

SINF22 Master Thesis

A Study Program Verifier

Based On Petri Nets

Appendices

By:

Colson Olivier

SINF22MS

50391000

Advisor: Lobelle Marc

UCL, 2014-2015

Table of contents

1) Installation guide	1
a. Installing Java.....	1
b. Installing PostgreSQL	1
c. Populating the database	1
d. Getting Glassfish	2
e. Running the application	2
f. Compiling the sources	3
i. Compiling BlindPN	3
ii. Compiling the application	4
g. Generating the documentation	4
i. For blindpn	4
ii. For the application	5
2) User manual.....	6
a. The login page	6
b. Logout	6
c. The index page	6
i. The tickets system	6
ii. Open tickets tab.....	7
iii. Closed tickets tab	7
iv. Search tab.....	7
v. Student: “Submit new program” and “Edit program” buttons.....	7
d. Student : Issue new study program	8
i. The studyProgramView page	8
ii. The studentJustificationView page	10
e. The ticketView page.....	11
i. Commission.....	11
ii. Student	13
3) Description of the test data.....	14
a. Test users.....	14
b. Test courses	15
c. Test programs	19

1) Installation guide

This section explains how to deploy the application presented in this thesis (named “Courses Coordinator”) on a local computer.

a. Installing Java

First thing is to install Java 8, update 45 or higher on your computer.

You can check your java version by opening a command line (both on Unix systems and Windows) and entering the following command:

```
java -version
```

This should display, among some other information, “java version ‘1.8.0_45’” (or any number above 45 if you install a more recent version).

If your java version is too low, or it is not installed on your computer, you can get it at the following URL: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html> (note that we recommend here installing the developer version of Java, in order to allow compiling the source and generating the documentation).

Once you have installed Java, open a new terminal and check your version as explained above.

b. Installing PostgreSQL

The application uses a local PostgreSQL test database.

To access the database in a more comfortable way, we install it together with a graphical client, PgAdmin.

An executable file installing together PostgreSQL and PgAdmin can be found here: <http://www.enterprisedb.com/products-services-training/pgdownload#windows>

(For some reason, Google Chrome seems unable to download the files on this page. If you encounter this problem, try with another web browser)

Note: When installing PostgreSQL, you will need to define a superuser password and a port to use. The application assumes the password is “GAT-X102”, and port 5432. If you wish to use another password or port, you will need to change that in the application’s classPathResources/META-INF/persistence.xml file. Then, you will need to rebuild the .war file of the application, as explained later in this section.

c. Populating the database

Now that you have installed PostgreSQL, it is time to inject our test data into the database.

Open PgAdmin, and click on file->Add server.

In the window that just popped out, write “coursesCoordinator” in the name field, and make sure port is 5432 and user name “postgres” (to use another user name, you will need to change

the persistence.xml file and rebuild the application, as previously explained for the password and port). Also, enter “127.0.0.1” in the host field, as we run the database locally. Finally, enter the database password (GAT-X102) in the “password” field. Pressing “OK” should close the window and create a new server called coursesCoordinator on the left of the screen. Double-click on it. It should now have expended the node in the tree view, and you should see a “Databases” subcategory. Right click on it and select “New database...”. Doing so, a new window should have appeared. In the “name” field, enter “test” (the name of our database). Then, click “OK”.

A “test” node should now be present under the “Datatabases” category. Right click on it and select “Restore...”. Then, in the new window that just appeared, select (for the “Filename” field) the provided testData.backup file, which contains a dump of the test database to use for the application. Finally, click “Restore”. Once the operation is completed, you can verify that it went well if the last of the messages displayed in restoration window indicates an exit code of 0.

d. Getting Glassfish

Next step is to get the Glassfish server to use to deploy our application. Nothing needs to be installed here, Glassfish can be downloaded at <https://glassfish.java.net/download.html> (make sure you select the “Java EE 7 Full Platform” version). Simply unzip the archive file in a place you can easily access from your terminal, as we will need to in the next step.

Caution: Glassfish does not like special characters in paths! Make sure you install it in a directory whose path does not contain any space or accentuated character. Otherwise, Glassfish will simply not run and give you a parsing error in some XML file of the application when trying to deploy it.

e. Running the application

Let us now explain how to deploy the application. For this, we need the .war file containing the whole website, which can be obtained by compiling the program.

For the sake of simplicity, the submitted application comes with a pre-built .war file, so that it is not necessary to compile it. We will hence consider in this point that the .war file is already available. To learn how to build the .war, see next point.

Open a terminal and enter the following command:

```
cd PATH_TO_GLASSFISH_BIN
```

where PATH_TO_GLASSFISH_BIN must be replaced by the path to the Glassfish “bin” directory, contained in the Glassfish server you downloaded. This path should be DIRPATH/glassfish-4.1/glassfish4/bin, where DIRPATH corresponds to the directory into which you unzipped the Glassfish archive.

Then, enter this command, to start a new domain responsible for our application:

asadmin start-domain

Wait until the domain has started, then enter this command:

asadmin deploy PATH_TO_WAR

where PATH_TO_WAR refers to the path to the .war file of the application to deploy.

Once the command has return, the application will be accessible through any web browser via the following link: <http://localhost:8080/coursesCoordinator/index.xhtml>

Once you have tested the application, it is a good idea to stop Glassfish. For this, enter the following command in the terminal you started the domain in:

asadmin stop-domain

f. Compiling the sources

We have explained how to deploy the application from a provided .war file, now we will see how to compile it entirely; hence, generate the .war.

i. Compiling BlindPN

As explain in this thesis' main paper, the application makes use of a home-made Petri Nets library called BlindPN. Of course, compiling the application also requires compiling BlindPN. This is what we will do first.

First, before compiling, it is important that you update your path in order for it to know the command corresponding to the compiler we use here: Javac. This is explained here, and basically consists of adding the “bin” directory of your Java installation to your system path: <https://docs.oracle.com/javase/tutorial/essential/environment/paths.html>

Once this is done, enter the following command to verify you have the right version of javac (1.8, update 45 or above), with the following command (in a new terminal):

javac -version

To compile BlindPN, just enter the provided blindpn directory (in the appendix archive) and run the following command:

*javac -d cpld *.java*

Then, to generate the .jar file, enter the following commands, in the same directory:

cd cpld

*jar cvmf ../MANIFEST.MF blindpn.jar **

After this command, the file blindpn.jar should have been created in the cpld directory.

ii. Compiling the application

Let us now see how to build the rest of the application.

1. Installing maven

To build the application, it is required to install a project builder called Maven. It is available here, with its installation instructions: <https://maven.apache.org/download.cgi>

2. Linking BlindPN

Before building the application, Maven must be told about the existence of BlindPN. Doing so, it will be able to resolve the dependency between the application and that library.

For this, open a terminal, go to the directory where the blindpn.jar is located (a precompiled version of it is provided in the “precompiled” directory of the appendix archive, but you can generate it yourself, as we have seen). And enter the following command:

```
mvn install:install-file -Dfile=blindpn.jar -DgroupId=org.blindpn -DartifactId=blindpn -Dversion=1.1 -Dpackaging=jar
```

Note that if this is the first time you do it, Maven will need to access the Internet to download its “install” plugin before executing the command.

3. Building the .war

Let us now build the .war file. As in the previous point, Maven will need to access the Internet to download dependencies the first time you will execute these instructions.

Go to the coursesCoordinator (provided in the appendix archive) directory in a terminal, and execute the following command:

```
mvn package
```

This will create the coursesCoordinator.war file inside the “target” directory.

g. Generating the documentation

i. For blindpn

To generate the documentation for blindpn, open a terminal into the blindpn directory and execute the following command:

```
javadoc -d doc *.java
```

The resulting Javadoc should then have been generated into the “doc” directory (inside the blindpn directory). Ignore the warnings given by the javadoc command.

Note that this command only works if the javadoc command is part of the system path. This should be the case if you added javac to your path (as explained earlier for compiling BlindPN).

ii. For the application

To generate the Javadoc with Maven, go into the `coursesCoordinator` directory and enter the following command:

```
mvn javadoc:javadoc
```

(Again, you will need to be connected to the Internet the first time you do that)

Do not pay attention to the errors displayed in the terminal (actually, the javadoc plugin of Maven is kind of extreme in terms of Javadoc usage). The documentation of the application is now generated in the `site/apidocs` folder of the “target” directory.

2) User manual

This section explains how to use the application.

We rely here on the key concepts define in the “Terminology” subsection of this thesis’ main report’s section 2.

The order of the following points reflects the typical order into which they come into play when a student wants to submit a new study program. Each time a page can be used by both the student and commission, we mention that this is the case, and explain the differences between them, if there is one. Some other pages are strictly reserved to one given type of user. These are also indicated clearly.

a. The login page

The first screen any user sees is the login page. This page simply asks the user to enter his user name and password, and redirects him to the index page if he provides correct credentials. In case a wrong (user name, password) pair is provided, an error message is displayed in the right corner of the screen, informing the user that the authentication process failed.

b. Logout

Once a user of any kind has successfully logged in, all the pages of the application he accesses to display, in the right corner of the screen, a “Logout” button. Clicking on it disconnects the user and redirects the web browser to the login page.

c. The index page

The index page corresponds to a kind of “main menu” for the application. It contains the option allowing a student to trigger the creation or edition of a study program, and gives access to both students and commission to all the messages they exchanged in the past.

i. The tickets system

The application uses the concept of “ticket” to model the exchange of messages between a given pair of users, or a user and a user group.

The tickets we use here are hence used by students to send study program proposal to the commission (in the form of the COMMISSION user group), and then by both the commission and student to dialog and discuss the study program until the commission accepts it.

A ticket always has an owner, a user or user group having the administration rights on it. These rights give the owner of a ticket the possibility to *close* it (by default, a ticket is always *open*). Once a ticket is closed, it is archived, and no more messages can be exchanged on it.

The messages contained in a ticket are called ticket parts. Conceptually speaking, they can contain any kind of data. In the current version of the application, they always contain a study program proposal, and some comments made about it.

ii. Open tickets tab

This tab of the index page shows, in a table, the open tickets in which the connected user is implied (either they are addressed to him or his group, or he created them). Tickets containing some ticket part that the user has not marked as read (see point e.) are displayed in bold and put at the top of the list, as they contain some new information for the user.

Clicking on one line of the table displayed in this tab opens the ticketView (see point e.) page, so that the information relative to the so selected ticket is displayed.

iii. Closed tickets tab

The index page also contains a “Closed tickets” tab, containing all the tickets linked to the current user that have been closed. Again, if some ticket part contained in these ticket parts was not marked as read, the corresponding ticket is displayed at the top of the list, and its name displayed in bold.

As with open ticket parts tab, clicking on one ticket in the displayed table opens it in the ticketView page.

iv. Search tab

This tab allows searching for tickets. It contains two check boxes, one for open tickets, and the other for closed tickets. A text field and a “Search” button are also present.

The text field is used to give the search parameter, that is, a user or group name for which we want to retrieve the tickets shared with the currently connected user. The two checkboxes indicate whether the search results should (checked) or not (unchecked) contain respectively the open and closed tickets. Of course, if no check box is checked, no result will ever be obtained, as there is no ticket that is neither open, nor closed. The “Search” button, as its name suggests, is used to trigger the search.

Once the “Search” button has been clicked, a table is displayed under it, listing all the tickets matching with the search. Again, tickets with unread ticket parts are marked in bold and put at the beginning of the list. Clicking on one of these tickets opens it in the ticketView page, as for the other tabs.

v. Student: “Submit new program” and “Edit program” buttons

For student users, it is also possible to have an additional option, in the form of a button above the tabs of the index page.

This button will be displayed as “Submit new program” if the student did not yet get some of his study program proposals accepted by the commission. Clicking it then triggers the creation of a new study program, via the studyProgramView page, as explained in point d.

If some program was already accepted by the commission for the current period, and if the student has the right to edit this program now (typically, several weeks are often left for students to modify their study program once the commission has accepted it), then the “Edit

program” button is displayed. This button also redirects the student to the studyProgramView page, but in edition mode, as explained in point e.ii.2.

d. Student : Issue new study program

Let us now explain how a student can make a new study program proposal and send it to the commission. As we have explained, this can be done with the “Submit program” button of the index page, which redirects the user to the studyProgramView page.

i. The studyProgramView page

The studyProgramView page is used to select courses and add them to a study program. It is vertically split in two parts, the left panel contains search tools and displays some information to the user, as the right panel displays the study program currently being built.

1. Getting information about courses

Each time a course is displayed in the left panel of the studyProgramView page, only its tag and name are shown (for example, “LSINF1160: Introduction à l’algorithmique et à la programmation, Partie 1”). Nothing more is displayed by the application, but it is possible to get some more information via the official website of the course, by simply clicking on it. It then opens the course’s website in a new tab, hence giving all the available information on it.

2. Getting information about programs

As with courses, very few information is displayed in the left panel of the studyProgramView page, but links to official web pages are provided, so that clicking on them opens a new tab displaying all the information available on the program.

The “Available programs” tab of the left panel lists all the programs that are available to the student with the programs list he has validated in his previous years of study (which is possibly empty, of course). It gives a better view to the student of what he can achieve.

3. Choosing courses

Let us now review the various ways a student has to add a course to his program.

a. Directly by tag

The first and most direct method supposes the student already knows exactly which course he wants to add. Then, he simply needs to select the tab of the study period he wants to insert the course in and enter the course tag (e.g. “LSINF1161”) into the text field next to the “Add course:” label, and then press the “Add” button, to the right. **Note that it is important to select the right study period in the tab pane below**, otherwise an error message will be shown, and nothing will be added.

Of course, trying to add a course that does not exist, that has already been added or that has already been validated in some previous year will result in an error message, without causing any additional effect.

b. Search for a course

Of course, a student has also the possibility to perform some search in the system to select his courses. This is achieved in the “Search courses” tab of the left panel. A text field at the top of this tab allows entering search parameters, and a popup button allows selecting what kind of search has to be performed with the provided parameter.

Whatever the kind of search selected, the results will always be displayed in the same manner: a list of links containing the name and tag of the course, with a “Add to program” button on the left of each. Clicking the “Add to program” button will have the effect of adding the course to the study program the user is building. Of course, if the student has succeeded this course in some previous year, or if this course has already been added to the program, an error message is displayed and nothing more is done.

Let us now review the different kind of searches available. Each of the three following points corresponds to one of the options in the “Search...” popup button.

i. By name and tag...

This kind of search simply displays all the courses whose tag or name contain the parameter provided in the text field (ignoring case).

ii. Needed by program...

This option searches for the program having as its tag the search parameter given in the text field. Then, it displays all courses offering at least one of the achievements needed by this program.

It is important to understand here that this does *not* necessarily mean that all the courses that are displayed need to be followed to validate the program. Indeed, it is possible that a program asks for achievement “a”, and that this achievement is available by following courses “1”, “2” or “3”. Those three courses will then be displayed after performing the search, but selecting only one of them will be perfectly fine regarding to the program requirements.

In the case a wrong program tag is entered as a search parameter, the search displays no result.

iii. Granting achievement...

This type of search simply displays all the courses whose achievements list contains the search parameter (hence needing an exact, case-sensitive match).

Of course, if no such course exist (most certainly because the achievement does not exist), no result is displayed.

4. Removing courses from selection

To remove some course from the study program he is modifying, a student needs to select the study period tab containing the course, then click on the “Delete course” popup button, select the course to delete in the list (containing the courses of the selected tab’s study period), and then press the “Delete selected courses” button, to the left of the popup.

5. Submitting

Once he is satisfied with his study program, a student can send it to the verifactor, in order to transmit it to the commission after that. This is achieved by clicking the “Submit program” button, at the bottom of the page. This button runs the verifactor on the study program proposal, and then reports its results in the studentJustificationView page, to which the user is redirected.

ii. The studentJustificationView page

This page is used to report verification results to the student and allow him justifying what the verifactor detected as errors and giving an additional comment to his program for the commission.

1. Completed programs list

The first thing that is contained in the studentJustificationView page is a frame containing the list of all the programs that will be completed by the student, taking his study program proposal, and previously validated courses and programs into account. If there is no such program, then it is reported in red in the frame.

2. Comment for the commission

This text editor allows the student adding some comment on his study program for the commission.

3. Justifying verification errors

Under the comments text zone, the errors found in the verification are listed as a succession of popup panels containing an error message and a text editor. The student *must* enter a justification to all the verification errors using these editors.

If no error was detected, then nothing is shown, and hence no justification to be given. The student can however give a comment before submitting if he wishes to.

4. Submitting

Submission in this page means sending the program proposal to the commission through the ticket system we presented at the beginning of this section.

Submission is only possible if there was no verification error or if all these were justified by the student. Indeed, what the vericator considers as “errors” can simply be abnormalities in a study program, which are perfectly fine because of some element that is not known by the vericator.

Submission is achieved with the “Submit” button, at the bottom of the page. It sends the study programs to the commission, and then redirects the user to the index page.

e. The ticketView page

Let us now consider the ticketView page, used to display tickets’ content.

This page consists of two parts. On the left of the screen, buttons allowing each to open a ticket part (they are sorted, from top to bottom of the page, from the most recent to the less one). On the right side of the screen, the currently selected (thanks to the left part’s button) ticket part is displayed.

This view is accessible to both the commission and students, but some parts of it (especially the options concerning ticket parts) diverge following the type of user accessing it. Hence, we will explain how the commission and student use this page separately, beginning with the commission.

i. Commission

1. Lock ticket

Each time we refer to “the commission”, we actually refer to the COMMISSION user group. This means that the commission can be composed of several members.

When a study program proposal is sent to the commission, it is hence actually sent to the entire COMMISSION group, making it available for all of its members. The members then have to share the workload of the whole commission. This can be done using the “lock” and “unlock” function, via the “Lock ticket to myself” or “Unlock ticket to my group” buttons (of course, depending on the ticket’s status, only one of them will be shown at a time).

Locking a ticket simply means that the commission member user becomes the ticket’s owner. Unlocking it sets the ticket’s owner back to COMMISSION group.

2. Close ticket

In the current version of the application, ticket parts always contained study program proposals, and tickets are only created by students to send theirs. The commission is hence always owner of these tickets, it can thus close them if they are open (see point c.i.), using the “Close ticket” button.

3. Ticket parts management

Let us now consider the ticket parts management options of the commission.

a. Mark as read

Of course, the commission members, like students, can mark ticket parts as read (see point c.i.), using the “Mark as read” button on an unread ticket part.

b. Accept or refuse student justifications

A ticket part sent to the commission always contains a study program proposal, together with its verification report. If this verification report contained verification errors, they have all been justified by the student at the time of submission (see point d.ii.4.). The commission must now accept or reject each of these justifications. Each verification error is listed in a popup frame, with an editor for the commission, and the student justification. Under the editor stands a button with two possible values, “Accepted” and “Refused” (default is “Refused”), telling whether or not this student justification has been accepted.

Of course, if the verification reported no error, the student gave no justification, and the commission has nothing to do on this perspective.

c. Comment student justifications

Whether it decides to accept a student justification or not, the commission *must* always comment its decision, for archival purpose, and to ease the communication with the student.

Of course, if the verification reported no error, the student gave no justification, and the commission has nothing to do on this perspective.

d. General comment

Just like a student submitting a program with the StudentJustificationView page, the commission can enter any global comment for the user before answering to his proposal.

e. Warnings

In some cases, some verification warnings will be reported in an additional popup frame to the commission. They don’t need to be commented, nor accepted, like verification errors, but give information that the commission should take into account before accepting a study program (for example, if a student acquires the same achievements too many time over the choice periods).

f. Answer to proposal

The “Answer” button at the bottom of the page gives the possibility to the commission to send the study program back with some comments to the user proposing it. Typically, this will be used when something is wrong with the proposal.

It is necessary for the commission to have commented all of the student justifications (if there were some) before answering; otherwise, nothing is done, and an error message is displayed.

Note that answering is also possible when no errors were reported during verification, as the commission may decide something is not right in the program, because of some criteria the verificador did not take into account.

g. Accept program

Of course, the commission can also accept study program proposals. For this, a commission member simply needs to click “Accept program” next to the “Answer” button.

Once a program has been accepted, it is recorded as the study program of the student for the current choice period. If edition is allowed, it can then be edited from index page, as we explained at the beginning of this manual.

Accepting a program is only possible if all the student justifications to verification errors for that program have been *both* accepted *and* commented by commission. This is important in order to better keep track of the decisions that were made and archive them. If this condition is not met, trying to accept a program will only display an error message.

ii. Student

1. Mark ticket parts as read

Just as the commission members, student can mark unread ticket parts as read with a dedicated button.

2. React to commission’s refusal

In this point, we briefly present the options of a student whose study program has just been rejected by commission.

a. Answer again

The first option of the student is simply to answer again, giving better explanations of why the verification errors should be ignored to the commission, or answering to some additional questions that the commission could have asked.

This is done through a view much similar to the one of the commission when it receives a study program proposal and displays the related ticket part, yet simpler.

b. Edit and submit new proposal in answer

The second possibility available to the student is to modify his proposal to take the commission's remarks into account.

He then has to click the "Edit program" button at the bottom of the page when displaying the commission's answer. This transmits the study program proposal to the studyProgramView page so that it can be used to edit it. This edition process is made in the same way as the creation one, which we depicted earlier.

Note that it is the commission accepted some, but not all of the student justifications to verification errors. If these verification errors occur again after editing the program, they will be *ignored* during the verification, and the student will not have to justify them again. They will however still be listed into the ticket parts of the study program proposal, for archiving and clarity purposes.

3. Edit already accepted program

The edition of an already accepted program, as we explained is made through the "Edit program" button of the index page. It follows exactly the same process as the edition of a refused proposal, described in the previous point.

3) Description of the test data

This section lists the content of the test database, in order to ease its use when trying the application. Of course, these data represent a small proportion of what the final system should contain, but they are enough to test its global behavior.

a. Test users

Here are the test users contained in the test data, listed with all their attributes. Of course, a program validated by a student also means that this student has acquired all the achievement needed by this program.

Name	Password	User type	Validated programs
ocolsonSINF11	SINF11	STUDENT	-
ocolsonSINF12	SINF12	STUDENT	SINF11BA
commission	commission	COMMISSION	N/A

In the provided data, only user ocolsonSINF12 has validated some courses. Here is the list of courses he has validated (information about those courses are given later on in this document):

LANGL1370
LCOPS1124
LECGE1113
LECGE1115
LESPO1122
LMAT1111B
LMAT1111E
LSINF1140
LSINF1151
LSINF1160
LSINF1161

b. Test courses

Here is a list of the courses contained in the test database. For readability purposes, we split the listing of the various attributes into several different tables.

Tag	Course name
LANGL1370	Compréhension à la lecture de textes de sciences appliquées
LCOPS1124	Philosophie
LECGE1113	Comptabilité I
LECGE1115	Economie politique
LESPO1122	Fondements du droit public et privé
LMAT1111B	Mathématiques générales 1 (1e quadrimestre)
LMAT1111E	Mathématiques générales 1 (2e quadrimestre)
LSINF1140	Bases électroniques de l'informatique
LSINF1151	Laboratoire; résolution informatique de problèmes
LSINF1160	Introduction à l'algorithmique et programmation, Partie 1
LSINF1161	Introduction à l'algorithmique et programmation, Partie 2
LANGL1372	English for Computer Scientists
LBIR1200	Mathématiques générales(II)
LBIR1203	Probabilités et statistiques (I)
LECGE1212	Macroéconomie
LECGE1222	Microéconomie
LFSAB1402	Informatique 2
LGLOR1172	Grammaire de base du grec
LGLOR1173	Langue et textes grecs
LJAPO1100	Japonais - niveau débutant
LSINF1225	Conception orientée objet et gestion de données
LSINF1250	Mathématiques pour l'informatique
LSINF1252	Systèmes informatiques 1
LTHEO1341	Hindouisme

LBIR1304	Probabilités et statistique II
LECGE1317	Théorie des organisations
LELEC1930	Introduction aux télécommunications
LFSAB1509	Projet 4 (en Informatique)
LGLOR1271	Langue grecque I
LINGI1101	Logique et structure discrètes
LINGI1113	Systèmes informatiques 2
LINGI1122	Méthodes de conception de programmes
LINGI1123	Calculabilité
LINGI1131	Computer language concepts
LJAPO1300	Japonais - niveau intermédiaire
LSINF1121	Algorithmique et structures de données
LTHEO1342	Bouddhisme
LJAPO1500	Japonais - niveau avance

Tag	Preconditions
LANGL1370	
LCOPS1124	
LECGE1113	
LECGE1115	
LESPO1122	
LMAT1111B	
LMAT1111E	mathFunctions, complexNbers
LSINF1140	
LSINF1151	java, objectOrientedProgammimg, guiBasics
LSINF1160	
LSINF1161	java,objectOrientedProgramming
LANGL1372	englishLvl1
LBIR1200	mathFunctions, complexNbers, integralCalculus, diffEqBasics
LBIR1203	
LECGE1212	macroEcoBasics
LECGE1222	microEcoBasics
LFSAB1402	objectOrientedProgramming
LGLOR1172	ancientGreekBasics
LGLOR1173	ancientGreekBasics
LJAPO1100	
LSINF1225	objectOrientedProgramming, java
LSINF1250	
LSINF1252	
LTHEO1341	
LBIR1304	
LECGE1317	
LELEC1930	
LFSAB1509	android
LGLOR1271	ancientGreekMedium, ancientGreekDeepGrammarLvl1
LINGI1101	

LINGI1113	lowLevelInformaticsLv1, unixLv1
LINGI1122	
LINGI1123	
LINGI1131	objectOrientedProgramming,ozLv1, functionalProgramming
LJAPO1300	japaneseLv1
LSINF1121	java, objectOrientedProgramming,
LTHEO1342	
LJAPO1500	japaneseLv2

Tag	Co-courses
LANGL1370	
LCOPS1124	
LECGE1113	
LECGE1115	
LESPO1122	
LMAT1111B	LMAT1111E
LMAT1111E	LMAT1111B
LSINF1140	
LSINF1151	
LSINF1160	LSINF1161
LSINF1161	LSINF1160
LANGL1372	
LBIR1200	
LBIR1203	
LECGE1212	
LECGE1222	
LFSAB1402	
LGLOR1172	LGLOR1173
LGLOR1173	LGLOR1172
LJAPO1100	
LSINF1225	
LSINF1250	
LSINF1252	
LTHEO1341	
LBIR1304	
LECGE1317	
LELEC1930	
LFSAB1509	
LGLOR1271	
LINGI1101	
LINGI1113	
LINGI1122	
LINGI1123	
LINGI1131	
LJAPO1300	
LSINF1121	

LTHEO1342	
LJAPO1500	

Tag	Achievements
LANGL1370	englishLvl1
LCOPS1124	philoBasics
LECGE1113	comptaBasics
LECGE1115	microEcoBasics, macroEcoBasics
LESPO1122	belgianLawBasics
LMAT1111B	mathFunctions, complexNbers
LMAT1111E	integralCalculus, diffEqBasics
LSINF1140	electroBasics, lowLevelArchitectureBasics
LSINF1151	bigSoftDevelopmentBasics
LSINF1160	java, objectOrientedProgramming,specificationBasics
LSINF1161	guiBasics,dataStructures
LANGL1372	englishLvl2
LBIR1200	lagrangianCalculus, taylor
LBIR1203	probaBasics
LECGE1212	macroEcoMedium
LECGE1222	microEcoMedium
LFSAB1402	ozLvl1, functionalProgramming, complexity
LGLOR1172	ancientGreekDeepGrammarLvl1
LGLOR1173	ancientGreekMedium
LJAPO1100	japaneseLvl1
LSINF1225	android, uml, databasesBasics
LSINF1250	propositionalLogic, basicGraphAlgorithms
LSINF1252	lowLevelInformaticsLvl1, c, unixLvl1
LTHEO1341	hinduismBasics
LBIR1304	statsBasics
LECGE1317	organizationsTheoryBasics
LELEC1930	physicsBasics
LFSAB1509	agile, bigSoftDevelopmentMedium
LGLOR1271	ancientGreekDeepGrammarLvl2
LINGI1101	propositionalLogic, formalSystems, formalStructures
LINGI1113	lowLevelInformaticsLvl2, unixLvl2
LINGI1122	specificationBasics
LINGI1123	complexity, computability
LINGI1131	ozLvl2, concurrencyModels
LJAPO1300	japaneseLvl2
LSINF1121	basicGraphAlgorithms, dataStructures
LTHEO1342	buddhismBasics
LJAPO1500	japaneseLvl3

c. Test programs

Here are the study programs available in the test data, together with all their attributes.

Program	Program requirements	Achievement requirements
SINF11BA		- englishLvl1 - philoBasics - comptaBasics - microEcoBasics - macroEcoBasics - belgianLawBasics - mathFunctions - complexNbers - integralCalculus - diffEqBasics - electroBasics - lowLevelArchitectureBasics - bigSoftDevelopmentBasics - java - objectOrientedProgramming - specificationBasics - guiBasics - dataStructures
SINF12BA	SINF11BA	- englishLvl2 - lagrangianCalculus - taylor - probaBasics - marcoEcoMedium - microEcoMedium - ozLvl1 - functionalProgramming - complexity - android - uml - databasesBasics - propositionalLogic - basicGraphAlgorithms - lowLevelInformaticsLvl1 - c - unixLvl1
SINF13BA	SINF12BA	- statsBasics - organizationsTheoryBasics - physicsBasics - agile - bigSoftDevelopmentMedium - propositionalLogic - formalSystems - formalStructures - lowLevelInformaticsLvl2 - unixLvl2

		- specificationBasics - complexity - computability - ozLvl2 - concurrencyModels - basicGraphAlgorithms - dataStructures
--	--	---