The Carnap Manual

Graham Leach-Krouse

2023-07-28

Table of contents

Preface

This is a guide to using Carnap for teaching and learning logic.

Here's an example of what Carnap can do:

Show ((P -> Q)-> P) -> P

It's still very much a work in progress. Please consider contributing!

Part I

Getting Started

1 Quick Start Guide

1.1 Get an account

Click the login link located in the upper right hand corner of most Carnap pages, and log in with a Google account (a Gmail account works, but you can also make a Google account with a different email).

Email Graham Leach-Krouse at gleachkr@gmail.com using your institutional email address to request instructor status.

1.2 Make your first assignment

1.2.1 Carnap's assignment format

An assignment in Carnap is a document which may include text (e.g., instructions for your students, where to find the relevant material in your textbook, etc.) but mainly will include *Carnap code blocks*. Each Carnap code block corresponds to one problem. They can be of any of the types listed on the main page. They are:

1. Syntax Checking
2. Translations
3. Truth Tables
4. Derivations
5. Model Checking
6. Qualitative Problems, e.g., multiple-choice

Each assignment document is a plain text file formatted in Markdown, a simple markup language. (Typically, the extension of Markdown text files is `.md`.) You can find a comprehensive introduction in the Markdown Guide including a list of editors that support Markdown. For a quick reference, try Learn Markdown in 60 seconds. Any plain text editor is fine, but Markdown (and Carnap) are sticklers about spaces and newlines in Markdown documents. So make sure your editor does not automatically wrap text.

1.2.2 Downloading shared documents

You can start with an existing document another instructor has shared. A list of shared documents can be found at carnap.io/shared. Each shared document has a download icon at the bottom ( ). If you click on that, you can save the source Markdown document of the shared page to your own computer. You can then edit it.

For instance, here is a sample assignment which you can download by adding `/download` to the end of the URL in your browser (or just click here).

1.2.3 Converting existing problem sets

If you already have problem sets in another format, you can turn them into Markdown for use on Carnap. If your problem sets are written in Word, there is a Markdown export plugin: Writeage. Or, select Save As (Plain Text) and then recreate the formatting in the resulting text file. If your sheets are in LaTeX you can use Pandoc. Here is an online converter (select from LaTeX to Markdown (pandoc)).

1.2.4 Carnap code blocks

Next you turn each problem into a Carnap code block. For instance, a translation exercise will be put into your Markdown document as:

```
~~~{.Translate .Prop}
3.1 P/\Q : People want to know what's going on and questions are unavoidable
~~~
```

`.Translate` tells Carnap that you want a translation exercise. `.Prop` tells Carnap it should use its propositional equivalence checker to test if the solution provided by the student is equivalent to the solution you provide. The number 3.1 indicates the exercise number. It is followed by a model solution. The colon separates the solution from the text that will be presented for translation. The result of the above is:

> `3.1 P/\Q : People want to know what's going on and questions are unavoidable`

Each Carnap code block takes a number of options as well, which are described in the linked pages above. The most important one is perhaps the `system` attribute, which determines the syntax and symbols used for sentences entered as solutions which Carnap will accept, and how Carnap formats formulas it displays to the student.

For instance,

~~~{.SynChecker .Match system="LogicBookSD"}
1.1 (P /\ Q) -> R
~~~
~~~{.SynChecker .Match system="thomasBolducAndZachTFL2019"}
1.1 (P /\ Q) -> R
~~~

will generate:

```
1.1 (P /\ Q) -> R
```

```
1.1 (P /\ Q) -> R
```

The attributes `LogicBookSD` and `thomasBolducAndZachTFL` correspond to the syntax and conventions of *The Logic Book*, and *forall x: Calgary*, respectively. The available `system` attributes are described on the Systems page.

Options common to all exercise types, such as how many points a problem is worth, are described on the Carnap Pandoc page. There you will also find a more in-depth description of the format of Carnap's documents.

1.3 Upload and test your document

On your Instructor Page on Carnap, there is a Manage Uploaded Documents tab where you will upload your finished assignment documents.

Once a document is uploaded, it will show up in your Instructor Page in the *Manage Uploaded Documents* tab. If you click on the file name, Carnap will display the document to you the same way that students will see it. You might first have to go through a bit of proof-reading. If the page contains an error message instead of the problem you expect, it's probably because you misspelled an option or did not use the correct syntax for your selected system.

1.4 Set up a course and assign your problems

Once you are ready to give your problem sets to students, you have to make a course for them to enrol in. You do this on your Instructor Page in the the *Manage Courses* card. You will provide your students with an enrolment link which you can find at the bottom of your class card. This video describes what this process looks like for a student. See the documentation of the Carnap Dashboard for more detail.

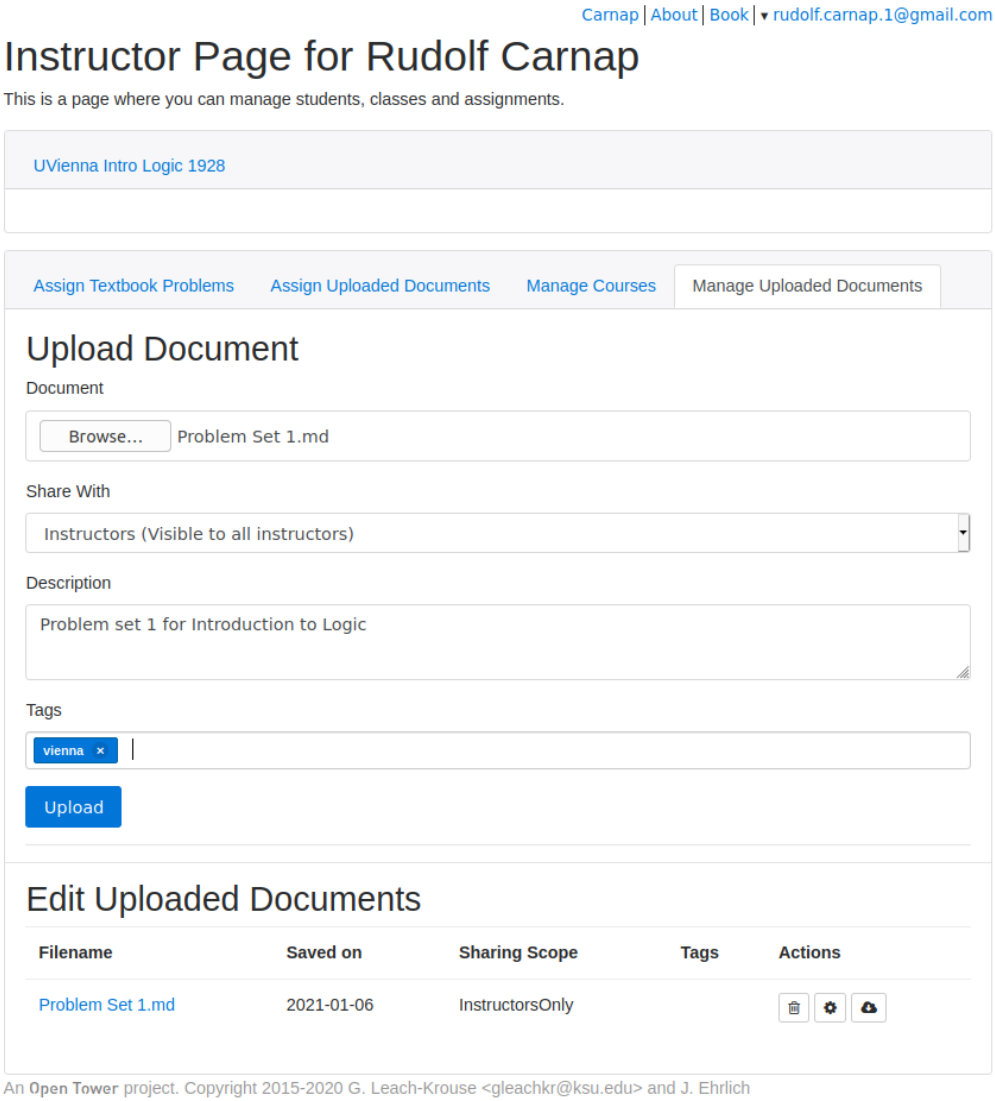


Figure 1.1: Document Upload tab

When you (and your students) are ready, you can assign any of your uploaded documents to your students. Carnap will allow them to complete and submit them. Once a problem is submitted, Carnap records the point value you have given to the problems as a score for the student. Students can see on their Student Home page which problems the have submitted and how many points they earned. You can also (in the course card for your course) see a list of enrolled students. Below the student roster, there is a link to download grades (per assignment, per problem). Your course card will look something like this:

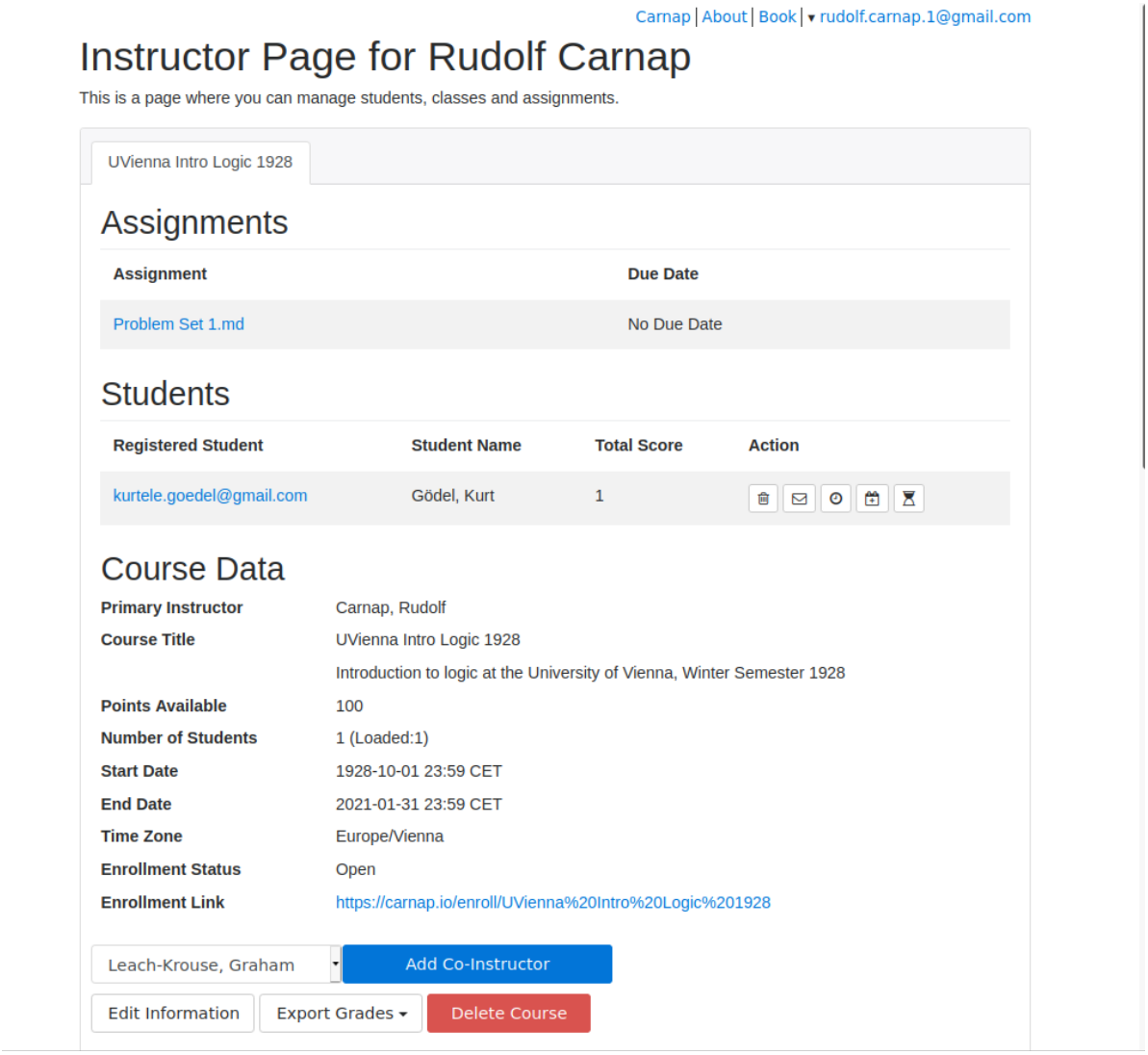


Figure 1.2: The course card

2 Frequently Asked Questions

How can I help my students recognize subproof boundaries better?

Carnap has a few options for indicating subproof boundaries. The two main methods are to either render the proof separately from the input, or to overlay indentation indicators that correspond to subproof boundaries.

The rendering option looks like this (details depending on the system):

```
|1.Show Q->P
|2.   Q          :AS
|3.   P          :PR
|4.:CD 3
```

and the overlay option looks like this (details depending on the system):

```
 P        :AS
 P/\P     :&I 1 1
 P        :&E 2
P->P      :->I 1-3
```

You can find details about overlaying indentation guides here, and details about the rendering option here.

How can I encourage my students to slow down and think when writing derivations?

This is a good question. One option to discourage students from typing before they think is to slow down the rate at which they get feedback. You can require a button-press for feedback on a proof, or turn off feedback about correctness entirely by using the settings `feedback="manual"` or `feedback="syntaxonly"` respectively. Here's the link for more details on the feedback settings.

How can I use Carnap to give students an exam?

If you're using Carnap, an exam is an assignment just like any other assignment. But depending on the type of format you want your exam to have, you may want to set some options on that assignment.

If you'd like your students to be able to submit incorrect answers, rather than having Carnap warn them when something is wrong, then you can add `exam` to the options attribute of your exercises, writing them something like this:

```
~~~{.ProofChecker options="fonts exam" }
1.1 :|-: P->P
~~~
```

All the exercise types except for the syntax-checker support the `exam` option; you can also set other options to disable checking or allow only certain types of feedback on the unsubmitted work. Check the documentation for individual exercise types to get the details on how to do that.

If your students are allowed to submit incorrect answers, you might want to review them and assign partial credit. You can do that using the review area that's associated with each assignment. There are also a few assignment settings that might also be helpful for an exam. In particular,

1. you can set a timer, so that exam needs to be completed in a certain number of minutes after its opened, and
2. you can set a "release date" for grades, so that students can only view grades after everyone has completed the exam.

Details on the review area, and assignment settings, can be found here: here.

You may also want to know how reset the exam timer for certain students, or how to offer accessibility accommodations for certain students. Information on how to do that can be found here

How can I adjust deadlines for assignments?

There are two major deadlines associated with an assignment. One is the *due date*. This is displayed on each student's user page, and determines whether work counts as late. The other is the *visible* date. After the *visible* date passes, the assignment is no longer visible to the student, and can't be accessed. You can configure both of these dates by pressing the small "gear" icon that appears next to the assignment listing in the "manage assignments" tab on the instructor page.

You can also adjust deadlines per-student in two ways. One is to offer the student an extension. You can do this by clicking the "calendar plus" icon that appears next to the student's name in the class roster on your instructor page. The extension will override both the due-date and the visibility date for that student. You can also set a deadline adjustment policy for a specific student by clicking the "clock" icon next to the student's name.

How can I control student credit for late assignments?

By default, students receive half-credit (rounding down) for problems that are submitted after the due date, but while the problem is still visible. However, late credit is configurable using the `late-credit` option, which applies to all exercises. You can set the `late-credit` option like this:

```
~~~{.SomeProblemType late-credit=4}
1.1 SOME PROBLEM
~~~
```

How can I control when grades are released?

Ordinarily, a student's score for a problem is visible (on the student's user page) immediately after the student submits a problem. In some situations (for example, during an exam), this can be undesirable.

If you want to release grades only after a certain time and date, then you can set a "Release Grades After" time when you create the assignment. You can also update this release time by pressing the "gear" icon next to the assignment on your instructor page, in the "manage assignments" tab.

How can I embed a video in an assignment?

Carnap's markdown dialect supports raw HTML, so you can embed videos by including them in the same way that you might include them in an ordinary webpage. There are two main options.

If your video is a file (a `.mp4`, `mov`, `avi` file or something similar) that you have uploaded to a file hosting service somewhere, then you can point a video tag at it by including something like this in your pandoc document:

```
<video controls
    src="https://archive.org/download/day_the_earth_stood_still/day_the_earth_stood_still_51
    width="560"></video>
```

Which will produce something like this (in browsers that support embedded videos):

More details on how to use `<video>` tags can be found here.

Note: *Please don't upload video files to the Carnap site.* We're not designed to serve these. Video files will need to be hosted elsewhere.

If your video hosted at a site like youtube or vimeo, rather than at a file hosting service, then you can instead use an "embed code". Instructions for obtaining an embed code for each of these services can be found here: for youtube, and for vimeo

The embed code, when you get it, should look a bit like this:

```
<iframe width="560" height="315"
    src="https://www.youtube.com/embed/nfeWlHVyBZQ" frameborder="0"
    allow="accelerometer; autoplay; clipboard-write; encrypted-media;
    gyroscope; picture-in-picture" allowfullscreen></iframe>
```

And should result in something like this

Want to offer a documentation suggestion or report a typo? Use the issue tracker here!

Or, if you just have a quick question that's not addressed above, feel free to send an email to gleachkr@gmail.com—if your question comes up a lot, we'll add it to the FAQ.

3 Carnap's Course Management Dashboard

This document gives a short description of the Carnap server's course-management dashboard.

The course management dashboard will be available once you have

1. *Created a user account.*

   To do this, just click the login link located in the upper right hand corner of most Carnap pages.

2. *Upgraded your account status to "instructor".*

   To do this, email Graham Leach-Krouse at gleachkr@gmail.com using your institutional email address.

Once you are logged in, you'll then be able to access the course management dashboard by first,

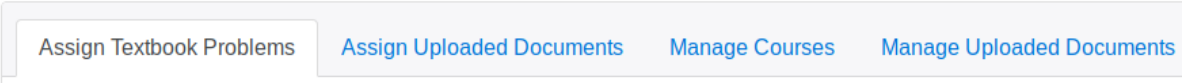
1. navigating to your user page, following the link to *Instructor Home* in the menu linked to your email address in the top right corner of most pages; then
2. from your user page, clicking the link at the very top that reads "your instructor page is here"

You should then see the course management dashboard. The dashboard has two main parts: The *course card*, where information for your different courses is listed, and the *assignment card*, where you can assign problem sets from the Carnap Book.[1] If this is your first time on the instructor page, your course card is probably empty. So we'll describe the assignment card first.

[1]Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

3.1 The Assignment Card

The top of the assignment card looks like this:



There are four main tabs. The first one you want to look at is probably “Manage Courses”.

3.1.1 Manage Courses

Within this tab, you can create a new course, by giving it a title, an optional description, a start-date, a total number of available points for the course, an end-date, and a time-zone. The time zone which is used to determine whether assignments have been received by a given due date and time. The start-date and end-date are used to determine when your course will be listed as an option for enrollment.

Figure 3.1: Creating a new course

Once you’ve created a course, during the period between the start date and end date, it will:

1. be visible as an enrollment option for people creating new accounts, or for people editing their user information from their user pages;
2. become visible to you as a target for assignments in the other tabs; and
3. appear as a tab in your course card (see below).

After the end-date of the course, the course will be archived at the bottom of the manage-courses tab, under the heading “Old Courses”. Once the course is archived, you’ll still be able

to edit the course's properties, download grades as a `.csv` file, and delete the course. If you need to reactivate an archived course, just set its end-date for some point in the future.

3.1.2 Assign Textbook Problems

Within this tab, you can assign textbook problem sets from the Carnap book to your courses, setting particular due-dates and due-times. There are currently 17 problem sets available, but more are likely to appear in the future. Once a problem set is assigned to a given course, it will appear on the user-page of everyone enrolled in that course.

You can also delete assigned problem sets, and replace them with new ones. Deleting an assigned problem set does not delete submitted student work, so first deleting then replacing is a safe way to change the due-date of an assignment.

By default, submitted problems received on time receive 5 points each, and late submissions receive 2 points each. Both the point amount awarded for a given problem and the treatment of late problems can be customized—details in the documentation for pandoc markup.

I find that leaving open the option to submit even *very* late work helps students who have fallen behind remain motivated, and gives them some incentive to do extra work that will make the later material easier for them to understand.

3.1.3 Assign Uploaded Documents

Within this tab, you can assign documents created using Carnap's pandoc markup. In order to assign a document, you need to first upload it in the Manage Uploaded Documents tab.

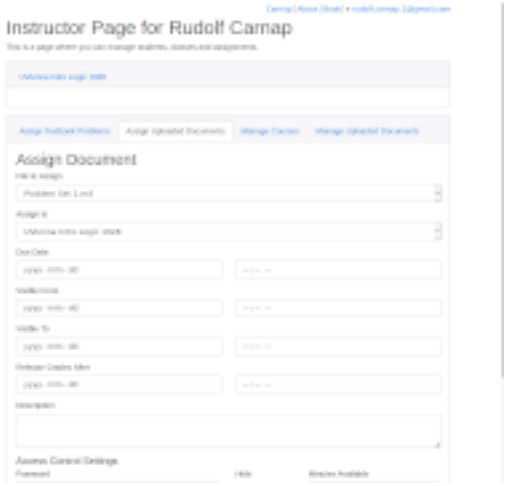


Figure 3.2: Assigning an uploaded document

Creating an assignment allows you to optionally associate a due-date, visibility range, grade release date, and description with the assignment.

Once an assignment is assigned to given course, it will appear on the user page of everyone enrolled in the course for the duration of the date and time range of you specify (or indefinitely, if no visibility range is given). It will not be possible for students to make new submissions after the assignment ceases to be visible.

The due-date affects when student work is counted as late, just like with a problem set. If the due date is omitted, then work can be turned in at any time for full credit. The grade release date determines when grades for the problems in this assignment will be released to students (before the release date, students will see that work has been submitted, but the point value will be listed as "-"). If it is omitted, scores will be released immediately.

Assignments can also be equipped with access controls. Access controls require setting a password, which students will need to enter before first accessing the assignment. Access-controlled assignments can be hidden from the course. If hidden, they will not appear on the user page, but will still be accessible via their URL. It's also possible to enter a time-limit for access controlled assignments. The time limit is measured from when the student enters first enters their password. After the limit expires the student will no longer be able to view the exam or submit problems. The time limit is based on a time-stamp of when the student began the exam, so it will expire after the designated period even if the student closes their browser.

In addition to creating assignments, you can also:

1. edit the descriptions and due dates associated with assignments from this tab, by clicking the gear icon ;
2. review the work submitted in connection with this assignment, by clicking the pencil icon , and assign partial credit, if any of the problems from the assignment have been set to allow for incomplete submissions; and
3. delete assignments, by clicking the trashcan icon next to the assignment. Deleting an assignment *deletes all student work associated with that assignment*, so be careful when doing this.

Submitted work for assignments is graded in much the same way as work submitted for textbook problem set. Each problem counts, by default, for five points on time or two points if late. Problems assigned a custom point value count for that value if received on time, and otherwise for half that value rounding down. Late point values can be customized by setting the `late-credit` attribute—for details, see the documentation for pandoc markup.

3.1.4 Manage Uploaded Documents

Within this tab, you can upload and manage your documents.

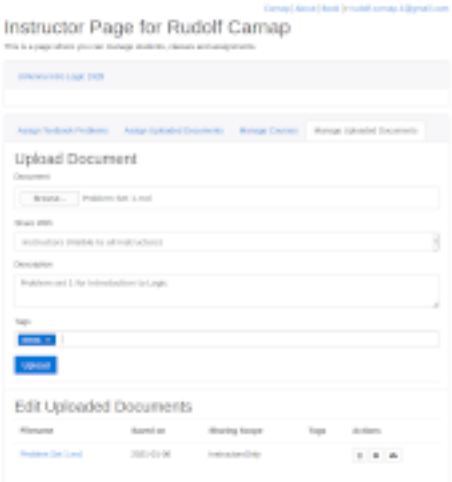


Figure 3.3: Uploading a document

Documents can be of three kinds:

- Problem sets and other materials created using pandoc markup.
- CSS stylesheets for custom styling of your course materials.
- JS scripts for more sophisticated customization of your course materials.

Uploaded files will be interpreted as problem sets unless they have a filename extension that indicates they are something else, so unless they are called something like `custom-stylesheet.css` or `my-script.js` (because of the extensions `.css` and `.js`). For more information on how to apply custom styling and behavior, see the "other features" section in the documentation of Carnap's pandoc markup.

Once documents are uploaded, they can also be assigned to students as problem sets, using the Assign Uploaded Documents tab.

When you create a document, you'll be able to select a "Sharing Scope". There are four options: Public, Private, Instructors-Only, and Link-Only. The sharing scope affects whether your document can be accessed via the list of shared document that be found at carnap.io/shared. Public documents are listed and available to anyone, Instructor-Only documents are only listed and available to instructors, and Private documents are unlisted and only available to you, Link-Only documents are unlisted but available (via link) to anyone. The tags you assign to a document are also used to help organize the document in the shared documents list.

You can share the the link to the document that appears in your document listing, e.g., via email or your LMS. However, only Public and Link-Only documents will be accessible via this link to your students. Also, you should not use this link to share assignments or tests with your students. Instead, share the link to the assignment in the course card.

The sharing scope doesn't affect whether you can assign the document to students. When a document is assigned to your class, students can access the assignment associated with the document, regardless of the sharing scope you set when you uploaded it. Note that documents you plan to assign with access controls (e.g., passwords or a time limit) should have their sharing scope set to Instructor-Only or Private, since otherwise they can be viewed without restrictions.

Again, to edit the attributes of an uploaded document, click the gear icon . You can also delete documents with the trashcan icon , or download them with the cloud icon .

3.2 The Course Card

Once you've created a course, the course card will look a bit like this:

Different courses are available under different tabs.

At the top ("Assignments") you will see a list of documents you have assigned to your course, sorted by name or by due date. You can share the link to the assignments given here with your students (e.g., in the LMS). Students have to be logged in to Carnap to be able to access assignments via this link, however.

Next you see a list of students registered in your course and total scores earned by those students. You can click on each student's email address to see details of their submitted problems and points earned. See below for the possible actions you can take for students using the bottons on the far right.

3.2.1 Course Information

Below the table of students, you'll see some course information, including the course title, points available, number of students registered, the start and end date for your course, the time zone, the textbook, whether the course is open for enrollment, an enrollment link you can give to students, and any co-instructors. You can update the course description, points available, start and end date, whether the course is open for enrollment, and the course textbook by clicking on "Edit Information".

3.2.1.1 Opening Enrollment for Your Students

Students who follow the enrollment link will be prompted to register (if they're not already registered) and automatically enrolled in your course. A short video explaining the process for students is available here: youtu.be/lmkWcxqxEZk. Feel free to send your students this link along with the enrollment link.

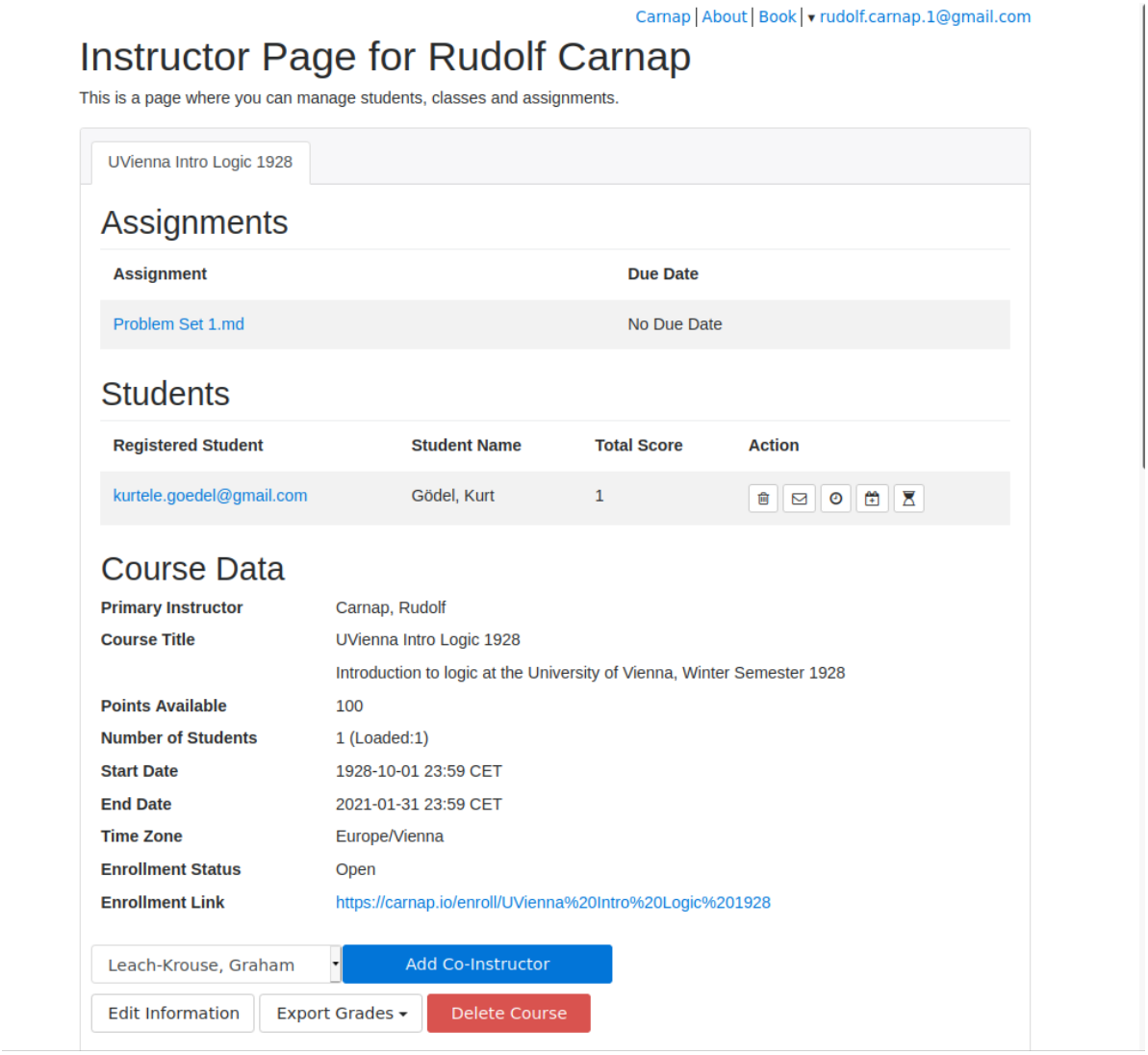


Figure 3.4: The course card

3.2.1.2 Choosing a Custom Textbook

You can also select a custom textbook for your course. A custom textbook is a Carnap document that students will be directed to when they click the "Book" link at the top of most pages (this overrides the ordinary behavior of directing students to the built-in Carnap textbook). The linked document would ordinarily function as a table of contents, directing students to various "chapters" (other assignments), containing problems that they can complete throughout the semester.

When selecting a custom textbook, you'll need to choose a document that has already been assigned to the course—so, you'll want to assign the textbook *first* and then set it as the textbook for the course.

Once assigned, the textbook will not be shown in the "Upcoming Problems" section of your students' user page. So you do not need to adjust the visibility of the assignment or mark it as hidden, in order to keep it from cluttering your assignment listings.

3.2.2 Actions on Students

A number of actions can be performed on individual students by clicking the icons in the "Actions" column:

1. Students can be dropped from a course by clicking the trashcan icon . A dropped student is removed from the course, but their data is otherwise intact. So if they are dropped by mistake, then can simply re-enroll.
2. Students can be emailed individually by clicking the "mail" icon visible in the action column to the right of their name.
3. Accessibility accommodations can be added for individual students by clicking the "clock" icon to the right of the student's name. A fixed number of minutes can be added to all of the students' timed exams, their exam times can be extended by a given factor (for example, 1.5 for time-and-a-half) and a fixed number of hours can be added to all of their due-dates. Adjustments to due-dates do not affect visibility dates for assignments, only the treatment of late work. Changes to due-dates will be reflected on the posted due-date seen on the student's user page.
4. Alternate due dates can be issued to individual students, for specific assignments, by clicking the "calendar plus" icon to the right of the student's name. Alternate due dates will be reflected on the posted due-date seen on the student's user page. Alternate due dates will affect whether an assignment is graded as late, and will override the visibility settings for that assignment, so that a student can continue to access the assignment and submit problems until their alternate due date expires. Assignments set as "hidden" however, will not be visible on the user page even if an alternate due date is set.

5. Access can be re-granted to timed assignments after the timer has expired, by clicking the "full hourglass" icon to the right of the student's name. Once access is re-granted, it will be as if the student never started the timed exam (although any work that they submitted on the first try will persist), so their next attempt at the exam will be timed in the same way as their first attempt.

3.2.3 Adding A Co-Instructor

At the very bottom of the course card, you have the options of adding a co-instructor, editing a course's information (including total points, description, and whether the course is open for enrollment), exporting grades, and also of deleting the course.

Co-instructors are other instructors who have the same type of access to the course as the course creator. They may be useful if you have a TA or if you're teaching a course with someone else. Co-instructors can be removed from within the course card by clicking the trash icon next to their name. Removing a co-instructor will also remove all assignments created by that co-instructor and all student work submitted in response to those assignments, so *always exercise caution when removing a co-instructor*!

3.2.4 Exporting Grades

Grades are exported in `.csv` format, which most spreadsheet programs should be able to import. Grades can either be tabulated per-assignment (for import into a CMS, for example), or per-problem (for detailed analysis of course performance, or for more complex grading schemes). Students are identified by first name, last name, University ID (an arbitrary string, like their student ID number or other useful identifying information, that you can ask them to enter when registering), and their email address. If a student has not entered some piece of information, its absence is indicated with a question mark.

3.2.5 Deleting a Course

Deleting the course will delete all assignments associated with the course and all student work submitted in response to these assignments, so *always exercise caution when deleting a course*!

4 Carnap's Pandoc MarkDown

Within Carnap, shared documents and problem sets are created using *pandoc* markdown, a simple formatting language (akin to LaTeX) developed by John McFarlane as part of the Pandoc project. Pandoc markdown extends John Gruber's original markdown syntax with some additional niceties.

It would be a little redundant to give the full specifications for pandoc markdown here, since there are many excellent resources already available. See, instead,

1. John Gruber's original markdown specification.
2. The Pandoc user's guide

If you learn best from example, it's also possible to download the source code for documents shared on Carnap.io from the shared document list, available at https://carnap.io/shared. The pandoc source for this document, should you wish to inspect it, is available here.

You can create pandoc documents in any text editing program. Pandoc is nothing but plain text that is written with some special notation to indicate how text is to be formatted. For example, you can write `*italic*` to generate the text "*italic*". However, there are some very good text-editor plugins and dedicated applications available for working with Pandoc. These provide extra features like live previews of the formatted text. A list of these can be found here. An online pandoc editor, with live previews of what the rendered document will look like, can be found at http://markup.rocks.

4.1 Pandoc Extensions

Carnap's pandoc parser incorporates the following pandoc extensions:

- raw_html
- markdown_in_html_blocks
- auto_identifiers
- tex_math_dollars
- fenced_divs
- bracketed_spans
- fenced_code_blocks
- backtick_code_blocks

- line_blocks
- fancy_lists
- startnum
- definition_lists
- example_lists
- simple_tables
- multiline_tables
- footnotes
- fenced_code_attributes
- inline_code_attributes
- shortcut_reference_links
- link_attributes
- yaml_metadata_block
- task_lists

For more details, about each of these extensions, please see the Pandoc Users Guide.

4.2 Exercises

4.2.1 General Pattern

Carnap's exercises are created by including special *code blocks* within a pandoc document. A code block looks something like this:

```
~~~
CONTENT GOES HERE
~~~
```

To indicate which type of exercise you want to create, you apply *classes* and *data attributes* to the code block. That would look something like this:

```
~~~{.SOMECLASS .ANOTHER attribute="value"}
label CONTENT GOES HERE
~~~
```

with classes indicated by a leading `.`, and attributes assigned with the `=` sign, and surrounded by quotes.

The leading `label` within the body of the code block will be used to generate an anchor tag to your exercise, so that you can link directly to the exercise at the URL `ASSIGNMENTURL#exercise-label` where `ASSIGNMENTURL` is the URL for the assignment, and `label` is the label you gave to that particular exercise.

4.2.2 Some common attributes

Here are some common attributes that can be applied to to any exercise

Name	Effect
`points`	Sets a custom point value
`late-credit`	Sets a custom point value for a late submission
`submission`	Controls how work is submitted. Set to "none" to disable submission
`options`	Exercise-specific options

the `options` attribute might require some extra explanation. This attribute is set to a string of space separated words, indicating what special options are set for the exercise-block it is attached to. Different types of exercises allow for different options.

4.2.3 Random problems

When you create a problem set, the different problems within a code block will generally be assigned numbers or some other identifier. So you might write

```
~~~
1.1 SOME PROBLEM
~~~
```

If the same identifier is used for a contiguous series of problems (all part of the same code block and next to one another), then when the problem set is assigned and viewed by a student, one of the problems in the series will be chosen randomly and displayed. So, for example given

```
~~~
1.1 PROBLEM A
1.1 PROBLEM B
~~~
```

Students will see PROBLEM A or PROBLEM B, but not both. A student reloading the page will continue to see the same problem, so if they see PROBLEM A on the first viewing, they'll continue to see PROBLEM A.

The selection of a random problem only takes place when the document is viewed as an assignment, so when the document is viewed through the instructor's "manage documents" tab, all the variant problems will be displayed.

4.2.4 Choose-One problems

If more than one problem is given the same label but the problems are *not* part of the same code block or are not next to one another, then both problems will be displayed. However, within each assignment, a student can submit at most one problem with a given label. So if you want to give your students the option to choose just one of several problems to complete, you can set those problems to all share a single label.

4.2.5 Exercise Types

There are currently eight types of exercises:

1. Syntax Checking
2. Translation
3. Truth Tables
4. Derivations
5. Model Checking
6. Qualitative Problems
7. Sequent Calculus Problems
8. Gentzen-Prawitz Natural Deduction Problems

To learn more about each one, follow the links above.

4.3 Other Features

4.3.1 Formula Parsing

Carnap's formula parser can be used to render and display formulas and sequents inline within a pandoc document. So for example, writing something like

```
`AxF(x)\/Ex-F(x)`{system="firstOrder"}
```

Will produce the output `AxF(x)\/Ex-F(x)`.

Any system that's available for derivations can be used in the formula parser. The available propositional systems are: `prop` `montagueSC` `LogicBookSD` `LogicBookSDPlus` `hausmanSL` `howardSnyderSL` `ichikawaJenkinsSL` `hausmanSL` `magnusSL` `magnusSLPlus` `thomasBolducAndZachTFL` `thomasBolducAndZachTFL2019` `gamutMPND`, `gamutIPND` `gamutPND` `gamutPNDPlus` `tomassiPL` and `hardegreeSL`. The available first-order systems are: `firstOrder` `montagueQC` `magnusQL` `thomasBolducAndZachFOL` `thomasBolducAndZachFOL2019` `thomasBolducAndZachFOLPlus2019` `LogicBookPD` `LogicBookPDPlus` `gamutND` `hausmanPL`

`howardSnyderPL` `ichikawaJenkinsQL` `hardegreePL` `goldfarbAltND` `goldfarbNDPlus` and `goldfarbAltNDPlus`. The available set theory systems are: `elementarySetTheory` and `separativeSetTheory`. The available second-order systems are: `secondOrder` and `PolySecondOrder`. The available propositional modal logic systems are: `hardegreeL` `hardegreeK` `hardegreeT` `hardegreeB` `hardegreeD` `hardegree4` and `hardegree5`. The available predicate modal logic system is `hardegreeMPL`, and the available "world theory" system is `hardegreeWTL`.

4.3.2 Custom CSS

The standard bootstrap CSS that is used for styling the appearance of an assignment can be overriden by including a `css` entry as part of a yaml metadata block within a carnap pandoc document.

The CSS entry can include either a url for a single CSS stylesheet, like so:

```
---
css: https://ghcdn.rawgit.org/Carnap/Carnap-Contrib/main/css/hide-points.css
---
```

or for several stylesheets, like so:

```
---
css:
- https://ghcdn.rawgit.org/Carnap/Carnap-Contrib/main/css/hide-points.css
- https://ghcdn.rawgit.org/Carnap/Carnap-Contrib/main/css/another-stylesheet.css
---
```

The stylesheets can be hosted anywhere, including on the Carnap server. In order to host a stylesheet, just upload it as if it were a pandoc document, but be sure to give it the filetype extension "css", so name it something like "mystylesheet.css". You'll then able able to access it with a metadata block of the form:

```
---
css:
- https://carnap.io/shared/youremail@gmail.com/custom.css
---
```

If you wish to not just partly override the bootstrap css, but to completely disable it, adding your new css to a blank slate (other than the css used to style Carnap's exercises), then you can instead use a `base-css` entry to set the new base css style. For example,

```
---
base-css:
- https://static.carnap.io/css/tufte.css
- https://static.carnap.io/css/tuftextra.css
---
```

will configure your document to have a tufte-like style, with no traces of bootstrap.

For more details about hosting your own stylesheets, please take a look at the documentation for the instructor dashboard.

4.3.3 Custom JavaScript

Documents can also include links to custom JavaScript. This can be done using the `raw_html` pandoc extension to include a `<script>` tag, or by including the JS in a metadata block. Like a CSS entry, a JS entry entry can include either a url for a single CSS stylesheet, like so:

```
---
js: https://carnap.io/shared/myemail@university.edu/myjs.js
---
```

or for several stylesheets, like so:

```
---
js:
- https://carnap.io/shared/myemail@university.edu/myjs1.js
- https://carnap.io/shared/myemail@university.edu/myjs2.js
---
```

The scripts can be hosted anywhere, including on the Carnap server. As with stylesheets, scripts can be uploaded as normal documents so long as they're given the proper filetype extension (in this case, ".js" rather than ".css")

4.3.4 Templates

Carnap can make use of pandoc's advanced templating capabilities, documented here. To create a template, upload a file with the extension `.template`. Your template file should contain HTML, enriched with template variables of the kind described in pandoc's documentation. To apply the template to a given document, include a `template` field in your document's YAML metadata, followed by the name of the template file. So for example, you might have metadata of the form:

```
---
js: https://carnap.io/shared/myemail@university.edu/myjs.js
template: test.template
---
```

Applied to an uploaded document `test.md`. Then, when `test.md` is accessed as a document or an assignment, its contents will be interpolated into `test.template` as the value of the `$body$` variable.

5 Community resources for Carnap

This document can be edited on GitHub

5.1 Assignment resources

- Carnap-Contrib - CSS stylesheets and JavaScript extensions for assignments

5.2 Textbooks

- forall x: UBC edition
- forall x: Calgary
- Introductory Logic, by Sean Walsh

5.3 Tools

- Carnap command line interface and GitHub Action:

  Automatically perform actions with the Carnap API such as uploading assignments

Part II

Exercises

6 Derivations

The `ProofChecker` class indicates that a code block will contain derivation exercises. The `Playground` class indicates that a code block will generate a "playground" in which instead of checking whether the proof establishes something set in advance, Carnap will figure out what the proof establishes and display it at the top of the proof-box. The formal systems used can be controlled using predefined classes, or by setting attributes on the code-block (see below).

6.1 Predefined classes

To get started, it's possible to create simple exercises in the propositional system of the Carnap book like this:

```
~~~{.ProofChecker .Prop}
1.1 P :|-: Q->P
~~~
```

Where the `:|-:` again indicates a turnstile. The result is then:

```
1.7 P:|-: Q->P
```

Similarly,

```
~~~{.Playground .Prop}
~~~
```

generates

```
```

The class `Prop` specifies that we wish to use the propositional formal system of the Carnap book, and specifies a couple of UI-defaults for displaying proofs in that system. The currently available predefined classes are:

Class Strings	Description
`.Prop`, `.FirstOrder`, `.SecondOrder`, `.PolySecondOrder`, `.MontagueSC`,`.MontagueQC`, `ElementaryST`, `SeparativeST`	Montague-Style systems, the first two of which are used in the Carnap book.
`.LogicBookSD`, `LogicBookSDPlus`, `LogicBookPD`, `.LogicBookPDPlus`	Fitch style systems based on Bergmann Moore and Nelson's *Logic Book*.
`.ForallxSL`, `.ForallxSLPlus`, `.ForallxQL`	Fitch-style systems based on Magnus's *Forall x*.
`.ZachTFL`, `.ZachTFL2019`, `.ZachFOL`, `.ZachFOL2019`, `.ZachFOLPlus2019`	Fitch-style systems based on the Calgary Remix of *Forall x*.
`.IchikawaJenkinsSL`, `.IchikawaJenkinsQL`	Fitch-style systems based on the Ichikawa-Jenkins Remix of *Forall x*.
`.GamutMPND`, `.GamutIPND`, `.GamutPND`, `.GamutPNDPlus .GamutND`,	Fitch-style systems based on the systems used in Gamut's *Introduction to Logic*, including minimal and intuitionistic fragments of propositional logic.
`HowardSnyderSL`,`HowardSnyderPL`	Systems based on Howard-Snyder's *The Power of Logic*.
`HausmanSL`,`HausmanPL`	Systems based on Hausman's *Logic and Philosophy*.
`.GoldfarbND`, `.GoldfarbAltND`, `.GoldfarbNDPlus`, `.GoldfarbAltNDPlus`	Lemmon-style systems based on Goldfarb's *Deductive Logic*.
`.TomassiPL`	Lemmon-style system based on Tomassi's *Logic*
`.HardegreeSL`, `.HardegreePL`, `.HardegreeWTL`, `.HardegreeL`, `.HardegreeK`, `.HardegreeT`, `.HardegreeB`, `.HardegreeD`, `.Hardegree4`, `.Hardegree5`, `.HardegreeMPL`	Hardegree-style systems based on Hardegree's *Modal Logic*
`.ZachPropEq`	Chain of equivalence prover.

6.2 Advanced Usage

6.2.0.1 Options

Like the other exercises, derivations allow for `points=VALUE` and `submission="none"`.

Derivations, however, currently have a little more depth than the other types of exercises. There are, correspondingly, more options available:

Name	Effect
`fonts`	Uses Fira Logic font, including ligatures for logical symbols
`popout`	Makes it possible to open the problem in a new window
`render`	Renders a picture of the proof as you type
`indent`	Automatically indents newlines to the level of the previous line
`tabindent`	Insert indent on tab press
`resize`	Automatically resizes proof area
`exam`	Allows for submission of work which is incomplete or incorrect

These can all be included in the "options" string supplied to the options attribute of the code block, like this:

```
~~~{.ProofChecker .Prop options="indent fonts popout resize render indent"}
1.8 P :|-: Q->P
|1.Show Q->P
|2.   Q:AS
|3.   P:PR
|4.:CD 3
~~~
```

The result of the above is:

```
1.8 P :|-: Q->P
|1.Show Q->P
|2.   Q:AS
|3.   P:PR
|4.:CD 3
```

An options string will override any options that are included by default in a given predefined class. For example, `.Prop` automatically turns on the `resize` option. However if we have just

```
~~~{.ProofChecker .Prop options="indent"}
1.9 P :|-: Q->P
~~~
```

We get a proofbox that resizes manually:

```
1.9 P :|-: Q->P
```

6.2.0.2 Partial Solutions

A partial solution to a problem can be included by following a problem with a derivation that is line-by-line prefixed with the `|` character, optionally (for readability) followed by a line number followed by a period, like this:

```
~~~{.ProofChecker .Prop}
1.7 P :|-: Q->P
|1.Show Q->P
|2.   Q:AS
|3.   P:PR
|4.:CD 3
~~~
```

This gives us:

```
1.7 P :|-: Q->P
|1.Show Q->P
|2.   Q:AS
|3.   P:PR
|4.:CD 3
```

Everything after the line number (or the `|` if you don't use numbers) is included in the proof, so be careful about spaces!

You can also include a partial derivation without specific solution, by replacing `.ProofChecker` with `.Playground`, writing for example:

```
~~~{.Playground .Prop}
|1.Show Q->P
|2.   Q:AS
|3.   P:PR
|4.:CD 3
~~~
```

which gives you a proof box that calculates what has been derived and displays it at the top, rather than calculating whether the derivation proves a given sequent, thus:

```
|1.Show Q->P
|2.   Q:AS
|3.   ?:PR
|4.:CD 3
```

6.2.0.3 Feedback

The defaults for feedback (check the proof with every keypress) can also be overridden by setting the feedback attribute. The override options for feedback are

Name	Effect
`manual`	require a button-press for feedback
`syntaxonly`	warn about syntax errors, but do not check proof for correctness
`none`	do not offer feedback

So,

```
~~~{.ProofChecker .Prop submission="none" feedback="manual"}
1.10 P :|-: Q->P
~~~
```

produces

```
1.10 P :|-: Q->P
```

6.2.0.4 Systems

If you don't want to use a predefined class, you can set the system option manually. This will give you complete control over which options are set—only the ones you select will be active. So you would write something like:

```
~~~{.ProofChecker system="prop" submission="none" feedback="manual"}
1.11 -Q :|-: Q->P
~~~
```

to produce

```
1.11 -Q :|-: Q->P
```

The available propositional systems are: `prop` `montagueSC` `LogicBookSD` `LogicBookSDPlus` `hausmanSL` `howardSnyderSL` `ichikawaJenkinsSL` `hausmanSL` `magnusSL` `magnusSLPlus` `thomasBolducAndZachTFL` `thomasBolducAndZachTFL2019` `gamutMPND` `gamutIPND`, `gamutPND` `gamutPNDPlus` `tomassiPL` and `hardegreeSL`. The available first-order systems are: `firstOrder` `montagueQC` `magnusQL` `thomasBolducAndZachFOL` `thomasBolducAndZachFOL2019`

`thomasBolducAndZachFOLPlus2019` `LogicBookPD` `LogicBookPDPlus` `gamutND` `hausmanPL` `howardSnyderPL` `ichikawaJenkinsQL` `hardegreePL` `goldfarbAltND` `goldfarbNDPlus` and `goldfarbAltNDPlus`. The available set theory systems are: `elementarySetTheory` and `separativeSetTheory`. The available second-order systems are: `secondOrder` and `PolySecondOrder`. The available propositional modal logic systems are: `hardegreeL` `hardegreeK` `hardegreeT` `hardegreeB` `hardegreeD` `hardegree4` and `hardegree5`. The available predicate modal logic system is `hardegreeMPL`, and the available "world theory" system is `hardegreeWTL`.

6.2.0.5 Initialization

The `init` attribute may be set to "now" in order to check the proof as soon as it is loaded, instead of waiting for a user interaction.

6.2.0.6 Indentation Guides

Besides just the indentation guide created by setting `guides` in the options string, there are several more refined overlays available for increasing the readability of a typed proof. These are configured by setting the `guides` attribute. The available options for guides are:

Name	Effect
`montague`	Montague style guides below show lines
`fitch`	A fitch style proof overlay
`hausman`	A Hausman style proof overlay
`howard-snyder`	A Howard-Snyder style proof overlay
`indent`	Simple indentation indicator guides

So,

```
~~~{.Playground .ForallxSL guides="fitch"}
 P       :AS
 P/\P    :&I 1 1
 P       :&E 2
P->P        :->I 1-3
~~~
```

Produces:

```
 P       :AS
 P/\P    :&I 1 1
 P       :&E 2
P->P     :->I 1-3
```

7 Truth Tables

The `TruthTable` class indicates that a code block will contain truth table exercises.

7.1 Simple Truth Tables

You can create simple truth table problems, checking to see if a formula is a tautology, by also adding the class `Simple`, like so:

```
~~~{.TruthTable .Simple}
2.1 ((P/\Q)\/R)<->((P\/R)/\(Q\/R))
~~~
```

the number 2.1 indicates the exercise number, and the formula is the one for which you will be constructing a truth table. This produces:

> 2.1 ((P/\Q)\/R)<->((P\/R)/\(Q\/R))

You can also use a comma-separated list of formulas, like so:

```
~~~{.TruthTable .Simple}
2.2 (P/\Q)\/R, (P\/R)/\(Q\/R)
~~~
```

This example produces:

> 2.2 (P/\Q)\/R, (P\/R)/\(Q\/R)

Simple truth tables can be (by default) checked for correctness, and will be considered correct when every row is filled in correctly. A problem can also be solved by indicating a counterexample to tautology. A row is considered a counterexample if *all* the formulas in the problem are false on that row.

7.2 Validity Truth Tables

If, instead of `Simple`, you add the class `Validity`, like so:

```
~~~{.TruthTable .Validity}
2.3 P :|-: ((P/\Q)\/R)<->((P\/R)/\(Q\/R))
~~~
```

the result is a truth-table for checking the validity of an argument (in this case, `P :|-: ((P/\Q)\/R)<->((P\/R)/\(Q\/R))`, where `:|-:` is just a stylized way of typing out the turnstile). The above produces:

> `2.3 P :|-: ((P/\Q)\/R)<->((P\/R)/\(Q\/R))`

Validity truth-table problems can include comma-separated lists of formulas to both the left and *right* of the turnstile. For example,

> `2.4 P,Q :|-: (P/\Q), (P\/R)/\(Q\/R)`

produces:

```
~~~{.TruthTable .Validity}
2.4 P,Q :|-: (P/\Q), (P\/R)/\(Q\/R)
~~~
```

Validity truth tables can be (by default) checked for correctness. A table is correct when every row has been filled in, with rows that are counterexamples to validity marked `F`, and rows that are not counterexamples to validity marked `T`. A counterexample to validity can also be provided directly by pressing the counterexample button and entering the truth values of the relevant row. A row is considered a counterexample to validity only if all the formulas to the left of the sequent are true, and *all* the formulas to the right of the sequent are false.[1]

7.3 Partial Truth Tables

You can also create truth table problems that involve filling in a single row. To do this, add the class `Partial`, like so:

[1]Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

```
~~~{.TruthTable .Partial}
2.5 (P/\Q), (P\/R)/\(Q\/R)
~~~
```

creating something like this:

> 2.5 (P/\Q), (P\/R)/\(Q\/R)

By default, the row is considered correct if it is filled in correctly, with any assignment of truth values to the atoms. However by using givens (see below), partial truth tables can be used to ask students to test for variety of different properties.

7.4 Advanced Usage

7.4.0.1 Options

In addition to setting a custom point value or turning off submission by adding `points=VALUE` and `submission="none"`, several other options are available for truth tables:

Name	Effect
`nocheck`	Disables the "check" button
`nocounterexample`	Disables the "counterexample" button
`exam`	Allows for submission of work which is incomplete or incorrect
`autoAtoms`	Prepopulates atomic sentence columns with truth values
`turnstilemark`	Uses and rather than `T` and `F` under the turnstile
`double-turnstile`	Displays a double turnstile (like so) rather than the single turnstile
`negated-double-turnstile`	Displays a negated double turnstile (like so) rather than the single turnstile
`immutable`	Makes the truth table immutable (useful for displaying truth tables)
`nodash`	Uses rather than `-` in empty cells (useful for printing worksheets)
`strictGivens`	Makes givens (see below) immutable
`hiddenGivens`	Hides givens (see below)

So for example,

```
~~~{.TruthTable .Validity options="nocheck nocounterexample"}
2.6 P :|-: Q
~~~
```

Generates:

2.6 P :|-: Q

7.4.0.2 Counterexamples

There are also a number of options that affect what counts as a counterexample. these are set using the `counterexample-to` attribute. The options are:

Name	Counterexample is
`validity`	a situation in which all formulas are false
`tautology`	same as `validity`
`equivalence`	a situation in which two formulas have different truth values
`inconsistency`	a situation in which all formulas are true
`contradiction`	same as `inconsistency`

In the case of simple truth tables, these apply to all formulas. In the case of validity truth tables, a counterexample is a situation in which the formulas to the left of the turnstile are all true, and the ones to the right of the turnstile have the counterexample property. Hence, a validity problem in which the counterexample is `equivalence` is basically using a truth table to test whether the formulas to the right of the turnstile are equivalent "under the assumption" that the formulas to the left of the turnstile are true.

Here's an example of a truth table looking for a counterexample to the equivalence of two formulas

```
```{.TruthTable .Simple counterexample-to="equivalence"}
2.7 P<->Q,  P->Q
```
```

which produces:

2.7 P<->Q, P->Q

7.4.0.3 Systems

The way that formulas are parsed and displayed can also be customized. This is done by setting the `system` attribute to indicate which formal system you are drawing your syntax from. So for example,

```
~~~{.TruthTable .Simple system="LogicBookSD"}
2.8 A > B & C
~~~
```

will generate:

> 2.8 A > B & C

The available systems are: `prop` `montagueSC` `LogicBookSD` `LogicBookSDPlus` `hausmanSL` `howardSnyderSL` `ichikawaJenkinsSL` `hausmanSL` `magnusSL` `magnusSLPlus` `thomasBolducAndZachTFL` `thomasBolducAndZachTFL2019` `tomassiPL` and `hardegreeSL`.

7.4.0.4 Givens

It is also possible to give a "partial solution" to a truth table problem, in which the truth table is partly filled in, and the student needs either to complete it or correct it. To pre-populate simple and validity tables with "givens" in this way, write the truth table you want, preceded by the bar character `|`, after the problem. So,

```
~~~{.TruthTable .Simple}
2.9 P \/~P
|   T - FT
|   F - TF
~~~
```

Generates

> 2.9 P \/~P
> | T - FT
> | F - TF

You do need to fill in every row entirely. If a row is the wrong length, or if there are the wrong number of rows, the table will not be pre-populated. If you wish to make the givens you assign immutable (to clarify that they are a hint, rather than something that needs to be corrected), you can use the `strictGivens` option.

Givens behave similarly for partial truth tables. For example,

```
~~~{.TruthTable .Partial}
2.10 P \/~P
|   F - TF
~~~
```

will produce:

```
2.10 P \/~P
|   F - TF
```

However: if a partial truth table is constructed with givens, then a solution will only be accepted if it is "consistent" with the givens. So in the above case, the only acceptable solution will be one that assigns `T` to `P`. The givens can be hidden, using `hiddenGivens` if you want to, for example, ask students to make a sentence truth and you want them to figure out the relation between the truth of a sentence and the truth value of the main connective.

if you wish to hide some givens, but not others, you can use a colon to separate the sentences that will have visible truth values from those that will not. For example, you can write

```
~~~{.TruthTable .Partial options="hiddenGivens"}
2.11 Q : Q->P
|    T   TT -
~~~
```

to create a problem in which the student must make `Q->P` true under the assumption that `Q` is true. The result is like this:

```
2.11 Q : Q->P
|    T   TT -
```

so this can also be used to create problems in which students are responsible for filling in a certain row of the truth table.

Finally, if more than one row of givens is provided to a partial truth table, then any solution which is compatible with *any one* of the rows will be accepted. So for example, you can write:

```
~~~{.TruthTable .Partial options="hiddenGivens"}
2.12 P/\Q, P\/Q
|    -T -  -F -
|    -F -  -T -
~~~
```

In order to ask students to provide a row that witnesses the inequivalence of these two sentences. The result will be:

```
2.12 P/\Q, P\/Q
|    -T -  -F -
|    -F -  -T -
```

7.4.0.5 Custom Marks

The marks for truth and falsity can be configured to something other than the usual `T` and `F` by setting the `falseMark` and `trueMark` attributes. So for example

````
```{.TruthTable .Validity trueMark="1" falseMark="0"}
2.13 P,Q:|-:P/\Q
```
````

will produce

```
2.13 P,Q:|-:P/\Q
```

The handling of givens remains the same in the presence of a configured `falseMark` or `trueMark` attribute. So,for example

````
```{.TruthTable .Validity trueMark="1" falseMark="0"}
2.14 P,Q:|-:P/\Q
| TT----
| FT----
| TF----
| FF----
```
````

will produce

```
2.14 P,Q:|-:P/\Q
| TT----
| FT----
| TF----
| FF----
```

8 Translations

the `Translate` class indicates that a code block will contain translation exercises that require symbolizing natural language sentences.

8.1 Propositional Logic

You can create propositional logic translation problems by also adding the class `Prop`, like so:

```
~~~{.Translate .Prop}
3.1 P/\Q : People want to know what's going on and questions are unavoidable
~~~
```

The number 3.1 indicates the exercise number, and the colon separates the solution from the text that will be presented for translation. The result of the above is:

> 3.1 P/\Q : People want to know what's going on and questions are unavoidable

To complete it, replace the text to the left of the submit solution button with your translation, press return to check, and then press "Submit Solution". Propositional translations are considered correct if they are logically equivalent to the original answer. So for example, `Q/\P` will be accepted above.

8.2 First-Order Translation

It is also possible to create first-order translation problems using the class `FOL`, thus:

```
~~~{.Translate .FOL}
3.2 AxF(x) : Everything is fine
~~~
```

with the result:

```
3.2 AxF(x) : Everything is fine
```

These are completed as above. Equivalence of first-order sentences is undecidable, so we can't check it, but we can catch most cases of "equivalent" translations by using some rewriting rules.[1] So, for example `~Ex~F(x)` will be accepted above.

8.3 Exact Translations

Using the class `Exact`, you can also create "translations" that don't accept logically equivalent answers. These may be useful if you wish to, for example, ask a student what the missing premise in some inference is. So, for example you might write

```
~~~{.Translate .Exact}
3.3 P : To make a modus ponens inference with P→Q, you need...
~~~
```

to generate:

```
3.3 P : To make a modus ponens inference with P→Q, you need...
```

Exact translations use the same syntax as `Prop` by default, but can be configured to use a large number of alternative syntaxes (see below)

8.4 Systems

The way that formulas are parsed can also be customized. This is done by setting the `system` attribute to indicate which formal system you are drawing your syntax from. So for example,

```
~~~{.Translate .FOL system="magnusQL"}
3.5 AxBx : Everything is bananas
~~~
```

will generate:

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

```
3.5 AxBx : Everything is bananas
```

For first-order translations, the available systems are: `firstOrder` `montagueQC` `magnusQL` `thomasBolducAndZachFOL` `thomasBolducAndZachFOL2019` `LogicBookPD` `LogicBookPDPlus` `hausmanPL` `howardSnyderPL` `ichikawaJenkinsQL` `hardegreePL` `goldfarbAltND` `goldfarbNDPlus` and `goldfarbAltNDPlus`.

For propositional translations, the available systems are: `prop` `montagueSC` `LogicBookSD` `LogicBookSDPlus` `hausmanSL` `howardSnyderSL` `ichikawaJenkinsSL` `hausmanSL` `magnusSL` `magnusSLPlus` `thomasBolducAndZachTFL` `thomasBolducAndZachTFL2019` `tomassiPL` and `hardegreeSL`.

For exact translations, the available systems are all of the above, together with modal logic systems `.HardegreeSL` `.HardegreePL` `.HardegreeWTL`, `.HardegreeL` `.HardegreeK` `.HardegreeT` `.HardegreeB` `.HardegreeD` `.Hardegree4` `.Hardegree5`, second order systems `.SecondOrder` `.PolySecondOrder`, and set theory systems `ElementaryST` and `SeparativeST`

8.5 Advanced usage

8.5.0.1 Multiple Solutions

If you wish to allow students to find one translation of a sentence that admits several formalizations, you can use a comma-separated list of admissible solutions. So,

```
~~~{.Translate .FOL}
3.4 (P /\ Q) \/ R, P/\(Q\/R) : Jack jumped the fence and was caught by the watchman or got a
~~~
```

generates

```
3.4 (P /\ Q) \/ R, P/\(Q\/R) : Jack jumped the fence and was caught by the watchman or got
```

8.5.0.2 Options and Attributes

In addition to allowing for custom point values with `points=VALUE`, and turning off submission with `submission="none"`, translations also have the following options

Name	Effect
`nocheck`	Disables checking solutions

Name	Effect
`exam`	Allows for submission of work which is incomplete or incorrect
`checksyntax`	When `exam` is active, blocks submission of syntactically incorrect work

These can be included in the space separated list supplied to the `options` attribute.

8.5.0.3 Translation tests

Finally, you can impose one or more extra tests on a translation. This is done by setting the `tests` attribute to indicate which tests you wish to require the translation to pass. The available tests for propositional translations are

Name	Effect
`CNF`	Requires conjunctive normal form
`DNF`	Requires disjunctive normal form
`maxCon:N`	Requires that the translation contain N or fewer connectives
`maxNot:N`	Requires that the translation contain N or fewer negations
`maxAnd:N`	Requires that the translation contain N or fewer conjunctions
`maxIff:N`	Requires that the translation contain N or fewer biconditionals
`maxIf:N`	Requires that the translation contain N or fewer conditionals
`maxOr:N`	Requires that the translation contain N or fewer disjunctions
`maxFalse:N`	Requires that the translation contain N or fewer falsity constants
`maxAtom:N`	Requires that the translation contain N or fewer atomic sentences

The available tests for first-order translations are all the propositional tests, plus:

Name	Effect
`PNF`	Requires prenex normal form

When using multiple tests, their names must be separated by spaces, so for example,

```
~~~{.Translate .FOL tests="PNF maxNeg:0"}
3.6 ~Ex~F(x) : Nothing is not bananas.
~~~
```

will generate:

```
3.6 ~Ex~F(x) : Nothing is not bananas.
```

8.5.0.4 Partial Solutions

It's possible to include a partial solution to a translation problem, by including the partial solution after a `|` following the problem. So for example,

```
~~~{.Translate .FOL options="nocheck"}
3.7 AxF(x) : Everything is fine
| For all x, x is fine
~~~
```

Generates

```
3.7 AxF(x) : Everything is fine
| For all x, x is fine
```

9 Countermodels

The `CounterModeler` class indicates that a code block will contain countermodel construction exercises.

9.1 Simple Countermodels

You can create simple countermodel problems, checking to see if one or more formulas are true in a constructed model like this:

````
```{.CounterModeler .Simple}
1.1 AxF(x), ExG(x)
```
````

The result will be:

```
1.1 AxF(x), ExG(x)
```

Here is an example with some more interesting vocabulary:

````
```{.CounterModeler .Simple}
1.2 AxAy(f(x,y) = f(y,x))
```
````

Producing:

```
1.2 AxAy(f(x,y) = f(y,x))
```

The domain of the model must consist of one or more natural numbers. The tuples making up the extensions of predicate and relation symbols are comma-separated and enclosed in brackets, parentheses, or `<>` pairs, like this: `[0,0]` or this `<1,1>`. The tuples themselves should be separated by commas, like this: `[0,0],[1,0]`.

The extension of each constant must be a natural number. The extension of a given function symbol will be a comma-separated list of tuples as above but with the arguments separated from the value by a semicolon, thus: `[0,0;1]`.

Formulas containing free variables are treated as equivalent to their universal closures.

9.2 Validity Countermodels

You can create a countermodel problem for showing invalidity like this:

````
```{.CounterModeler .Validity}
1.3 AxEyF(x,y) :|-: ExAyF(y,x)
```
````

Creating:

```
1.3 AxEyF(x,y) :|-: ExAyF(y,x)
```

Validity countermodel problems can include comma-separated lists of formulas to both the left and *right* of the turnstile. A countermodel is successful if it makes all the formulas to the left of the turnstile true and all the formulas to the right of the turnstile false.

9.3 Constraint Countermodels

You can create a countermodel with some implicit constraints on the allowable models like this:

````
```{.CounterModeler .Constraint }
1.4 ExEy(not x = y) : AxAyF(x,y)
```
````

Resulting in:

```
1.4 ExEy(not x = y) : AxAyF(x,y)
```

A countermodel with implicit constraints is successful if it makes all the formulas to the right and the left of the `:` true.

9.4 Advanced Usage

9.4.0.1 Options

In addition to setting a custom point value or turning off submission by adding `points=VALUE` and `submission="none"`, several other options are available for countermodels:

Name	Effect
`nocheck`	Disables the "check" button
`exam`	Allows for submission of work which is incomplete or incorrect
`strictGivens`	Makes givens immutable (see below)
`double-turnstile`	Displays a double turnstile (like so) rather than the single turnstile
`negated-double-turnstile`	Displays a negated double turnstile (like so) rather than the single turnstile

9.4.0.2 Counterexamples

There are also a number of options that affect what counts as a successful countermodel. these are set using the `counterexample-to` attribute. The options are:

Name	Counterexample is
`validity`	a situation in which all formulas are false
`tautology`	Same as `validity`
`equivalence`	a situation in which two formulas have different truth values
`inconsistency`	a situation in which all formulas are true

In the case of simple countermodels, these apply to all formulas. In the case of validity countermodels, a successful countermodel is a situation in which the formulas to the left of the turnstile are all true, and the ones to the right of the turnstile have the counterexample property. Similarly in the case of a constraint countermodel problem—all the constraints must be true, and the other formulas must have the counterexample property.

9.4.0.3 Systems

The way that formulas are parsed and displayed can also be customized. This is done by setting the `system` attribute to indicate which formal system you are drawing your syntax from. So for example,

~~~{.CounterModeler .Simple system="LogicBookPD"}
1.5 (Ax)(Ax->Bx)
~~~

will generate:

> 1.5 (Ax)(Ax->Bx)

The available systems are `firstOrder` `montagueQC` `magnusQL` `thomasBolducAndZachFOL` `thomasBolducAndZachFOL2019` `thomasBolducAndZachFOLPlus` `LogicBookPD` `LogicBookPDPlus` `hausmanPL` `howardSnyderPL` `ichikawaJenkinsQL` `hardegreePL` `goldfarbAltND` `goldfarbNDPlus` and `goldfarbAltNDPlus`.

9.4.0.4 Givens

It is also possible to give a "partial solution" to a countermodeling problem, in which the model is partly filled in, and the student needs either to complete it or correct it. To pre-populate models with "givens" in this way, write each field you want filled in, preceded by the bar character `|` and followed by a colon and what you want it filled in with, like this:

```
```{.CounterModeler .Simple}
1.1 AxF(x), ExG(x)
| Domain : 0,1,2
| F(_) : 1,2
| a : 1
```
```

Which will produce:

> 1.1 AxF(x), ExG(x), H(a)
> | Domain : 0,1,2
> | F(_) : 1,2
> | a : 1

The field-names for the givens should match how the field-names would be displayed in the exercise.

Ordinarily, givens function as hints. However, you can make it impossible for students to change them (thereby turning them into requirements) by adding the `strictGivens` option to your problem. Use this, for instance, to require students to use a specific domain, or to prevent them from making universal sentences vacuously true.

10 Gentzen-Prawitz Natural Deduction

The `TreeDeduction` class indicates that a code block will contain Gentzen-Prawitz natural deduction exercises, which require the production of a Gentzen-Prawitz deduction tree.

These problems use ProofJS for their user interface. An initial click may be required to select a node. Once a node is selected, `Enter` will create a sibling premise (on any node but the root), `Ctrl-Enter` will create a new premise above the focused node, and `Ctrl-Shift-Enter` will create a new conclusion node below the focused node (on any node but the root node). Nodes can be selected by either pressing `Tab` and `Shift-Tab` to cycle, or by using the mouse. Changes can be undone and redone with `Ctrl-Z` and `Shift-Ctrl-Z` respectively. The subtree above a selected node can be deleted with `Ctrl-Backspace`, and subtrees can be cut-copy-pasted with `Shift-Ctrl-X`, `Shift-Ctrl-C` and `Shift-Ctrl-V` respectively.

10.1 Available Systems

At the moment, four systems are available:

System	Description
propNK	A system based on the propositional fragment of Gentzen's NK
propNJ	A system based on the propositional fragment of Gentzen's NJ
openLogicNK	The propositional fragement of the Open Logic project's natural deduction
openLogicFOLNK	The full (first-order with equality) Open Logic project natural deduction
openLogicArithNK	openLogicFOLNK for the language of arithmetic
openLogicExArithNK	openLogicFOLNK for the language of arithmetic with arbitrary predicates and functic
openLogicSTNK	openLogicFOLNK for the basic language of set theory with arbitrary predicates and fu
openLogicExSTNK	openLogicFOLNK for the basic language of set theory
openLogicESTNK	openLogicFOLNK for an extended language of set theory
openLogicExESTNK	openLogicFOLNK for an extended language of set theory with arbitrary predicates an
openLogicSSTNK	openLogicFOLNK for an extended language of set theory with separation abstracts
openLogicExSSTNK	openLogicFOLNK for an extended language of set theory with separation abstracts an

Exercises are given by specifying the system, and the sequent to be proved. So an exercise can be constructed like so:

```{.TreeDeduction .propNK}
1.1 P\/Q, ~P :|-: Q
```

which produces:

```
1.1 P\/Q, ~P :|-: Q
```

Instead of `.propNK` etc, you can also use `system="propNK"`.

(Remember to click on a node in order to interact, and to press `Ctrl-Enter` to create the first child node)

A completed proof will look like this:

```
1.2 :|-: (P->Q)->((R->P)->(R->Q))
| {
|   "label": "(P->Q)->((R->P)->(R->Q))",
|   "rule": "->I(1)",
|   "ident": 0,
|   "forest": [
|     {
|       "label": "(R->P)->(R->Q)",
|       "rule": "->I(2)",
|       "ident": 1,
|       "forest": [
|         {
|           "label": "R->Q",
|           "rule": "->I(3)",
|           "ident": 2,
|           "forest": [
|             {
|               "label": "Q",
|               "rule": "->E",
|               "ident": 3,
|               "forest": [
|                 {
|                   "label": "P->Q",
|                   "rule": "(1)",
|                   "ident": 4,
|                   "forest": [
|                     {
|                       "label": "",
```

```
                    "rule": "",
                    "ident": 5,
                    "forest": []
                  }
                ]
              },
              {
                "label": "P",
                "rule": "->E",
                "ident": 6,
                "forest": [
                  {
                    "label": "R->P",
                    "rule": "(2)",
                    "ident": 7,
                    "forest": [
                      {
                        "label": "",
                        "rule": "",
                        "ident": 8,
                        "forest": []
                      }
                    ]
                  },
                  {
                    "label": "R",
                    "rule": "(3)",
                    "ident": 9,
                    "forest": [
                      {
                        "label": "",
                        "rule": "",
                        "ident": 10,
                        "forest": []
                      }
                    ]
                  }
                ]
              }
            ]
          }
```

```
|           ]
|         }
|       ]
|     }
|   ]
| }
```

With rule names to the right of inference lines, and assumptions labeled to the right of the rule citation (with or without parentheses). Discharged assumptions are marked using an inference with empty premise, and the assumption label on its own to the right of the inference line.

10.2 Rules for `.propNJ` and `.propNK`

Here's a brief summary of NJ's propositional rules. The notation [ ]/ indicates that an assumption can be discharged from the subproof establishing

Rule	Premises	Conclusion
I	,	
E		OR
I		OR
E	, [ ]/ , [ ]/	
→I	[ ]/	→
→E	, →	
¬I	[ ]/	¬
¬E	, ¬	
¬E		

NK results from the addition of one more rule:

Rule	Premises	Conclusion
LEM		¬

The syntax of formulas accepted for is that for the propositional systems for Kalish & Montague/The Carnap Book in the Systems Reference.

10.3 Rules for Open Logic Systems

For the systems `.openLogicNK`, etc., the rules are:

Rule	Premises	Conclusion
`I`	,	
`E`		OR
`I`		OR
`E`	, [ ]/ , [ ]/	
`→I`	[ ]/	→
`→E`	, →	
`I`	[ ]/ , [ ]/	
`E`	,	
	,	
`¬I`	[ ]/	¬
`¬E`	, ¬	
`X`		
`IP`	[¬]/	

For the first order systems, we also have the rules:

Rule	Premises	Conclusion
`I`	(a)	x (x)
`VE`	x (x)	(t)
`I`	(t)	x (x)
`E`	x (x), [(a)]/	
`=I`		t=t
`=E`	(t),t=s	(s)

The syntax of accepted for the Open Logic systems is described in the Systems Reference. The natural deduction systems for arithmetic and set theory only differ in the syntax; there are no axioms.

Here is an example of a derivation in the language of arithmetic:

```
|{"ident":7,"label":"Ax ~ x < 0","rule":"AI","forest":[{"ident":11,"label":"~ a < 0","rule
```

10.4 Advanced Usage

10.4.1 Options

In addition to the standard `points=VALUE` and `submission="none"` options, Gentzen-Prawitz natural deduction exercises allow for you to set `init="now"` to have proofchecking begin as soon as the proof is loaded (rather than waiting for input) as well as the following allowed arguments to `options="…"`:

Option name	Effect
`prepopulate`	Prepopulates the endformula of an exercises with the conclusion to be shown
`displayJSON`	`Ctrl-?` will toggle display of an editable JSON representation of the proof

10.4.2 Runtime Axioms and Rules

The "extended" mathematical systems, `openLogicExArithNK` `openLogicExSTNK` `openLogicExESTNK` and `openLogicExSSTNK` can be equipped with extra axioms and rules. To set an axiom or a rule for a system, include an option of the form `axiom-NAME="RULEVARIANTS"` where `NAME` is the name that you want your axiom or rule to have, and `RULEVARIANTS` is a semicolon-separated list of sequents representing the variant schematic forms of the rule. The rule should be cited as `Ax-NAME`. Axiom names are case-insensitive, so don't rely on upper or lower-case variants of names to distinguish rules.

When entering the schematic sequents representing the forms of a rule, you should indicate which sentence letters, constants, and function symbols are to be read as schematic by preceding each such symbol with a prime, like so: `'P`. So, for example:

```
~~~{.TreePlayground .openLogicExArithNK init="now" axiom-flip="'P('a*'b) :|-: 'P('b*'a); 'P(
{ "ident": 13, "label": "a+b=c", "rule": "Ax-flip", "forest": [
    { "ident": 15, "label": "b+a=c", "rule": "", "forest": [] }
]}
~~~
```

will produce:

```
{ "ident": 13, "label": "a+b=c", "rule": "Ax-flip", "forest": [
    { "ident": 15, "label": "b+a=c", "rule": "", "forest": [] }
]}
```

10.4.3 JSON Serialization

Here's an incomplete proof, showing how to use the `displayJSON` option:

```
1.1 P\/Q, ~P :|-: Q
| {"ident":12,"label":"Q","rule":"\\/E", "forest":[
|   {"ident":13,"label":"P\\/Q","rule":"", "forest":[]},
|   {"ident":14,"label":"Q","rule":"(1)",  "forest":[
|       {"ident":17,"label":"","rule":"","forest":[]}
|   ]},
|   {"ident":15,"label":"Q","rule":"?","forest":[
|       {"ident":18,"label":"?","rule":"","forest":[]}
|   ]}
| ]}
```

To show the JSON representation, click to focus one of the input areas in the proof and press `Ctrl-?`. To edit the deduction by editing the JSON, try replacing one of the `Q`s in the JSON panel with a `P`. The deduction will update to reflect the JSON, so long as the JSON is a well-formed representation of a deduction.

The above was generated with

````
```{.TreeDeduction .propNK init="now" options="displayJSON"}
1.1 P\/Q, ~P :|-: Q
| {"ident":12,"label":"Q","rule":"\\/E", "forest":[
|   {"ident":13,"label":"P\\/Q","rule":"", "forest":[]},
|   {"ident":14,"label":"Q","rule":"(1)",  "forest":[
|       {"ident":17,"label":"","rule":"","forest":[]}
|   ]},
|   {"ident":15,"label":"Q","rule":"?","forest":[
|       {"ident":18,"label":"?","rule":"","forest":[]}
|   ]}
| ]}
```
````

The `displayJSON` option is useful for saving and communicating proofs, since one can reproduce a proof by pasting its JSON representation into the panel where the JSON representation is displayed. It's also useful for creating exercises in which the problems are partially completed, since, as in the example above, one can prefill an exercise by supplying a JSON representation below the statement of the problem.

10.4.4 Playgrounds

One can generate a Gentzen-Prawitz playground (where there is no goal, but where what you've proved is displayed) using something like the following:

````
```{.TreePlayground .propNK init="now" options="displayJSON"}
| {"ident":12,"label":"Q","rule":"\\/E (1) (2)","forest":[
|   {"ident":13,"label":"P\\/Q","rule":"","forest":[]},
|   {"ident":14,"label":"Q","rule":"(1)","forest":[
|       {"ident":17,"label":"","rule":"","forest":[]}
|   ]},
|   {"ident":15,"label":"Q","rule":"-E","forest":[
|       {"ident":18,"label":"!?","rule":"-E","forest":[
|           {"ident":24,"label":"P","rule":"(2)","forest":[
|               {"ident":27,"label":"","rule":"","forest":[]}
|           ]},
|           {"ident":25,"label":"-P","rule":"","forest":[]}
|       ]}
|   ]}
| ]}
```
````

The result, in this case, will be:

```
| {"ident":12,"label":"Q","rule":"\\/E (1) (2)","forest":[
|   {"ident":13,"label":"P\\/Q","rule":"","forest":[]},
|   {"ident":14,"label":"Q","rule":"(1)","forest":[
|       {"ident":17,"label":"","rule":"","forest":[]}
|   ]},
|   {"ident":15,"label":"Q","rule":"-E","forest":[
|       {"ident":18,"label":"!?","rule":"-E","forest":[
|           {"ident":24,"label":"P","rule":"(2)","forest":[
|               {"ident":27,"label":"","rule":"","forest":[]}
|           ]},
|           {"ident":25,"label":"-P","rule":"","forest":[]}
|       ]}
|   ]}
| ]}
```

Try editing the proof to see how the displayed sequent changes!

11 Sequent Calculus Deductions

The `Sequent` class indicates that a code block will contain sequent-calculus deduction exercises, which require the production of a sequent calculus deduction.

These problems use ProofJS for their user interface. An initial click may be required to select a node. Once a node is selected, `Enter` will create a sibling node (on any node but the root), `Ctrl-Enter` will create a new child node, and `Ctrl-Shift-Enter` will create a new parent node (on any node but the root node). Nodes can be selected by either pressing `Tab` and `Shift-Tab` to cycle, or by using the mouse. Changes can be undone and redone with `Ctrl-Z` and `Shift-Ctrl-Z` respectively. The subtree above a selected node can be deleted with `Ctrl-Backspace`, and subtrees can be cut-copy-pasted with `Shift-Ctrl-X`, `Shift-Ctrl-C` and `Shift-Ctrl-V` respectively.

11.1 Available Systems

At the moment, four systems are available:

System	Description
propLK	A system based on the propositional fragment of Gentzen's LK
propLJ	A system based on the propositional fragment of Gentzen's LJ
foLK	A system based on the full first order system of LK
foLJ	A system based on the full first order system of LJ

Exercises are given by specifying the desired endsequent. So an exercise can be constructed like so:

```
```{.Sequent .propLK}
1.1 P->Q, Q->R :|-: P->R
```
```

which produces:

```
1.1 P->Q, Q->R :|-: P->R
```

(Remember to click on a node in order to interact, and to press `Ctrl-Enter` to create the first child node)

A completed proof will look like this:

```
1.2 P, P->Q :|-: Q
| {"ident":11,"label":"P->Q,P:|-:Q","rule":"->L","forest":[{"ident":12,"label":"Q,P:|-:Q",
```

With `:|-:` indicating a turnstile, and rule names to the right of inference lines.

11.2 Rules for LK and LJ

Here's a brief summary of LK's propositional "operational" rules:

Rule	Premise Sequents	Conclusion Sequent
`&R`	$\Gamma \quad \Delta, \quad ; \Gamma \quad \Delta, \quad$	$\Gamma \quad \Delta, \quad \&$
`&L`	$\quad , \Gamma \quad \Delta$	$\quad \& \quad , \Gamma \quad \Delta$
`&L`	$\quad , \Gamma \quad \Delta$	$\quad \& \quad , \Gamma \quad \Delta$
`L`	$\quad , \Gamma \quad \Delta \; ; \quad , \Gamma \quad \Delta$	$\quad , \Gamma \quad \Delta$
`R`	$\Gamma \quad \Delta,$	$\Gamma \quad \Delta,$
`R`	$\Gamma \quad \Delta,$	$\Gamma \quad \Delta,$
`→L`	$\Gamma \quad \Delta, \; ; \quad , \Gamma \quad \Delta$	$\quad \rightarrow \quad , \Gamma \quad \Delta$
`→R`	$\quad , \Gamma \quad \Delta,$	$\Gamma \quad \Delta, \quad \rightarrow$
`¬L`	$\Gamma \quad \Delta,$	$\neg \quad , \Gamma \quad \Delta$
`¬R`	$\quad , \Gamma \quad \Delta$	$\Gamma \quad \Delta, \neg$

Some structural rules (interchange, and contraction) are applied automatically to the parts of the sequent that match the Γ and Δ, so there is some room for a little bit of laxness there.[1]

The following explicit structural inferences are allowed:

Rule	Premise Sequents	Conclusion Sequent
`CR`	$\Gamma \quad \Delta$	$\Gamma \quad \Delta$
`CL`	$\Gamma \quad \Delta$	$\Gamma \quad \Delta$

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

Rule	Premise Sequents	Conclusion Sequent
`XR`	Γ Δ	Γ Δ
`XL`	Γ Δ	Γ Δ
`WR`	Γ Δ	Γ Δ,Δ'
`WL`	Γ Δ	Γ,Γ' Δ
`Cut`	, Γ Δ ; Γ' Δ',	Γ,Γ' Δ,Δ'
`Ax`		

The contraction and exchange rules are actually redundant here, because of the fact that structural rules are applied automatically to the formulas substituted in for Γ,Δ. They're included here for presentation. Weakening is modified to allow for an arbitrary number of new formulas to be added.

The first order system for LK adds the following rules to the above:

Rule	Premise Sequents	Conclusion Sequent
`L`	( ), Γ Δ	( ), Γ Δ
`L`	( ), Γ Δ	( ), Γ Δ
`R`	Γ Δ, ( )	Γ Δ, ( )
`R`	Γ Δ, ( )	Γ Δ, ( )

The usual eigenvariable restrictions apply to `L` and `R`: the term must be a constant or variable that does not occur anywhere in the conclusion of the inference (i.e. not in Γ,Δ, or ( )).

The propositional and first-order systems for LJ are just like those for LK with the additional restriction that at most one formula can occur on the right hand side of a sequent.

11.3 Syntax for LK and LJ

As usual, in writing formulas and inference rules, any of `&`, or `/\` can be used for conjunction, and any of , `v`, and `\/` can be used for disjunction. Any of `->` or → can be used for the conditional, and any of ¬, `-`, or `~` for negation. Sentence letters are `P` through `W` or `P` with subscripts like so: `P_1`. Predicates are `F` through `L`, or `F` with a subscript. Variables are `v` through `z` or `x` with a subscript and constants are `a` through `e` or `c` with a subscript.

11.4 Advanced Usage

11.4.1 Options

In addition to the standard `points=VALUE` and `submission="none"` options, sequent calculus exercises allow for you to set `init="now"` to have proofchecking begin as soon as the proof is loaded (rather than waiting for input) as well as the following options:

Name	Effect
`displayJSON`	`Ctrl-?` will toggle display of an editable JSON representation of the proof

11.4.2 JSON Serialization

Proof 1.2 above was generated using the displayJSON option, with the following invocation:

````
```{.Sequent .propLK init="now" options="displayJSON"}
1.2 P, P->Q :|-: Q
| {"ident":11,"label":"P->Q,P:|-:Q","rule":"->L","forest":[{"ident":12,"label":"Q,P:|-:Q","ru
```
````

The `displayJSON` option is useful for saving and communicating proofs, since one can reproduce a proof by pasting its JSON representation into the panel where the JSON representation is displayed. It's also useful for creating exercises in which the problems are partially completed, since, as in the example above, one can prefill an exercise by supplying a JSON representation below the statement of the problem (beginning each line of the JSON with a bar character `|`).

11.4.3 Playgrounds

You can generate a sequent calculus playground (where there is no goal, but where what you've proved is displayed) using something like the following:

````
```{.SequentPlayground .propLK init="now" options="displayJSON"}
{"ident":11,"label":"P->Q,P:|-:Q","rule":"->L","forest":[{"ident":12,"label":"Q,P:|-:Q","rule
```
````

Or like:

```{.SequentPlayground .propLK init="now" options="displayJSON"}
```

The result, in the first case, will be:


```
{"ident":11,"label":"P->Q,P:|-:Q","rule":"->L","forest":[{"ident":12,"label":"Q,P:|-:Q","r
```


and in the second will be:


```
```


Try editing each proof, and see how the displayed sequent changes!

12 Qualitative Problems

the `QualitativeProblem` class indicates that a code block will contain qualitative problems, either short-answer or multiple choice questions.

12.1 Short Answer

To create a short-answer problem, use the class `ShortAnswer`, and simply provide a short answer question, like so:

````
```{.QualitativeProblem .ShortAnswer}
1.1 How are you feeling today?
```
````

The number indicates the exercise number. This is immediately followed by the question. The result will be:

> 1.1 How are you feeling today?

Student submissions to short-answer prompts will be recorded, but by default will not be assigned any credit. They will, however, be visible in the review section for the assignment, so it will be possible to assign them partial credit, as if they were an exam problem. The behavior for awarding credit can also be configured with the `give-credit` option, see Advanced Usage below.

12.2 Multiple Choice

To create a multiple-choice problem, use the class `MultipleChoice`, and provide a question along with a set of answers (one per line), like so:

````
```{.QualitativeProblem .MultipleChoice}
1.2 How are you feeling today?
| Good!
````
````

```
| +OK!
| Meh.
```

The answers should be typed as you want them to be displayed. Carnap doesn't currently render formulas or LaTeX within multiple choice answers, and HTML escape codes like `&or;` will cause some trouble for the answer-parser. If you want to include a formula with symbols, just use the symbols directly in the answer. You can copy-paste from here: ¬,→, , , , .

Answers marked with a `+` or a `*` will be accepted as correct. Answers marked with a `-` or a `+` will be pre-selected. So the result of the above is:

```
1.2 How are you feeling today?
| Good!
| +OK!
| Meh.
```

Which has "OK!" preselected and just one correct answer, namely "OK!".

12.3 Multiple Selection

To create a multiple-selection problem, use the class `MultipleSelection`,

To create a multiple-choice problem, use the class `MultipleChoice`, and provide a question along with a set of answers (one per line), like so:

```
```{.QualitativeProblem .MultipleSelection}
1.2 How are you feeling today?
| - Bad!
| + OK!
| * I contain multitudes.
```
```

As above, answers should be typed as you want them displayed.

Answers marked with a `+` or a `*` must be selected for a correct answer, and other answers must not be selected. Answers marked with a `-` or a `+` will be pre-selected. So the result of the above is:

```
1.2 How are you feeling today?
| - Bad!
| + OK!
```

```
| * I contain multitudes.
```

12.4 Numerical

To create a numerical problem, use the class `.Numerical` and provide a question and answer, in the format `ANSWER : QUESTION`, optionally pre-filling with hint written below the question, like so:

````
```{.QualitativeProblem .Numerical}
1.3 8 : How many bits in a byte?
| 2^3
```
````

The above produces:

```
1.3 8 : How many bits in a byte?
| 2^3
```

Answers must be given as decimal expressions or fractions (so, the above 2^3 won't be accepted without being evaluated). They'll be saved as decimal expressions.

12.5 Advanced Usage

12.5.0.1 Options and Attributes

In addition to allowing for turning off submission with `submission="none"`, qualitative problems also have the following options, which can be configured using the `options` field, for example setting `options="exam"` or `options="exam check"`. See the general pandoc documentation for more details.

Name	Effect
`exam`	Allows for submission of work which is incomplete or incorrect
`check`	Adds a button which checks results without submitting

This won't have any effect on a short-answer question, since there's no notion of "correct" to apply. Qualitative questions also permit assigning custom point values with the `points=VALUE`, like most other graded problems.

Short answer questions also accept a `give-credit` option, which has the following available settings

Name	Effect
`onSubmission`	Counts all submissions as correct

so, adding `give-credit="onSubmisson"` to a short answer question will award credit automatically for any submission.

For more complicated formatting of multiple choice and multiple selection options, you can also use the `content-format` option, which has the following available settings

Name	Effect
`html`	Applies html tags that appear in the answer options

So, for example, one can write:

````
```{.QualitativeProblem .MultipleChoice options="check" content-format="html"}
1. "Truth" is:
| a <b>propery</b> of propositions
| a logical <b>object</b>
| *A five-letter word
```
````

to generate

```
1. "Truth" is:
| a <b>propery</b> of propositions
| a logical <b>object</b>
| *A five-letter word
```

13 Truth Trees (Semantic Tableaux)

Carnap supports making truth trees. A sample problem with one follows:

```
1.1 A&B,~(C\/D),(~B\/C)\/E,~E
```

Note that there is unfortunately a bug where long lines get cut off. When you begin solving the exercise, it will fix itself.

The source for this sample is this:

````
```{.TruthTree .Prop}
1.1 A&B,~(C\/D),(~B\/C)\/E,~E
```
````

Here is a screenshot of the solved version:

Description of the above truth tree

There's a downward continuation from line 4 with two lines, "A", line 5, and "B", line 6, with "1" as the row and "&" as the rule in the right margin.

There's another with "~C", line 7, and "~D", line 8, following from row 2, with rule "~/".

There's a split branch on line 9 from line 3 into "~B C" and "E" with the rule "/". "E" on the right branch of line 9 is marked as a contradiction between lines 9 and 4.

The left branch of line 9 further branches into "~B" and "C". "~B" is marked as a contradiction between lines 6 and 10, and "C" is marked as a contradiction between lines 7 and 10.

To solve it, the student should do the following:

1. Right click on line 1 ("A&B"), select "Continue branch with 2 formulas". Enter "A" as the first and "B" as the second. Row is 1, and rule is "&".
2. Right click on line 1, select "Mark as resolved" to put a checkmark on the line.
3. Right click on line 2 ("~(C D)"), select "Continue branch with 2 formulas". Enter "~C" as the first, "~D" as the second. Row is 2, rule is "~/".
4. Mark line 2 as resolved as above.
5. Right click on line 3, select "Split branch with 1 formula" and enter "~B C" in the left box and "E" in the right box. Row is 3, rule is "/".

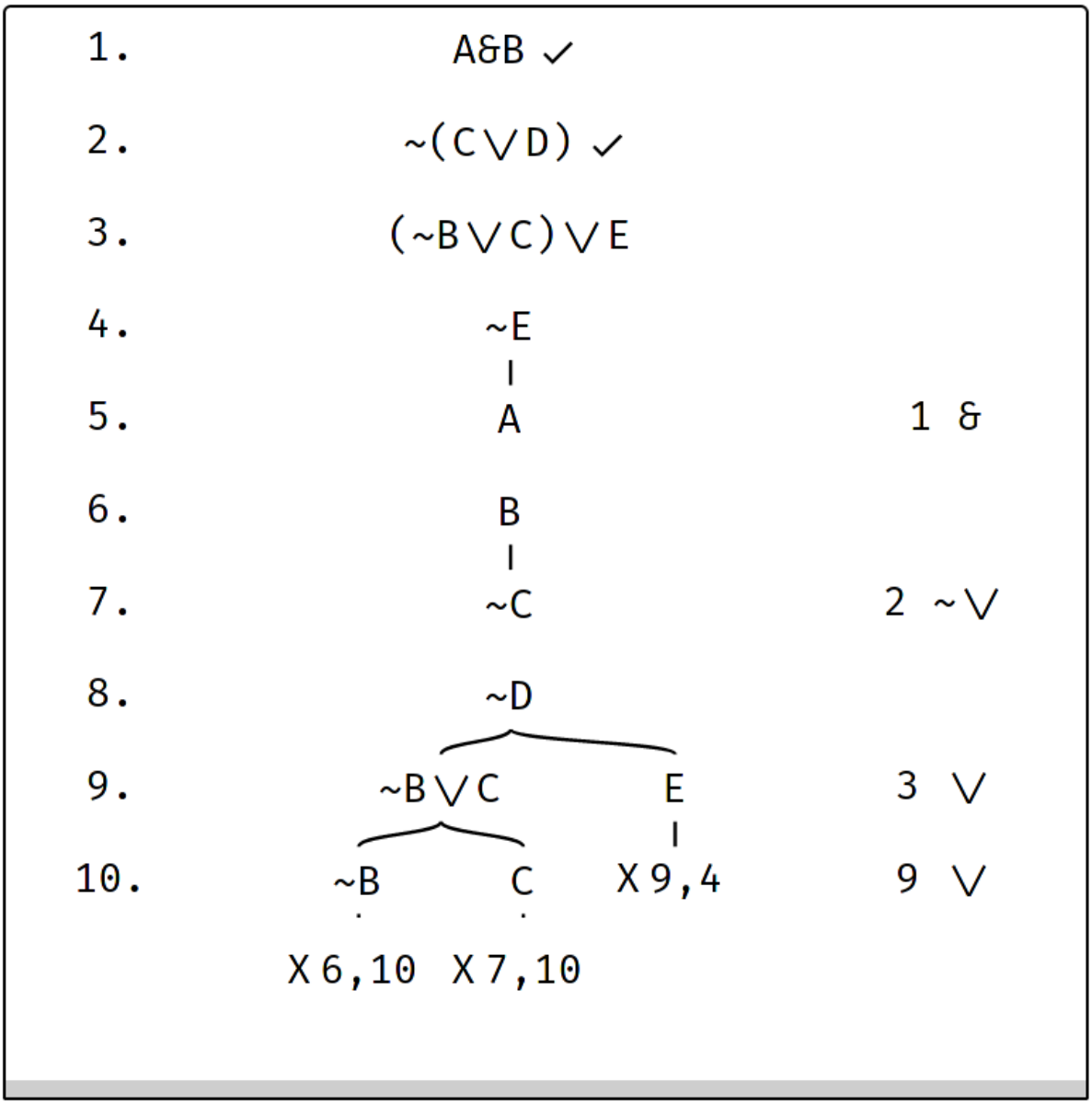


Figure 13.1: screenshot of the solved truth tree, described next

6. Right click the right branch “E” on line 9, select “Mark branch as contradiction” and enter “9,4” in the box.
7. Right click the left branch “~B C” and select “Split branch with 1 formula” and enter “~B” in the left box, “C” in the right box. Row is 9, rule is “/”.
8. Right click “~B” on line 10 to close it as a contradiction with line 6. Enter “6,10” in the box.
9. Right click “C” on line 10 to close it as a contradiction with line 7. Enter “7,10” in the box.
10. Mark line 3 as resolved.

13.1 Setting another system

By default, truth trees operate on Ichikawa-Jenkins SL. You can change this to something else using the `checkFunction` attribute (default value is `"checkIchikawaJenkinsSLTableau"`).

For example,

````
```{.TruthTree .Prop checkFunction="checkIchikawaJenkinsQLTableau"}
1.1 P,P->Q,~Q
```
````

14 Syntax Check Exercises

Chapter 1 of the Carnap book includes exercises that require identifying the main connective of a formula and of its sub-formulas in order to break that formula down into atomic sentences. This helps students learn how sentences are structured by connectives.

To add a formula-parsing exercise of this kind, use the classes `SynCheck` (to indicate that you're checking syntax) and `Match` to indicate that you want to display as many parentheses as possible.

So, following text:

```
~~~{.SynChecker .Match}
1.1 P /\ Q /\ R
~~~
```

will generate:

> 1.1 P /\ Q /\ R

`1.1` is a problem number that will be used to save the problem when the student submits it, and `P /\ Q /\ R` is the formula to be parsed.

In Chapter 2 of the Carnap book, this exercise is repeated, but with some parentheses omitted by the precedence rules for the Boolean connectives. For a formula parsing exercise more like the chapter 2 problems, just change `Match` to `MatchClean`, so that you write:

```
~~~{.SynChecker .MatchClean}
1.2 P /\ Q /\ R
~~~
```

The result will be

> 1.2 P /\ Q /\ R

14.1 Syntax Check Options

You can require explicit "parsing of atoms" (pressing return with when an atom is highlighted to acknowledge that it contains no connectives) by including adding `parseAtoms` in the options attribute. So for example,

```
~~~{.SynChecker .MatchClean submission="none" options="parseAtoms"}
1.3 P->Q
~~~
```

Generates

```
1.3 P->Q
```

Part III

Supported Systems

Carnap supports multiple "systems", i.e., languages with different symbols and syntax conventions, in the various types of exercises. For derivation exercises, a system implies a derivation system (set of rules, and derivation style). For truth table, translation, and counter-modeler exercises, the system implies a parser and a formula renderer, i.e., it implies which formulas are accepted as correct, how to parse them, and how to render formulas when they are displayed by Carnap.

All sentence letters, predicate symbols, constants, and function symbols (if allowed) take subscripts (e.g., `P_1` for P_1). Predicate and function symbols also take superscripts (e.g., `P^1` for P^2) to indicate arity. The parser does not enforce the arity, i.e., the arity is always determined by the number of arguments actually given.

The systems supported are:

- Bergmann, Moore & Nelson, *The Logic Book*
- Bonevac, *Deduction*
- Gallow, *forall x: Pittsburgh*
- Gamut, *Logic, Language, and Meaning*
- Goldfarb, *Deductive Logic*
- Hardegree, *Symbolic Logic* and *Modal Logic*
- Hausman, Kahane & Tidman, *Logic and Philosophy*
- Howard-Snyder, Howard-Snyder & Wasserman, *The Power of Logic*
- Hurley, *Concise Introduction to Logic*
- Ichikawa-Jenkins, *forall x: UBC*
- Johnson, *forall x: Mississippi State*
- Leach-Krouse, *The Carnap Book*
- Kalish & Montague, *Logic*
- Magnus, *forall x*
- Open Logic Project
- Thomas-Bolduc & Zach, *forall x: Calgary*
- Tomassi, *Logic*

Bergmann, Moore & Nelson, *The Logic Book*

Sentential logic

For the corresponding proof systems, see here.

- Selected with `system="..."`: `LogicBookSD` `LogicBookSDPlus`
- Sentence letters:`A`…`Z`
- Brackets allowed `(`, `)`
- Associative ∧, ∨: left
- Connectives:

Connective	Keyboard
	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
~	`-`, `~`, `not`

Example:

```
`A /\ B /\ (C_1 -> (~R_2 \/ (S <-> T)))`{system="LogicBookSD"}
```

produces `A /\ B /\ (C_1 -> (~R_2 \/ (S <-> T)))`.

Predicate logic

- Selected with `system="..."`: `LogicBookPD`
- Sentence letters: `A` … `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `v`
- Function symbols: no
- Variables: `w`…`z`
- Atomic formulas: Fax
- Identity: no
- Quantifiers: $(\forall x)$, $(\exists x)$

Quantifiers:

Connective	Keyboard
	`A`
	`E`

Example:

```
`(Ax)(Gabx -> (Ew)(Hxw /\ P))`{system="LogicBookPD"}
```

produces `(Ax)(Gabx -> (Ew)(Hx /\ P))`

Predicate logic with equality

- Selected with `system="..."`: `LogicBookPDE`
- Function symbols: `a…t`
- Variables: `w…z`

Connective	Keyboard
$=$	`=`

Example:

```
`(Ax)(Gabx -> (Ew)(Hxw /\ P /\ ~f(x)=w))`{system="LogicBookPDE"}
```

produces `(Ax)(Gabx -> (Ew)(Hxw /\ P /\ ~f(x)=w))`

Bonevac, *Deduction*

Sentential logic

- Selected with `system="..."`: `bonevacSL`
- Sentence letters: `a…z`
- Brackets allowed `(`, `)`
- Associative $\wedge$, $\vee$: no
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

Example:

```
`(a & b) & (c_1 -> (~r_2 \/ (s <-> t)))`{system="bonevacSL"}
```

produces `(a /\ b) /\ (c_1 -> (~r_2 \/ (s <-> t)))`.

Quantificational logic

- Selected with `system="..."`: `bonevacQL`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `w`
- Function symbols: none
- Variables: `x`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

Example:

```
`Ax(Gabx -> Ey(Hxy & Pa & ~x=v))`{system="bonevacQL"}
```

produces `Ax(Gabx -> Ey(Hxy & Pa & ~x=v))`

Gallow, *forall x: Pittsburgh*

For the corresponding proof systems, see here.

Sentential logic

- Selected with `system="..."`: `gallowSL`
- Sentence letters: `A`…`Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative $\wedge$, $\vee$: left

Connective	Keyboard

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`

Example:

```
`A /\ B /\ (C_1 -> (~R_2 \/ [S <-> T]))`{system="gallowSL"}
```

produces `A /\ B /\ (C_1 -> (~R_2 \/ [_|_ <-> T])).`

First-order logic

- Selected with `system="..."`: `gallowPL`
- Sentence letters: `A` …. `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `v`
- Function symbols: none
- Variables: `w`…`z`
- Identity: no
- Atomic formulas: Fax
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`

Example:

```
`Ax(Gabx -> Ew((Hxw /\ P) /\ Qs))`{system="gallowPL"}
```

produces `Ax(Gabx -> Ew((Hxw /\ P) /\ Qs))`

Gamut, *Logic, Language, and Meaning*

Propositional logic

- Selected with `system="..."`: `gamutIPND` `gamutPND` `gamutPNDPlus`
- Sentence letters: `a`…`z`
- Brackets allowed `(`, `)`
- Associative $\wedge$, $\vee$: no
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`

Example:

```
`(a /\ b) /\ (c_1 -> (~r_2 \/ (_|_ <-> t)))`{system="gamutIPND"}
```

produces `(a /\ b) /\ (c_1 -> (~r_2 \/ (_|_ <-> t)))`.

Predicate logic

- Selected with `system="..."`: `gamutND`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `r`
- Function symbols: none
- Variables: `s`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
=	`=`

Example:

```
`Ax(Gabx -> Ew(Hxw /\ (_|_ \/ ~x=w)))`{system="gamutND"}
```

produces `Ax(Gabx -> Ew(Hxw /\ (_|_ \/ ~x=w)))`

Goldfarb, *Deductive Logic*

Propositional logic

- Selected with `system="..."`: `goldfarbPropND`
- Sentence letters: `a`…`z`
- Brackets allowed `(`, `)`
- Associative ∧, ∨: no
- Connectives:

Connective	Keyboard
	`>`, `->`, , `→`
	`.`, ,
	`\/`, `|`, `or`
	`<->`, `<=>`, `<>`
-	`-`, `~`, `not`
	`!?`, `_|_`

Example:

```
`(a /\ b) /\ (c_1 -> (~r_2 \/ (_|_ <-> t)))`{system="goldfarbPropND"}
```

produces `(a /\ b) /\ (c_1 -> (~r_2 \/ (_|_ <-> t)))`.

Predicate logic

- Selected with `system="..."`: `goldfarbND`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: none
- Function symbols: none
- Variables: `a` … `z`
- Atomic formulas: Fxy
- Identity: no
- Quantifiers: $(\forall x)$, $(\exists x)$

Quantifiers:

Connective	Keyboard
	`A`
	`E`

Example:

```
`(Ax)(Gabx -> (Ew)(Hxw /\ P))`{system="goldfarbND"}
```

produces `Ax(Gax -> Ew(Hxw /\ Pw))`

Hardegree, *Symbolic Logic*

Sentential logic

- Selected with `system="..."`: `hardegreeSL`
- Sentence letters: `A` … `Z`
- Brackets allowed `(`, `)`
- Associative $\wedge$, $\vee$: left
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`

Connective	Keyboard
$\sim$	`-`, `~`, `not` `!?`, `_|_`

Example:

```
`A & G & (R_1 -> (~R_2 \/ (_|_ <-> T)))`{system="hardegreeSL"}
```

produces `A & G & (R_1 -> (~R_2 \/ (_|_ <-> T)))`.

Predicate logic

- Selected with `system="..."`: `hardegreePL`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `s`
- Function symbols: none
- Variables: `t`…`z`
- Atomic formulas: Fax
- Identity: no
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`

Example:

```
`Ax(Gabx -> Ev(Hxe & Ov & _|_))`{system="hardegreePL"}
```

produces `Ax(Gabx -> Ev(Hxe & Ov & _|_))`

Hausman, Kahane & Tidman, *Logic and Philosophy*

Sentential logic

- Selected with `system="..."`: `hausmanSL`
- Sentence letters: `A`…`Z`
- Brackets allowed `[`, `]`, `(`, `)`, `{`, `}`, (only in that order)
- Associative $\wedge$, $\vee$: no
- Connectives:

Connective	Keyboard
	, `->`,`>`
	`.`, ` `
	`\/`, `|`, `or`
	`<->`, `<=>`, `<>`
$\sim$	`-`, `~`, `not`

Example:

```
`[A . B] . [C_1 > (~R_2 \/ {S <> T})]`{system="hausmanSL"}
```

produces `[A . B] . [C_1 > (~R_2 \/ {S <> T})]`.

Predicate logic

- Selected with `system="..."`: `hausmanPL`
- Sentence letters: `A` … `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `t`
- Function symbols: no
- Variables: `u`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: (x), $(\exists x)$

Quantifiers:

Connective	Keyboard
	`E`

Connective	Keyboard
=	=

Example:

```
`(x)[Gabx > (Eu)(Hxu . {P \/ ~x=u})]`{system="hausmanPL"}
```

produces `(x)[Gabx > (Eu)(Hxu . {P \/ ~x=u})]`

Howard-Snyder, Howard-Snyder & Wasserman, *The Power of Logic*

Sentential logic

- Selected with `system="..."`: `howardSnyderSL`
- Sentence letters: `A`…`Z`
- With subscripts: ?
- Brackets allowed `(`, `)`, `[`, `]`, `{`, `}`
- Associative ∧, ∨: no
- Connectives:

Connective	Keyboard
→	`->`, `=>`,`>`
	`.`, ` `
	`\/`, `|`, `or`
	`<->`, `<=>`, `<>`
~	`-`, `~`, `not`

Example:

```
`(A . B) . [C_1 -> (~R_2 \/ {S <-> T})]`{system="howardSnyderSL"}
```

produces `(A . B) . [C_1 -> (~R_2 \/ {S <-> T})]`.

Predicate logic

- Selected with `system="..."`: `howardSnyderPL`
- Sentence letters: `A` … `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `u`
- Function symbols: no
- Variables: `v`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: (x), $(\exists x)$

Quantifiers:

Connective	Keyboard
	`E`
$=$	`=`

Example:

```
`(x)[Gabx -> (Ev)(Hxv . (P \/ ~x=v))]`{system="howardSnyderPL"}
```

produces `(x)[Gabx -> (Ev)(Hxv . (P \/ ~x=v))]`

Hurley, *Concise Introduction to Logic*

Sentential logic

- Selected with `system="..."`: `hurleySL`
- Sentence letters: `A`…`Z`
- With subscripts: ?
- Brackets allowed `(`, `)`, `[`, `]`, `{`, `}`
- Associative $\wedge$, $\vee$: no
- Connectives:

Connective	Keyboard
	, `->`, `>`
	`.`, ,
	`\/`, `|`, `or`

Connective	Keyboard
	`<->`, `<=>`, `<>`
~	`-`, `~`, `not`

Example:

```
`(A . B) . [C_1 > (~R_2 \/ {S <-> T})]`{system="hurleySL"}
```

produces `(A . B) . [C_1 > (~R_2 \/ {S <-> T})]`.

Predicate logic

- Selected with `system="..."`: `hurleyPL`
- Sentence letters: `A` … `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `w`
- Function symbols: no
- Variables: `x`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: (x), $(\exists x)$

Quantifiers:

Connective	Keyboard
	`E`
$=$	`=`

Example:

```
`(x)[Gabx > (Ey)(Hxy . {P \/ ~x=y})]`{system="hurleyPL"}
```

produces `(x)[Gabx > (Ey)(Hxy . {P \/ ~x=y})]`

Ichikawa-Jenkins, *forall x: UBC*

Sentential logic

- Selected with `system="..."`: `ichikawaJenkinsSL`
- Sentence letters: `A`…`Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative $\wedge$, $\vee$: yes
- Connectives:

Connective	Keyboard
	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

Example:

```
`A /\ B /\ (C_1 -> (~R_2 \/ [S <-> T]))`{system="ichikawaJenkinsSL"}
```

produces `A /\ B /\ (C_1 -> (~R_2 \/ [S <-> T]))`.

Quantificational logic

- Selected with `system="..."`: `ichikawaJenkinsQL`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `w`
- Function symbols: none
- Variables: `x`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`

Connective	Keyboard
	`E`
=	`=`

Example:

```
`Ax(Gabx -> Ey(Hxy /\ Pa /\ ~x=y))`{system="ichikawaJenkinsQL"}
```

produces `Ax(Gabx -> Ey(Hxy /\ (Pa /\ ~x=y)))`

Johnson, *forall x: Mississippi State*

For the corresponding proof systems, see here.

- Selected with `system="..."`: `johnsonSL`
- Sentence letters: `A…Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative ∧, ∨: yes
- Connectives:

Connective	Keyboard
→	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`v`, `\/`, `|`, `or`
	`<->`, `<=>`
¬	`-`, `~`, `not`

Example:

```
`A & B & (C_1 -> (~R_2 \/ [S <-> T]))`{system="johnsonSL"}
```

produces `A & B & (C_1 -> (~R_2 \/ [S <-> T]))`.

Leach-Krouse, *The Carnap Book*

Kalish & Montague, *Logic*

For the corresponding proof systems, see here.

Propositional logic

- Selected with `system="..."`: `prop`, `montagueSC`
- Sentence letters: `P` … `W`
- Brackets allowed `(`, `)`
- Associative $\wedge$, $\vee$: left
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

Example:

```
`P /\ Q /\ (R_1 -> (~R_2 \/ (S <-> T)))`{system="prop"}
```

produces `P /\ Q /\ (R_1 -> (~R_2 \/ (S <-> T)))`.

First-order logic

- Selected with `system="..."`: `firstOrder`, `montagueQC`
- Sentence letters: `P` … `W`
- Predicate symbols: `F` …`O`
- Constant symbols: `a` … `e`
- Function symbols: `f` … `h`
- Variables: `v`…`z`
- Atomic formulas: $F(a, x)$
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
=	`=`

Example:

```
`Ax(G(a,f(b),x) -> Ev(H(x,v) /\ P /\ ~(x=v)))`{system="firstOrder"}
```

produces `Ax(G(a,f(b),x) -> Ev(H(x,v) /\ P /\ ~(x=v)))`

Monadic second-order logic

- Selected with `system="..."`: `secondOrder`
- Second-order variables: `X` … `Z`

Symbols:

Connective	Keyboard
	`\`

Example:

Example:

```
`AX(\x[Ey(F(y) /\ X(x))](a))`{system="secondOrder"}
```

produces `AX(\x[Ey(F(y) /\ X(x))](a))`

Polyadic second-order logic

- Selected with `system="..."`: `polyadicSecondOrder`
- Second-order variables: `Xn` … `Zn`
- Arity: given by n

Example:

```
`AX2(\x[Ay(F(y) -> X2(x,y))](a))`{system="polyadicSecondOrder"}
```

produces `AX2(\x[Ay(F(y) -> X2(x,y))](a))`

Magnus, *forall x*

Sentential logic

- Selected with `system="..."`: `magnusSL` `magnusSLPlus`
- Sentence letters: `A`…`Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative $\wedge$, $\vee$: yes
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

Example:

```
`A & B & (C_1 -> (~R_2 \/ [S <-> T]))`{system="magnusSL"}
```

produces `A & B & (C_1 -> (~R_2 \/ [S <-> T]))`.

Quantificational logic

- Selected with `system="..."`: `magnusQL`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `w`
- Function symbols: none
- Variables: `x`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

Example:

```
`Ax(Gabx -> Ey(Hxy & Pa & ~x=v))`{system="magnusQL"}
```

produces `Ax(Gabx -> Ey(Hxy & Pa & ~x=v))`

Open Logic Project

Plain propositional and first-order logic uses the same syntax as the TFL and FOL systems of *forall x: Calgary*, 2019+ version (see below. The parser for `openLogicNK` and `openLogicLK` is synonymous with `thomasBolducAndZachTFL2019`; and `openLogicFOLNK` and `openLogicFOLLK` with `thomasBolducAndZachFOL2019`.

Two OLP proof systems are supported: sequent calculus and natural deduction.

In addition, there are special systems supporting the language of arithmetic and the language of set theory.

Arithmetic

- Selected with `system="..."`: `openLogicArithNK`
- Predicate symbols: `<` (two-place, infix)
- Constant symbols: `a` … `r`, `0`
- Function symbols: `'` (one-place, postfix), `+`, `*` (two-place, infix)
- Variables: `s`…`z`
- Identity: `=`,

Symbol	Keyboard
×	*
	!=

Example:

```
`AxAy x * y' = (x * y) + y /\ Ax(0!=x -> 0<x)`{system="openLogicArithNK"}
```

produces `AxAy x * y' = (x * y) + y /\ Ax(0!=x -> 0<x)`

Extended Arithmetic

- Selected with `system="..."`: `openLogicExArithNK`
- Predicate symbols: strings beginning with uppercase letter, `<` (two-place, infix)
- Constant symbols: strings beginning with lowercase letter, `0`
- Function symbols: strings beginning with lowercase letter, `'` (one-place, postfix), `+`, `*` (two-place, infix)
- Variables: `s`…`z`
- Identity: `=`,

Symbol	Keyboard
×	*
	!=

Example:

```
`Q_1(0,0') /\ Ax(0<x -> Sq_a(0,x))`{system="openLogicExArithNK"}
```

produces `Q_1(0,0') /\ Ax(0<x -> Sq_a(0,x))`

Set theory

- Selected with `system="..."`: `openLogicSTNK`
- Predicate symbols: (two-place, infix)
- Constant symbols: `a` … `r`
- Variables: `s`…`z`
- Identity: `=`,

The system `openLogicExSTNK` is like the above but adds arbitrary string predicates and function symbols:

- Selected with `system="..."`: `openLogicExESTNK`
- Predicate symbols: strings beginning with uppercase letter
- Constant symbols: strings beginning with lowercase letter
- Function symbols: strings beginning with lowercase letter

Symbol	Keyboard
	`<<`, `<e`
	`!=`

Example:

```
`Ex(Ay ~y<<x /\ Az(z!=x -> Eu u<<z))`{system="openLogicSTNK"}
```

produces `Ex(Ay ~y<<x /\ Az(z!=x -> Eu u<<z))`

Extended set theory

- Selected with `system="..."`: `openLogicESTNK`, `openLogicExESTNK`
- Predicate symbols: , (two-place, infix)
- Constant symbols: , `a` … `r`
- Function symbols: , , `/`, `Pow` (two-place, infix)
- Variables: `s`…`z`
- Identity: `=`,

The system `openLogicExESTNK` is like the above but adds arbitrary string predicates and function symbols:

- Selected with `system="..."`: `openLogicExESTNK`
- Predicate symbols: strings beginning with uppercase letter
- Constant symbols: strings beginning with lowercase letter

- Function symbols: strings beginning with lowercase letter

Symbol	Keyboard
	`{}`, `empty`
	`<<`, `<e`, `in`
	`<(`, `<s`, `within`, `sub`
	`U`, `cup`
	`I`, `cap`
/	`\`
Pow	`P`
	`!=`

Example:

```
`Ex(Ay ~y<<x /\ Az(z!={} -> Eu u <( P(z)))`{system="openLogicESTNK"}
```

produces `Ex(Ay ~y<<x /\ Az(z!={} -> Eu u <( P(z)))`

Thomas-Bolduc & Zach, *forall x: Calgary*

For the corresponding proof systems, see here.

Fall 2019 and after

The `2019` systems refer to the syntax used in *forall x: Carnap*, Fall 2019 and after. The TFL system allows leaving out extra parentheses in conjunctions and disjunctions of more than 2 sentences. The pre-2019 systems do not, so can be used if stricter parenthesis rules are desired.

The major change is in the syntax of the FOL systems, which wrap arguments in parentheses. This allows support of function symbols and terms. The `FOL2019` system also allows entering negated identities as `x != y`. This is not done in *forall x: Calgary*, but is the convention in the Open Logic Project.

Truth-functional logic

- Selected with `system="..."`: `thomasBolducAndZachTFL2019`
- Sentence letters: `A`…`Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative $\wedge$, $\vee$: left

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`

Example:

```
`A /\ B /\ (C_1 -> (~R_2 \/ [S <-> T]))`{system="thomasBolducAndZachTFL2019"}
```

produces `A /\ B /\ (C_1 -> (~R_2 \/ [_|_ <-> T]))`.

First-order logic

- Selected with `system="..."`: `thomasBolducAndZachFOL2019`, `thomasBolducAndZachFOLPlus2019`
- Sentence letters: `A` …. `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `r`
- Function symbols: `a`…`t`
- Variables: `s`…`z`
- Atomic formulas: $F(a, x)$
- Identity: `=`,
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

Connective	Keyboard
	`!=`

Example:

```
`Ax(G(a,f(b),x) -> Es(H(x,s) /\ P /\ x!=s))`{system="thomasBolducAndZachFOL2019"}
```

produces `Ax(G(a,f(b),x) -> Es(H(x,s) /\ P /\ x!=s))`

pre-Fall 2019

These syntax conventions were in force before the Fall 2019 edition of *forall x: Calgary*. They are still useful: (a) They coincide with the syntax conventions of Tim Button's *forall x: Cambridge* and the text by Sean Ebbels-Duggan. (b) The TFL system requires all parentheses be included and displays with all parentheses. So it can be used in exercises that require TFL sentences be entered or displayed *without* bracketing conventions.

Truth-functional logic

- Selected with `system="..."`: `thomasBolducAndZachTFL`, `ebelsDugganTFL`
- Sentence letters: `A`…`Z`
- Brackets allowed `(`, `)`, `[`, `]`
- Associative ∧, ∨: no
- Connectives:

Connective	Keyboard
→	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
¬	`-`, `~`, `not`
	`!?`, `_|_`

Example:

```
`(A /\ B) /\ (C_1 -> (~R_2 \/ [_|_ <-> T]))`{system="thomasBolducAndZachTFL"}
```

produces `(A /\ B) /\ (C_1 -> (~R_2 \/ [_|_ <-> T]))`.

Example:

```
`(A /\ B) /\ (C_1 -> (~R_2 \/ [_|_ <-> T]))`{system="ebelsDugganTFL"}
```

produces `(A /\ B) /\ (C_1 -> (~R_2 \/ [_|_ <-> T]))`.

First-order logic

- Selected with `system="..."`: `thomasBolducAndZachFOL`
- Sentence letters: `A` …. `Z`
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `r`
- Function symbols: none
- Variables: `s`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

Example:

```
`Ax(Gabx -> Es(Hxs /\ P /\ ~x=s))`{system="thomasBolducAndZachFOL"}
```

produces `Ax(Gabx -> Es(Hxs /\ (P \/ ~x=s)))`

Tomassi, *Logic*

Propositional logic

- Selected with `system="..."`: `tomassiPL`
- Sentence letters: `P` … `W`
- Brackets allowed `(`, `)`
- Associative $\wedge$, $\vee$: left
- Connectives:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\sim$	`-`, `~`, `not`

Example:

```
`P /\ Q /\ (R_1 -> (~R_2 \/ (S <-> T)))`{system="tomassiPL"}
```

produces `P /\ Q /\ (R_1 -> (~R_2 \/ (S <-> T)))`.

Predicate logic

- Selected with `system="..."`: `tomassiQL`
- Sentence letters: none
- Predicate symbols: `A` …`Z`
- Constant symbols: `a` … `t`
- Function symbols: none
- Variables: `u`…`z`
- Atomic formulas: Fax
- Identity: `=`
- Quantifiers: $\forall x$, $\exists x$

Quantifiers:

Connective	Keyboard
	`A`

Connective	Keyboard
	`E`
=	`=`

Example:

```
`Ax[Gabx -> Ev(Hxv /\ Pr /\ ~(x=v))]`{system="tomassiQL"}
```

produces `Ax[Gabx -> Ev(Hxv /\ Pr /\ ~(x=v))]`

Part IV

Todo:

The available set theory systems are: `elementarySetTheory` and `separativeSetTheory`. The available propositional modal logic systems are: `hardegreeL` `hardegreeK` `hardegreeT` `hardegreeB` `hardegreeD` `hardegree4` and `hardegree5`. The available predicate modal logic system is `hardegreeMPL`, and the available "world theory" system is `hardegreeWTL`.

15 Chains of equivalences

This document gives a short description of how Carnap presents the systems of "chains of equivalences" that allow transformations of sentences into equivalent sentences, e.g., in disjunctive or conjunctive normal form.

15.1 Notation

The system uses the parser/syntax of *forall x: Calgary* and the *Open Logic Text*.

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`
	`A`, `@`
	`E`, `3`

Proofs consist of a series of lines. Every line contains a formula followed by a `:` and then a rule abbreviation. The first line is justified by `:PR`. The available rules are all equivalences, and can be applied to subformulas of a given formula. It is assumed that every line follows from the immediately preceding line by one of the equivalence rules below.

15.2 Equivalences for TFL (propositional logic)

The equivalence proof checker is selected using `.ZachPropEq`.

```
Ex1 A \/ ~(B \/ C) :|-: (A \/ ~C) /\ (A \/ ~B)
|A \/ ~(B \/ C) :PR
```

```
|A \/ ~(C \/ B) :Comm
|A \/ (~C /\ ~B) :DeM
|(A \/ ~C) /\ (A \/ ~B) :Dist
```

The following exchange rules are allowed. They can be used within a propositional context Φ, and in both directions. In other words, any formula occurrence on the left of the table below can be replaced by the corresponding formula on the right, and vice versa.

Rule	Abbreviation	Premises	Conclusion
Double Negation	`DN`	$\neg\neg\varphi$	φ
Conditional	`Cond`	$(\varphi \to \psi)$	$(\neg\varphi \vee \psi)$
		$(\neg\varphi \to \psi)$	$(\varphi \vee \psi)$
Biconditional Exchange	`Bicond`	$(\varphi \leftrightarrow \psi)$	$((\varphi \to \psi) \wedge (\psi \to \varphi))$
DeMorgan's Laws	`DeM`	$\neg(\varphi \wedge \psi)$	$(\neg\varphi \vee \neg\psi)$
		$\neg(\varphi \vee \psi)$	$(\neg\varphi \wedge \neg\psi)$
Commutativity	`Comm`	$(\varphi \wedge \psi)$	$(\psi \wedge \varphi)$
		$(\varphi \vee \psi)$	$(\psi \vee \varphi)$
Associativity	`Assoc`	$(\varphi \wedge (\psi \wedge \chi))$	$((\varphi \wedge \psi) \wedge \chi)$
		$(\varphi \vee (\psi \vee \chi))$	$((\varphi \vee \psi) \vee \chi)$
Distributivity	`Dist`	$(\varphi \vee (\psi \wedge \chi))$	$((\varphi \vee \psi) \wedge (\varphi \vee \chi))$
		$(\varphi \wedge (\psi \vee \chi))$	$((\varphi \wedge \psi) \vee (\varphi \wedge \chi))$
		$((\varphi \wedge \psi) \vee (\varphi \wedge \chi))$	$(\varphi \wedge (\psi \vee \chi))$
Idempotence	`Id`	$(\varphi \wedge \varphi)$	φ
		$(\varphi \vee \varphi)$	φ
Absorption	`Abs`	$(\varphi \wedge (\varphi \vee \psi))$	φ
		$(\varphi \vee (\varphi \wedge \psi))$	φ
Simplification	`Simp`	$(\varphi \wedge (\psi \vee \neg\psi))$	φ
		$(\varphi \vee (\psi \wedge \neg\psi))$	φ
		$(\varphi \vee (\psi \vee \neg\psi))$	$(\psi \vee \neg\psi)$
		$(\varphi \wedge (\psi \wedge \neg\psi))$	$(\psi \wedge \neg\psi)$

The rules for distributivity, absorption, and simplification allow for implicit commutativity, so e.g., `Distr` also allows replacing $(\psi \wedge \chi) \vee \varphi$ by $(\psi \vee \varphi) \wedge (\chi \vee \varphi)$, and `Simp` allows replacing $(\neg\psi \vee \psi) \wedge \varphi$ by φ.

If the conclusion of the target entailment is left empty, and the `tests` option is set, the checker will accept any proof where the last line meets the test. So for instance, to require students to transform a sentence into conjunctive normal form, you would assign

```
```{.ProofChecker .ZachPropEq tests="CNF"}
Ex1 A \/ ~(B \/ C) :|-:
```
```

```
|A \/ ~(B \/ C) :PR
```

```
Ex1 A \/ ~(B \/ C) :|-:
|A \/ ~(B \/ C) :PR
```

A proof playground is also supported.

```
```{.Playground .ZachPropEq}
```

The available tests are the same as for translation exercises, and can be combined. If combined, multiple tests have to be separated by spaces. Tests also work on playgrounds.

Name	Effect
`CNF`	Requires conjunctive normal form
`DNF`	Requires disjunctive normal form
`maxCon:N`	Requires that the translation contain N or fewer connectives
`maxNot:N`	Requires that the translation contain N or fewer negations
`maxAnd:N`	Requires that the translation contain N or fewer conjunctions
`maxIff:N`	Requires that the translation contain N or fewer biconditionals
`maxIf:N`	Requires that the translation contain N or fewer conditionals
`maxOr:N`	Requires that the translation contain N or fewer disjunctions
`maxFalse:N`	Requires that the translation contain N or fewer falsity constants
`maxAtom:N`	Requires that the translation contain N or fewer atomic sentences

15.3 The FOL Systems

The system `.ZachFOLEq` extends the rules of `.ZachPropEq` by equivalence rules for quantifiers, also applied in arbitrary contexts Φ, and in either direction. Those are:

Rule	Abbreviation	Premises	Conclusion
Variable Renaming	`VR`	$\forall x\,\varphi(x)$	$\forall y\,\varphi(y)$
		$\exists x\,\varphi(x)$	$\exists y\,\varphi(y)$
Quantifier Exchange	`QX`	$\forall x \forall y\,\varphi(x,y)$	$\forall y \forall x\,\varphi(y)$
		$\exists x \exists y\,\varphi(x,y)$	$\exists y \exists x\,\varphi(x,y)$
Quantifier Negation	`QN`	$\neg\forall x \varphi(x)$	$\exists x \neg\varphi(x)$
```

Rule	Abbreviation	Premises	Conclusion
		$\neg\exists x\varphi(x)$	$\forall x\neg\varphi(x)$
Quantifier Distribution	`QD`	$\forall x(\varphi(x) \land \psi(x))$	$\forall x\,\varphi(x) \land \forall x\,\psi(x)$
		$\exists x(\varphi(x) \lor \psi(x))$	$\exists x\,\varphi(x) \lor \exists x\,\psi(x)$
Quantifier Shift for $\forall$	`QSA`	$\forall x(\varphi(x) \land \psi)$	$\forall x\,\varphi(x) \land \psi$
		$\forall x(\varphi(x) \lor \psi)$	$\forall x\,\varphi(x) \lor \psi$
		$\forall x(\varphi(x) \to \psi)$	$\exists x\,\varphi(x) \to \psi$
		$\forall x(\psi \to \varphi(x))$	$\psi \to \forall x\,\varphi(x)$
Quantifier Shifts for $\exists$	`QSE`	$\exists x(\varphi(x) \land \psi)$	$\exists x\,\varphi(x) \land \psi$
		$\exists x(\varphi(x) \lor \psi)$	$\exists x\,\varphi(x) \lor \psi$
		$\exists x(\varphi(x) \to \psi)$	$\forall x\,\varphi(x) \to \psi$
		$\exists x(\psi \to \varphi(x))$	$\psi \to \exists x\,\varphi(x)$

Note that Carnap actually considers all formulas equal up to renaming of bound variables. Thus, the `VR` rules are provided just for completeness (and will allow any number of renamings of bound variables). Any variable renaming necessary to apply a quantifier shift can be done implicitly without first invoking the `VR` rule.

Testing of correctness can become quite slow, so it is recommended to not do this on every button press and use `feedback="manual"` instead.

The FOL system has an additional test, `PNF`, that requires the final line to be in prenex normal form.

````
```{.ProofChecker .ZachFOLEq options="resize" feedback="manual" tests="PNF"}
Ex2 ~(Ax P(x) <-> Ey Q(y)) :|-:
|~(Ax P(x) <-> Ey Q(y)) :PR
```
````

```
Ex2 ~(Ax P(x) <-> Ey Q(y)) :|-:
|~(Ax P(x) <-> Ey Q(y)) :PR
|~((Ax P(x) -> Ey Q(y)) /\ (Ey Q(y) -> Ax P(x))) :Bicond
|~(Ax P(x) -> Ey Q(y)) \/ ~(Ey Q(y) -> Ax P(x)) :DeM
|~(Ax P(x) -> Ey Q(y)) \/ ~Ay( Q(y) -> Ax P(x)) :QSA
|~(Ax P(x) -> Ey Q(y)) \/ ~AyAx( Q(y) ->  P(x)) :QSA
|~(Ax P(x) -> Ey Q(y)) \/ Ey~Ax( Q(y) ->  P(x)) :QN
|~(Ax P(x) -> Ey Q(y)) \/ EyEx~( Q(y) ->  P(x)) :QN
|(Ax P(x) /\ ~Ey Q(y)) \/ EyEx~( Q(y) ->  P(x)) :Cond
|(Ax P(x) /\ Ay~ Q(y)) \/ EyEx~( Q(y) ->  P(x)) :QN
|(Ax P(x) /\ Ax~ Q(x)) \/ EyEx~( Q(y) ->  P(x)) :VR
|Ax(P(x) /\ ~Q(x)) \/ EyEx~( Q(y) ->  P(x)) :QD
```

```
|Ey[Ax(P(x) /\ ~Q(x)) \/ Ex~( Q(y) ->  P(x))] :QSE
|EyEx[Ax(P(x) /\ ~Q(x)) \/ ~( Q(y) ->  P(x))] :QSE
```

16 Natural deduction in the *forall x: Mississippi State* systems

This document gives a short description of how Carnap presents the systems of natural deduction from Greg Johnson's *forall x: Mississippi State*. At least some prior familiarity with Fitch-style proof systems is assumed.

The syntax of formulas accepted is described in the Systems Reference.

16.1 Notation

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
$\&$	`/\`, `and`
	`v`, `\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

The available sentence letters are A through Z, together with the infinitely many subscripted letters $P_1, P_2, \ldots$ written `P_1`, `P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, or a separator line containing two dashes, thus: `--`. A justification consists of a rule abbreviation followed by zero or more numbers (citations of particular lines) and pairs of numbers separated by a dash (citations of a subproof contained within the given line range).

A subproof is begun by increasing the indentation level. The first line of a subproof should be more indented than the containing proof, and the lines directly contained in this subproof should maintain this indentation level. (Lines indirectly contained, by being part of a sub-subproof, will need to be indented more deeply.) The subproof ends when the indentation level of the containing proof is resumed; hence, two contiguous sub-proofs of the same containing proof can be distinguished from one another by inserting a separator line between them at the

same level of indentation as the containing proof. The final *unindented* line of a derivation will serve as the conclusion of the entire derivation.

Here's an example derivation, using the TFL system `.JohnsonSL`:

```
Ex ~P \/ (R & Q) :|-: P -> Q
|~P \/ (R & Q) :PR
|    P :AS
|    ~~P :DN 2
|    R & Q :\/E 1,3
|    Q :&E 4
|P -> Q :->I 2-5
```

Or, with a Fitch-style guides overlay (activated with `guides="fitch"`):

```
Ex ~P \/ (R & Q) :|-: P -> Q
|~P \/ (R & Q) :PR
|    P :AS
|    ~~P :DN 2
|    R & Q :\/E 1,3
|    Q :&E 4
|P -> Q :->I 2-5
```

There is also a playground mode:

```

```

The system for Johnson's *forall x: Mississippi State* (the system used in a proofchecker constructed with `.JohnsonSL` in Carnap's Pandoc Markup) has the following set of rules for direct inferences:

Rule	Abbreviation	Premises	Conclusion
And-Elim	`E`	$\varphi \wedge \psi$	φ / ψ
And-Intro	`I`	φ, ψ	$\varphi \wedge \psi$
Or-Elim	`E`	$\neg\psi, \varphi \vee \psi$	φ
		$\neg\varphi, \varphi \vee \psi$	ψ
Or-Intro	`I`	φ	$\varphi \vee \psi$
		ψ	$\varphi \vee \psi$
Conditional-Elim	`→E`	$\varphi, \varphi \rightarrow \psi$	ψ
Biconditional-Elim	`E`	$\varphi, \varphi \leftrightarrow \psi$	ψ
		$\psi, \varphi \leftrightarrow \psi$	φ
Biconditional-Intro	`I`	$\varphi \rightarrow \psi, \psi \rightarrow \varphi$	$\varphi \leftrightarrow \psi$
Double Negation	`DN`	φ	$\neg\neg\varphi$
Reiteration	`R`	φ	φ

Rule	Abbreviation	Premises	Conclusion

We also have four rules for indirect inferences:

1. `→I`, which justifies an assertion of the form $\varphi \to \psi$ by citing a subproof beginning with the assumption φ and ending with the conclusion ψ;
2. `¬I`, which justifies an assertion of the form $\neg\varphi$ by citing a subproof beginning with the assumption φ and ending with a pair of lines ψ,$\neg\psi$.
3. `¬E`, which justifies an assertion of the form by citing a subproof beginning with the assumption $\neg\varphi$ and ending with a pair of lines ψ,$\neg\psi$.

Finally, `PR` can be used to justify a line asserting a premise, and `AS` can be used to justify a line making an assumption. A note about the reason for an assumption can be included in the rendered proof by writing `A/NOTETEXTHERE` rather than `AS` for an assumption. Assumptions are only allowed on the first line of a subproof.

17 Natural Deduction in the *forall x: Pittsburgh* systems

This document gives a short description of how Carnap presents the systems of natural deduction from *forall x: Pittsburgh*, the remix by Dimitri Gallow of Aaron Thomas-Bolduc and Richard Zach's Calgary version of P.D. Magnus's *forall x*.

The syntax of formulas accepted is described in the Systems Reference.

17.1 Truth-functional logic

17.1.1 Notation

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`

The available sentence letters are A through Z, together with the infinitely many subscripted letters $P_1, P_2, \ldots$ written `P_1, P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, or a separator line containing two dashes, thus: `--`. A justification consists of a rule abbreviation followed by zero or more numbers (citations of particular lines) and pairs of numbers separated by a dash (citations of a subproof contained within the given line range).

A subproof is begun by increasing the indentation level. The first line of a subproof should be more indented than the containing proof, and the lines directly contained in this subproof

should maintain this indentation level. (Lines indirectly contained, by being part of a sub-sub-proof, will need to be indented more deeply.) The subproof ends when the indentation level of the containing proof is resumed; hence, two contiguous sub-proofs of the same containing proof can be distinguished from one another by inserting a separator line between them at the same level of indentation as the containing proof. The final *unindented* line of a derivation will serve as the conclusion of the entire derivation.

Here's an example derivation, using the system `.GallowSL`:

```
Ex :|-: (A \/ B) -> (B \/ A)
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

Or, `.GallowSLPlus` with a Fitch-style guides overlay (activated with `guides="fitch"`):

```
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

Simple indent guides overlay (activated with `guides="indent"`) with system `.GallowPL`:

```
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

17.1.2 The SL systems

The system `.GallowSLPlus` allows all rules below. `.GallowSL` is like `.GallowSLPlus` except it *disallows* all derived rules, i.e., the only allowed rules are `R`, and the I and E rules for the connectives and for .

It has the following set of rules for direct inferences:

Basic rules:

Rule	Abbreviation	Premises	Conclusion
And-Elim.	`E`	$\varphi \wedge \psi$	φ/ψ
And-Intro.	`I`	φ, ψ	$\varphi \wedge \psi$
Or-Intro	`I`	φ/ψ	$\varphi \vee \psi$
Contradiction-Intro	`I`	$\varphi, \neg\varphi$	$\bot$
Contradiction-Elim	`E`	$\bot$	ψ
Biconditional-Elim	`E`	$\varphi/\psi, \varphi \leftrightarrow \psi$	ψ/φ
Reiteration	`R`	φ	φ

Derived rules:

Rule	Abbreviation	Premises	Conclusion
Disjunctive Syllogism	`DS`	$\neg\psi/\neg\varphi, \varphi \vee \psi$	φ/ψ
Modus Tollens	`MT`	$\varphi \rightarrow \psi, \neg\psi$	$\neg\varphi$
Double Negation Elim.	`DNE`	$\neg\neg\varphi$	φ
DeMorgan's Laws	`DeM`	$\neg(\varphi \wedge \psi)$	$\neg\varphi \vee \neg\psi$
		$\neg(\varphi \vee \psi)$	$\neg\varphi \wedge \neg\psi$
		$\neg\varphi \vee \neg\psi$	$\neg(\varphi \wedge \psi)$
		$\neg\varphi \wedge \neg\psi$	$\neg(\varphi \vee \psi)$

We also have five rules for indirect inferences:

1. `→I`, which justifies an assertion of the form $\varphi \rightarrow \psi$ by citing a subproof beginning with the assumption φ and ending with the conclusion ψ;
2. `I`, which justifies an assertion of the form $\varphi \leftrightarrow \psi$ by citing two subproofs, beginning with assumptions φ, ψ, respectively, and ending with conclusions ψ, φ, respectively;
3. `¬I`, which justifies an assertion of the form $\neg\varphi$ by citing a subproof beginning with the assumption φ and ending with a conclusion $\bot$.
4. `E`, which justifies an assertion of the form by citing a disjunction $\psi \vee \chi$ and two subproofs beginning with assumptions ψ, χ respectively and each ending with the conclusion φ.
5. `¬E` (indirect proof), which justifies an assertion of the form φ by citing a subproof beginning with the assumption $\neg\varphi$ and ending with a conclusion $\bot$.

6. `LEM` (Law of the Excluded Middle), which justifies an assertion of the form ψ by citing two subproofs beginning with assumptions $\varphi, \neg\varphi$ respectively and each ending with the conclusion ψ. LEM is a derived rule.

Finally, `PR` can be used to justify a line asserting a premise, and `AS` can be used to justify a line making an assumption. A note about the reason for an assumption can be included in the rendered proof by writing `A/NOTETEXTHERE` rather than `AS` for an assumption. Assumptions are only allowed on the first line of a subproof.

17.2 Predicate logic

There are two proof systems corresponding to the Pittsburgh remix of *forall x*. All of them allow sentence letters in first-order formulas. The available relation symbols are the same as for SL: A through Z, together with the infinitely many subscripted letters $F_1, F_2, ...$ written `F_1, F_2` etc. However, the available constants and function symbols are only a through v, together with the infinitely many subscripted letters $c_1, c_2, ...$ written `c_1, c_2,…`. The available variables are w through z, with the infinitely many subscripted letters $x_1, x_2, ...$ written `x_1, x_2,…`.

Connective	Keyboard
	`A`, `@`
	`E`, `3`

The predicate logic system `.GallowPL` of *forall x: Pittsburgh* extend the rules of the system `.GallowSL` with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Existential Introduction	`I`	$\varphi(\sigma)$	$\exists x\varphi(x)$
Universal Elimination	`E`	$\forall x\varphi(x)$	$\varphi(\sigma)$
Universal Introduction	`E`	$\varphi(\sigma)$	$\forall x\varphi(x)$

Where Universal Introduction is subject to the restriction that σ must not appear in $\varphi(x)$, or any undischarged assumption or in any premise of the proof.[1]

There is one new rule for indirect derivations: `E`, which justifies an assertion ψ by citing an assertion of the form $\exists x\varphi(x)$ and a subproof beginning with the assumption $\varphi(\sigma)$ and ending

[1]Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

with the conclusion ψ, where σ does not appear in $\psi, \exists x\varphi(x)$, or in any of the undischarged assumptions or premises of the proof.

The proof system `.GallowPLPlus` for the Pittsburgh version of *forall x* adds, in addition to the (basic and derived) rules of `.GallowPL`, the rules

Rule	Abbreviation	Premises	Conclusion
Conversion of Quantifiers	`CQ`	$\neg\forall x\varphi(x)$	$\exists x\neg\varphi(x)$
		$\exists x\neg\varphi(x)$	$\neg\forall x\varphi(x)$
		$\neg\exists x\varphi(x)$	$\forall x\neg\varphi(x)$
		$\forall x\neg\varphi(x)$	$\neg\exists x\varphi(x)$

18 Natural deduction in the *forall x: Calgary* systems

This document gives a short description of how Carnap presents the systems of natural deduction from *forall x: Calgary*, the remix by Aaron Thomas-Bolduc and Richard Zach of Tim Button's Cambridge version of P.D. Magnus's *forall x*.

The systems supported come in two versions, with slightly different rules and different syntax in the first-order part: those for versions of the book before Fall 2019, and those for the Fall 2019 edition and after. The versions for the pre-2019 editions are practically the same as the systems in Tim Button's *forall x: Cambridge*, except that the LEM rule is called TND in Button's text.

The syntax of formulas accepted is described in the Systems Reference.

18.1 Truth-functional logic

18.1.1 Notation

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`, `>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`
	`!?`, `_|_`

The available sentence letters are A through Z, together with the infinitely many subscripted letters $P_1, P_2, \ldots$ written `P_1`, `P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, or a separator line containing two dashes, thus: `--`. A justification consists of a rule abbreviation followed by zero or more numbers

(citations of particular lines) and pairs of numbers separated by a dash (citations of a subproof contained within the given line range).

A subproof is begun by increasing the indentation level. The first line of a subproof should be more indented than the containing proof, and the lines directly contained in this subproof should maintain this indentation level. (Lines indirectly contained, by being part of a sub-sub-proof, will need to be indented more deeply.) The subproof ends when the indentation level of the containing proof is resumed; hence, two contiguous sub-proofs of the same containing proof can be distinguished from one another by inserting a separator line between them at the same level of indentation as the containing proof. The final *unindented* line of a derivation will serve as the conclusion of the entire derivation.

Here's an example derivation, using the TFL system `.ZachTFL`:

```
Ex :|-: (A \/ B) -> (B \/ A)
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

Or, with a Fitch-style guides overlay (activated with `guides="fitch"`):

```
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

Simple indent guides overlay (activated with `guides="indent"`):

```
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
```

```
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

18.1.2 The TFL systems

The system `.ZachTFL` allows all rules below. `.ZachTFL2019` is like `.ZachTFL` except it *disallows* all derived rules, i.e., the only allowed rules are `R`, `X`, `IP`, and the I and E rules for the connectives.

It has the following set of rules for direct inferences:

Basic rules:

Rule	Abbreviation	Premises	Conclusion
And-Elim.	`E`	$\varphi \wedge \psi$	φ/ψ
And-Intro.	`I`	φ, ψ	$\varphi \wedge \psi$
Or-Intro	`I`	φ/ψ	$\varphi \vee \psi$
Negation-Elim	`¬E`	$\varphi, \neg\varphi$	$\bot$
Explosion	`X`	$\bot$	ψ
Biconditional-Elim	`E`	$\varphi/\psi, \varphi \leftrightarrow \psi$	ψ/φ

Derived rules:

Rule	Abbreviation	Premises	Conclusion
Reiteration	`R`	φ	φ
Disjunctive Syllogism	`DS`	$\neg\psi/\neg\varphi, \varphi \vee \psi$	φ/ψ
Modus Tollens	`MT`	$\varphi \rightarrow \psi, \neg\psi$	$\neg\varphi$
Double Negation Elim.	`DNE`	$\neg\neg\varphi$	φ
DeMorgan's Laws	`DeM`	$\neg(\varphi \wedge \psi)$	$\neg\varphi \vee \neg\psi$
		$\neg(\varphi \vee \psi)$	$\neg\varphi \wedge \neg\psi$
		$\neg\varphi \vee \neg\psi$	$\neg(\varphi \wedge \psi)$
		$\neg\varphi \wedge \neg\psi$	$\neg(\varphi \vee \psi)$

We also have five rules for indirect inferences:

1. `→I`, which justifies an assertion of the form $\varphi \rightarrow \psi$ by citing a subproof beginning with the assumption φ and ending with the conclusion ψ;
2. `I`, which justifies an assertion of the form $\varphi \leftrightarrow \psi$ by citing two subproofs, beginning with assumptions φ, ψ, respectively, and ending with conclusions ψ, φ, respectively;

3. ¬I, which justifies an assertion of the form $\neg\varphi$ by citing a subproof beginning with the assumption φ and ending with a conclusion $\bot$.
4. E, which justifies an assertion of the form by citing a disjunction $\psi \vee \chi$ and two subproofs beginning with assumptions ψ, χ respectively and each ending with the conclusion φ.
5. IP (indirect proof), which justifies an assertion of the form φ by citing a subproof beginning with the assumption $\neg\varphi$ and ending with a conclusion $\bot$.
6. LEM (Law of the Excluded Middle), which justifies an assertion of the form ψ by citing two subproofs beginning with assumptions $\varphi, \neg\varphi$ respectively and each ending with the conclusion ψ. LEM is a derived rule.

Finally, PR can be used to justify a line asserting a premise, and AS can be used to justify a line making an assumption. A note about the reason for an assumption can be included in the rendered proof by writing A/NOTETEXTHERE rather than AS for an assumption. Assumptions are only allowed on the first line of a subproof.

18.2 First-order logic

There are three proof systems corresponding to the Calgary remix of *forall x*. All of them allow sentence letters in first-order formulas. The available relation symbols are the same as for TFL: A through Z, together with the infinitely many subscripted letters $F_1, F_2, \ldots$ written F_1, F_2 etc. However, the available constants and function symbols are only a through r, together with the infinitely many subscripted letters $c_1, c_2, \ldots$ written c_1, c_2,…. The available variables are s through z, with the infinitely many subscripted letters $x_1, x_2, \ldots$ written x_1, x_2,….

As of the Fall 2019 edition of *forall x: Calgary*, the syntax for first-order formulas has arguments to predicates in parentheses and with commas (e.g., $F(a, b)$); prior to that edition, the convention was the same as in the original and Cambridge editions of *forall x* (e.g., Fab).

Connective	Keyboard
	A, @
	E, 3
$=$	=

The first-order *forall x: Calgary* systems for FOL extend the rules of the system TFL with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Existential Introduction	I	$\varphi(\sigma)$	$\exists x\varphi(x)$
Universal Elimination	E	$\forall x\varphi(x)$	$\varphi(\sigma)$
Universal Introduction	E	$\varphi(\sigma)$	$\forall x\varphi(x)$

Rule	Abbreviation	Premises	Conclusion
Equality Introduction	`=I`		$\sigma = \sigma$
Equality Elimination	`=E`	$\sigma = \tau, \varphi(\sigma)/\varphi(\tau)$	$\varphi(\tau)/\varphi(\sigma)$

Where Universal Introduction is subject to the restriction that σ must not appear in $\varphi(x)$, or any undischarged assumption or in any premise of the proof.[1]

There is one new rule for indirect derivations: `E`, which justifies an assertion ψ by citing an assertion of the form $\exists x\varphi(x)$ and a subproof beginning with the assumption $\varphi(\sigma)$ and ending with the conclusion ψ, where σ does not appear in $\psi, \exists x\varphi(x)$, or in any of the undischarged assumptions or premises of the proof.

The original proof system for the Calgary version of *forall x*, `.ZachFOL` adds, in addition to the (basic and derived) rules of `.ZachTFL`, the rules

Rule	Abbreviation	Premises	Conclusion
Conversion of Quantifiers	`CQ`	$\neg\forall x\varphi(x)$	$\exists x\neg\varphi(x)$
		$\exists x\neg\varphi(x)$	$\neg\forall x\varphi(x)$
		$\neg\exists x\varphi(x)$	$\forall x\neg\varphi(x)$
		$\forall x\neg\varphi(x)$	$\neg\exists x\varphi(x)$

The 2019 versions of the FOL systems use the new syntax, i.e., $F(a, b)$ instead of Fab. The system `.ZachFOL2019` allows only the basic rules of the TFL system and the basic rules of FOL. The system `.ZachFOLPlus2019` allows the basic and derived rules of `.ZachTFL` and the CQ rules listed above.

In summary:

System	TFL Rules	FOL Syntax	FOL Rules
`.ZachTFL`	Basic + Derived		
`.ZachFOL`	Basic + Derived	Fab	Basic + CQ
`.ZachTFL2019`	Basic		
`.ZachFOL2019`	Basic	F(a,b)	Basic
`.ZachFOLPlus2019`	Basic + Derived	F(a,b)	Basic + CQ

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

19 Natural deduction in the original *forall x* systems

This document gives a short description of how Carnap presents the systems of natural deduction from P.D. Magnus' *forall x*. At least some prior familiarity with Fitch-style proof systems is assumed.

There are several alternate versions (remixes) of *forall x*, which use slightly different syntax and/or rules. The versions supported by Carnap are:

- The original version of *forall x* by P.D. Magnus, described on this page.
- *forall x: Calgary*
- *forall x: Mississippi State*
- *forall x: Pittsbugh*
- *forall x: UBC*

19.1 Notation

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
$\&$	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

The available sentence letters are A through Z, together with the infinitely many subscripted letters $P_1, P_2, \ldots$ written `P_1`, `P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, or a separator line containing two dashes, thus: `--`. A justification consists of a rule abbreviation followed by zero or more numbers (citations of particular lines) and pairs of numbers separated by a dash (citations of a subproof contained within the given line range).

A subproof is begun by increasing the indentation level. The first line of a subproof should be more indented than the containing proof, and the lines directly contained in this subproof should maintain this indentation level. (Lines indirectly contained, by being part of a sub-sub-proof, will need to be indented more deeply.) The subproof ends when the indentation level of the containing proof is resumed; hence, two contiguous sub-proofs of the same containing proof can be distinguished from one another by inserting a separator line between them at the same level of indentation as the containing proof. The final *unindented* line of a derivation will serve as the conclusion of the entire derivation.

Here's an example derivation, using system SL of P.D. Magnus *forall x*, activated in Carnap by `.ForallxSL`:

```
Ex :|-: (A \/ B) -> (B \/ A)
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

Or, with a Fitch-style guides overlay (activated with `guides="fitch"`):

```
|   A \/ B:AS
|      A:AS
|      B \/ A:\/I 2
|   --
|      B:AS
|      B \/ A:\/I 5
|   B \/ A:\/E 1,2-3,5-6
|(A \/ B) -> (B \/ A):->I 1-7
```

There is also a playground mode:

```
```

19.2 Sentential logic

19.2.1 *forall x* System SL

The minimal system SL for P.D. Magnus' *forall x* (the system used in a proofchecker constructed with `.ForallxSL` in Carnap's Pandoc Markup) has the following set of rules for direct inferences:

Rule	Abbreviation	Premises	Conclusion
And-Elim	`E`	$\varphi \wedge \psi$	φ / ψ
And-Intro	`I`	φ, ψ	$\varphi \wedge \psi$
Or-Elim	`E`	$\neg\psi, \varphi \vee \psi$	φ
		$\neg\varphi, \varphi \vee \psi$	ψ
Or-Intro	`I`	φ	$\varphi \vee \psi$
		ψ	$\varphi \vee \psi$
Conditional-Elim	`→E`	$\varphi, \varphi \rightarrow \psi$	ψ
Biconditional-Elim	`E`	$\varphi, \varphi \leftrightarrow \psi$	ψ
		$\psi, \varphi \leftrightarrow \psi$	φ
Reiteration	`R`	φ	φ

We also have four rules for indirect inferences:

1. `→I`, which justifies an assertion of the form $\varphi \rightarrow \psi$ by citing a subproof beginning with the assumption φ and ending with the conclusion ψ;
2. `I`, which justifies an assertion of the form by citing two subproofs, beginning with assuptions φ, ψ, respectively, and ending with conclusions ψ, φ, respectively;
3. `¬I`, which justifies an assertion of the form $\neg\varphi$ by citing a subproof beginning with the assumption φ and ending with a pair of lines ψ,$\neg\psi$.
4. `¬E`, which justifies an assertion of the form by citing a subproof beginning with the assumption $\neg\varphi$ and ending with a pair of lines ψ,$\neg\psi$.

Finally, `PR` can be used to justify a line asserting a premise, and `AS` can be used to justify a line making an assumption. A note about the reason for an assumption can be included in the rendered proof by writing `A/NOTETEXTHERE` rather than `AS` for an assumption. Assumptions are only allowed on the first line of a subproof.

19.2.2 *forall x* System SL Plus

The extended system SL Plus for P.D. Magnus' *forall x* (the system used in a proofchecker constructed with `.ForallxSLPlus` in Carnap's Pandoc Markup) also adds the following rules:

Rule	Abbreviation	Premises	Conclusion
Dilemma	`DIL`	$\varphi \vee \psi, \varphi \to \chi, \psi \to \chi$	χ
Hypothetical Syllogism	`HS`	$\varphi \to \psi, \psi \to \chi$	$\varphi \to \chi$
Modus Tollens	`MT`	$\varphi \to \psi, \neg\psi$	$\neg\varphi$

As well as the following exchange rules, which can be used within a propositional context Φ:

Rule	Abbreviation	Premises	Conclusion
Commutativity	`Comm`	$\Phi(\varphi \wedge \psi)$	$\Phi(\psi \wedge \varphi)$
		$\Phi(\varphi \vee \psi)$	$\Phi(\psi \vee \varphi)$
		$\Phi(\varphi \leftrightarrow \psi)$	$\Phi(\psi \leftrightarrow \varphi)$
Double Negation	`DN`	$\Phi(\varphi)/\Phi(\neg\neg\varphi)$	$\Phi(\neg\neg\varphi)/\Phi(\varphi)$
Material Conditional	`MC`	$\Phi(\varphi \to \psi)$	$\Phi(\neg\varphi \vee \psi)$
		$\Phi(\neg\varphi \vee \psi)$	$\Phi(\varphi \to \psi)$
		$\Phi(\varphi \vee \psi)$	$\Phi(\neg\varphi \to \psi)$
		$\Phi(\neg\varphi \to \psi)$	$\Phi(\varphi \vee \psi)$
BiConditional Exchange	`ex`	$\Phi(\varphi \leftrightarrow \psi)$	$\Phi(\varphi \to \psi \wedge \psi \to \varphi)$
		$\Phi(\varphi \to \psi \wedge \psi \to \varphi)$	$\Phi(\varphi \leftrightarrow \psi)$
DeMorgan's Laws	`DeM`	$\Phi(\neg(\varphi \wedge \psi))$	$\Phi(\neg\varphi \vee \neg\psi)$
		$\Phi(\neg(\varphi \vee \psi))$	$\Phi(\neg\varphi \wedge \neg\psi)$
		$\Phi(\neg\varphi \vee \neg\psi)$	$\Phi(\neg(\varphi \wedge \psi))$
		$\Phi(\neg\varphi \wedge \neg\psi)$	$\Phi(\neg(\varphi \vee \psi))$

19.3 Quantificational logic

The proof system for Magnus's *forall x*, QL, is activated using `.ForallxQL`.

19.3.1 Notation

The different admissible keyboard abbreviations for quantifiers and equality is as follows:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

The *forall x* first-order systems do not contain sentence letters.

Application of a relation symbol is indicated by directly appending the arguments to the symbol.

The available relation symbols are A through Z, together with the infinitely many subscripted letters F_1, F_2, ... written 'F_1, F_2. The arity of a relation symbol is determined from context.

The available constants are a through w, with the infinitely many subscripted letters c_1, c_2, ... written `c_1, c_2,….`

The available variables are x through z, with the infinitely many subscripted letters x_1, x_2, ... written `x_1, x_2,….`

19.3.2 Basic Rules

The first-order *forall x* systems QL (the systems used in proofcheckers constructed with `.ForallxQL`) extend the rules of the system SL with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Existential Introduction	`I`	$\varphi(\sigma)$	$\exists x\varphi(x)$
Universal Elimination	`E`	$\forall x\varphi(x)$	$\varphi(\sigma)$
Universal Introduction	`E`	$\varphi(\sigma)$	$\forall x\varphi(x)$
Equality Introduction	`=I`		$\sigma = \sigma$
Equality Elimination	`=E`	$\sigma = \tau, \varphi(\sigma)/\varphi(\tau)$	$\varphi(\tau)/\varphi(\sigma)$

Where Universal Introduction is subject to the restriction that σ must not appear in $\varphi(x)$, or any undischarged assumption or in any premise of the proof.[1]

It also adds one new rule for indirect derivations: `E`, which justifies an assertion ψ by citing an assertion of the form $\exists x\varphi(x)$ and a subproof beginning with the assumption $\varphi(\sigma)$ and ending with the conclusion ψ, where σ does not appear in $\psi, \exists x\varphi(x)$, or in any of the undischarged assumptions or premises of the proof.

19.3.3 forall x QL Plus

The system QL Plus, activated with `.ForallxQLPlus`, includes all the rules of SL Plus, as well as the following exchange rules, which can be used within a context Φ:

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

Rule	Abbreviation	Premises	Conclusion
Quantifier Negation	`QN`	$\Phi(\forall x \neg \varphi(x)))$	$\Phi(\neg \exists x \varphi(x)))$
		$\Phi(\neg \forall x \varphi(x)))$	$\Phi(\exists x \neg \varphi(x)))$
		$\Phi(\exists x \neg \varphi(x)))$	$\Phi(\neg \forall x \varphi(x)))$
		$\Phi(\neg \exists x \varphi(x)))$	$\Phi(\forall x \neg \varphi(x)))$

20 Natural Deduction in Logic Book Systems

This document gives a short description of how Carnap presents the systems of natural deduction from Bergmann Moore and Nelson's *Logic Book*.

20.1 Propositional Systems

20.1.1 Notation

The different admissible keyboard abbreviations for the different connectives are as follows:

Connective	Keyboard
	`->`, `=>`,`>`
&	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
~	`-`, `~`, `not`

The available sentence letters are A through Z, together with the infinitely many subscripted letters $P_1, P_2, ...$ written `P_1, P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, or a separator line containing two dashes, thus: `--`.[1] A justification consists of zero or more numbers (citations of particular lines) and pairs of numbers separated by a dash (citations of a subproof contained within the given line range), followed by the name of a rule being applied.

A subproof is begun by increasing the indentation level. The first line of a subproof should be more indented than the containing proof, and the lines directly contained in this subproof should maintain this indentation level. (Lines indirectly contained, by being part of a sub-subproof, will need to be indented more deeply.) The subproof ends when the indentation level

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

of the containing proof is resumed; hence, two contiguous sub-proofs of the same containing proof can be distinguished from one another by inserting a separator line between them at the same level of indentation as the containing proof. The final *unindented* line of a derivation will serve as the conclusion of the entire derivation.

Here's an example derivation, using system SD from the *Logic Book*

```
   P         :AS
      P      :AS
      P      :2 R
   P > P     :2-3 CI
P > (P > P) :1-4 CI
```

and here is one using system PD

```
(Ax)Fx         :PR
(Ax)(Fx > Gx)  :PR
Fa             :1 AE
Fa > Ga        :2 AE
Ga             :3,4 >E
(Ax)Gx         :5 AI
```

20.1.2 Basic Rules

20.1.2.1 Logic Book System SD

The minimal system SD from the *Logic Book* (the system used in a proofchecker constructed with `.LogicBookSD` in Carnap's Pandoc Markup) has the following set of rules for direct inferences:

Rule	Abbreviation	Premises	Conclusion
And-Elim.	`&E`	$\varphi\&\psi$	φ/ψ
And-Intro.	`&I`	φ, ψ	$\varphi\&\psi$
Or-Intro	`\/I`	φ/ψ	$\varphi \vee \psi$
Conditional-Elim	`->E.CE`	$\varphi, \varphi \supset \psi$	ψ
Biconditional-Elim	`<->E,BE`	$\varphi/\psi, \varphi \leftrightarrow \psi$	ψ/φ
Reiteration	`R`	φ	φ

We also have five rules for indirect inferences:

1. `->I` (also denoted `CI`), which justifies an assertion of the form $\varphi \supset \psi$ by citing a subproof beginning with the assumption φ and ending with the conclusion ψ;

2. `<->I`, (also denoted `BI`) which justifies an assertion of the form by citing two subproofs, beginning with assuptions φ, ψ, respectively, and ending with conclusions ψ, φ, respectively;
3. `~I`, which justifies an assertion of the form φ by citing a subproof beginning with the assumption φ and ending with a pair of lines ψ, ψ.
4. `~E`, which justifies an assertion of the form by citing a subproof beginning with the assumption φ and ending with a pair of lines ψ, ψ.
5. `E`, which justifies an assertion of the form by citing a line , and two subproofs beginning the assumptions φ, ψ respectively and both ending with a line .

Finally, `PR` or `Assumption` (to keep with the *Logic Book* terminology) can be used to justify a line asserting a premise, and `AS` can be used to justify a line making an assumption. `A/SOMEADDITIONALTEXT` (where `SOMEADDITIONALTEXT` indicates some additional text to be included alongside the assumption in the rendered proof) can also be used to justify an assumption. Assumptions are only allowed on the first line of a subproof.

20.1.2.2 Logic Book System SD Plus

The extended system SD Plus (the system used in a proofchecker constructed with `.LogicBookSDPlus` in Carnap's Pandoc Markup) also adds the following rules:

Rule	Abbreviation	Premises	Conclusion
Modus Tollens	`MT`	$\varphi \supset \psi, \sim \psi$	$\sim \varphi$
Hypothetical Syllogism	`HS`	$\varphi \supset \psi, \psi \supset \chi$	$\varphi \supset \chi$
Disjunctive Syllogism	`DS`	$\sim \psi, \varphi \vee \psi$	φ
		$\sim \varphi, \varphi \vee \psi$	ψ

As well as the following exchange rules, which can be used within a propositional context Φ:

Rule	Abbreviation	Premises	Conclusion
Commutation	`Comm`	$\Phi(\varphi \& \psi)$	$\Phi(\psi \& \varphi)$
		$\Phi(\varphi \vee \psi)$	$\Phi(\psi \vee \varphi)$
Association	`Assoc`	$\Phi(\varphi \& (\psi \& \chi))$	$\Phi((\varphi \& \psi) \& \chi)$
		$\Phi((\varphi \& \psi) \& \chi)$	$\Phi(\varphi \& (\psi \& \chi))$
		$\Phi(\varphi \vee (\psi \vee \chi))$	$\Phi((\varphi \vee \psi) \vee \chi)$
		$\Phi((\varphi \vee \psi) \vee \chi)$	$\Phi(\varphi \vee (\psi \vee \chi))$
Implication	`Impl`	$\Phi(\varphi \supset \psi) / \Phi(\sim \varphi \vee \psi)$	$\Phi(\varphi \supset \psi) / \Phi(\sim \varphi \vee \psi)$
Double Negation	`DN`	$\Phi(\varphi) / \Phi(\sim\sim \varphi)$	$\Phi(\sim\sim \varphi) / \Phi(\varphi)$
Idempotence	`Idem`	$\Phi(\varphi)$	$\Phi(\varphi \& \varphi)$
		$\Phi(\varphi \& \varphi)$	$\Phi(\varphi)$

Rule	Abbreviation	Premises	Conclusion
		$\Phi(\varphi)$	$\Phi(\varphi \vee \varphi)$
		$\Phi(\varphi \vee \varphi)$	$\Phi(\varphi)$
DeMorgan's Laws	`DeM`	$\Phi(\sim(\varphi\&\psi))$	$\Phi(\sim\varphi\vee\sim\psi)$
		$\Phi(\sim(\varphi\vee\psi))$	$\Phi(\sim\varphi\&\sim\psi)$
		$\Phi(\sim\varphi\vee\sim\psi)$	$\Phi(\sim(\varphi\&\psi))$
		$\Phi(\sim\varphi\&\sim\psi)$	$\Phi(\sim(\varphi\vee\psi))$
Transposition	`Trans`	$\Phi(\varphi \supset \psi)$	$\Phi(\sim\psi\supset\sim\varphi))$
		$\Phi(\sim\psi\supset\sim\varphi))$	$\Phi(\varphi \supset \psi)$
Exportation	`Exp`	$\Phi(\varphi \supset (\psi \supset \chi))$	$\Phi(\varphi\&\psi \supset \chi)$
		$\Phi(\varphi\&\psi \supset \chi)$	$\Phi(\varphi \supset (\psi \supset \chi))$
Distribution	`Dist`	$\Phi(\varphi\&(\psi\vee\chi))$	$\Phi((\varphi\&\psi)\vee(\varphi\&\chi))$
		$\Phi((\varphi\&\psi)\vee(\varphi\&\chi))$	$\Phi(\varphi\&(\psi\vee\chi))$
		$\Phi(\varphi\vee(\psi\&\chi))$	$\Phi((\varphi\vee\psi)\&(\varphi\vee\chi))$
		$\Phi((\varphi\vee\psi)\&(\varphi\vee\chi))$	$\Phi(\varphi\vee(\psi\&\chi))$
Equivalence	`Equiv`	$\Phi(\varphi \leftrightarrow \psi)$	$\Phi(\varphi \supset \psi\&\psi \supset \varphi)$
		$\Phi(\varphi \supset \psi\&\psi \supset \varphi)$	$\Phi(\varphi \leftrightarrow \psi)$
		$\Phi(\varphi \leftrightarrow \psi)$	$\Phi((\varphi\&\psi)\vee(\sim\psi\&\sim\varphi))$
		$\Phi((\varphi\&\psi)\vee(\sim\psi\&\sim\varphi))$	$\Phi(\varphi \leftrightarrow \psi)$

20.2 First-Order Systems

20.2.1 Notation

The different admissible keyboard abbreviations for quantifiers are as follows:

Connective	Keyboard
	`A`
	`E`

The Logic Book first order systems contain sentence letters A through Z, together with the infinitely many subscripted letters $P_1, P_2, ...$ written `P_1, P_2` and so on.

Application of a relation symbol is indicated by directly appending the arguments to the symbol.

The available relation symbols are A through Z, together with the infinitely many subscripted letters $F_1, F_2, ...$ written `F_1, F_2,…`. The arity of a relation symbol is determined from context.

The available constants are a through v, with the infinitely many subscripted letters $c_1, c_2, ...$ written `c_1, c_2,…`.

The available variables are w through z, with the infinitely many subscripted letters $x_1, x_2, ...$ written `x_1, x_2,…`.

Quantificational phrases are formed by appending a variable to a quantifier, and wrapping the result in parentheses.

20.2.2 Basic Rules

The first-order *Logic Book* systems PD and PD+ (the systems used in proofcheckers constructed with `.LogicBookPD`, and `LogicBookPDPlus` respectively) extend the rules of the system SD and SD+ respectively with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Existential Introduction	`EI`	$\varphi(\sigma)$	$(\exists x)\varphi(x)$
Universal Elimination	`AE`	$(\forall x)\varphi(x)$	$\varphi(\sigma)$
Universal Introduction	`AI`	$\varphi(\sigma)$	$(\forall x)\varphi(x)$

Where Universal Introduction is subject to the restriction that σ must not appear in $\varphi(x)$, or any undischarged assumption or in any premise of the proof.[2]

It also adds one new rule for indirect derivations: `EE`, which justifies an assertion ψ by citing an assertion of the form $(\exists x)\varphi(x)$ and a subproof beginning with the assumption $\varphi(\sigma)$ and ending with the conclusion ψ, where σ does not appear in $\psi, (\exists x)\varphi(x)$, or in any of the undischarged assumptions or premises of the proof.

To these rules, PD+ also adds the exchange rule:

Rule	Abbreviation	Premises	Conclusion
Quantifier Negation	`QN`	$\Phi(\sim(\forall x\varphi)(x))$	$\Phi((\exists x) \sim \varphi(x))$
		$\Phi((\exists x) \sim \varphi(x))$	$\Phi(\sim(\forall x)\varphi(x))$
		$\Phi(\sim(\exists x\varphi)(x))$	$\Phi((\forall x) \sim \varphi(x))$
		$\Phi((\forall x) \sim \varphi(x))$	$\Phi(\sim(\exists x)\varphi(x))$

[2]The fact that $\varphi(a)$ needs to be cited as a premise is undesirable, and will be changed relatively soon, pending some improvements to the proof checking algorithm.

21 Natural Deduction in the Carnap Book

This document gives a short description of the systems of natural deduction used in the Carnap book, and the two available second-order extensions of these systems. At least some prior familiarity with Montague-style proof systems is assumed.

21.1 The Propositional System

21.1.1 Notation

The different admissible keyboard abbreviations for the different connectives is as follows:

Connective	Keyboard
$\rightarrow$	`->`, `=>`,`>`
	`/\`, `&`, `and`
	`\/`, `|`, `or`
	`<->`, `<=>`
$\neg$	`-`, `~`, `not`

The available sentence letters are P through W, and the infinitely many subscripted letters $P_1, P_2, ...$ written `P_1, P_2` and so on.

Proofs consist of a series of lines. A line is either an assertion line containing a formula followed by a `:` and then a justification for that formula, a show line consisting of the word `show` followed by a formula, or a QED line containing a `:` followed by a rule for an indirect inference.

A subproof begins a new show line. The next line should be more indented than the containing proof, and the lines directly contained in this subproof should maintain this indentation level. (Lines indirectly contained, by being part of a sub-sub-proof, will need to be indented more deeply.) The subproof ends with a QED line at the same level of indentation as the containing proof.

For example:

```
Show P -> (P -> P)
   P:AS
   Show: P->P
      P:AS
   :CD 4
:CD 3
```

21.1.2 Basic Rules

The propositional part of the system (the system used in a proofchecker constructed with `.Prop` in Carnap's Pandoc Markup) has the following set of rules for direct inferences:

Rule	Abbreviation	Premises	Conclusion
Simplification	`S`	$\varphi \wedge \psi$	φ / ψ
Adjunction	`ADJ`	φ, ψ	$\varphi \wedge \psi$
Modus Ponens	`MP`	$\varphi, \varphi \rightarrow \psi$	ψ
Modus Tollens	`MT`	$\neg\psi, \varphi \rightarrow \psi$	$\neg\varphi$
Modus Tollendo Ponens	`MTP`	$\neg\psi / \neg\varphi, \varphi \vee \psi$	φ / ψ
Addition	`ADD`	φ / ψ	$\varphi \vee \psi$
Conditional to Bicond.	`CB`	$\psi \rightarrow \varphi, \varphi \rightarrow \psi$	$\psi \leftrightarrow \varphi$
Biconditional to Cond.	`BC`	$\psi \leftrightarrow \varphi$	$\psi \rightarrow \varphi / \varphi \rightarrow \psi$
Double Negation Intro.	`DNI`	φ	$\neg\neg\varphi$
Double Negation Elem.	`DNE`	$\neg\neg\varphi$	φ
Double Negation	`DN`	$\varphi / \neg\neg\varphi$	$\neg\neg\varphi / \varphi$

We also have three rules for indirect inferences: `DD`, which closes a show line of the form $Show : \varphi$ by citing an available line containing φ , `CD` which closes a show line of the form $Show : \varphi \rightarrow \psi$ by citing an available line containing ψ, and cancels the assumption φ if it is used to support this line, and `ID`, which choses a show line of the form $Show : \neg\varphi$ by citing two available lines containing ψ and $\neg\psi$, canceling the assumption φ if it is used to support these lines.

Finally, `PR` can be used to justify a line asserting a premise, and `AS` can be used to justify a line making an assumption.

21.1.3 Derived Rules

Additional rules previously derived by users are allowed. These are regular rules of direct inference, and use abbreviations of the form `D-NAME`, where `NAME` is chosen by the user.

21.2 The First Order System

21.2.1 Notation

The different admissible keyboard abbreviations for quantifiers and equality is as follows:

Connective	Keyboard
	`A`
	`E`
$=$	`=`

Sentence letters are as in the propositional system.

Application of a function or relation symbol is indicated by wrapping a comma-separated list of arguments in parentheses, and appending this to the symbol.

The available relation symbols are F through O, with the infinitely many subscripted letters $F_1, F_2, ...$ written `F_1, F_2,….` The arity of a relation symbol is determined from context.

The available constants are a through e, with the infinitely many subscripted letters $c_1, c_2, ...$ written `c_1, c_2,….`

The available function symbols are f through h, with the infinitely many subscripted letters $f_1, f_2, ...$ written `f_1, f_2,….` The arity of a function symbol is determined from context.

The available variables are v through z, with the infinitely many subscripted letters $x_1, x_2, ...$ written `x_1, x_2,….`

21.2.2 Basic Rules

The first-order part of the system (the system used in a proofchecker constructed with `.FirstOrder`) extends the propositional part of the system with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Leibniz's Law	`LL`	$\tau = \sigma, \varphi(\tau)/\varphi(\sigma)$	$\varphi(\sigma)/\varphi(\tau)$
Leibniz's Law (negative)	`LL`	$\varphi(\tau), \neg\varphi(\sigma)$	$\neg(\tau = \sigma)$
Euclid's Law	`EL`	$\tau = \sigma$	$\theta(\tau) = \theta(\sigma)$
Universal Instantiation	`UI`	$\forall x\varphi(x)$	$\varphi(\tau)$
Existential Generalization	`EG`	$\varphi(\tau)$	$\exists x\varphi(x)$
Quantifier Negation	`QN`	$\neg\forall x\varphi(x)/\neg\exists x\varphi(x)/\exists\neg x\varphi(x)/\forall\neg x\varphi(x)$	$\exists\neg x\varphi(x)/\forall\neg x\varphi(x)/\neg\forall x\varphi(x)/\neg\exists x$
Symmetry	`Sm`	$\tau = \sigma$	$\sigma = \tau$

Rule	Abbreviation	Premises	Conclusion
Reflexivity	`ID`		$\tau = \tau$

It also adds two new rules for indirect derivations: `UD`, which closes a show line of the form $Show : \forall x \varphi(x)$ by citing an available line $\varphi(a)$, where a is a fresh constant[1] , and `ED` which closes a show line of the form $Show : \psi$ by citing an assertion of the form $\exists x \varphi(x)$, an assumption of the form $\varphi(a)$ and an assertion of the form ψ, and cancels the assumption $\varphi(a)$, where a is a fresh constant.[2]

21.2.3 Traditional Kalish and Montague Rules

The basic rules for first-order logic listed above are not precisely the same as the rules given by Kalish and Montague in *Formal Logic*. In particular, universal derivations work differently, and existential derivations are replaced by a rule of existential instantiation.

Universal Derivations in *Formal Logic* closing a show line of the form $Show : \forall x \varphi(x)$ are required to end by citing an available line $\phi(x)$ in which ϕ occurs with precisely the same variable free as we see bound in $\forall x \phi(x)$. Furthermore, the variable x must not occur free on any line available from the salient show line.

The rule of `EI` of existential instantiation allows one to infer from a premise of the form $\exists x \phi(x)$ that $\phi(y)$ for some variable not occurring *free or bound* on any earlier line in the proof.

These rules can be activated by using `.MontagueQC` (for "Quantifier Calculus") instead of `.FirstOrder` in the construction of the proof-checker.

21.3 The Monadic Second-Order System

21.3.1 Notation

The different admissible keyboard abbreviations for monadic second-order lambdas is as follows:

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

[2] The fact that $\varphi(a)$ needs to be cited as a premise is undesirable, and will be changed relatively soon, pending some improvements to the proof checking algorithm.

Connective	Keyboard
	\

First order syntax is as in the first-order system.

The available second-order variables are X through Z, with the infinitely many subscripted variables $X_1, X_2, ...$ written `X_1, X_2,…`.

A lambda abstract has the form $\lambda x[\varphi(x)]$, and can be applied like a predicate. Lambdas can only be applied to formulas, so the only abstracts that can be produced are for monadic relations.

21.3.2 Basic Rules

The monadic second-order system (the system used in a proofchecker constructed with `.SecondOrder`) extends the first-order part of the system with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
Abstraction	`ABS`	$\varphi(\sigma)$	$\lambda x[\varphi(x)](\sigma)$
Application	`APP`	$\lambda x[\varphi(x)](\sigma)$	$\varphi(\sigma)$
Monadic Univ. Instantiation	`UI`	$\forall X\Phi(X)$	$\Phi(\zeta)$
Monadic Ex. Generalization	`EG`	$\Phi(\zeta)$	$\exists X\Phi(X)$

Where is schematic for a unary lambda term.

And with rules for indirect derivation UD and EG exactly analogous to the first-order case, except that a fresh second-order variable is used rather than a fresh constant.

21.4 The Polyadic Second-Order System

21.4.1 Notation

First order syntax is as in the first-order system.

The available 2nd-order n-ary variables are XN through ZN (where N is replaced by the arity of the variable), with the infinitely many subscripted variables $XN_1, XN_2, ...$ written `XN_1, XN_2,…`.

A lambda abstract has the form $\lambda x_1 ... \lambda x_n[\varphi(x_1, ..., x_n)]$, and can be applied like an n-ary relation symbol.

21.4.2 Basic Rules

The polyadic second-order system (the system used in a proofchecker constructed with `.PolySecondOrder`) extends the first-order part of the system with the following set of new basic rules:

Rule	Abbreviation	Premises	Conclusion
n-Abstraction	`ABSN`	$\varphi(\sigma_1, ..., \sigma_n)$	$\lambda x_1 ... x_n[\varphi(x_1, ..., x_n)](\sigma_1, ...\sigma_n)$
n-Application	`APPN`	$\lambda x_1 ... x_n[\varphi(x_1, ..., x_n)](\sigma_1, ...\sigma_n)$	$\varphi(\sigma_1, ..., \sigma_n)$
n-ary Univ. Instantiation	`UIN`	$\forall XN\Phi(XN)$	$\Phi(\zeta^n)$
n-ary Ex. Generalization	`EGN`	$\Phi(\zeta^n)$	$\exists XN\Phi(XN)$

Where ζ^n is schematic for an n-ary lambda term.

And with rules of n-ary universal and existential derivation `UDN` and `EGN` exactly analogous to the first-order case, except that a fresh second-order variable is used rather than a fresh constant.

22 Set Theory Demo

Here's a demo of Carnap's (alpha-stage) support for natural deduction in set theory.

The rules are those of the Carnap book's first-order system (The one used in a proof checker constructed with `.FirstOrder`—See the documentation for Montague Systems), plus the following bi-directional replacement rules:

Rule	Abbreviation	Premises	Conclusion
Union Definition	`Def-U`	$\Phi(\tau \in \sigma \vee \tau \in \theta)$	$\Phi(x \in \sigma \cup \theta)$
Intersection Definition	`Def-I`	$\Phi(\tau \in \sigma \wedge \tau \in \theta)$	$\Phi(x \in \sigma \cap \theta)$
Complement Definition	`Def-C`	$\Phi(\tau \in \sigma \wedge \neg\tau \in \theta)$	$\Phi(x \in \sigma/\theta)$
Power Set Definition	`Def-P`	$\Phi(\tau \subseteq \theta)$	$\Phi(\tau \in Pow(\theta))$
Subset Definition	`Def-S`	$\Phi(\forall x(x \in \tau \rightarrow x \in \theta))$	$\Phi(\tau \subseteq \theta)$
Separation Definition	`Def-{}`	$\Phi(\tau \in \theta \wedge \varphi(\tau))$	$\Phi(\tau \in x \in \theta \vert \varphi(x))$
Equality Definition	`Def-=`	$\Phi(\forall x(x \in \tau \leftrightarrow x \in \sigma))$	$\Phi(\tau = \sigma)$

Symbols are:

Symbol	Keyboard
	`I`
	`U`,
	`in`, `<<`, `<e`
	`within`, `<(`, `<s`

Here's an example proof:

```
Ax(x in a -> x within a) :PR
Show P(a) within P(P(a))
 Show Ax(x in P(a) -> x in P(P(a)))
  Show b in P(a) -> b in P(P(a))
   b in P(a) :AS
```

```
   b within a:Def-P 5
   Ax(x in b -> x in a):Def-S 6
   Show b in P(P(a))
    Show Ax(x in b -> x in P(a))
     Show c in b -> c in P(a)
      c in b :AS
      c in b -> c in a:UI 7
      c in a :MP 11 12
      c in a -> c within a :UI 1
      c within a:MP 13 14
      c in P(a):Def-P 15
     :CD 16
    :UD10
    b within P(a):Def-S 9
    b in P(P(a)):Def-P 19
   :DD 20
  :CD 8
 :UD 4
 P(a) within P(P(a)):Def-S 3
:DD 24
```

Part V

Advanced Usage

23 Carnap deployment and administration

23.1 General information

The build infrastructure around Carnap supports two deployment modes: Stack builds for use with the Keter web server on traditional Linux servers and Nix builds for use on Docker or NixOS servers. Stack with Keter is used on Carnap.io. An experimental prebuilt Docker image is available.

If you would like to build Carnap yourself using Nix directly, see the development README.

We provide Nix packaging as the `server` attribute returned by `default.nix` for use in NixOS and NixOps.

If you wish to work with Carnap's Nix infrastructure outside the Docker images, it is highly recommended to use Cachix to avoid having to build anything that our Continuous Integration has already built.

23.2 Requirements

- PostgreSQL database with a user (TODO: which privileges do they need on their database?)
- Writable persistent directory to give as the `dataroot`
- Google API key (see below for information on how to configure it)

23.3 Server setup

23.3.1 Docker

There is experimental Docker support for Carnap. Images are available via the GitHub container registry at `ghcr.io/carnap/carnap/carnap:latest`.

There is a sample docker-compose environment with automatically-managed Caddy based HTTPS termination, the recommended PostgreSQL database, and full setup instructions available in the Carnap documentation repository here.

Docker configuration details

Carnap in Docker can most effectively be configured with environment variables. See the example settings file for a list. A volume should be provided at `/data` for persistent data such as documents.

At minimum, the following environment variables must be configured:

- `APPROOT`
- `GOOGLEKEY`
- `GOOGLESECRET`

23.3.2 Manual setup summary

If you want to set up Carnap manually (Docker is recommended instead):

Files:

- `/var/lib/carnap` (or whatever you're using as `DATAROOT`), writable by your `carnap` user you're running the server as, with subdirectories:
  - `static` from `Carnap-Server/static` (*copying* symlinks if you're deploying it)
  - `config` from `Carnap-Server/config`
  - `book` from `Carnap-Book`
  - `book/cache` directory created
  - `data` directory created

Environment Variables:

- `APPROOT=https://carnap.example.com`
- `DATAROOT=/var/lib/carnap`
- `BOOKROOT=/var/lib/carnap/book`

Assuming you're using PostgreSQL (`...` values are replaced with their respective real values):

- `SQLITE=false`
- `PGHOST=...`
- `PGPORT=...`
- `PGUSER=...`
- `PGPASS=...`
- `PGDATABASE=...`

23.3.3 Settings file

Carnap uses a settings file, `settings.yml`, to store its configuration. You can find the example version at `Carnap-Server/config/settings-example.yml`

The version in `Carnap-Server/config/settings.yml` is bundled within the executable. To utilize binary caching, Nix builds directly use the example configuration file, and users are expected to use environment variables or provide a different YAML file at runtime.

You can provide a different settings file at runtime by passing it as the first argument to the `Carnap-Server` executable. Also, all settings are configurable by environment variables, which is useful for Docker deployments (see the example configuration for a list of these settings).

23.3.4 `dataroot`

Carnap requires a writable `dataroot`, configured with the `DATAROOT` environment variable, to store documents and the cookie encryption key in.

This directory can also contain a subdirectory `srv`, whose contents are directly available at `/srv/` on the Carnap instance. This is used for documentation on the production instance at `carnap.io`.

23.3.5 Database

Due to a bug that breaks migrations on SQLite in our database library, production deployments of Carnap should use PostgreSQL databases. Set the environment variable `SQLITE=false` and supply `PGUSER`, `PGPASS`, `PGHOST`, and if required, `PGPORT` and `PGDATABASE` for your PostgreSQL database instance.

If you wish to use peer authentication via Unix socket on a locally hosted PostgreSQL database, set all of `PGUSER`, `PGPASS`, `PGHOST` and `PGPORT` to empty strings.

23.3.6 NixOps

The Carnap team at UBC uses NixOps to automatically deploy Carnap on DigitalOcean servers running NixOS. It's currently experimental and documentation is a work-in-progress, but the configuration files are public.

23.4 Authentication

Two authentication schemes are supported by Carnap, both of which require some configuration. Production Carnap deployments currently require Google authentication to be configured for administration purposes, even if it is not intended to be used by students.

See the LTI 1.3 documentation for details on how to configure that system after completing setup.

23.4.1 Google authentication

Setting up Google authentication requires setting up a Google APIs project.

First, create the project: https://console.developers.google.com/cloud-resource-manager

Then, create an OAuth2 client ID: https://console.developers.google.com/apis/credentials

It will show up like this:

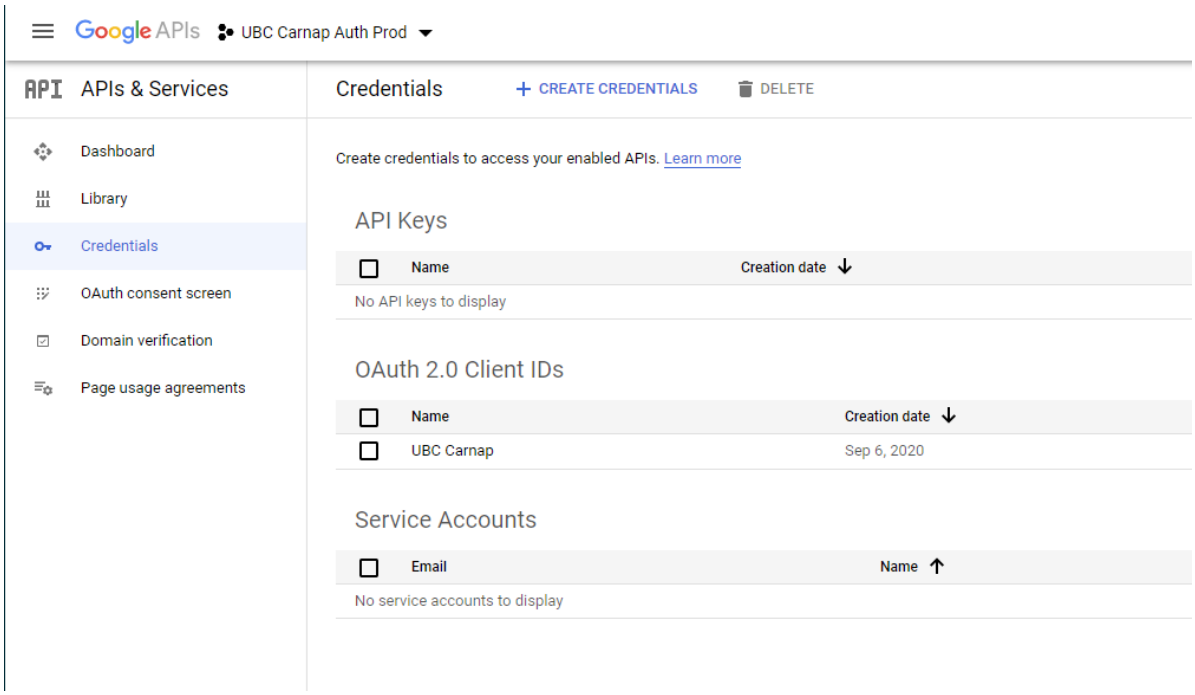


Figure 23.1: image of the client ID page

On the client ID page, set an Authorized Redirect URI for `https://YOURDOMAINHERE/auth/page/google/callb`

On your OAuth2 Consent Screen tab, no scopes need to be added as Carnap just needs emails to log in.

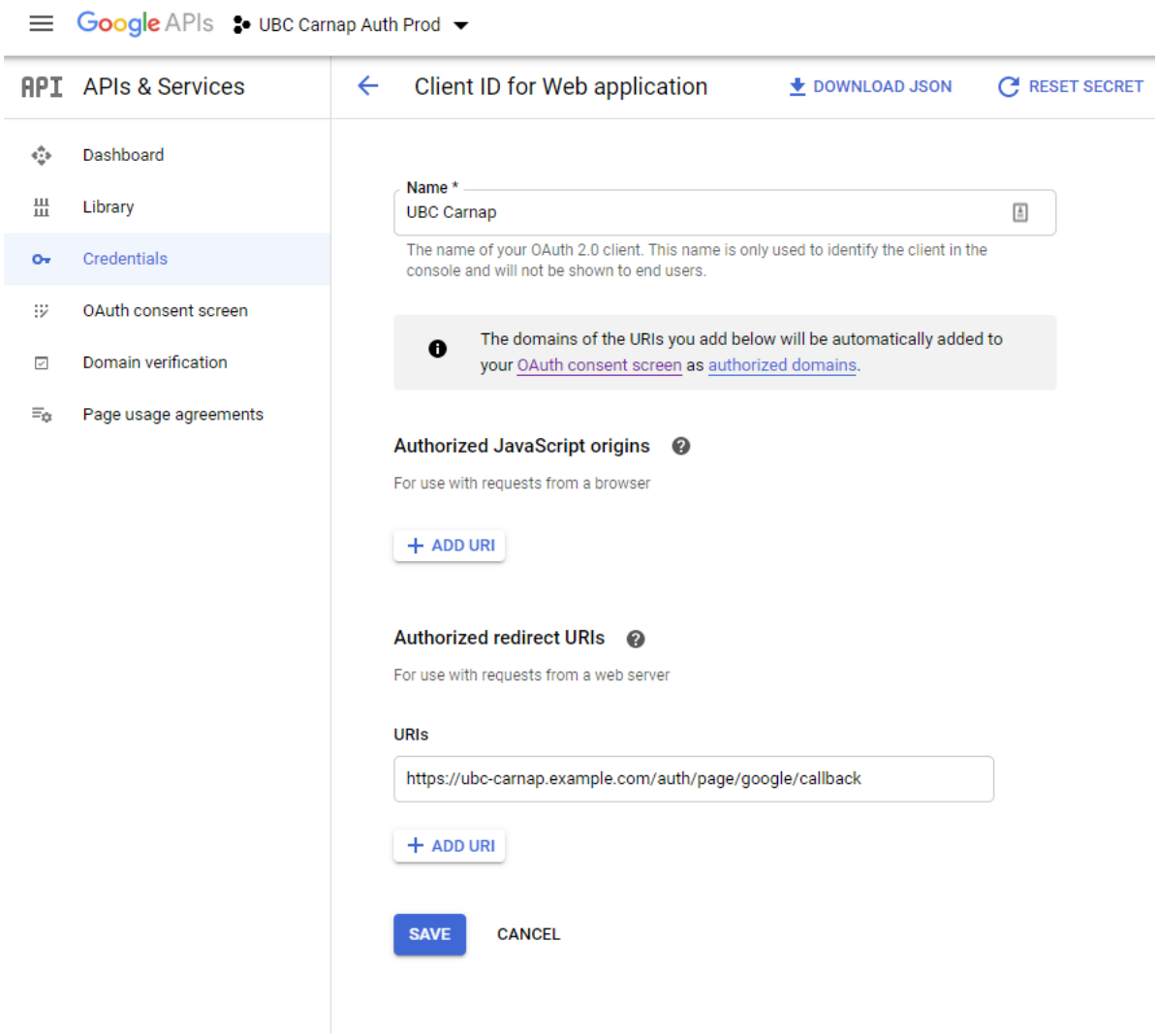


Figure 23.2: image of the authorized redirect URIs page

Once you've configured all the information on the Google side, fill it in the Carnap configuration file or environment variables:

- `google-api-key`/`GOOGLEKEY`: your Google Client ID
- `google-secret`/`GOOGLESECRET`: your Google Client secret

23.5 Post installation first-time setup

23.5.1 Becoming administrator

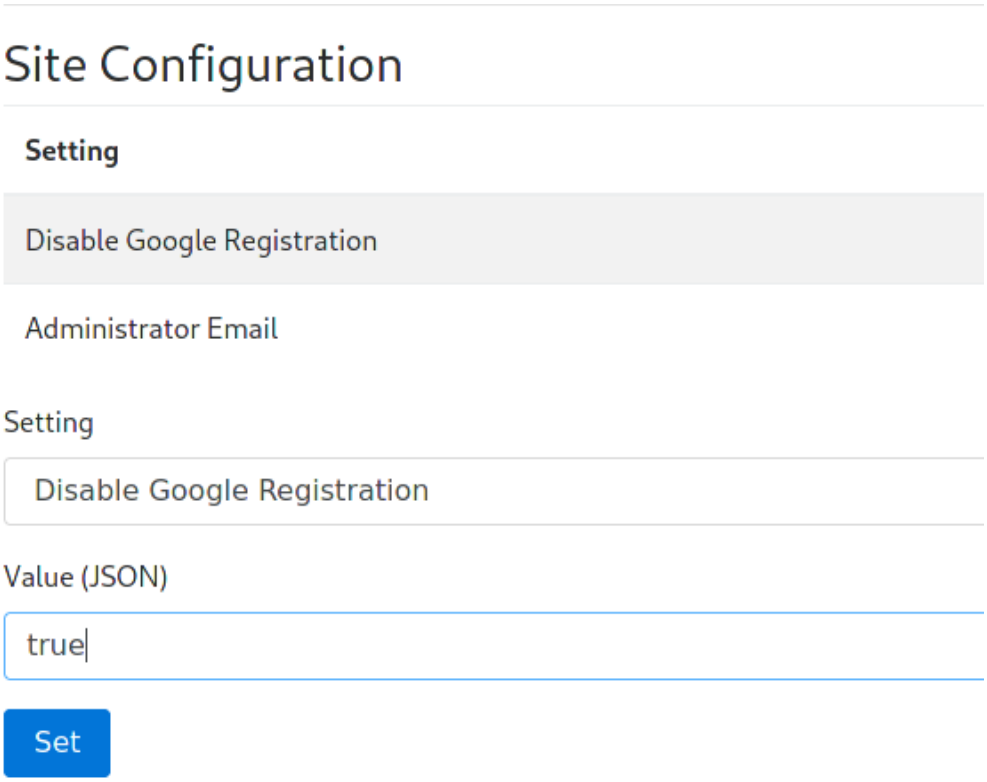
Go to your newly minted Carnap instance and log in with Google. Enter your name and register. Then, go to `https://YOURDOMAINHERE/admin_promote` and click the button. You will now be the administrator of this instance. Multiple administrators are supported (with manual database editing), but there is not yet user interface to enable this.

You can manage the site including promoting instructors, managing students, and configuring LTI platforms at `https://YOURDOMAINHERE/master_admin`.

23.5.2 Once you're an administrator

After you have become administrator, check the following items on the `master_admin` page:

1. Ask anyone who needs to be instructor to log in and register, then promote them to instructor.
2. If you don't want to allow further Google registrations (if you intend to use LTI with your learning management system for student login, setting this option is recommended), enter `true` for "Disable Google Registration" under the "Site Configuration" heading:



Anyone who has logged in already with Google can still log in, but new registrations will be prevented.

This can be disabled temporarily at any time if more instructors need to be added.

3. Configure login with LTI 1.3 compatible learning management systems using the LTI guide.

24 LTI 1.3

Carnap supports LTI 1.3 for login.[1] This allows Carnap to be launched directly from a Learning Management System, and to hand over management of user account data (like email, name, and student ID) as well as login, to the LMS.

Various major LMS implementations including Moodle, Brightspace, and Canvas support LTI 1.3 natively.

24.1 Basic Setup Outline

Connecting Carnap to your university's LMS involves three steps.

1. Your LMS needs to be configured to recognize Carnap
2. Your Carnap server needs to be configured to recognize your LMS
3. Your Carnap class needs to be configured to auto-enroll students who are launching Carnap from within your LMS.

Below, you'll find sections on each of these steps.

24.2 Step 1 - Configuring your LMS to recognize Carnap

There are a few generic pieces of information your LMS will need in order to connect to Carnap. These are as follows:

- `oidc_initiation_url`: https://carnap.io/auth/page/lti13/initiate
- `target_link_uri`: https://carnap.io
- Public JWK URL: https://carnap.io/auth/page/lti13/jwks
- Redirect URLs: https://carnap.io/auth/page/lti13/authenticate

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

If you are running your own instance of the Carnap server, you will use the same paths but with your custom domain name replacing "carnap.io".

However, different LMSes need this information to be entered in different places. This will generally be the responsibility of an LMS administrator, rather than the responsibility of an individual instructor, so you may need to work with your IT department to arrange the initial setup of the LMS integration.

Now - here are instructions on configuring your LMS, for the LMSes we have experience with. If something is missing or incorrect, please consider opening an issue here or sending an email to gleachkr@gmail.com.

Configuring Canvas to use Carnap

You'll need to begin by configuring an LTI key. This requires administrator access to your Canvas instance, so if you're an instructor, you may need to speak to IT. Instructions for configuring an LTI key for Canvas can be found here: Configuring an LTI key.

Canvas allows configurations to be imported rather than set manually, via the "Paste JSON" option described in the documentation linked above. A JSON file for easy configuration of a Canvas instance using the "Paste JSON" option is reproduced below.

Once the LTI key is configured in your Canvas instance, you'll also need to add the "Client Id" number from the key to your course. (If an adminstrator configured the key for you, you may need to ask them for the Client ID.) Instructions on adding Carnap to your course using the Client ID can be found here: Adding Carnap To Your Course.

For more details on Canvas setup with LTI 1.3, see:

- Technical documentation. This is also a nice general overview of the protocol.
- Configuring an LTI key
- Adding Carnap To Your Course

24.2.0.1 JSON Configuration with Canvas.

```
{
   "title": "Carnap",
   "description": "Carnap Logic Framework",
   "oidc_initiation_url": "https://carnap.io/auth/page/lti13/initiate",
   "target_link_uri": "https://carnap.io/",
   "public_jwk_url": "https://carnap.io/auth/page/lti13/jwks",
   "scopes": [
      "https://purl.imsglobal.org/spec/lti-ags/scope/lineitem",
      "https://purl.imsglobal.org/spec/lti-ags/scope/result.readonly",
      "https://purl.imsglobal.org/spec/lti-ags/scope/score",
```

```
        "https://purl.imsglobal.org/spec/lti-nrps/scope/contextmembership.readonly"
    ],
    "extensions": [
        {
            "domain": "carnap.io",
            "tool_id": "Carnap.io",
            "platform": "canvas.instructure.com",
            "settings": {
                "text": "Carnap",
                "selection_height": 800,
                "selection_width": 800,
                "privacy_level": "public",
                "placements": [
                    {
                        "text": "Carnap",
                        "enabled": true,
                        "placement": "course_navigation",
                        "message_type": "LtiResourceLinkRequest",
                        "target_link_uri": "https://carnap.io/",
                        "windowTarget": "_blank"
                    }
                ]
            }
        }
    ]
}
```

Note: It is possible to use Carnap in an `iframe` (so it appears in the Canvas page without opening a new tab), but there are caveats, especially around support for Safari and other WebKit browsers, since they are very aggressive about third-party cookie blocking. If you want to try this, remove the `"windowTarget": "_blank"` in the JSON.

Configuring Brightspace/D2L to use Carnap

Under LTI Security Settings, make sure that the following are checked to allow for automatic registration in your Carnap courses:

- Name
- Org Unit Information

Carnap will also use the Email property to fill in user information if it is present, but it is not required.

It appears that Brightspace doesn't support sending student ID numbers in the `lis` claim as `person_sourcedid` like Canvas does. If this is something that you need for gradebook purposes, it is possible we can make Carnap accept the Brightspace specific "Org Defined ID" or "User ID" properties for the Carnap "University ID" field.

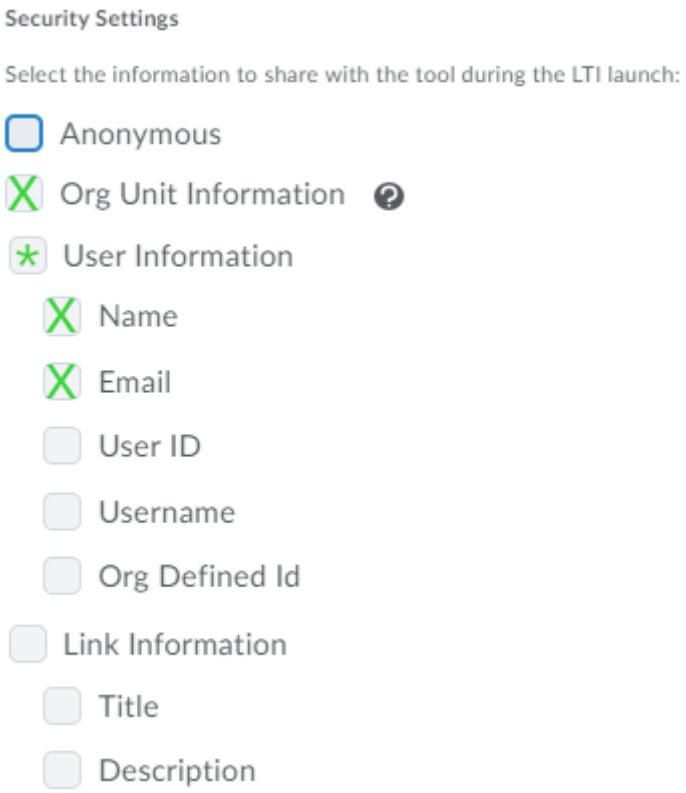


Figure 24.1: screenshot from brightspace documentation of the "security settings" subsection with "name" and "org unit information" checked

For more details on LTI configuration in Brightspace/D2L, see these pages:

- LTI Integration Guide (links to other pages)
- LTI Security Settings and Platform information
- LTI Advantage (LTI 1.3) Administrator Guide

24.3 Step 2 - Configuring Carnap to recognize your LMS

After your LMS is configured to talk to Carnap, it needs to be registered on the Carnap server. For setup with the public Carnap instance at Carnap.io, contact Graham with the following details from your LMS:

- Public JWKs URL
- Authorization Redirect URL
- Client ID

For example, for a cloud Canvas (production environment), these would be:

- JWK URL: `https://canvas.instructure.com/api/lti/security/jwks`
- Authorization redirect URL: `https://canvas.instructure.com/api/lti/authorize_redirect`
- Client ID: from your LTI developer key
- Deployment ID: not required for Carnap

If you are running a self-hosted instance of Carnap, you can configure LTI Platforms (Learning Management Systems) on the admin page at `https://carnap.example.com/master_admin`.

24.4 Step 3 - Configuring your Carnap class for auto-enrollment

Automatic registration links a class on the Carnap server to a class in an LMS, and automatically registers students in the Carnap class when they log in to Carnap from the associated course in the LMS LMS.

Once you've completed steps 1 and 2 above, attempt to log in to Carnap from the LMS course that you want to connect to Carnap. A message will appear on the Carnap user registration page giving you an autoregistration ID. The ID will look something like this:

```
{"ltiDeploymentId":"…", "ltiContextId":"…","ltiIssuer":"…","label":"…"}
```

but with concrete values replacing the ellipses.

Copy this ID down (the whole thing, including the enclosing brackets) and go to your instructor page on Carnap. Select the course that you wish to associate with your LMS course, and edit the course information. In the course information, there will be a field where you can paste the autoregistration ID.

Once that ID has been configured, all future launches from your LMS course will be registered in the associated Carnap course automatically.

24.4.1 Notes

- Automatic registrations are allowed even if your course is closed (even if you've unchecked the "course open" box on your instructor page). So, you can set registration to be LTI only simply by setting your course to be closed.

- Students will have their user information automatically synchronized with the LMS on every launch, so if they want to change their name or other details, that should be accomplished in the LMS or other upstream systems.

25 LTI Information for Carnap Developers

Setting up a Canvas instance or other LMS is quite burdensome for doing LTI testing. Therefore, the UBC Carnap team has kindly set up a LTI Reference Implementation platform here:

https://lti-ri.imsglobal.org/platforms/1255/

Configure it in Carnap at `/master_admin` with the following:

iss

client_id

OIDC Auth Endpoint

JWK URL

aaaaa

abcde

https://lti-ri.imsglobal.org/platforms/1255/authorizations/new

https://lti-ri.imsglobal.org/platforms/1255/platform_keys/1248.json

To perform launches, use the "Resource Links" page.

25.1 Debugging

Carnap records LTI tokens (with student data removed) to its server logs for the purposes of debugging automatic LTI registration, allowing to find out why automatic registration is failing. If that is not sufficient, some debugging can be done client-side by looking at browser request logs in the network tab of the developer tools:

Most of the process can be traced with dev tools. Note that Chrome now has a feature called "Auto-open DevTools for popups" that is ideal for debugging LTI failing to initiate properly in a popup. Remember to enable "Preserve log" before testing the problematic path.

These settings are accessible here:

Click the gear icon at the top right of the DevTools panel:

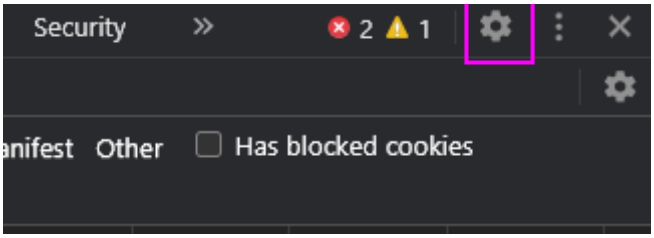


Figure 25.1: where that icon is

Then the setting is in the bottom right under Global:

Using this feature and the network tab, you can find the failing `initiate`/other request and figure out what the LMS is sending with its requests and how Carnap reacted.

Let's go through a successful sequence of requests in DevTools. My Canvas is at `http://ubuntu-vm:8900` and my Carnap is at `https://ubc-carnap-staging.lfcode.ca` (these are both test instances running on virtual machines on my computer).

I click the "Carnap" link in the Canvas sidebar. A new tab opens, Canvas goes through a couple of pages itself before sending us to Carnap.

First we have a POST to Carnap's `/auth/page/lti13/initiate` with the following parameters in the form data:

```
iss: https://canvas.instructure.com
login_hint: f326d6a8a55f30f47b2480586f97991ab9e602bb
client_id: 10000000000002
target_link_uri: https://ubc-carnap-staging.lfcode.ca/
lti_message_hint: some-long-jwt-that-is-opaque-per-the-standard
canvas_region: not_configured
```

The main thing to note here is that this is where Carnap has to decide where to send the user back to. This is done with a combination of `iss` and `client_id`, the latter of which is optional and we do our best if it's not there.

Carnap finds a record in its LTI platforms with the `issuer` and `client_id` pair, so it sends a 303 redirect to `http://ubuntu-vm:8900/api/lti/authorize_redirect` as configured in the Carnap LTI settings.

Carnap sends the browser a redirect to this URL:

Figure 25.2: where the setting is

```
http://ubuntu-vm:8900/api/lti/authorize_redirect
    ?scope=openid
    &response_type=id_token
    &client_id=10000000000002
    &redirect_uri=https%3A%2F%2Fubc-carnap-staging.lfcode.ca%2Fauth%2Fpage%2Flti13%2Fauthenti
    &login_hint=f326d6a8a55f30f47b2480586f97991ab9e602bb
    &state=3CjT-Wa3RKZjgge6bjPNEycuOToQdTUa1ELW3Jx38Gh6&response_mode=form_post
    &nonce=eara1fJmG39MIIijSv-ehGUMJsX2SVhlYhGZdxcsCJKa
    &prompt=none
    &lti_message_hint=some-long-jwt-that-is-opaque-per-the-standard
```

Things of note in this request:

- We generate the `redirect_uri`.
- The `client_id`, `login_hint`, and `lti_message_hint` are just from Canvas' initiation
- `state` and `nonce` are just randomly generated strings we check on the way back.
- Some of these parameters are kept in the session by Carnap, so if cookies don't work, we fail either at this request or the next.

The browser hits Canvas at that URL, and it bounces around a bit before coming back to Carnap with a POST `https://ubc-carnap-staging.lfcode.ca/auth/page/lti13/authenticate`.

Its parameters are:

```
utf8:
authenticity_token: VtRVOQdeoLQqfnzuNSa3kfg8vyuhcWj8lhsehEsmhx4ukCdsPxiT/3MYF416c+Tiyw7zGZFA
id_token: ey -- THIS IS A VERY LONG JWT TOKEN --
state: 3CjT-Wa3RKZjgge6bjPNEycuOToQdTUa1ELW3Jx38Gh6
```

We don't care about any of these except for the very long `id_token`, which is a JSON Web Token.

Carnap receives this, performs autoregistration, and goes to the user page. Thus, this is where you will want to start looking if you are having issues with attributes getting across, for example names, emails, etc.

View the token in a JWT debugging tool

26 JavaScript Extensions for Carnap

26.1 Including JavaScript

Carnap allows you to incorporate external JavaScript to an assignment, in order to add extra interactive behavior to your problem sets. This can be done using the `raw_html` pandoc extension to include a `<script>` tag, or by including the JS in a metadata block. Like a CSS entry, a JS entry entry can include either a url for a single script, like so:

```
---
js: https://carnap.io/shared/myemail@university.edu/myjs.js
---
```

or for several scripts, like so:

```
---
js:
- https://carnap.io/shared/myemail@university.edu/myjs1.js
- https://carnap.io/shared/myemail@university.edu/myjs2.js
---
```

The scripts can be hosted anywhere, including on the Carnap server. As with stylesheets, scripts can be uploaded as normal documents so long as they're given the proper filetype extension (in this case, ".js" rather than ".css")

26.2 Advanced Usage

26.2.1 Interacting with the Server

JavaScript running as part of an assignment will have access to an object called `CarnapServerAPI`, which lets your JavaScript interact with the server, to retrieve information about the current student and assignment, and to save information on the server for later retrieval.

Currently, the useful properties of `CarnapServerAPI` are as follows:

Property	Data
`CarnapServerAPI.user.firstName`	Student's first name
`CarnapServerAPI.user.lastName`	Student's first name
`CarnapServerAPI.assignment.dueDate`	Assignment due date ECMAScript epoch time (milliseconds si
`CarnapServerAPI.assignment.description`	Assignment description

There are two methods for dealing with saving data, one for saving and one for retrieving. Data is always saved under a "namespace", so that different JavaScript extensions can avoid overwriting one another's saved information. Each student has their own saved state for each assignment - state is not shared between students or between assignments.

Method	Result
`CarnapServerAPI.putAssignmentState(ns,state)`	Serializes `state` and inserts it under namespace `ns` on t
`CarnapServerAPI.getAssignmentState()`	Retrieves the assignment state asynchronously, as a `Pro`

The first call to `CarnapServerAPI.getAssignmentState` retrieves the state from the server, and subsequent calls return a locally cached state that's updated by `CarnapServerAPI.putAssignmentState`. So, while the first call to `CarnapServerAPI.getAssignmentState` might take a moment, subsequent calls should be quite fast.

26.3 Events

In order to make it possible for interactive JS behavior to be triggered by successful or unsuccessful completion of exercises, Carnap provides a few events that you can listen for.

When Carnap finishes its initial loading, and exercise elements are properly formatted and loaded, it will fire a global `carnap-loaded` event, which can be observed with `document.addEventListener("carnap-loaded", ...)`.

Whenever an exercise is successfully checked or submitted, an `exercise-success` event will fire on the main exercise element (the one carries the `data-carnap-*` attributes). An `exercise-failure` event fires when an exercise is checked and detected as incorrect, or a submission is rejected. Both `exercise-success` and `exercise-failure` are disabled if an exercise has the `exam` option set.

Finally, a `problem-submision` event fires on the submission button when a problem is submitted.

27 Installing Carnap

This guide will provide instructors for getting started with Carnap development: installing and building a local copy of the software that you can tinker with on your own machine. This is a useful thing to be able to do if you want to contribute changes and bugfixes to the project, or if you eventually want to deploy your own instance Carnap.

27.1 Requirements

Carnap has two components: `Carnap-Client` contains the code to display and check exercises. `Carnap-Server` contains the code for the Carnap web application (administering courses, uploading exercises, submitting solutions, etc.). Both have to be built and installed for Carnap to work.

To install Carnap you need: - A computer to build and run Carnap on. Currently Linux and MacOS are supported, but it is possible to build and run Carnap on Windows using the Windows Subsystem for Linux. - The source code for Carnap from GitHub. - Software to build Carnap.

Warning: building Carnap requires a large amount of memory (about 12GB) to compile. You should probably close all other applications and background services (Zoom, Slack, etc.) when doing it.

27.2 Building Carnap

27.2.1 Installing Nix

The easiest way to build Carnap is using the Nix package manager. Carnap comes with information Nix uses to build Carnap and all its dependencies automatically, without you having to install anything else by hand. The build using Nix will use self-contained tools to reduce the risk of incompatibilities between, e.g., a Haskell compiler you have installed and what Carnap requires.

To install Nix, follow the Installation instructions for your platform. On Linux, this just requires the command

```
$ sh <(curl -L https://nixos.org/nix/install)
```

from the terminal command line as a user who has `sudo` access.

(The `$` indicates the command line prompt, you only enter the commands to the right of it.)

If you want to compile using nix on a machine where you don't have `sudo` access, some alternative nix installation approaches are described in the nixos wiki. In particular a `nix-user-chroot` installation has been confirmed to work for compiling Carnap.

You can significantly speed up builds by using binaries for Carnap dependencies from Carnap's Cachix instance, which has support for both Linux and macOS. To use it, say:

```
$ nix-env -iA cachix -f https://cachix.org/api/v1/install
$ cachix use carnap
```

27.2.2 Download the Source Code

The latest version of Carnap's source code is available in the GitHub repository. Click on the "Code" button. If you have `git` installed and configured for ssh access, this command should do the trick:

```
$ git clone git@github.com:Carnap/Carnap.git
```

This downloads the source code into the directory `Carnap`. Change to that directory by saying

```
$ cd Carnap
```

27.2.3 Run the Development Server

Once you're in the `Carnap` directory, you can use `nix` to build the different components of the system. Some useful recipes for building in different ways have been recorded in Carnap's included Makefile. If you're interested in building a simple development server (which doesn't require any extra database or authentication configuration) then you can simply issue the command

```
make devel
```

This will build Carnap's client-side component (the JavaScript proof-checker that runs in the browser), and will then build and activate a Carnap server. By default, you'll be able to reach this server by navigating to `http://localhost:3000` on the computer where the server is running. Warning: as noted above, the initial build may take some time, and may require a lot of memory.

Issuing `make devel` again after the initial build will reactivate the server, unless you've made any changes to your copy of Carnap source code. If you have made changes, then reissuing `make devel` will rebuild any parts of the client and server that are downstream from your changes, and will then reactivate the server. Some behavior of the development server (like whether the server appears at http://localhost:3000, or somewhere else) can be controlled by changing environment variables. For details, see the server configuration documentation

27.3 Server Setup

Once your server is configured and running, you have to provide yourself with an administrator account if you want to create any instructors.

Go to your newly minted Carnap instance and log in - because you're running in development mode, you can log in as whatever you'd like by just typing in an identifier. Then go to `http://localhost:3000/admin_promote` (or to the corresponding address if you've edited the settings file so that you're serving on something other than `localhost:3000`) and click the button. You will now be the administrator of this instance. Multiple administrators are supported, but there is not yet a user interface to enable this.

You can now administrate your server (including promoting instructors, and configuring LTI platforms) at `http://localhost:3000/master_admin`

27.4 Carnap's user documentation

> Does this section belong in server configuration?

The data directory (by default, `Carnap/dataroot`) can also contain a subdirectory `srv`, whose contents are directly available at `/srv/` on the Carnap instance. This is used for documentation on the production instance at `carnap.io`. The menu for Carnap instructors contains a link "Documentation" which points to `/srv/doc/index.md`.

If you want site-specific documentation, you should create the directory `srv/doc/` inside your data directory, and place a MarkDown file `index.md` there as the starting point. If you want to provide Carnap's official documentation on your Carnap instance, you can download it into said directory:

```
$ cd dataroot
$ git clone git@github.com:Carnap/Carnap-Documentation.git srv
```

28 Carnap API

Carnap has an experimental API, which is currently under development (see the tracking issues below).

28.1 Tracking issues

- Issue #226 "Perform instructor actions via an API"
- Issue #231 "Documents API"
- Issue #230 "Copy course"

28.2 Instructor setup

To use the API, you need an API key. Your API key should be kept a secret, since it allows anyone who has it to access and modify your courses.

You can create an API key for yourself on the instructor page, beneath the "Manage Uploaded Documents" tab, here:

28.3 Usage

28.3.1 Example Scripts

Here are some examples to help you start using the API with different programming languages. We assume that you're testing, so the URL begins with `http://localhost:3000`. This should instead be `https://YOURSERVERRDOMAIN` when you're using this with a non-local server

28.3.1.1 Python

Create the file `apitest.py`:

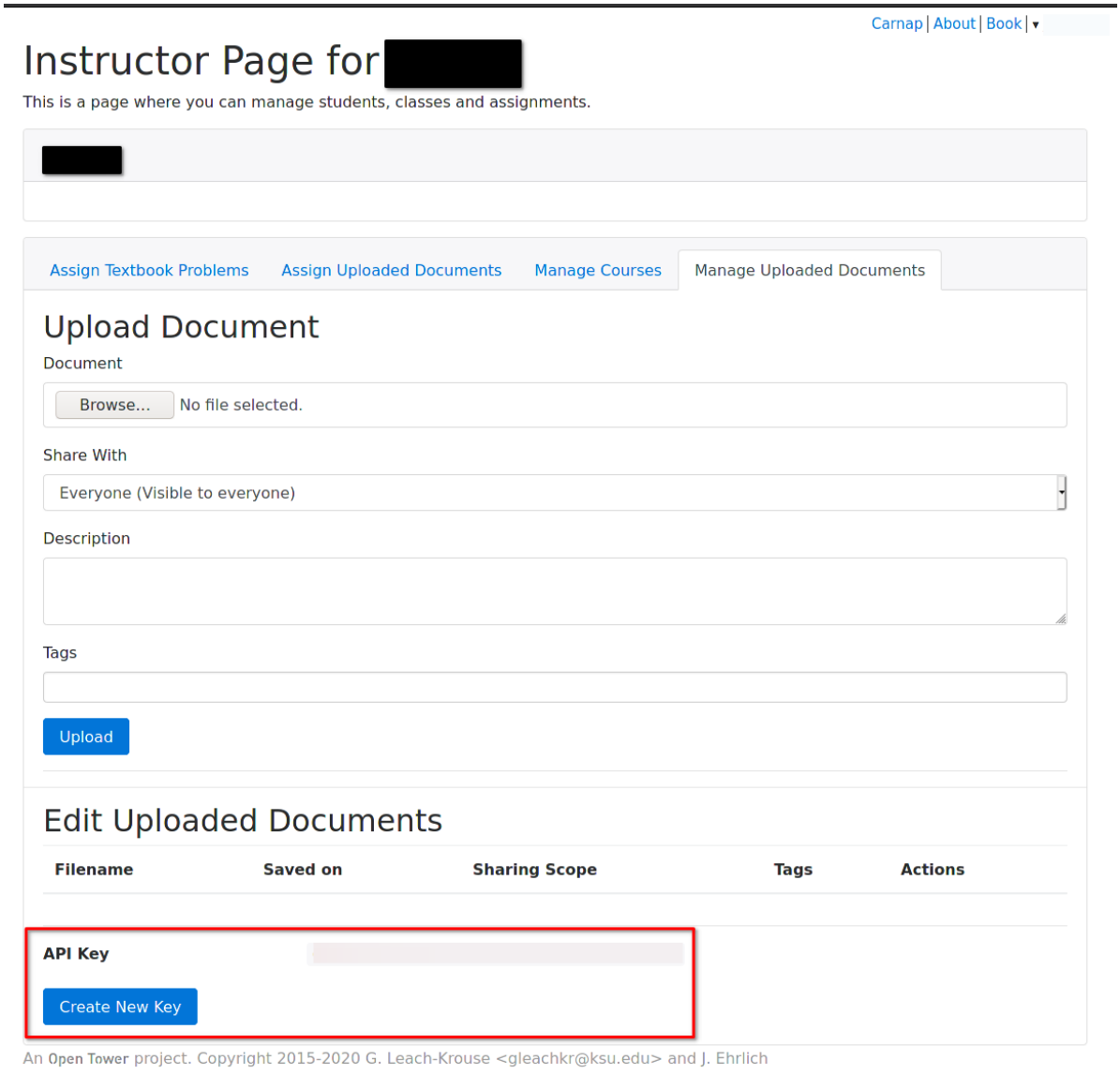


Figure 28.1: image showing the API key field at the bottom of the screen

```
import requests

apikey = 'REDACTED'

base = 'http://localhost:3000/api/v1'

def rq(meth, p, *args, key=apikey, **kwargs):
    h = {
        'X-API-KEY': key,
    }
    return requests.request(meth, base + p, *args, **kwargs, headers=h)
```

To issue the commands displayed below interactively `ipython -i apitest.py`, or `python -i apitest.py`.

28.3.1.2 Bash

Just use `curl` as indicated below.

28.4 Available API Methods

28.4.1 Document API

All of the methods in the documents API require that `:instructorIdent` is the same as the user who owns the API key (i.e. they do not work on other users' documents yet).

Tracking issue

28.4.1.0.1 GET `/instructors/:instructorIdent/documents`

Retrieves a list of documents owned by the given instructor and their metadata.

Here are some example commands:

Python

```
rq('GET', '/instructors/yourname@gmail.com/documents').json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEYHERE" localhost:3000/api/v1/instructors/yourname@youremail.co
```

Result

And here's what your result will look like:

```
[{'creator': 1,
  'date': '2021-02-16T09:41:39.445672522Z',
  'scope': 'Public',
  'id': 1,
  'description': None,
  'filename': 'api.md'}]
```

28.4.1.0.2 POST `/instructors/:instructorIdent/documents`

Creates a new document with empty contents.

Should be followed by a `PUT` at `/instructors/:instructorIdent/documents/:documentId/data` in order to fill in the document contents.

Python

```
rq('POST', '/instructors/yourname@gmail.com/documents',
    json={
        "filename": "myfile.md",
        "scope": "Private",
        "description": "My file",
    }
).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" \
     -H "Content-Type: application/json" \
     -d '{"filename":"myfile.md","scope":"Private", "description":"My file"}' \
     localhost:3000/api/v1/instructors/yourname@gmail.com/documents
```

Result

The ID of the new document:

```
3
```

or an error is returned as an encoded JSON string.

The response will also include a `Location` header pointed at the new resource. `scope` indicates the sharing scope of the document, and can be one of `Private`, `Public`, `LinkOnly` or

`InstructorsOnly`. Both the scope and description fields can be omitted, with scope defaulting to `Private`.

28.4.1.0.3 GET `/instructors/:instructorIdent/documents/:documentId`

Like GET `/instructors/:instructorIdent/documents` but for a single document.

Python

```
rq('GET', '/instructors/yourname@gmail.com/documents/1').json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEYHERE" localhost:3000/api/v1/instructors/yourname@youremail.co
```

Result

```
{'creator': 1,
 'date': '2021-02-16T09:41:39.445672522Z',
 'scope': 'Public',
 'description': None,
 'filename': 'api.md'}
```

28.4.1.0.4 PATCH `/instructors/:instructorIdent/documents/:documentId`

Updates the metadata for a single document.

Python

```
rq('PATCH', '/instructors/yourname@gmail.com/documents/1',
    json={"scope": "Private"}).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" -X "PATCH" -d '{"scope":"Public"}' \
     localhost:3000/api/v1/instructors/yourname@youremail.com/documents/1
```

Result

```
{'creator': 1,
 'date': '2021-02-16T09:41:39.445672522Z',
 'scope': 'Private',
```

```
 'description': None,
 'filename': 'api.md'}
```

Currently `scope` and `description` fields can be updated. Passing in a null value for `description` will cause the document description to be cleared entirely.

28.4.1.0.5 GET `/instructors/:instructorIdent/documents/:documentId/data`

Gets the content of the given document by ID and returns it.

Python

```
rq('GET', 'instructors/yourname@youremail.com/documents/1/data').text
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEYHERE" \
     localhost:3000/api/v1/instructors/yourname@youremail.com/documents/1/data
```

Result

The contents of your document, as text

28.4.1.0.6 PUT `/instructors/:instructorIdent/documents/:documentId/data`

Overwrites the content of the given document by ID.

These examples use "aaaaaa" as a thing you might insert as the document contents.

Python

```
rq('PUT', '/instructors/yourname@gmail.com/documents/1/data', data='aaaaaa')
```

Bash

```
echo aaaaaa | curl -H "X-API-KEY:YOURAPIKEYHERE"  -T "-" \
    localhost:3000/api/v1/instructors/gleachkr@gmail.com/documents/1940/data
```

Result

```
<Response [200]>
```

28.4.2 Course API

Note: `:courseTitle` in the below needs to be properly URL-encoded for `curl`, which means in particular that each space needs to be replaced with `%20`. The python requests module does URL-encoding automatically, so spaces can be used verbatim. The examples below are for a course with the title "test course".

28.4.2.0.1 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses`

Retrieves a list of courses.

```
rq('GET', '/instructors/gleachkr@gmail.com/courses/test course/students').json()
```

```
curl -H "X-API-KEY:YOURAPIKEYHERE" \
    'localhost:3000/api/v1/instructors/gleachkr@gmail.com/courses'
```

Result

```
[
  {
    "textBook": 12,
    "instructor": 1123,
    "enrollmentOpen": false,
    "endDate": "2021-05-19T04:59:59Z",
    "startDate": "2021-01-26T05:59:59Z",
    "textbookProblems": {
      "readAssignmentTable": [
        [
          1,
          "2021-05-15T04:59:59Z"
        ],
        ...
      ]
    },
    "totalPoints": 0,
    "title": "PHIL0680 - Independent Study",
    "timeZone": "America/Chicago",
    "description": null
  }
]
```

The `instructor` field gives the main instructor's instructorId, which is distinct from their userId. The `textBook` field gives the `assignmentId` for an associated textbook, which is the same data as the `id` field for an assignment, retrieved via GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/assignments`

28.4.2.0.2 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves a list of student data for a given course, including unique `:studentId` identifiers for each student.

```
rq('GET', '/instructors/gleachkr@gmail.com/courses/test course/students').json()
```

```
curl -H "X-API-KEY:YOURAPIKEYHERE" \
    'localhost:3000/api/v1/instructors/gleachkr@gmail.com/courses/test%20course/students'
```

Result


```
[
  {
    "email": "groundworker@gmail.com",
    "lastName": "Kant",
    "universityId": null,
    "userId": 1231,
    "firstName": "Immanuel",
    "isAdmin": false,
    "id": 1313,
    "enrolledIn": 141,
    "instructorId": null,
    "isLti": false
  },
  {
    "email": "thinker@gmail.com",
    "lastName": "Descartes",
    "universityId": null,
    "userId": 1232,
    "firstName": "Rene",
    "isAdmin": false,
    "id": 1314,
    "enrolledIn": 141,
    "instructorId": null,
    "isLti": false
```

```
  },
]
```

The `:studentId` for a student is the number in the `id` field, *not* the number in the `userId` field.

28.4.2.0.3 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves student data for the student with id `:studentId`

```
rq('GET', '/instructors/gleachkr@gmail.com/courses/test course/students/1231').json()
```

```
curl -H "X-API-KEY:YOURAPIKEYHERE" \
    'localhost:3000/api/v1/instructors/gleachkr@gmail.com/courses/test%20course/students/1
```

Result

```
{
    "email": "groundworker@gmail.com",
    "lastName": "Kant",
    "universityId": null,
    "userId": 1231,
    "firstName": "Immanuel",
    "isAdmin": false,
    "id": 1313,
    "enrolledIn": 141,
    "instructorId": null,
    "isLti": false
}
```

The `:studentId` for a student is the number in `id` field, *not* the number in the `studentId` field.

28.4.2.0.4 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves a list of all submission data for the student with id `:studentId`

```
rq('GET', '/instructors/gleachkr@gmail.com/courses/test course/students/1231/submissions')
```

```
curl -H "X-API-KEY:YOURAPIKEYHERE" \
    'localhost:3000/api/v1/instructors/gleachkr@gmail.com/courses/test%20course/students/1
```

Result

```
[
  {
    'problemSubmissionAssignmentId': 3001,
    'problemSubmissionCorrect': False,
    'problemSubmissionData': {..depends on problem type..},
    'problemSubmissionSource': {'tag': 'Assignment', 'contents': 'AssignmentMetadataKey {unAs
    'problemSubmissionIdent': 'Exercise-24',
    'problemSubmissionUserId': 10385,
    'problemSubmissionLateCredit': None,
    'problemSubmissionType': 'Qualitative',
    'problemSubmissionTime': '2021-04-26T19:14:06.563900857Z',
    'problemSubmissionExtra': None
  }
]
```

28.4.2.0.5 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves a list of all assignment extensions granted to the student with id `:studentId`

RESULT

```
[
  {
    "onAssignment": 1238,
    "until": "2022-06-02T04:59:59Z",
    "forUser": 12313,
    "id": 499
  }
]
```

28.4.2.0.6 POST `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/st`

Grants an extension to the student with id `:studentId`

```
rq('POST', '/instructors/yourname@gmail.com/courses/test course/students/12313/extensions'
    json={
        "onAssignment": 2756,
        "until": "2022-06-02",
        "forUser": 12313,
    }
).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" \
     -H "Content-Type: application/json" \
     -d '{ "onAssignment": 2756, "until": "2022-06-02", "forUser": 12313 }' \
     localhost:3000/api/v1//instructors/yourname@gmail.com/courses/test%20course/students/
```

Dates can be given in the format "2021-06-30T21:40:00Z", for UTC time, or in the format "2022-01-01 10:10" or simply "2022-01-01", in which case the date will assume the time zone associated with the course.

The template for the POST JSON is:

```
{
  "onAsssignment": ...,
  "until": ...,
  "forUser": ...,
}
```

where the user field must the *userId* of the student, the contents of the `userId` field of the data returned by GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu` *not* the contents of the `id` field returned by that GET request.

Result

The ID of the new document:

```
3
```

or an error is returned as an encoded JSON string.

28.4.2.0.7 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves a list of accommodations active for the student with id `:studentId`

So if the student's time allowed on exams is `(3 × STANDARD TIME) + 0 minutes`, and they have their due-dates extended by one hour, then the result will be:

```
{
  "timeFactor": 3,
  "timeExtraMinutes": 0,
  "forUser": 1314,
  "id": 62,
  "dateExtraHours": 1,
  "forCourse": 121
}
```

28.4.2.0.8 PATCH `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/s`

Modifies the accommodations active for the student with id `:studentId`

For example:

```
rq('PATCH', '/instructors/yourname@gmail.com/courses/test course/students/11231/accomodati
    json={"timeFactor": "2"}).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" -X "PATCH" \
     -H "Content-Type: application/json" \
     -d '{"timeFactor":"2"}' \
     /instructors/yourname@gmail.com/courses/test%20course/students/11231/accomodations
```

The template for the PATCH JSON (with all fields optional) is:

```
{
  "timeFactor": ...,
  "timeExtraMinutes": ...,
  "dateExtraHours": ...,
}
```

28.4.2.0.9 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/stu`

Retrieves a list of all access-restricted assignments that have been accessed by the student with id `:studentId`, along with access time for each.

RESULT


```
[
  {
    "createdAt": "2021-06-10T19:10:42.757612899Z",
    "user": 1,
    "id": 17927,
    "assignment": 3188
  }
]
```


28.4.2.0.10 DELETE `https://carnap.io/api/v1/instructor/:instructorIdent/courses/:courseTitle/`

Deletes the record of a student having accessed an access-restricted assignment, allowing them to e.g. retake an exam for which a certain amount of time was allotted.

RESULT


```
"deleted token"
```


28.4.2.0.11 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/as`

Retrieves a list of assignments for a given course.

RESULTS


```
[
  {
    "gradeRelease": "2021-05-20T19:30:00Z",
    "totalProblems": null,
    "visibleFrom": "2021-05-19T17:58:00Z",
    "availability": null,
    "date": "2021-05-18T21:41:35.12705102Z",
    "document": 4293,
    "pointValue": null,
    "course": 258,
    "id": 3276,
    "duedate": "2021-05-19T19:30:00Z",
    "title": "The Hardest Logic Puzzle Ever",
    "visibleTill": "2021-06-30T21:40:00Z",
```

```
    "assigner": null,
    "description": null
  }
]
```

Times are reported in UTC

28.4.2.0.12 POST `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/a`

Creates a new assignment.

```
rq('POST', '/instructors/yourname@gmail.com/courses/test course/assignments',
    json={
        "document": 2756,
        "title": "A new assignment",
        "due-date": "2022-01-01 10:10",
    }
).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" \
     -H "Content-Type: application/json" \
     -d '{"document": 2756, "title": "A new assignment", "due-date": "2022-01-01 10:10"}'
     localhost:3000/api/v1/instructors/yourname@gmail.com/courses/test%20course/assignment
```

The full template for the POST json is

```
{
  "gradeRelease": ...,
  "totalProblems": ...,
  "visibleFrom": ...,
  "availability": ...,
  "document": ...,
  "pointValue": ...,
  "duedate": ...,
  "title": ...,
  "visibleTill": ...,
  "description": ...
}
```

The only mandatory fields are `document` and `title`. Dates can be given in the format "2021-06-30T21:40:00Z", for UTC time, or in the format "2022-01-01 10:10" or simply "2022-01-01", in which case the date will assume the time zone associated with the course.

RESULT

The ID of the new assignment:

```
3
```

or an error is returned as an encoded JSON string.

28.4.2.0.13 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/as`

Retrieves the details of the assignment with id `:assignmentId` .

RESULT

```
{
  "gradeRelease": "2021-05-20T19:30:00Z",
  "totalProblems": null,
  "visibleFrom": "2021-05-19T17:58:00Z",
  "availability": null,
  "date": "2021-05-18T21:41:35.12705102Z",
  "document": 4293,
  "pointValue": null,
  "course": 258,
  "id": 3276,
  "duedate": "2021-05-19T19:30:00Z",
  "title": "The Hardest Logic Puzzle Ever",
  "visibleTill": "2021-06-30T21:40:00Z",
  "assigner": null,
  "description": null
}
```

28.4.2.0.14 PATCH `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/`

Modifies the details of the assignment with id `:assignmentId` .

For example:

```
rq('PATCH', '/instructors/yourname@gmail.com/courses/test course/assignments/1313',
    json={"gradeRelease": "2022-02-02"}).json()
```

Bash

```
curl -H "X-API-KEY:YOURAPIKEY" -X "PATCH" \
     -H "Content-Type: application/json" \
     -d '{"gradeRelease": "2022-02-02"}' \
     /instructors/yourname@gmail.com/courses/test%20course/assignments/1313
```

The template for the PATCH JSON (with all fields optional) is:

```
{
  "gradeRelease": ...,
  "totalProblems": ...,
  "visibleFrom": ...,
  "availability": ...,
  "pointValue": ...,
  "duedate": ...,
  "title": ...,
  "visibleTill": ...,
  "description": ...,
}
```

To delete the contents of any field, except for `title`, you can send an explicit `null` value in the patch for that field.

28.4.2.0.15 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/as`

Retrieves a list of all submitted problems for the assignment with id `:assignmentId` .

RESULT

```
[
  {
    'problemSubmissionAssignmentId': 3001,
    'problemSubmissionCorrect': False,
    'problemSubmissionData': {..depends on problem type..},
    'problemSubmissionSource': {'tag': 'Assignment', 'contents': 'AssignmentMetadataKey {unA
    'problemSubmissionIdent': 'Exercise-24',
    'problemSubmissionUserId': 10385,
```

```
    'problemSubmissionLateCredit': None,
    'problemSubmissionType': 'Qualitative',
    'problemSubmissionTime': '2021-04-26T19:14:06.563900857Z',
    'problemSubmissionExtra': None
  }
]
```

28.4.2.0.16 GET `https://carnap.io/api/v1/instructors/:instructorIdent/courses/:courseTitle/ass`

Retrieves a list of all submitted problems from the student with id `:studentId`, for the assignment with id `:assignmentId` .

RESULT

```
[
  {
    'problemSubmissionAssignmentId': 3001,
    'problemSubmissionCorrect': False,
    'problemSubmissionData': {..depends on problem type..},
    'problemSubmissionSource': {'tag': 'Assignment', 'contents': 'AssignmentMetadataKey {unAs
    'problemSubmissionIdent': 'Exercise-24',
    'problemSubmissionUserId': 10385,
    'problemSubmissionLateCredit': None,
    'problemSubmissionType': 'Qualitative',
    'problemSubmissionTime': '2021-04-26T19:14:06.563900857Z',
    'problemSubmissionExtra': None
  }
]
```

29 Configuring Carnap

Carnap uses a settings file, `settings.yml`, to store its configuration. You can find the example version at `Carnap-Server/config/settings-example.yml`. On the initial build, to utilize binary caching, our Makefile tries to copy this example set of settings to `settting.yml`, so that your build will exactly match a standard build, hopefully enabling Nix to use previously cached information stored on cachix to speed things up for you.

You can edit `settings.yml`, and rebuild the server to change the default settings. You can also change the settings at runtime (that is, *after* building the server) by using environment variables or providing a different settings file as an argument to the Carnap executable. Runtime configuration is useful for Docker deployments, where you may wish to simply run the latest docker image, without building your own server (deployment with docker is out of the scope of this document). The settings file lets you set a number of variables, the most important of which are:

Name	Purpose
STATIC_DIR	The location of the `static` directory where Carnap looks for things like CSS style sheets
APPROOT	The address at which the Carnap server will be accessed.
DATAROOT	The location of Carnap's data directory.
BOOKROOT	The location of the directory holding the Carnap book.
GOOGLEKEY	The API key for Google authentication
GOOGLESECRET	The Google Auth secret string

29.0.1 Directories

The Carnap server needs access to three directories. These are:

- The data directory, which holds uploaded documents,the cookie encryption key, and the database containing user, course, and submission data (if Carnap uses `sqlite` to administer the database, and not a database server). It must be writable by the Carnap server process, and its location is given as the `DATAROOT` variable. This can be anywhere you like, but the build environments have it in `Carnap/dataroot`. The directory must exist when you run the server.

- The directory holding static files such as CSS style sheets and JavaScript files. Its location is given as the `STATICROOT` variable. In the source distribution, this directory is located in `Carnap/Carnap-Server/static`.(If you want Carnap to locate these files somewhere else you can instead set the `STATICROOT` variable to the full URL of the static directory.)
- The book directory holds the Carnap book. In the source distribution, this directory is located in `Carnap/Carnap-Book`.

Paths to these directories should be absolute, or relative to the directory from which you run the Carnap server. For instance, if you want to run Carnap from the source directory, your `mysettings.yml` file should contain the following lines:

```
static-dir: "_env:STATIC_DIR:Carnap-Server/static"
data-root: "_env:DATAROOT:dataroot"
book-root: "_env:BOOKROOT:Carnap-Book"
```

29.0.2 HTTP access

Carnap is obviously a web application, and the Carnap server needs to know where you (or your users) will access it. The two relevant variables are `APPROOT`, which is the URL of the Carnap server, and `PORT`, which is the TCP port where Carnap will be provided on the local machine.

For testing purposes, it is often enough to provide the Carnap app locally on your machine only, and not make it accessible from the network. In such a situation, your `mysettings.yml` file should contain the lines:

```
port: "_env:PORT:3000"
approot: "_env:APPROOT:http://localhost:3000"
```

This will make Carnap available on your machine only at the URL http://localhost:3000.

If you are making Carnap available over the network, you need to know the URL at which Carnap will be available, and set a reverse proxy on the server with that address to the host/port where Carnap is hosted. For instance, say you want Carnap to be accessible at `https://carnap.bigstateu.edu`, and Carnap runs on the server `webappserv.bigstateu.edu` on port 3000. Then your web administrator has to set a reverse proxy that translates HTTP requests for `https://carnap.bigstateu.edu` internally to `webappserv.bigstateu.edu:3000`. Your `mysettings.yml` file should then contain the lines:

```
port: "_env:PORT:3000"
approot: "_env:APPROOT:https://carnap.bigstateu.edu"
```

If your machine is accessible from the internet and you want to make your own Carnap server available, you could use a simple solution such as caddy.

29.0.3 Database

If the setting variable `SQLITE` in settings.yml is set to true (default), or if the environment variable SQLITE is set to true, Carnap will generate a lightweight database for storing its data, using (SQLite).

If SQLITE is set to false, Carnap will instead use a PostgreSQL database. Set the environment variable `SQLITE=false` and supply `PGUSER`, `PGPASS`, `PGHOST`, and if required, `PGPORT` and `PGDATABASE` for your postgresql database instance.

Which database should you use? Sqlite is simpler to manage, and should be adequate for most installations with fewer than 500 users. For larger installations, PostgreSQL may be appropriate.[1]

If you wish to use peer authentication via Unix socket on a locally hosted PostgreSQL database, set all of `PGUSER`, `PGPASS`, `PGHOST` and `PGPORT` to empty strings.

29.1 Authentication

Carnap servers currently require Google authentication to be configured for administration purposes, even if it is not intended to be used by students. (See the LTI 1.3 documentation for details on how to configure LTI authentication, after completing setup).

> Is there a way to run Carnap without Google Auth once it's built without the `dev` flag? If not, how do you build it without Google Auth if you really just want to play on your laptop?

Setting up Google authentication requires setting up a Google APIs project.

First, create a project in Google's developer console. Then, create an OAuth2 client ID under Google credentials.

It will show up like this:

Click "Create Credentials". On the client ID page, set an Authorized Redirect URI for `APPROOT/auth/page/google/callback`, where `APPROOT` should be replaced by the value of your `APPROOT` settings variable.

[1] Why? PostgreSQL is preferred in industry, and may have some security advantages depending on your server configuration. Sqlite also doesn't have the same level of support for concurrent writes to the database as PostgreSQL, but Carnap retries database transactions that fail because the database is busy, so this shouldn't generally cause any trouble in low-traffic scenarios.

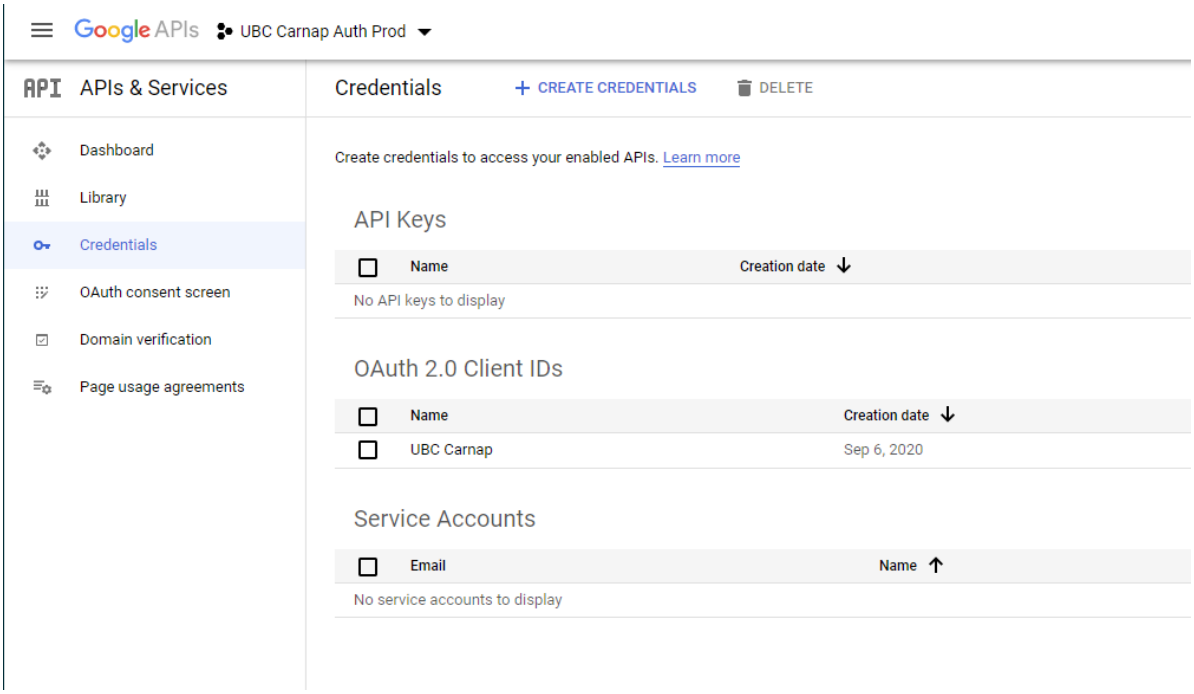


Figure 29.1: image of the client ID page

> The following didn't make sense to me and I don't recall seeing anything about scopes.

On your OAuth2 Consent Screen tab, no scopes need to be added as Carnap just needs emails to log in.

Once you've configured all the information on the Google side, fill it in the Carnap settings file:

```
google-api-key: "_env:GOOGLEKEY:<your Google client ID>"
google-secret: "_env:GOOGLESECRET:<your Google secret>"
```

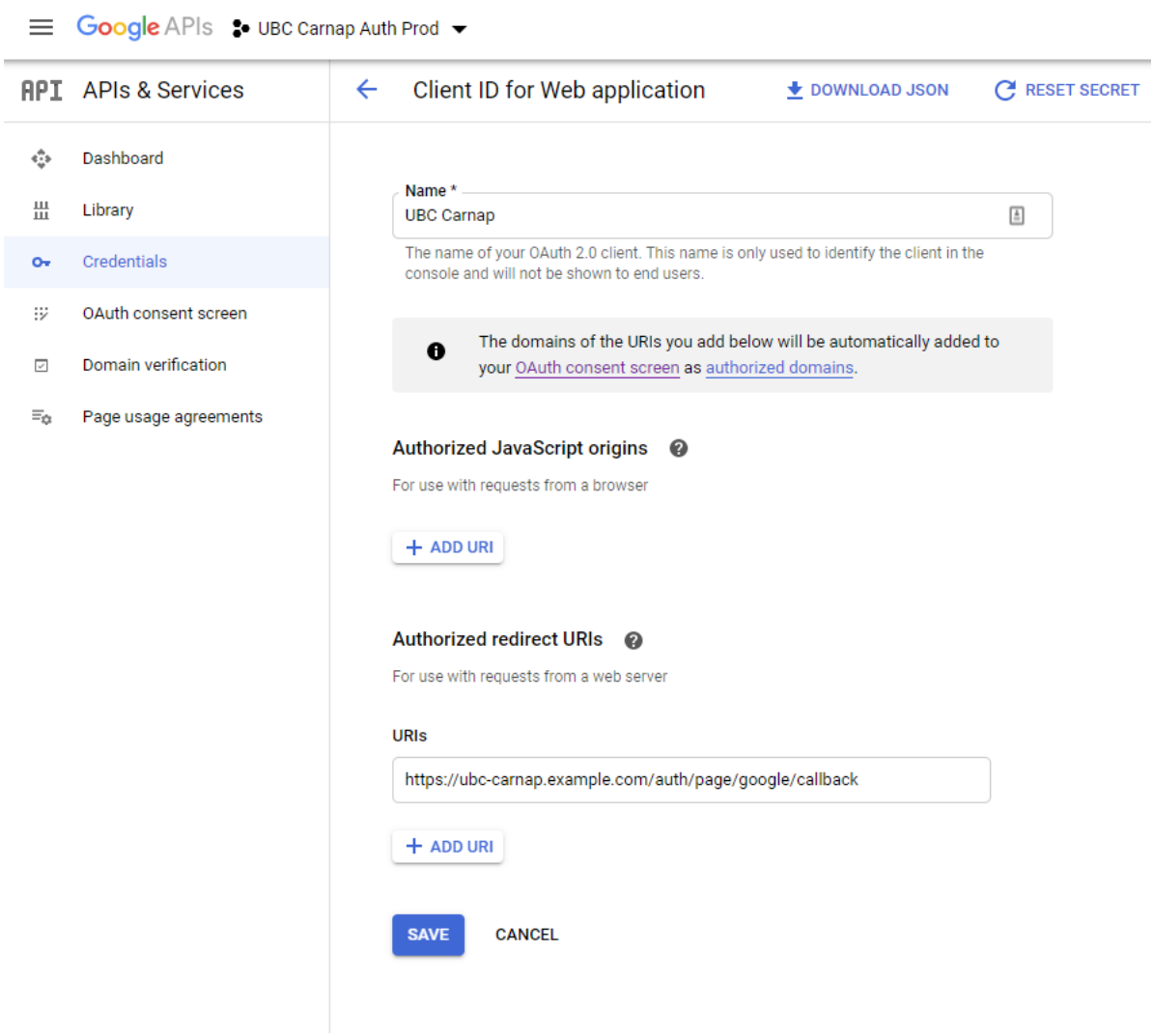


Figure 29.2: image of the authorized redirect URIs page

References