number of cfu he requested, the solutions proposed by this approach are not always acceptable. If there is a possible ordering to achieve the right number of cfu, it has to be found. But there are a lot of constraint to satisfy: we have to satisfy prerequisites (which is no more a problem) but also to understand if it is possible to achieve the right number of cfu. To do so, we use the rule *goal_cfu_backtrack_*:

```
rule(goal_cfu_backtrack_(PartialOrdering, RequiredCfu, SubYear,
                        SubSemester, Cdl, Ordering) if
    taught_in(Exam, Cdl, _, _) and
    teaching(Exam, _, _, _) and
    callp(\+ member(Exam, PartialOrdering)) and
    callp(append(PartialOrdering, [Exam], NewOrdering)) and
    valid_teaching_list(NewOrdering, Cdl, SubYear, SubSemester) and
    respects_prerequisites([Exam], PartialOrdering, Cdl) and
    sum_cfu(NewOrdering, CfuSum) and
    callp(CfuSum < RequiredCfu) and
    goal_cfu_backtrack_(NewOrdering, RequiredCfu, SubYear,
                        SubSemester, Cdl, Ordering)
).
```

```
rule(goal_cfu_backtrack_(PartialOrdering, RequiredCfu, SubYear,
                        SubSemester, Cdl, Ordering) if
    taught_in(Exam, Cdl, _, _) and
    teaching(Exam, _, _, _) and
    callp(\+ member(Exam, PartialOrdering)) and
    callp(append(PartialOrdering, [Exam], Ordering)) and
    valid_teaching_list(NewOrdering, Cdl, SubYear, SubSemester) and
    respects_prerequisites([Exam], PartialOrdering, Cdl) and
    sum_cfu(Ordering, CfuSum) and
    callp(CfuSum = RequiredCfu)
).
```

This rule will be true for every ordering which satisfies both prerequisites and total number of cfu. But what if a number of cfu is not achievable? For example, there is no combination of exams of 3,6,9,12 cfu which sums to 22. To do so, we can substitute the equal sign with a greater or equal in the second rule. But in that case, we have to take the result which has the minimum number of total cfu.

```
rule(goal_cfu_backtrack(RequiredCfu, SubYear, SubSemester, Cdl,
                        Ordering) if
    callp(find_all(AnOrdering,
                    (goal_cfu_backtrack([], RequiredCfu, SubYear,
                                         SubSemester, Cdl, AnOrdering)),
                    Orderings)) and
    callp(random_permutation(Orderings, ShuffledOrderings)) and
    sum_multiple_cfu(ShuffledOrderings, CfuSum) and
    callp(min_1(ShuffledOrderings, CfuSum, Ordering, _))
).
```

This rule aims to do so, but applying it with a goal of a number higher than 20-30 cfu, will take a lot of time to get an answer. Trying every combination of exams and every time also check for the prerequisites it's computational expensive. And we are also counting all the cfu for each ordering.

This approach is correct and using it we are sure we'll get the optimal ordering, after some time. Since it does not scale for a big number of cfu, we decided to keep also the implementation based on sampling, which is of course naive but can retrieve an answer fast also for a big number of cfu.

Code for sampling

```
% generate a list of probabilities for each exam, which is
% cfu/total cfus
rule(cfu_to_probability([Exam|Exams], Cdl, [P|Ps]) if
    teaching(Exam, Cfu, _, _) and
    taught_in(Exam, Cdl, _, _) and
    cdl(Cdl, _, _, TotalExamsCfu) and
    callp(P is (Cfu/TotalExamsCfu)) and
    cfu_to_probability(Exams, Cdl, Ps)
).

rule(cfu_to_probability([], _, []) if true).

rule(cfu_to_probability([Exam|Exams], Cdl, TotalExamsCfu, [P|Ps]) if
    teaching(Exam, Cfu, _, _) and
    taught_in(Exam, Cdl, _, _) and
    callp(P is (Cfu/TotalExamsCfu)) and
```

```

        cfu_to_probability(Exams, Cdl, TotalExamsCfu, Ps)
    ).
rule(cfu_to_probability([], _, _, []) if true).

rule(sample_exam(ToDoExams, Cdl, Exam) if
    % sum the cfus in ToDoExams
    sum_cfu(ToDoExams, CfuSum) and
    % divide cfu by the sum
    cfu_to_probability(ToDoExams, Cdl, CfuSum, Probability) and
    % sample an exam
    callp(sample(ToDoExams, Probability, Exam))
).

sample([X], [P], Cumulative, Rand, X) :-
    Rand < Cumulative + P, !.
sample([X|_], [P|_], Cumulative, Rand, X) :-
    Rand < Cumulative + P, !.
sample([_|Xs], [P|Ps], Cumulative, Rand, Y) :-
    Cumulative1 is Cumulative + P,
    Rand >= Cumulative1,
    sample(Xs, Ps, Cumulative1, Rand, Y).

```

Code for goal

```

rule(goal_cfu(Cdl, DoneExams, RequiredCfu, Ordering) if
    cdl(Cdl, _, NoYears, _) and
    goal_cfu(Cdl, DoneExams, NoYears, 2, RequiredCfu, Ordering)
).

rule(goal_cfu(Cdl, DoneExams, SubYear, RequiredCfu, Ordering) if
    cdl(Cdl, _, _, _) and
    goal_cfu(Cdl, DoneExams, SubYear, 2, RequiredCfu, Ordering)
).

rule(goal_cfu(Cdl, DoneExams, SubYear, SubSemester, RequiredCfu,
    Ordering) if
    % check integrity of the parameters
    cdl(Cdl, _, NoYears, _) and
    callp(SubYear <= NoYears) and
    callp(SubSemester <= 2) and
    valid_teaching_list(DoneExams, Cdl, SubYear, SubSemester) and
    valid_cfu_number(DoneExams, Cdl, RequiredCfu) and

```



```

cdl_teachings(Cdl, SubYear, SubSemester, Teachings) and
% remove teachings which are already done
callp(subtract(Teachings, DoneExams, ToDoExams)) and
add_optional_exams(ToDoExams, Cdl, NoYears,
                   SubYear, SubSemester, ToShuffleExams) and
% random permute the exams
callp(random_permutation(ToShuffleExams, ShuffledExams)) and
% find an ordering which respects prerequisites
cfu_ordering(ShuffledExams, DoneExams, RequiredCfu, Cdl, Ordering)
).

% sample an exam and call cfu_ordering_
rule(cfu_ordering(ToDoExams, DoneExams, RequiredCfu, Cdl,
                  Ordering) if
    callp(\+ (ToDoExams = [])) and
    % sample an exam
    sample_exam(ToDoExams, Cdl, ToDoExam) and
    % remove it from ToDoExams
    callp(subtract(ToDoExams, [ToDoExam],
                  ToDoExamsWithoutNewExam)) and
    % set sampled exam as first exam to do
    cfu_ordering_([ToDoExam|ToDoExamsWithoutNewExam], DoneExams,
                  RequiredCfu, Cdl, Ordering)
).

% case prerequisites are not satisfied
rule(cfu_ordering_([ToDoExam|ToDoExams], DoneExams, RequiredCfu,
                  Cdl, Ordering) if
    % check prerequisites
    set_of_prioritized_prerequisites(ToDoExam, DoneExams,
                                     Cdl, Prerequisites) and
    % case prerequisites are not satisfied
    callp(\+Prerequisites = []) and
    % pick an exam from the ones which can help satisfying prerequisites
    pick_an_exam(Prerequisites, PrerequisiteExam) and
    % remove exam from todo list to avoid duplicates
    callp(subtract(ToDoExams, [PrerequisiteExam],
                  ToDoExamsWithoutNewPrerequisite)) and

```

```

% recursively call the ordering rule
cfu_ordering(
    [PrerequisiteExam, ToDoExam|ToDoExamsWithoutNewPrerequisite],
    DoneExams, RequiredCfu, Cdl, Ordering)
).

```

5.3.4 Adjust ordering

We also need to take into account suggested prerequisites, which are of course non mandatory, but a wise student might decide to follow them in order to have a full background knowledge to do all the exams. To do so, we decided to start from an ordering which already satisfies prerequisites and can be found with the other goals, and to refine it making it compliant also with the suggested prerequisite. In this way, for each goal we can have the correspondent ordering which respects all prerequisites, mandatory or not. We build a rule which makes the ordering respect the suggested prerequisites of the first exam which has some of them, and then try to find a fixed point with that rule. If an exam has some suggested prerequisites which are already done before that exam in the ordering, then the ordering is already correct. If not, we try to bring the prerequisite exams before the selected exam.

```

% this is true if the adjusted ordering respects suggested prerequisites
rule(adjust_ordering(Ordering, Cdl, AdjustedOrdering) if
    adjust_ordering_(Ordering, [], FirstAdjustedOrdering) and
    (
        (
            % if it is a fixed point, no need to adjust
            callp(Ordering = FirstAdjustedOrdering) and
            callp(AdjustedOrdering = FirstAdjustedOrdering)
        )
        or
        (
            % if it is not a fixed point, call recursively
            adjust_ordering(FirstAdjustedOrdering, Cdl,
                AdjustedOrdering) and
            % ensure that the adjusted ordering still
            % respects prerequisites
            respects_prerequisites(AdjustedOrdering, [], Cdl)
        )
        or
        (
            % if it doesn't respect the prerequisites,
            % adjusting is breaking something.

```

```

        callp(AdjustedOrdering = Ordering)
        % this is not happening, but is here for completeness
    )
)
).

% this is true if the adjusted ordering respects suggested
% prerequisites of the first exam in the ordering which has some
rule(adjust_ordering_([OrderedExam|OrderedExams], BeforeExams,
    AdjustedOrdering) if
set_of_suggested_prerequisites(OrderedExam,
    SuggestedPrerequisites) and
(
    (
        % if it is a subset, suggested prerequisites
        % have been already done
        callp(subset(SuggestedPrerequisites, BeforeExams)) and
        % set the first exam as checked
        callp(append(BeforeExams, [OrderedExam],
            BeforeExamsWithOrderedExam)) and
        % recursively call the adjusting rule
        adjust_ordering_(OrderedExams, BeforeExamsWithOrderedExam,
            AdjustedOrdering)
    )
    or
    (
        % if it is not a subset, some suggested prerequisites
        % have not been done yet
        % remove the ones already done
        callp(subtract(SuggestedPrerequisites, BeforeExams,
            RemainingPrerequisites)) and
        % remove the suggested prerequisites from the exams
        % in the ordering to avoid duplicates
        callp(subtract(OrderedExams, RemainingPrerequisites,
            OrderedExamsWithoutPrerequisites)) and
        % set the suggested prerequisites as first in the ordering
        callp(append(BeforeExams, RemainingPrerequisites,
            BeforeExamsWithPrerequisites)) and
        callp(append(BeforeExamsWithPrerequisites,
            [OrderedExam|OrderedExamsWithoutPrerequisites],
            AdjustedOrdering))
    )
)

```

```

    )
).
rule(adjust_ordering_([], BeforeExams, _, BeforeExams) if true).

```

Now that we have all the rules that we need to achieve the goals, we have a working system. We try in the next sections to have a more simple interface based on some natural language queries to get the intent of the user and query SWI-Prolog for him.

6 Linguistic Module

In order to be able to recognize intents in some natural language queries, we decide to use *Python* with *PySwip*, a *SWI-Prolog* bridge enabling library to query SWI-Prolog in Python programs.

6.1 Intent Handling

We define this set of intents:

- **help**, which will trigger an explanation on what the system can do,
- **hello**, to say hi to the user,
- **thanks**, to answer when an user says thanks to the system,
- **cdl_ordering**, to handle the cdl goal,
- **cfu_ordering**, to handle the cfu goal,
- **covers_ordering**, to handle the covers goal,
- **suggested_books**, to show the list of suggested books for a teaching,
- **table_teachings**, to show the list of teachings in a cdl,
- **teacher_information**, to show the information about teachers of a teaching,
- **covered_topics**, to show the covered topics for a teaching,
- **goodbye**, to interrupt the communication with the system

For intents regarding goals or general information, we have a prolog query associated to the intent, and we have to get the query parameters by the user. For each intent we define some regular expression to trigger them, and some template answer. In the *dialogue* folder of the project we can find the json files in which all this data is stored. It is easy to add answers and intent expressions. To add intents, we also have to define its behaviour (a prolog query). The parameters which are not specified by the user, are than asked by the system to complete the set of parameters, run the query and show

the results. Once the system answers, it waits for the next intent, unless you trigger the *goodbye* intent. The *intent.json* is structured in this way:

- Each entry has a tag (which is the intent name), a list of patterns to trigger the intent, the list of parameters that can be captured by that set of patterns.
- An intent can have more entries, based on the parameters that can be captured by the regular expressions. For example, a set of patterns for the cdl goal is able to get the information about the cdl name. So, for that entry, the list of parameters will contain "*cdl_name*". Another set of parameters is more general and is not able to detect the name of the cdl, so it will be then asked to the user.

Here is the json file containg regular expressions for each intent.

```
{
  "intents" : [
    {
      "tag" : "help",
      "patterns" : [
        "^help$",
        "what can you do?\\?",
        "what can you do for me?\\?",
        "how can you help me?\\?"
      ],
      "params" : []
    },
    {
      "tag" : "hello",
      "patterns" : [
        "^hello$",
        "^hi$",
        "^hey$",
        "^howdy$"
      ],
      "params" : []
    },
    {
      "tag" : "thanks",
      "patterns" : [
        "thanks",
        "thank you",
        "thank you very much",
        "thank you so much",
        "tysm"
      ],
      "params" : []
    },
    {
      "tag" : "cdl_ordering",
      "patterns" : [
        "(?:can you(?:please)?|will you(?:please)?)?suggest(?:me)?an exam ordering(?:for|in) a cdl ?\\??",
        "params" : []
      ],
    },
    {
      "tag" : "cdl_ordering",
      "patterns" : [
        "(?:can you(?:please)?|will you(?:please)?)?suggest(?:me)?an exam ordering(?:for|in) ([a-zA-Z ]+)?year [1-3] semester [1-2] ?\\??",
        "params" : [
          "cdl_name",
          "year",
          "semester"
        ]
      ],
    },
    {
      "tag" : "cdl_ordering",
      "patterns" : [
        "(?:can you(?:please)?|will you(?:please)?)?suggest(?:me)?an exam ordering(?:for|in) ([a-zA-Z ]+)? ?\\??",
        "params" : [
          "cdl_name",
          "year",
          "semester"
        ]
      ],
    }
  ]
}
```

```

    "params" : ["cdl_name"]},

{"tag" : "cfu_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering to
    ↪ (?:acquire|gain|get|achieve) ([0-9]+) cfu (?:for|in) a cdl
    ↪ ?\\??"],
 "params" : ["no_cfu"]},

{"tag" : "cfu_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering to
    ↪ (?:acquire|gain|get|achieve) (?:some|a number|many) cfu
    ↪ (?:for|in) a cdl ?\\??"],
 "params" : []},

{"tag" : "cfu_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering to
    ↪ (?:acquire|gain|get|achieve) ([0-9]+) cfu (?:for|in) ([a-zA-Z
    ↪ ]+) ?\\??"],
 "params" : ["no_cfu","cdl_name"]},

{"tag" : "covers_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering (?:to learn |to cover |for
    ↪ learning )some topics (?:for|in) a cdl ?\\??"],
 "params" : []},

{"tag" : "covers_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering (?:to learn |to cover |for
    ↪ learning )some topics for ([a-zA-Z ]+) ?\\??"],
 "params" : ["cdl_name"]},

{"tag" : "covers_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
    ↪ )?)?suggest (?:me )?an exam ordering to learn ([a-zA-Z
    ↪ ]+(?:,[a-zA-Z ]+)* ) for a cdl ?\\??"],
 "params" : ["topics"]},

```

```

{"tag" : "covers_ordering",
 "patterns" : ["(?:can you (?:please )?|will you (?:please
 ↪  )?)?suggest (?:me )?an exam ordering to learn ([a-zA-Z
 ↪  ]+(?:,[a-zA-Z ]+)* ) for ([a-zA-Z ]+) ?\\{?"",
 "params" : ["topics", "cdl_name"]},

{"tag" : "suggested_books",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?suggested books for a teaching ?\\{?"",
 "params" : []},

{"tag" : "suggested_books",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?suggested books for ([a-zA-Z ]+) ?\\{?"",
 "params" : ["teaching_name"]},

{"tag" : "table_teachings",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?(?:the )?teachings for a cdl ?\\{?"",
 "params" : []},

{"tag" : "table_teachings",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?(?:the )?teachings for ([a-zA-Z ]+) ?\\{?"",
 "params" : ["cdl_name"]},

{"tag" : "teacher_information",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?teachers for a teaching ?\\{?"",
 "params" : []},

{"tag" : "teacher_information",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?teachers for ([a-zA-Z ]+) ?\\{?"",
 "params" : ["teacher_name"]},

{"tag" : "covered_topics",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
 ↪  (?:me )?covered topics for a teaching ?\\{?"",

```

```

    "params" : []},

{"tag" : "covered_topics",
 "patterns" : ["(?:can you (?:please )?|will you (?:please )?)?show
↪  (?:me )?covered topics for ([a-zA-Z ]+) ?\\{?"],
 "params" : ["teaching_name"]},

{"tag" : "goodbye",
 "patterns" : ["(no(?:,)?(?: )?)?you can't", "(no(?:,)?(?: )?)?i
↪  don't need help", "(no(?:,)?(?: )?)?i dont need help",
↪  "(no(?:,)?(?: )?)?i'm ok", "goodbye", "bye", "see you",
↪  "^no$"],
 "params" : []}
]}

```

7 User Interface

For this system we built a command line interface. To show results with a nice formatting, we used the *Rich* library. We show some screenshots of the system while in use.

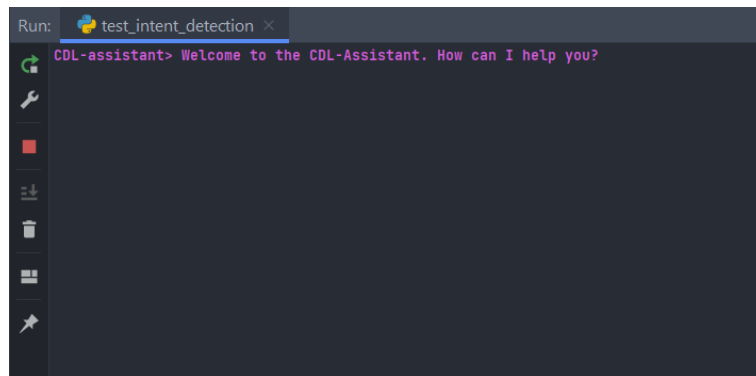


Figure 2: Welcome message


```
Run: test_intent_detection X
CDL-assistant> Welcome to the CDL-Assistant. How can I help you?
help
CDL-assistant> You can ask me for:
-Exam ordering for a cdl
-Exam ordering which covers some topics of your choice
-Exam ordering to achieve a given number of cfu
-Courses for a cdl
-Teacher information
-Suggested books for an exam
CDL-assistant> Can I help you with anything else?
```

Figure 3: Help message

```
Run: test_intent_detection X
CDL-assistant> Can I help you with anything else?
can you show covered topics for ingegneria della conoscenza?
Ingegneria Della Conoscenza's covered topics
```

Topic
Sistemi intelligenti basati su conoscenza
Rappresentazione della conoscenza
Logica proposizionale
Ragionamento automatico
Deduzione
Abduzione
Induzione
Ragionamento con incertezza
Acquisizione della conoscenza
Modelli di classificazione
Modelli probabilistici

```
CDL-assistant> Can I help you with anything else?
```

Figure 4: Covered topics for a teaching

Run: test_intent_detection ×

will you show me teachings for informatica?

Informatica's teachings

Name	Year	Semester	Cfu	Ssd	Optional
Architettura Degli Elaboratori E Sistemi Operativi	1	1	9	ING-INF/05	False
Matematica Discreta	1	1	9	MAT/03	False
Programmazione	1	1	12	INF/01	False
Analisi Matematica	1	2	9	MAT/05	False
Linguaggi Di Programmazione	1	2	9	INF/01	False
Laboratorio Di Informatica	1	2	6	INF/01	False
Lingua Inglese	1	2	6	L-LIN/12	False
Algoritmi E Strutture Dati	2	1	9	INF/01	False
Basi Di Dati	2	1	9	INF/01	False
Calcolo Numerico	2	1	6	MAT/08	False
Fondamenti Di Fisica	2	1	6	FIS/07	False
Ingegneria Del Software	2	2	9	INF/01	False
Metodi Avanzati Di Programmazione	2	2	9	ING-INF/05	False
Calcolo Delle Probabilità E Statistica	2	2	6	MAT/06	False
Calcolabilità E Complessità	2	2	6	INF/01	False
Reti Di Calcolatori	3	1	9	ING-INF/05	False

Figure 5: Teachings for a cdl

```
Run: test_intent_detection ×
CDL-assistant> Welcome to the CDL-Assistant. How can I help you?
will you please suggest me an exam ordering for informatica?
CDL-assistant> What are the names of the exams you have taken?
Please enter the names of the exam you have taken, separated by commas. If you haven't taken any exam, leave this blank.
Programmazione, Matematica Discreta
CDL-assistant> What year are you in?
Enter an integer. For example, if you are enrolled in the first year, please enter 1.
3
CDL-assistant> Which semester are you enrolled in?
Enter an integer. For example, if you are enrolled in the second semester, please enter 2.
2
CDL-assistant> Did you pass the admission test?
no
CDL-assistant> Do you want to respect the suggested prerequisites?
sure
```

Figure 6: Parameter asking for a goal

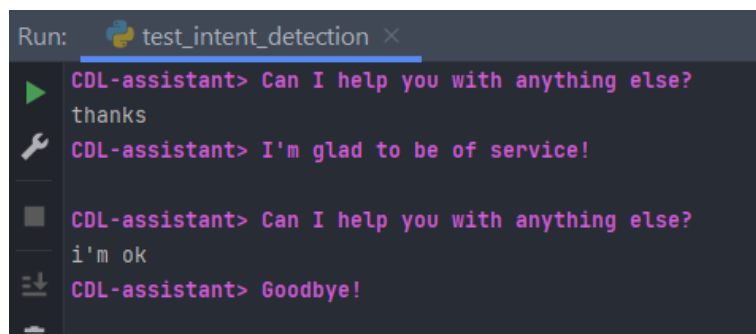
Run: test_intent_detection ×

Informatica's teachings ordering, having done ['matematica discreta', 'programmazione']

Order	Name	Year	Semester	Cfu	Ssd	Optional
1	Linguaggi Di Programmazione	1	2	9	INF/01	False
2	Analisi Matematica	1	2	9	MAT/05	False
3	Lingua Inglese	1	2	6	L-LIN/12	False
4	Laboratorio Di Informatica	1	2	6	INF/01	False
5	Modelli E Metodi Per La Sicurezza Delle Applicazioni	3	2	6	INF/01	True
6	Interazione Uomo-Macchina	3	1	6	INF/01	False
7	Architettura Degli Elaboratori E Sistemi Operativi	1	1	9	ING-INF/05	False
8	Reti Di Calcolatori	3	1	9	ING-INF/05	False
9	Calcolo Delle Probabilità E Statistica	2	2	6	MAT/06	False
10	Algoritmi E Strutture Dati	2	1	9	INF/01	False
11	Metodi Per Il Ritrovamento Dell'Informazione	3	1	9	ING-INF/05	False
12	Ingegneria Del Software	2	2	9	INF/01	False
13	Ingegneria Della Conoscenza	3	1	6	ING-INF/05	False
14	Calcolabilità E Complessità	2	2	6	INF/01	False
15	Sviluppo Di Videogiochi	3	2	6	INF/01	True
16	Calcolo Numerico	2	1	6	MAT/08	False
17	Basi Di Dati	2	1	9	INF/01	False
18	Fondamenti Di Fisica	2	1	6	FIS/07	False
19	Metodi Avanzati Di Programmazione	2	2	9	ING-INF/05	False

CDL-assistant> Can I help you with anything else?

Figure 7: Result of goal cdl



A terminal window titled "Run: test_intent_detection" with a close button. The window contains a series of interactions between a user and a CDL-assistant. The assistant's prompts are in pink, and the user's responses are in white. The interactions are as follows:

```
CDL-assistant> Can I help you with anything else?  
thanks  
CDL-assistant> I'm glad to be of service!  
CDL-assistant> Can I help you with anything else?  
i'm ok  
CDL-assistant> Goodbye!
```

Figure 8: Thanks and goodbye intents

8 Conclusions and Future Developments

We developed a system which is able to find valid orderings for the teachings of a cdl, knowing the prerequisites. The knowledge base has only a cdl, but of course in future it is easily possible to add more cdls, also from other departments or universities. To this aim, there is the need of having the information about cdls and teachings well structured, in order to be able to scrape all the data automatically.

Actually the system relies on regular expressions to recognize intents and answer. A future development could involve the use of intent detection through machine learning, using tools such as *DialogFlow* by Google.

It would be useful, especially for the covers goal, to have an explanation on why the exams are added in the ordering:

- A teaching is added because it covers a topic selected by the student
- Or because it is prerequisite to another teaching

It would be also nice to have an explanation on why an exam is placed before another one in the ordering, so that the student may understand better the ordering and trust the job made by the system.

It is also possible to involve more the student while searching for an ordering, in a way that he can pick from a set of exams the ones he wants to do and the ones he wants to avoid. This could be useful for the cfu and covers goal, not for the cdl goal, since if the goal is to get the degree, there is no choice between exams, you have to do them all.

Since the ultimate goal is to have a system which can have a dialogue with the user and also use the knowledge base to help him, a Telegram Bot could be more appropriate than a command line interface. And also easier to use, not having to install anything on you device but just a chat app that is probably already installed.