WebEx: Web Exercises for RISCAL[*]

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

October 25, 2018

Abstract

We report on a software framework "WebEx" for developing web-based student exercises whose correctness is checked with the help of the RISCAL (RISC Algorithm Language) software. This framework allows to generate from an appropriately annotated RISCAL specification file an HTML file that serves as the frontend to a remote execution service. Student input (RISCAL fragments) are transmitted to this execution service which generates from the annotated specification file and the input a plain RISCAL specification on which RISCAL is invoked (in a non-interactive mode); the success status of the execution and the produced output are reported back to the web interface. For each successful exercise the server produces a digitally signed certificate that is returned to the user who may submit this certificate as a proof of successful completion of the exercise (which may be subsequently automatically checked). Furthermore each annotated RISCAL specification may serve as a template that may be instantiated with other data to produce a set of exercise instances. The WebEx software is mostly independent of RISCAL; it may be also used to provide a web front end for other scientific software of a similar nature.

[*]Supported by the Austrian OEAD WTZ program and the Slovak SRDA agency under the contract SK 14/2018 "SemTech — Semantic Technologies for Computer Science Education" and by the Johannes Kepler University Linz, Linz Institute of Technology (LIT), Project LOGTECHEDU "Logic Technology for Computer Science Education".

Contents

1 Introduction

The RISC Algorithm Language (RISCAL) [1, 2] is is a specification language and associated software system for modeling mathematical algorithms, formally specifying their behavior based on mathematical theories, and validating the correctness of algorithms, specifications, and theories by checking that the executions of algorithms satisfy the annotations and by evaluating the validity of generated verification conditions.

RISCAL is used in educational scenarios [3] where students are expected to actively engage with the presented content by developing (respectively adapting/completing/refining) RISCAL specifications (mathematical theories, algorithms, specifications, and annotations). So far this has required the local execution of RISCAL (with its graphical user interface) on the student's computer (based on a predefined virtual machine that has RISCAL installed) or on a remote server (based on the remote desktop solution X2Go). Students develop specification files and submit these (together with exercise reports that show the produced checker outputs) as the results of the exercises.

However, especially in undergraduate education it would be advisable to provide access to RISCAL also via the web (such that no local software installation is required). Furthermore, the elaboration of an exercise should not necessarily expose a complete RISCAL specification but also those parts that are relevant for the student (hiding in particular those parts that describe how the correctness of the exercise is checked). The "WebEx" software described in this document is intended to satisfy this demand by allowing to turn with moderate effort annotated RISCAL specifications to web-based exercises.

The remainder of this document is organized as follows. In Section 2 we give an overview on the architecture of the software and its use. Section 3 documents the syntax of the annotation language from which the web exercises are generated. Section 4 explains in detail an example of a web exercise produced with the framework. Section 5 describes the installation and configuration of the software. Section 6 concludes and outlines further work. Appendix A includes the screenshots of various web exercises and the annotated RISCAL sources from which they have been produced.

The WebEx software is mainly implemented in Python 3 with a JavaScript library and a CSS file for the web frontend. It is freely available under the GNU 3 license from the following URL:

https://www.risc.jku.at/research/formal/software/RISCAL/WebEx

2 The Software

Figure 1 displays the architecture of the WebEx software and the workflow of its use:

- The command `webex` generates from an annotated RISCAL text file *exercise.txt* (that is deployed on the execution server) an HTML file *exercise.html* that represents the exercise form; this file is deployed on the web server.

- The client downloads *exercise.html* from the web server to her web browser. She enters her name into the exercise form and performs some task described in the form by providing some input and then triggering some RISCAL action to check that input.

- To perform the action, the web browser generates a request to the service `webex.wsgi` running on the execution server; this request contains the user input (including the user name and the action triggered) and a unique identifier *uid* (a large random integer) that can be subsequently used to identify the request.

- The execution server produces from *exercise.txt* and the user input a plain RISCAL file and starts a RISCAL process to execute the action denoted by the user on that file. The output generated by RISCAL is permanently logged in an output file *uid.log*. The service waits for the termination of the process (a timeout value is set which ensures that this happens within a bounded amount of time) but also starts a monitor thread to wait for an abortion request (see below).

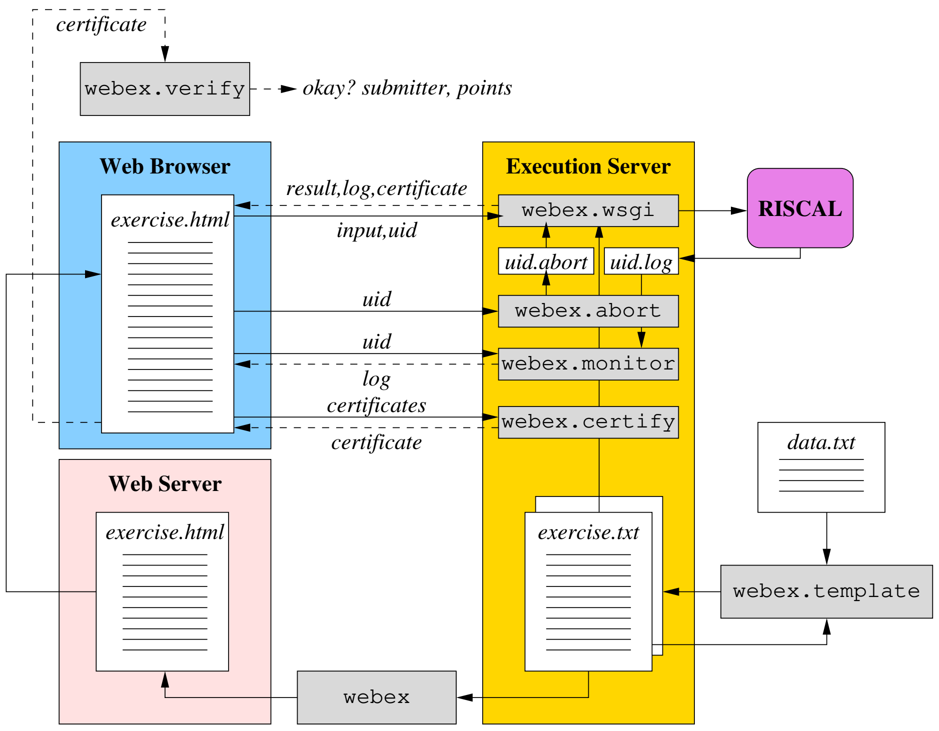


Figure 1: The Software Architecture

- The user may in the web interface abort the execution of the request which causes the browser to send to the service `webex.abort` the request uid; this service then generates a trigger file *uid.abort*. When the monitor thread detects this file, it aborts the RISCAL process.
- The web browser periodically sends a request to the service `webex.monitor` with the request uid; this service then returns the content of the output file *uid.log* to the web browser which thus continuously displays the progress of the execution.
- When the RISCAL process terminates (normally, by user abortion, or by timeout), `webex.wsgi` returns the result status of the RISCAL process (success or failure) together with the content of *uid.log* and a digital certificate of the execution (signed with a private key of the execution server) to the web browser; the certificate includes the user name, the action performed, and the grade points earned.
- The web browser keeps track of the certificates that the user has gathered by performing the tasks described in the exercise. Upon user request, this collection of certificates is transferred to the service `webex.certify` which checks the authenticity of these certificates and generates a summary certificate which indicates the total number of grade points that the user has earned by performing the tasks; this digitally signed summary certificate (which includes also the collection of the individual certificates) is returned to the web browser[1].
- The user stores the summary certificate in a file and submits it as the result of the exercise. The lecturer may use the script `webex.verify` (which needs access to the private server key) to check the authenticity of the certificate and to print the name of the user that has performed the exercise and the number of grade points that she has earned[2].

[1]The summary certificate is strictly speaking not necessary since all the information is contained in the collection of individual certificates; the summary, however, simplifies further processing.

[2]The summary certificate guarantees that the number of grade reported indeed corresponds to the performed tasks. It does, however,

- The script `webex.template` may be invoked to generate from an already existing exercise file *exercise.txt* a new instance by letting the data provided in a file `data.txt` replace the corresponding data in the original file. Thus it becomes easy to duplicate exercises by only providing the data that should change.

Above workflow involves the following commands to be executed by the creator of the exercise:

```
webex exercise.txt > exercise.html
```

Thus the script `webex` generates from the annotated file *exercise.txt* an HTML file and prints it to the standard output (which is redirected to file *exercise.html*).

```
webex.verify exercise.wxc
```

Thus the script `webex.verify` checks the authenticity of the certificate stored in file *exercise.wxc* (the file is in plain HTML format; the extension *wxc* just discourages the user from opening the file in an editor). If the file is not authentic (the digital signature does not match the content), an error message is printed. If the file is authentic, the submitter of the exercise and the number of grade points are printed.

```
webex.template exercise.txt data.txt >exercise-data.txt
```

Thus the script `webex.template` generates from the file *exercise.txt* a new instance that is identical to the original file except that it overwrites all data contained in *data.txt*. The instance is printed to the standard output (which is redirected to file *exercise-data.txt*).

3 The Annotation Language

An exercise file is a text file whose content is written in the language of the software system in which the exercise is to be executed. We assume that this language allows to write comments and define a syntactic subset of such comments that embed annotations which WebEx uses for the generation of web exercises.

For instance, RISCAL supports as comments single line comments preceded by the marker `//` and multi-line comments embedded between the markers `/*` and `*/`. WebEx assumes that any single-line comment preceded by `//@` and every multi-line comment between `/*@` and `@*/` embeds WebEx annotations (except for white space, no other text is allowed in these comments). Each such comment may contain one of the following kinds of annotations:

`<webex id="..." type="..." header="..." href="...">`

This annotation must appear as the first one (and only as the first one) in the file. Furthermore, the comment embedding this annotation must appear at the very beginning of the file (no text is allowed before the comment, also not whitespace).

The mandatory attribute `id` specifies the name of the exercise that the execution server uses to look up the exercise file; typically this name is the base name of the file (the file name without extension).

The mandatory attribute `type` specifies the type of the software system used for running the exercise; currently only the attribute value `"riscal"` (for RISCAL) is supported.

The optional attribute `header` gives a header line to be displayed in the generated exercise. If also the optional attribute `href` is given, the header line becomes a link to the URL specified by `href`.

not guarantee that these tasks have been indeed been performed by the named user; the authentication of users (and the prevention of cheating by collaboration) is not addressed by the framework. The inclusion of a user name in the certificate, however, prevents the trivial duplication of certificates; for each named user, the tasks have to be performed separately.

`<display id="...">...</display>`

The text "..." between the tags `<display>` and `</display>` is displayed in the web page. This text is *not* formal exercise content but arbitrary text that is just used for display (empty or whitespace text may be used to visually separate formal content before and after this tag).

`<public>`

`<private>`

By default, the content of the exercise file is *not* displayed in the generated web page, i.e., the exercise file is processed in a "private" mode. However, if the tag `<public>` is given, the mode switches to "public", i.e., all subsequent content becomes part of the web page (it is displayed in a grey box which indicates the formal content of the exercise; the formatting of the content is preserved). A subsequent tag `<private>` switches the mode back to "private" again.

`<data id="...">`

`</data>`

The tag `<data>` marks the begin of a portion of the exercise that is given the name specified by the value of the mandatory `id` attribute (which must be an alphanumeric identifier); the end of this portion is marked by the subsequent `</data>` tag (no other annotations may appear between these tags). This is useful (only) if the exercise file shall serve as a template where this portion of the exercise can be replaced by instantiation.

`<input id="..." label="..." rows="..." cols="...>`

`</input>`

The tag `<input>` marks the begin of a portion of the exercise that represents user input; this portion is terminated by the tag `</input>`. In the web page a corresponding input box is displayed that is prefilled with the content between `<input>` and `</input>` but is expected to be overwritten by user input.

The optional attribute `id` (whose value must be an alphanumeric identifier) is (only) needed if the exercise shall serve as a template where this portion of the exercise can be replaced by instantiation. If the optional attribute `label` is given, the box is prefixed with the content of the label. If the optional attribute `rows` is given, its content must denote a positive integer which is interpreted as the number of rows of the input box (if the attribute is not given, this number is one). If the optional attribute `cols` is given, its content must denote a positive integer which is interpreted as the number of columns of the input box (if the attribute is not given, a moderate value is chosen). In any case, in the web page the input box can be resized by the user.

`<button id="..." label="..." action="..." points="...">...</button>`

This element causes the generation of a button in the web page that can be pressed by the user to trigger a particular execution of the software system on the exercise file with the given user input.

The value of the mandatory attribute `id` must be an alphanumerical identifier that uniquely identifies the button. The value of the mandatory attribute `label` is used to label the button in the web page. The value of the mandatory attribute `action` denotes the action to be performed in the execution of the software system (in RISCAL, this value must be the name of an operation (function/predicate/theorem/procedure) defined in the exercise).

The value of the optional attribute `points` must be an integer number that denotes how many grade points the user earns, if the execution of the action indicates the successful completion of the task (if the attribute is omitted, one grade point is assumed).

The content "..." between <button> and </button> is a (possibly empty) sequence of elements of one of the following forms:

```
<argument value="...">
<argument option="...">
```

In case of an <argument> with attribute `value`, the value of this attribute must be a string that denotes a command line argument to be passed to an invocation to the software system for running the exercise. If instead the attribute `option` is used, its value must be the identifier of one of the <option> elements in the exercise file (see below). In that case, the user-selected value of this option is passed as a command line argument to the software system.

`<option id="..." label="..." type="..." min="..." max="...">`

An <option> element is translated into a web page element that allows to user to set some execution option for the software system running the exercise.

The value of the mandatory attribute `id` must be an alphanumeric identifier that uniquely identifies the option. The value of the mandatory attribute `label` is a string that is used to label the web page element. The value of the mandatory attribute `type` may be one of the following:

`bool` In this case, the web page element is a selection box; if this box is ticked, the corresponding command line argument is `1`, otherwise it is `0`.

`int` or **`long`** In this case, the web page element is a text box that allows to enter a non-negative 32 bit respectively 64 bit integer value. If the attribute `min` is given, its value must be a non-negative integer that specifies the smallest value that may be chosen; if the attribute `max` is given, its value must be a non-negative integer that specifies the largest value that may be chosen. The corresponding command line argument is the decimal representation of the chosen value.

`string` In this case, the web page element is a text box that allows the user to enter an arbitrary string whose value represents a command line argument to the software system.

If we use the command

```
webex.template exercise.txt data.txt >exercise-data.txt
```

to instantiate a template file *exercise.txt* with the data from the file *data.txt*, the content of *data.txt* is essentially an annotation file (with annotations embedded in comments) with the following considerations respectively restrictions:

- In the <webex> element the value of the attribute `id` must identify the instantiated file, not the template file (e.g., value `"exercise-data"` rather than `"exercise"` in the example above).
- Every other element in the data file must have an attribute `id` whose value is identical to the corresponding attribute value of an element *with the same tag* in the template file. The identified element in the template file will be replaced by the element in the data file to yield the instantiated file.

The following section demonstrates the annotation language and the use of the various commands by an example.

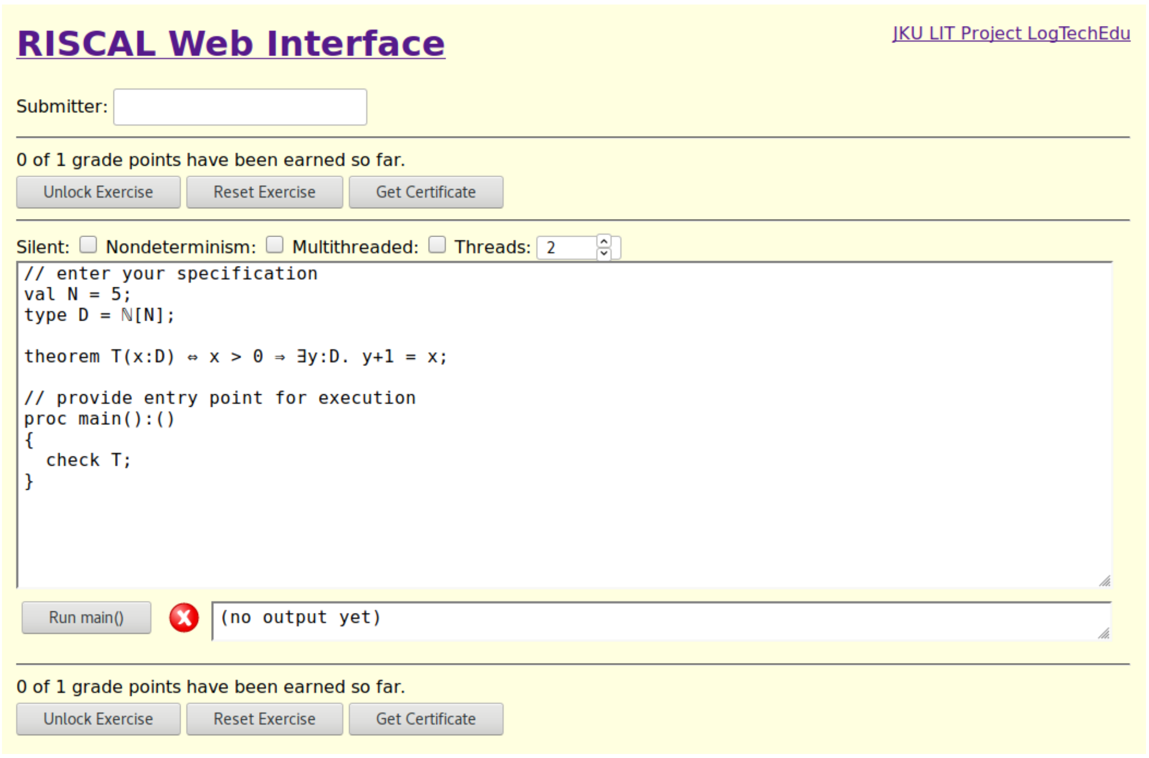
4 An Example

Figure 2 displays the content of the exercise file `RISCAL.txt`; this exercise actually represents a generic user interface for RISCAL in which the user can enter arbitrary RISCAL specifications and check them.

The described user interface consists of the following parts:

```
/*@ <webex id="RISCAL" type="riscal"
       header="RISCAL Web Interface"
       href="https://www.risc.jku.at/research/formal/software/RISCAL">
@*/

//@ <option id="silent" label="Silent:" type="bool">
//@ <option id="nondet" label="Nondeterminism:" type="bool">
//@ <option id="multi" label="Multithreaded:" type="bool">
//@ <option id="threads" label="Threads:" type="int" min="2" max="4">

//@ <input id="in" cols="98" rows="15">
// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}
//@ </input>

/*@ <button id="run" label="Run main()" action="main">
    <argument value="-opt-si"> <argument option="silent">
    <argument value="-opt-nd"> <argument option="nondet">
    <argument value="-opt-mt"> <argument option="multi">
    <argument value="-opt-tr"> <argument option="threads">
    </button>
@*/
```

Figure 2: RISCAL Web Interface



0 of 1 grade points have been earned so far.

Silent: ☐ Nondeterminism: ☐ Multithreaded: ☐ Threads: 2

```
// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}
```

```
(no output yet)
```

0 of 1 grade points have been earned so far.

Figure 3: RISCAL Web Interface (Before Execution)

1. A header with a link to the RISCAL web page.
2. A number of options that the user may select for invoking RISCAL.
3. An input field with some predefined content that the user may override.
4. A button to invoke the operation `main()` specified in the input field.

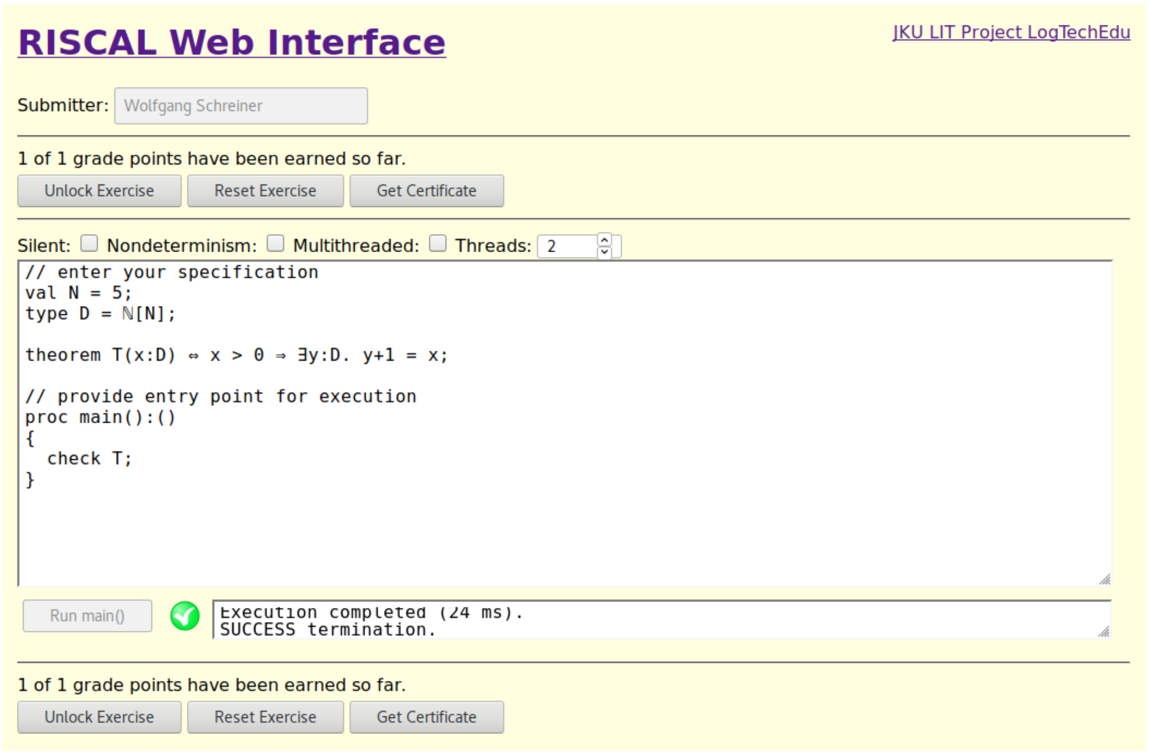
The invocation

```
webex RISCAL.txt > RISCAL.html
```

generates the web page illustrated in Figure 3.

This interface can be used as follows:

- The user has to enter her name into the “Submitter” field.
- The user may set the RISCAL execution options “Silent” (output is minimized), “Nondeterminism” (all execution/evaluation branches are chosen), “Multithreaded” (execution proceeds with at least two concurrent threads), “Threads” (the number of threads may chosen in the range from 2 to 4).
- The user may enter in the text field arbitrary RISCAL definitions; one of these definitions should be an operation `main()`.
- The user may press the “Run main()” button to execute the operation with the selected options; the output of the execution is shown in the attached output field. The success status is displayed by the icon next to the button (green for success).


RISCAL Web Interface

Submitter: Wolfgang Schreiner

1 of 1 grade points have been earned so far.

Silent: ☐ Nondeterminism: ☐ Multithreaded: ☐ Threads: 2

```
// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}
```

```
Execution completed (24 ms).
SUCCESS termination.
```

1 of 1 grade points have been earned so far.

Figure 4: RISCAL Web Interface (After Execution)

After pressing the button, the display changes as shown in Figure 4. The submitter field and the the pressed button are now locked which indicates that we have performed the task and are ready to submit a certificate (to unlock the form and perform further changes, one may press the button “Unlock Exercise”; the button “Reset Exercise” resets the exercise to its initial form).

The output field contains the output produced by the software; resizing this field shows the complete output:

```
RISC Algorithm Language 2.6.1 (October 18, 2018)
http://www.risc.jku.at/research/formal/software/RISCAL
(C) 2016-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCAL -h" to see the available command line options.
-----------------------------------------------------------------
Reading file /tmp/tmp_wfgpn1w
Computing the value of N...
Type checking and translation completed.
Executing main().
Run of deterministic function main():
Executing T(ℤ) with all 6 inputs.
Run 0 of deterministic function T(0):
Result (0 ms): true
Run 1 of deterministic function T(1):
Result (6 ms): true
Run 2 of deterministic function T(2):
Result (0 ms): true
Run 3 of deterministic function T(3):
Result (0 ms): true
Run 4 of deterministic function T(4):
Result (0 ms): true
Run 5 of deterministic function T(5):
Result (0 ms): true
Execution completed for ALL inputs (10 ms, 6 checked, 0 inadmissible).
Result (23 ms): ()
Execution completed (24 ms).
```

```
SUCCESS termination.
```

By pressing the "Get Certificate" button a dialog pops up that allows us to save the generated file *RISCAL.wxc* with the digitally signed certificate. Its content (which is of no concern for the user) is essentially as follows:

```
<html>
<head>
<meta charset="utf-8">
</meta></head>
<body>
<pre id="certificate"><<<EXERCISE CERTIFICATE>>>
Webex: RISCAL
Submitter: Wolfgang Schreiner
Time: Thu Oct 25 13:48:52 2018
Successes: 1
Points: 1
<<<END CERTIFICATE>>></pre>
<pre id="signature">d364...5ec5b6498359b59fa375312b22fe559e</pre>
<pre id="certificate0"><<<SUCCESS CERTIFICATE>>>
Webex: RISCAL
Submitter: Wolfgang Schreiner
Time: Thu Oct 25 13:41:24 2018
Action: run
Points: 1
Inputs:
in=// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}

threads=2
Log:
RISC Algorithm Language 2.6.1 (October 18, 2018)
http://www.risc.jku.at/research/formal/software/RISCAL
(C) 2016-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCAL -h" to see the available command line options.
-----------------------------------------------------------------
...
Execution completed (24 ms).
SUCCESS termination.
<<<END CERTIFICATE>>></pre>
<pre id="signature0">ac4bc8bcbb1f89...96e16b562100b2c59ada7</pre>
</body>
</html>
```

This content represents the digitally signed summary certificate indicating the submitter of the exercise and the number of grade points earned as well as the digitally signed certificate of the execution with all user input and the output produced by the software. The invocation

```
webex.verify ~/Downloads/RISCAL.wxc
Wolfgang Schreiner: 1
```

verifies the authenticity of the certificate and displays the submitter and the number of grade points earned.

To generate an instance of this exercise with other predefined input data we use the data file *RISCAL-data.txt* listed in Figure 5. By executing the command

```
webex.template RISCAL.txt RISCAL-data.txt > RISCAL-instance.txt
```

we generate the instance file *RISCAL-instance.txt* listed in Figure 6.

More (also more representative) examples of RISCAL-based web exercises are given in the appendix.

```
/*@ <webex id="RISCAL-instance" type="riscal"
       header="RISCAL Web Interface"
       href="https://www.risc.jku.at/research/formal/software/RISCAL">
@*/

//@ <input id="in" cols="98" rows="15">
proc main(): ()
{
  print 123;
}
//@ </input>
```

Figure 5: A Data File

```
/*@ <webex href="https://www.risc.jku.at/research/formal/software/RISCAL"
      id="RISCAL-instance" header="RISCAL Web Interface" type="riscal">
@*/

//@<option id="silent" type="bool" label="Silent:">
//@<option id="nondet" type="bool" label="Nondeterminism:">
//@<option id="multi" type="bool" label="Multithreaded:">
//@<option min="2" max="4" id="threads" type="int" label="Threads:">

//@<input cols="98" rows="15" id="in">
proc main(): ()
{
  print 123;
}
//@</input>

/*@
<button action="main" id="run" label="Run main()">
<argument value="-opt-si">
<argument option="silent">
<argument value="-opt-nd">
<argument option="nondet">
<argument value="-opt-mt">
<argument option="multi">
<argument value="-opt-tr">
<argument option="threads">
</button>
@*/
```

Figure 6: An Instance File

5 Installation and Configuration

The software distribution consists of the following files:

```
webex/
  *.py  ... the Python 3 files
  *.js  ... the client-side JavaScript library
  *.svg ... the client-side images
  *.css ... the client-side style sheets
bin/
  webex          ... script to generate the HTML exercise file
  webex.template ... script to produce exercise instances
  webex.verify   ... script to verify certificates
test/
  *.txt          ... examples of annotated RISCAL files
  *.html         ... the generated HTML files
doc/
  main.pdf       ... the manual
```

To install the software, some web-accessible directory *www* on the web server has to be chosen to host the generated HTML files; into this directory the files `*.js`, `*.svg`, and `*.css` have to be copied. The file `webex.js` has to be edited by providing URLS to the various services of the execution server:

```
webex_url = "http://localhost/webex";
abort_url = "http://localhost/webexabort";
certify_url = "http://localhost/webexcertify";
monitor_url = "http://localhost/webexmonitor";
```

Now some web-accessible directory *webex* on the execution server has to be chosen to host the Python 3 code; into this directory the files `*.py` have to be copied. The scripts in the `bin` directory have to be edited to point to the respective Python paths and they must be copied into a directory that is in the current `PATH`.

Among the Python 3 files, the file *config.py* has to be adapted to one's setup:

```
# either * to allow request from any origin or http://url to restrict origin
# https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin
access_control_allow_origin = "*"

# path to PEM file with private/public RSA keys
pem_path = "/var/www/webex.pem"

# maximum time (seconds) spent in service execution
timeout = 60

# path to specification file directory for every WebEx type
spec_dir = {
    'riscal': "/var/www/RISCAL"
}

# extension of specification files for every WebEx type
spec_ext = {
    'riscal': ".txt"
}
```

```
# command to execute for every WebEx type
spec_command = {
    'riscal': "/software/bin/RISCAL"
}
```

Now we have to configure the execution server (which may or may not be identical to the web server). Assuming that the execution server runs Apache 2, this server has to install Python 3 and the WSGI adaptor module `mod-wsgi` for Python 3. On a Debian 9 GNU/Linux distribution, this can be achieved by executing the following commands.

```
apt-get install python3
apt-get install libapache2-mod-wsgi-py3
```

Into the configuration file for this host (located typically in a file lik

```
/etc/apache2/sites-enabled/000-default.conf
```

a configuration of the following kind has to be entered:

```
<VirtualHost *:80>
  ...

  # replace by your name
  ServerName server.at

  # directory that contains the "webex" directory with the python files
  WSGIDaemonProcess server.at python-path=/var/www

  # replace by your name
  WSGIProcessGroup server.at

  # enter the paths of the python files
  WSGIScriptAlias /webex /var/www/webex/wsgi.py
  WSGIScriptAlias /webexcertify /var/www/webex/certify.py
  WSGIScriptAlias /webexmonitor /var/www/webex/monitor.py
  WSGIScriptAlias /webexabort /var/www/webex/abort.py

  # give access to the "webex" directory
  <Directory "/var/www/webex">
    Require all granted
  </Directory>

</VirtualHost>
```

This completes the setup.

6 Conclusions

The WebEx software toolset is an attempt to provide a low-threshold solution to the problem of running web-based exercises with scientific software that is typically used via an interactive user interface but also allows supports some form of scriptable access (via input/output files respectively standard input/output). An example of this software is the RISCAL software which we use in courses related to formal methods, logic, and formal modeling for which we have up to now required students to install the software locally or access it on a remote server via remote display software.

While the software is currently targeted to RISCAL, it should be rather easily adaptable to other pieces of software of a similar nature. We are currently still in an experimental phase; the WebEx software will certainly be further refined and revised. In the near future, however, we hope to profit from it in real-life educational scenarios.

References

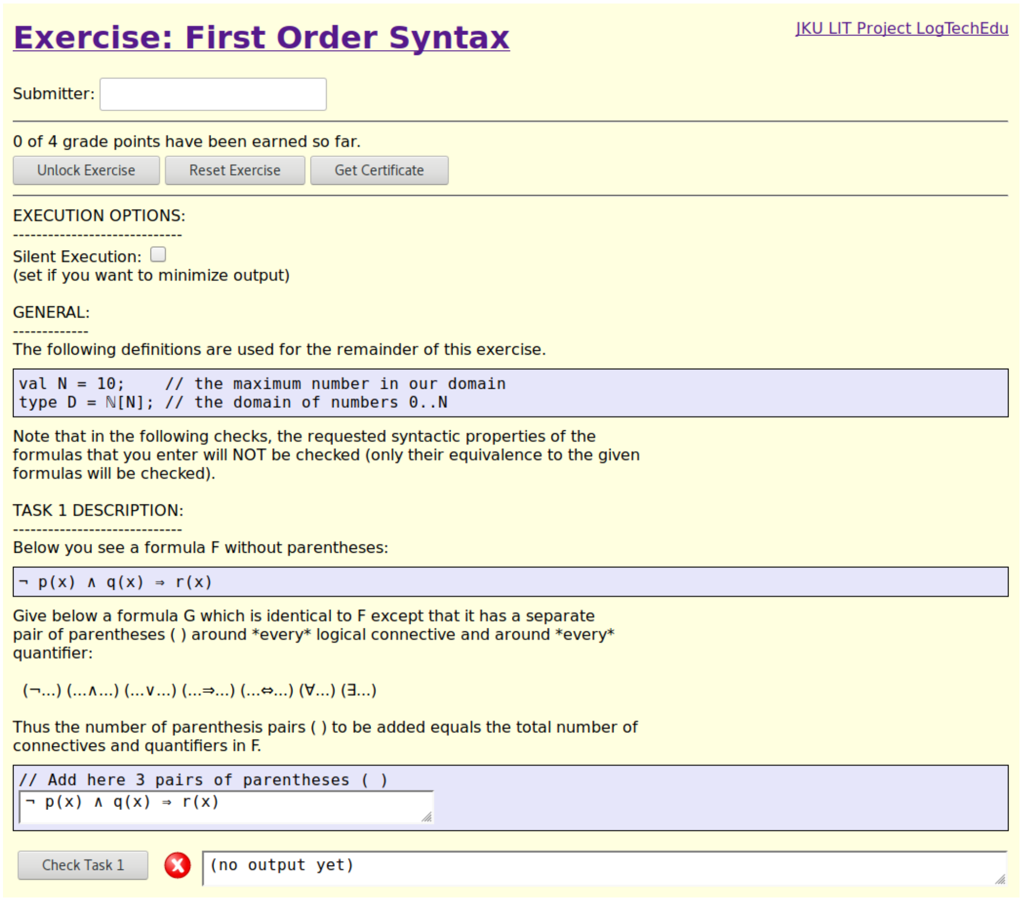
[1] RISCAL. *The RISC Algorithm Language (RISCAL)*. Mar. 2017. URL: `https://www.risc.jku.at/research/formal/software/RISCAL`.

[2] Wolfgang Schreiner. *The RISC Algorithm Language (RISCAL) — Tutorial and Reference Manual (Version 1.0)*. Technical Report. Download from [1]. Johannes Kepler University, Linz, Austria: RISC, Mar. 2017.

[3] Wolfgang Schreiner, Alexander Brunhuemer, and Christoph Fürst. "Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models". In: *Post-Proceedings ThEdu'17, 6th International Workshop on Theorem proving components for Educational software*. Ed. by Pedro Quaresma and Walther Neuper. Vol. 267. Electronic Proceedings in Theoretical Computer Science (EPTCS). Gothenburg, Sweden, August 6, 2017: Open Publishing Association, 2018, pp. 120–139. URL: `https://doi.org/10.4204/EPTCS.267.8`.

A RISCAL Web Exercises

A.1 First Order Syntax



Exercise: First Order Syntax

JKU LIT Project LogTechEdu

Submitter:

0 of 4 grade points have been earned so far.

EXECUTION OPTIONS:

Silent Execution: ☐
(set if you want to minimize output)

GENERAL:

The following definitions are used for the remainder of this exercise.

```
val N = 10;    // the maximum number in our domain
type D = ℕ[N]; // the domain of numbers 0..N
```

Note that in the following checks, the requested syntactic properties of the formulas that you enter will NOT be checked (only their equivalence to the given formulas will be checked).

TASK 1 DESCRIPTION:

Below you see a formula F without parentheses:

```
¬ p(x) ∧ q(x) ⇒ r(x)
```

Give below a formula G which is identical to F except that it has a separate pair of parentheses ( ) around *every* logical connective and around *every* quantifier:

(¬...) (...∧...) (...∨...) (...⇒...) (...⇔...) (∀...) (∃...)

Thus the number of parenthesis pairs ( ) to be added equals the total number of connectives and quantifiers in F.

```
// Add here 3 pairs of parentheses ( )
¬ p(x) ∧ q(x) ⇒ r(x)
```

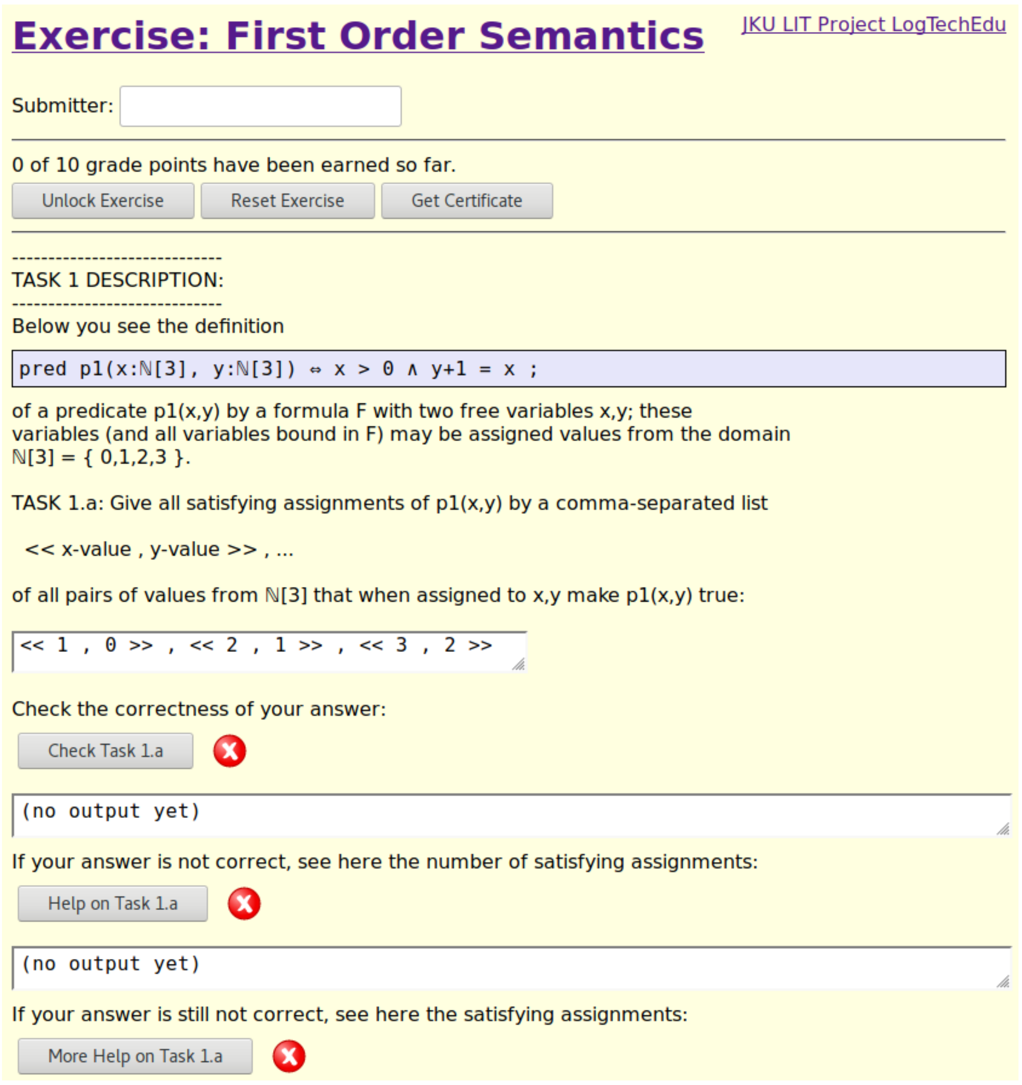
```
(no output yet)
```

```
/*@ <webex id="firstOrderSyntax" type="riscal"
       header="Exercise: First Order Syntax"
       href="http://fmv.jku.at/logik/index.html">
@*/
// ================================================================
//
// firstOrderSyntax.txt
// An demonstration exercise in first order logic (syntax).
// (c) 2018, Project LogTechEdu, http://fmv.jku.at/logtechedu/
//
// This exercise consists of 4 tasks TASK 1-4 where each task
// consists of one ore more questions a,b,...
//
// This file is for demonstration; it provides both the questions
// and corect answers to these questions. Study the questions, its
// answers, and run the checks of the correctness of the answers.
//
// ================================================================

/*@
<display>
EXECUTION OPTIONS:
-----------------------------
</display>
@*/

//@ <option id="silent" label="Silent Execution:" type="bool">
/*@<display>
(set if you want to minimize output)
```

```
GENERAL:
-------------
The following definitions are used for the remainder of this exercise.
</display>@*/
//@ <public>
val N = 10;    // the maximum number in our domain
type D = ℕ[N]; // the domain of numbers 0..N
/*@ <display id="task1display">
Note that in the following checks, the requested syntactic properties of the
formulas that you enter will NOT be checked (only their equivalence to the given
formulas will be checked).

TASK 1 DESCRIPTION:
----------------------------
Below you see a formula F without parentheses:
</display>@*/
//@ <private>
pred p(x:D) ⇔ x > 0;
pred q(x:D) ⇔ x < N;
pred r(x:D) ⇔ x > 0 ∧ x < N;

pred withoutPar1(x:D) ⇔
//@ <public>
//@ <data id="task1data">
¬ p(x) ∧ q(x) ⇒ r(x)
//@ </data>
//@ <private>
;
/*@ <display>
Give below a formula G which is identical to F except that it has a separate
pair of parentheses ( ) around *every* logical connective and around *every*
quantifier:

  (¬...) (...∧...) (...∨...) (...⇒...) (...⇔...) (∀...) (∃...)

Thus the number of parenthesis pairs ( ) to be added equals the total number of
connectives and quantifiers in F.
</display> @*/

//@ <private>
pred withPar1(x:D) ⇔
//@ <public>
//@ <data id="task1help">
// Add here 3 pairs of parentheses ( )
//@ </data>
//@ <input id="task1input">
¬ p(x) ∧ q(x) ⇒ r(x)
//@ </input>
//@ <private>
;

theorem correctPar1(x:D) ⇔ withoutPar1(x) ⇔ withPar1(x);

/*@ <display>
</display>@*/
/*@
<button id="task1" label="Check Task 1" action="correctPar1">
<argument value="-opt-si"> <argument option="silent">
</button>
@*/

//@ <private>
/*@ <display>

TASK 2 DESCRIPTION:
----------------------------
Below you see a formula F without parentheses:
</display> @*/

pred withoutPar2(x:D) ⇔
//@ <public>
//@ <data id="task2data">
  ∃y:D. x + 1 = y
//@ </data>
//@ <private>
;

/*@ <display>
Give a definition of a formula G as described in the TASK 1 DESCRIPTION. However
*also* add in formula G a pair of parentheses ( ) around every application of an
infix function/predicate symbol:

   (...+...) (...=...) ...
</display>@*/

pred withPar2(x:D) ⇔
//@ <public>
//@ <data id="task2help">
// add here 3 pairs of parentheses ( )
//@ </data>
//@ <input id="task2input">
∃y:D. x + 1 = y
//@ </input>
//@ <private>
;

theorem correctPar2(x:D) ⇔ withoutPar2(x) ⇔ withPar2(x);

/*@
<button id="task2" label="Check Task 2" action="correctPar2">
<argument value="-opt-si"> <argument option="silent">
</button>
@*/

/*@ <display>

TASK 3 DESCRIPTION:
-----------------------------
Below you see the definition of a predicate by a formula F. Remove from the
parameter list all those variables x:D,... that are *not* free in F.
</display> @*/

//@ <public>
// remove here as many variables as possible
pred varFormula3(/*@<input id="task3input" cols="20">@*/x:D, y:D, z:D/*@</input>@*/) ⇔
//@ <data id="task3data">
  x < z ⇒ ∃y:D. y > 0 ∧ x+y = z
//@ </data>
;
//@ <private>
/*@<display>
Your answer is correct, if removing any more variables (after pressing the
"Check" button) would give an error message.
</display>@*/

/*@
<button id="task3" label="Check Task 3" action="varFormula3">
<argument value="-opt-si"> <argument option="silent">
</button>
@*/

/*@<display>

TASK 4 DESCRIPTION:
-----------------------------
Below you see the description of an informal statement.
</display>@*/
/*@<display id="task4data">

    x is greater than 0 and there is no such number
    (i.e., no number greater than 0) that is smaller than x
    (this means that x is the smallest number greater than 0.)
```

```
</display>@*/
/*@<display>
Give a definition of a predicate

   pred formula4(x:D,y:D,z:D) ⇔ F ;

with a formula F that expresses this statement in the *most literal* way (i.e.,
it should be a straight-forward translation from natural language to logic
operators).

Furthermore remove from "formula4(x:D,y:D,z:D)" all those variables that do
*not* occur freely in F.

Your answer is *very likely* correct if your definition (after pressing the
"Check" button does not give an error message and not the message "theorem is
not true".
</display>@*/
//@ <public>
// give here your formula F.
pred formula4(/*@<input id="task4input1" cols="20">@*/x:D, y:D, z:D/*@</input>@*/) ⇔ // remove variables
/*@<input id="task4input2" cols="40" rows="3">@*/
x > 0 ∧ ~∃y:D. y > 0 ∧ y < x
/*@</input>@*/  // replace "⊤"
;
//@ <private>
theorem correctFormula4(x:D,y:D, z:D) ⇔ // remove variables
//@ <public>


// correspondingly remove variables from application of formula here
/*@<input id="task4input3">@*/formula4(x,y,z)/*@</input>@*/
//@ <private>
  ⇔
  x = 1
;
/*@ <button id="task4" label="Check Task 4" action="correctFormula4">
    <argument value="-opt-si"> <argument option="silent">
    </button>
  @*/


// ===============================================================
// END OF FILE
// ===============================================================
```

A.2 First Order Semantics



Exercise: First Order Semantics

JKU LIT Project LogTechEdu

Submitter:

0 of 10 grade points have been earned so far.

TASK 1 DESCRIPTION:

Below you see the definition

```
pred p1(x:ℕ[3], y:ℕ[3]) ⇔ x > 0 ∧ y+1 = x ;
```

of a predicate p1(x,y) by a formula F with two free variables x,y; these variables (and all variables bound in F) may be assigned values from the domain ℕ[3] = { 0,1,2,3 }.

TASK 1.a: Give all satisfying assignments of p1(x,y) by a comma-separated list

<< x-value , y-value >> , ...

of all pairs of values from ℕ[3] that when assigned to x,y make p1(x,y) true:

```
<< 1 , 0 >> , << 2 , 1 >> , << 3 , 2 >>
```

Check the correctness of your answer:

```
(no output yet)
```

If your answer is not correct, see here the number of satisfying assignments:

```
(no output yet)
```

If your answer is still not correct, see here the satisfying assignments:

```
/*@ <webex id="firstOrderSemantics" type="riscal"
       header="Exercise: First Order Semantics"
       href="http://fmv.jku.at/logik/index.html">
@*/

// ================================================================
//
// firstOrderSemantics.txt
// An demonstration exercise in first order logic (semantics).
// (c) 2018, Project LogTechEdu, http://fmv.jku.at/logtechedu/
//
// This exercise consists of 5 tasks TASK 1-5 where each task
// consists of one ore more questions a,b,...
//
// This file is for demonstration; it provides both the questions
// and corect answers to these questions. Study the questions, its
// answers, and run the checks of the correctness of the answers.
//
// ================================================================

/*@ <display>
-----------------------------
TASK 1 DESCRIPTION:
-----------------------------
Below you see the definition
```

```
</display> @*/
//@ <public>
pred p1(x:ℕ[3], y:ℕ[3]) ⇔ x > 0 ∧ y+1 = x ;
//@ <private>
/*@ <display>
of a predicate p1(x,y) by a formula F with two free variables x,y; these
variables (and all variables bound in F) may be assigned values from the domain
ℕ[3] = { 0,1,2,3 }.

TASK 1.a: Give all satisfying assignments of p1(x,y) by a comma-separated list

  << x-value , y-value >> , ...

of all pairs of values from ℕ[3] that when assigned to x,y make p1(x,y) true:

</display> @*/
val a1 = {//@ <input id="task1in">
<< 1 , 0 >> , << 2 , 1 >> , << 3 , 2 >>
//@ </input>
} ;

theorem ass1() ⇔ a1 = { ⟨x,y⟩ | x:ℕ[3], y:ℕ[3] with p1(x,y) };
fun num1():ℕ[16] = |{ ⟨x,y⟩ | x:ℕ[3], y:ℕ[3] with p1(x,y) }|;
/*@ <display>

Check the correctness of your answer:
</display> @*/
/*@
<button id="task1" label="Check Task 1.a" action="ass1">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@ <display>
If your answer is not correct, see here the number of satisfying assignments:
</display> @*/
/*@
<button id="task1help1" label="Help on Task 1.a" action="num1" points="0">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@ <display>
If your answer is still not correct, see here the satisfying assignments:
</display> @*/
/*@
<button id="task1help2" label="More Help on Task 1.a" action="p1" points="0">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@<display>
TASK 1.b: The formula p1(x,y) is satisfiable. True? (enter "true" or "false")

</display> @*/
theorem sat1() ⇔ /*@<input id="task1satin" cols="4">@*/true/*@</input>@*/
⇔ ∃x:ℕ[3],y:ℕ[3]. p1(x,y);
/*@
<button id="task1sat" label="Check Task 1.b" action="sat1">
<argument value="-opt-si"> <argument value="1">
</button>
@*/
/*@<display>
TASK 1.c: The formula p1(x,y) is valid. True? (enter "true" or "false")

</display>@*/
//@ <private>
theorem valid1() ⇔
/*@<input id="task1validin" cols="4">@*/false/*@</input>@*/
⇔ ∀x:ℕ[3],y:ℕ[3]. p1(x,y);
/*@
<button id="task1valid" label="Check Task 1.c" action="valid1">
<argument value="-opt-si"> <argument value="1">
</button>
```

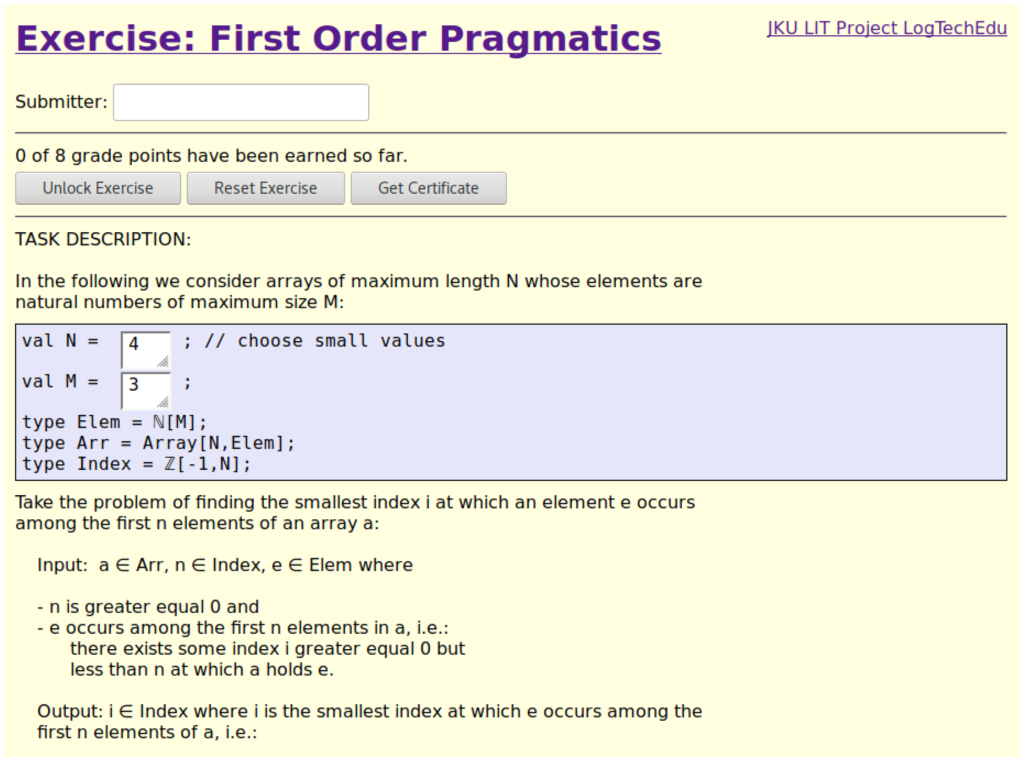
```
@*/

/*@ <display>
-----------------------------
TASK 2 DESCRIPTION:
-----------------------------
Below you see the definition
</display> @*/
//@ <public>
pred p2(x:ℕ[3]) ⇔ ∃y:ℕ[3]. y+1 = x ;
//@ <private>
/*@ <display>
of a predicate p2(x) by a formula F with one free variable x; this variable
(and all variables bound in F) may be assigned values from the domain
ℕ[3] = { 0,1,2,3 }.

TASK 2.a: Give all satisfying assignments of p2(x) by a comma-separated list

  x-value , ...

of all values from ℕ[3] that when assigned to x make p2(x) true.

</display> @*/
val a2 = {//@ <input id="task2in">
1, 2, 3
//@ </input>
} ;
//@ <private>
theorem ass2() ⇔ a2 = { x | x:ℕ[3] with p2(x) };
fun num2():ℕ[4] = |{ x | x:ℕ[3] with p2(x) }|;
/*@ <display>

Check the correctness of your answer:
</display> @*/
/*@
<button id="task2" label="Check Task 2.a" action="ass2">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@ <display>
If your answer is not correct, see here the number of satisfying assignments:
</display> @*/
/*@
<button id="task2help1" label="Help on Task 2.a" action="num2" points="0">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@ <display>
If your answer is still not correct, see here the satisfying assignments:
</display> @*/
/*@
<button id="task2help2" label="More Help on Task 2.a" action="p2" points="0">
<argument value="-opt-si"> <argument value="0">
</button>
@*/
/*@<display>
TASK 2.b: The formula p2(x) is satisfiable. True? (enter "true" or "false")

</display> @*/
theorem sat2() ⇔ /*@<input id="task2satin" cols="4">@*/true/*@</input>@*/
⇔ ∃x:ℕ[3],y:ℕ[3]. p1(x,y);
/*@
<button id="task2sat" label="Check Task 2.b" action="sat2">
<argument value="-opt-si"> <argument value="1">
</button>
@*/
/*@<display>
TASK 2.c: The formula p2(x) is valid. True? (enter "true" or "false")

</display>@*/
//@ <private>
```

```
theorem valid2() ⇔
/*@<input id="task2validin" cols="4">@*/false/*@</input>@*/
⇔ ∀x:ℕ[3],y:ℕ[3]. p1(x,y);
/*@
<button id="task2valid" label="Check Task 2.c" action="valid2">
<argument value="-opt-si"> <argument value="1">
</button>
@*/

/*@ <display>
-----------------------------
TASK 3 DESCRIPTION:
-----------------------------
Below you see the definition
</display> @*/
//@ <public>
pred p3() ⇔ ∃y:ℕ[3]. ∀x:ℕ[3]. x > 0 ⇒ ∃y:ℕ[3]. y+1 = x ;
//@ <private>
/*@ <display>
of a predicate p3() by a closed formula F.

Give the truth value of p3() (enter "true" or "false"):

</display> @*/
theorem valid3() ⇔ p3() ⇔ /*@<input id="task3in" cols="4">@*/true/*@</input>@*/;

/*@
<button id="task3valid" label="Check Task 3" action="valid3">
<argument value="-opt-si"> <argument value="1">
</button>
@*/

/*@ <display>
-----------------------------
TASK 4 DESCRIPTION:
-----------------------------
In the following we consider formulas interpreted over the domain ℕ[2] = { 0,1,2 }.

Below you see two definitions
</display> @*/
type Formula1 = Map[ℕ[2],Bool];
//@ <public>
pred a4(p:Formula1, q:Formula1) ⇔
  (∃x:ℕ[2]. p[x]) ∧ (∃x:ℕ[2]. q[x]) ;
pred b4(p:Formula1, q:Formula1) ⇔
  ∃x:ℕ[2]. p[x] ∧ q[x] ;
//@ <private>

fun implies4a(): Set[Tuple[Formula1,Formula1]] =
  { ⟨p,q⟩ | p:Formula1, q:Formula1 with ¬(a4(p,q) ⇒ b4(p,q)) };
/*@ <display>
with formulas in which predicates p[x] and q[x] occur; please note that
applications of these predicates have to be written with square brackets [ ].

Task 4.a: F2 a logical consequence of F1. True? (enter "true" or "false")
</display> @*/
theorem task4a() ⇔
/*@<input id="task4in" cols="4">@*/false/*@</input>@*/
⇔ (∀p:Formula1, q:Formula1. a4(p,q) ⇒ b4(p,q));

/*@
<button id="task4a" label="Check Task 4.a" action="task4a">
<argument value="-opt-si"> <argument value="1">
</button>
@*/

/*@<display>
TASK 4.b: [ Only if your answer to TASK 4.a is "false". ]

Give examples of two predicates p[x] and q[x] for which a4 is true and b4 is
false. These examples may be simply written in the form of a two sequences
```

```
  p: p[0], p[1], p[2]
  q: q[0], q[1], q[2]

of truth values that list the value of p[x] and q[x] for each value of x.

</display>@*/

fun p():Formula1 = choose p:Formula1 with
  p[0] = /*@<input id="pin0" cols="4" label="p:">@*/true/*@</input>@*/
∧ p[1] = /*@<input id="pin1" cols="4">@*/false/*@</input>@*/
∧ p[2] = /*@<input id="pin2" cols="4">@*/false/*@</input>@*/
;

//@<display></display>

fun q():Formula1 = choose q:Formula1 with
  q[0] = /*@<input id="qin0" cols="4" label="q:">@*/false/*@</input>@*/
∧ q[1] = /*@<input id="qin1" cols="4">@*/true/*@</input>@*/
∧ q[2] = /*@<input id="qin2" cols="4">@*/false/*@</input>@*/
;

theorem task4b() ⇔ a4(p(),q()) ∧ ~b4(p(),q());

/*@<display>

</display>@*/

/*@
<button id="task4b" label="Check Task 4.b" action="task4b">
<argument value="-opt-si"> <argument value="1">
</button>
@*/

/*@ <display>
-----------------------------
TASK 5 DESCRIPTION:
-----------------------------
In the following we consider formulas interpreted over the domain ℕ[2] = { 0,1,2 }.

Below you see a definition
</display>@*/
type Formula2 = Map[ℕ[2],Formula1];
//@ <public>
pred a5(p:Formula1, q:Formula2) ⇔
  ¬ ∀x:ℕ[2]. p[x] ⇒ ∃y:ℕ[2]. q[x][y]
;
/*@ <display>
with a formula in which predicates p[x] and q[x][y] occur; please note that
applications of these predicates have to be written with square brackets [ ]
around every argument.

Give a definition
</display>@*/
//@ <public>
pred b5(p:Formula1, q:Formula2) ⇔
/*@<input id="task5in" cols="40" rows="1">@*/∃x:ℕ[2]. p[x] ∧ ∀y:ℕ[2]. ¬q[x][y]/*@</input>@*/

;
//@ <private>
/*@ <display>
with a formula that is logically equivalent to the one given above but in which
the negation connective ¬ is only applied to atomic formulas (i.e., "push the
negation as much inside as possible").

Note: the following will only check the equivalence of your formula to the
given one, not whether your formula indeed satisfies the syntactic description
given above.

</display>@*/
```

```
theorem task5() ⇔ ∀p:Formula1, q:Formula2. a5(p,q) ⇔ b5(p,q);

/*@
<button id="task5" label="Check Task 5" action="task5">
<argument value="-opt-si"> <argument value="1">
<argument value="-opt-nd"> <argument value="1">
</button>
@*/
```

A.3 First Order Pragmatics


Exercise: First Order Pragmatics

JKU LIT Project LogTechEdu

Submitter:

0 of 8 grade points have been earned so far.

TASK DESCRIPTION:

In the following we consider arrays of maximum length N whose elements are
natural numbers of maximum size M:

```
val N = 4 ; // choose small values
val M = 3 ;
type Elem = ℕ[M];
type Arr = Array[N,Elem];
type Index = ℤ[-1,N];
```

Take the problem of finding the smallest index i at which an element e occurs
among the first n elements of an array a:

Input: a ∈ Arr, n ∈ Index, e ∈ Elem where

- n is greater equal 0 and
- e occurs among the first n elements in a, i.e.:
  there exists some index i greater equal 0 but
  less than n at which a holds e.

Output: i ∈ Index where i is the smallest index at which e occurs among the
first n elements of a, i.e.:

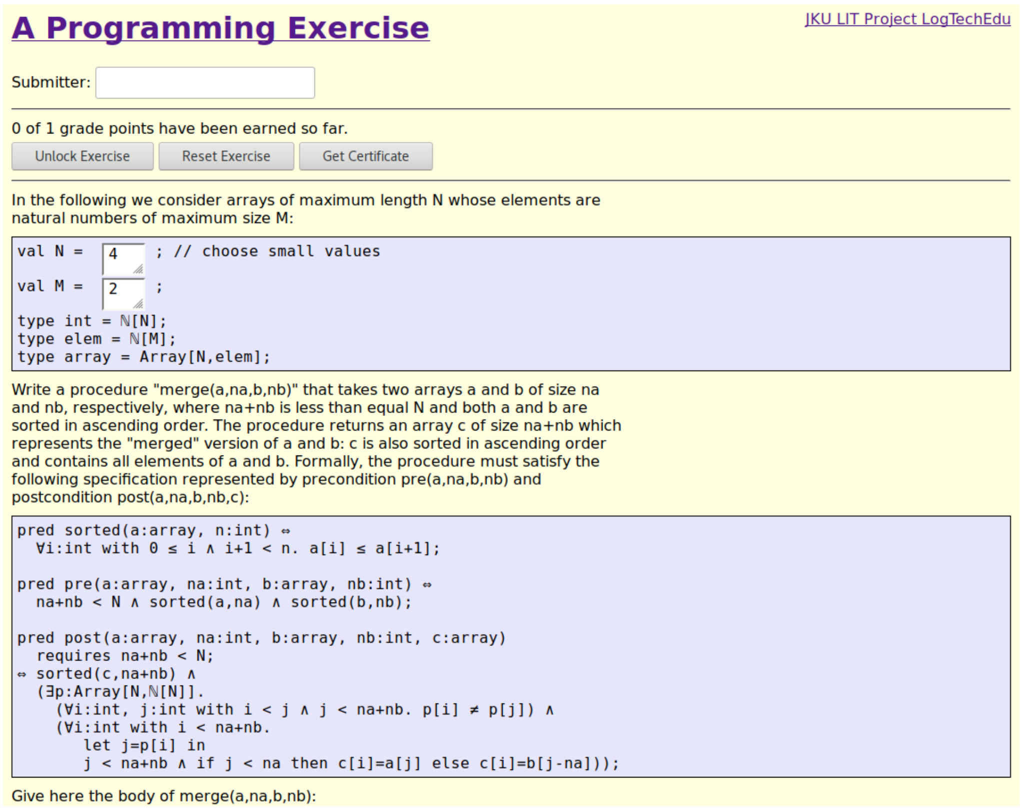
```
/*@ <webex id="firstOrderPragmatics" type="riscal"
       header="Exercise: First Order Pragmatics"
       href="http://fmv.jku.at/logik/index.html">
@*/

/*@ <display>
TASK DESCRIPTION:

In the following we consider arrays of maximum length N whose elements are
natural numbers of maximum size M:
</display> @*/

//@ <public>
val N = /*@<input id="N" cols="2">@*/4/*@</input>@*/; // choose small values
val M = /*@<input id="M" cols="2">@*/3/*@</input>@*/;
type Elem = ℕ[M];
type Arr = Array[N,Elem];
type Index = ℤ[-1,N];
//@ <private>

/*@ <display>
Take the problem of finding the smallest index i at which an element e occurs
among the first n elements of an array a:

    Input:  a ∈ Arr, n ∈ Index, e ∈ Elem where

    - n is greater equal 0 and
    - e occurs among the first n elements in a, i.e.:
          there exists some index i greater equal 0 but
          less than n at which a holds e.

    Output: i ∈ Index where i is the smallest index at which e occurs among the
    first n elements of a, i.e.:

    - i is greater equal 0 but less than n and a holds e at i
    - i is less than equal all indices i0 that have the same property

The task is to develop a formal specification of this problem, i.e., to define a
```

```
predicate P(a,n,e), the input condition of the problem, and a predicate Q(a,n,e,i),
the output condition.
</display> @*/

//@ <public>
pred P(a:Arr,n:Index,e:Elem) ⇔
//@ <input id="P" cols="70" rows="3">
  // formulate here the input condition
  0 ≤ n ∧
  ∃i:Index. 0 ≤ i ∧ i < n ∧ a[i] = e
//@ </input>
;

pred Q(a:Arr,n:Index,e:Elem,i:Index) ⇔
//@ <input id="Q" cols="70" rows="3">
  // formulate here the output condition
  0 ≤ i ∧ i < n ∧ a[i] = e ∧
  ∀i0:Index. 0 ≤ i0 ∧ i0 < n ∧ a[i0] = e ⇒ i ≤ i0
//@ </input>
;

/*@ <display>

TASK a: investigate the inputs/outputs allowed by your specification
(see for which inputs it allows which outputs):
</display> @*/
//@ <public>
fun compute(a:Arr,n:Index,e:Elem):Index
  requires P(a,n,e);
= choose i:Index with Q(a,n,e,i);
//@ <private>

/*@ <button id="compute" label="Run compute()"
      action="compute">
      <argument value="-opt-si"> <argument value="0">
      <argument value="-opt-nd"> <argument value="1">
</button>@*/

/*@ <display>

TASKS b and c: check whether your input condition is satisfiable but not trivial.
</display> @*/

//@ <public>
theorem satP()     ⇔ ∃a:Arr,n:Index,e:Elem. P(a,n,e);
theorem nontrivP() ⇔ ∃a:Arr,n:Index,e:Elem. ¬P(a,n,e);
//@ <private>

/*@ <button id="satP" label="Check satP()"
      action="satP">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

//@ <display></display>

/*@ <button id="nontrivP" label="Check nontrivP()"
      action="nontrivP">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

/*@ <display>

TASKS d and e: check whether your output condition is satisfiable but not trivial.
</display> @*/

//@ <public>
pred satQ()     ⇔ ∀a:Arr,n:Index,e:Elem.
                    P(a,n,e) ⇒ ∃i:Index. Q(a,n,e,i);
pred nontrivQ() ⇔ ∃a:Arr,n:Index,e:Elem.
                    P(a,n,e) ∧ ∃i:Index. ¬Q(a,n,e,i);
//@ <private>
```

```
/*@ <button id="satQ" label="Check satQ()"
      action="satP">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

//@ <display></display>

/*@ <button id="nontrivQ" label="Check nontrivQ()"
      action="nontrivP">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

/*@ <display>

TASK f: check whether your output condition defines the result uniquely:
</display> @*/

//@ <public>
pred uniqueQ() ⇔
  ∀a:Arr,n:Index,e:Elem. P(a,n,e) ⇒
    ∀i1:Index,i2:Index. Q(a,n,e,i1) ∧ Q(a,n,e,i2) ⇒ i1 = i2;
//@ <private>

/*@ <button id="uniqueQ" label="Check uniqueQ()"
      action="uniqueQ">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

/*@ <display>

TASK g: check whether your specification adequately specifies the following
code (is not too strong nor too weak):
</display> @*/

//@ <public>
proc search(a:Arr,n:Elem,e:Elem):Index
  requires P(a, n, e);
  ensures Q(a, n, e, result);
{
  var i:Index := 0;
  var result:Index := -1;
  while i < n ∧ result = -1 do
  {
    if a[i] = e then result := i;
    i := i+1;
  }
  return result;
}
//@ <private>

//@ <display></display>

//@ <public>
theorem correct(a:Arr,n:Elem,e:Elem)
  requires P(a,n,e);
⇔ let i = search(a,n,e) in Q(a,n,e,i);
//@ <private>

/*@ <button id="correct" label="Check correct()"
      action="correct">
      <argument value="-opt-si"> <argument value="0">
</button>@*/

//@ <public>
theorem complete(a:Arr,n:Elem,e:Elem)
  requires P(a,n,e);
⇔ ∀i:Index. Q(a,n,e,i) ⇒ i = search(a,n,e);
//@ <private>

/*@ <button id="complete" label="Check complete()"
```

```
      action="complete">
      <argument value="-opt-si"> <argument value="0">
</button>@*/
```

A.4 A Programming Exercise



0 of 1 grade points have been earned so far.

In the following we consider arrays of maximum length N whose elements are natural numbers of maximum size M:

```
val N = 4 ; // choose small values
val M = 2 ;
type int = ℕ[N];
type elem = ℕ[M];
type array = Array[N,elem];
```

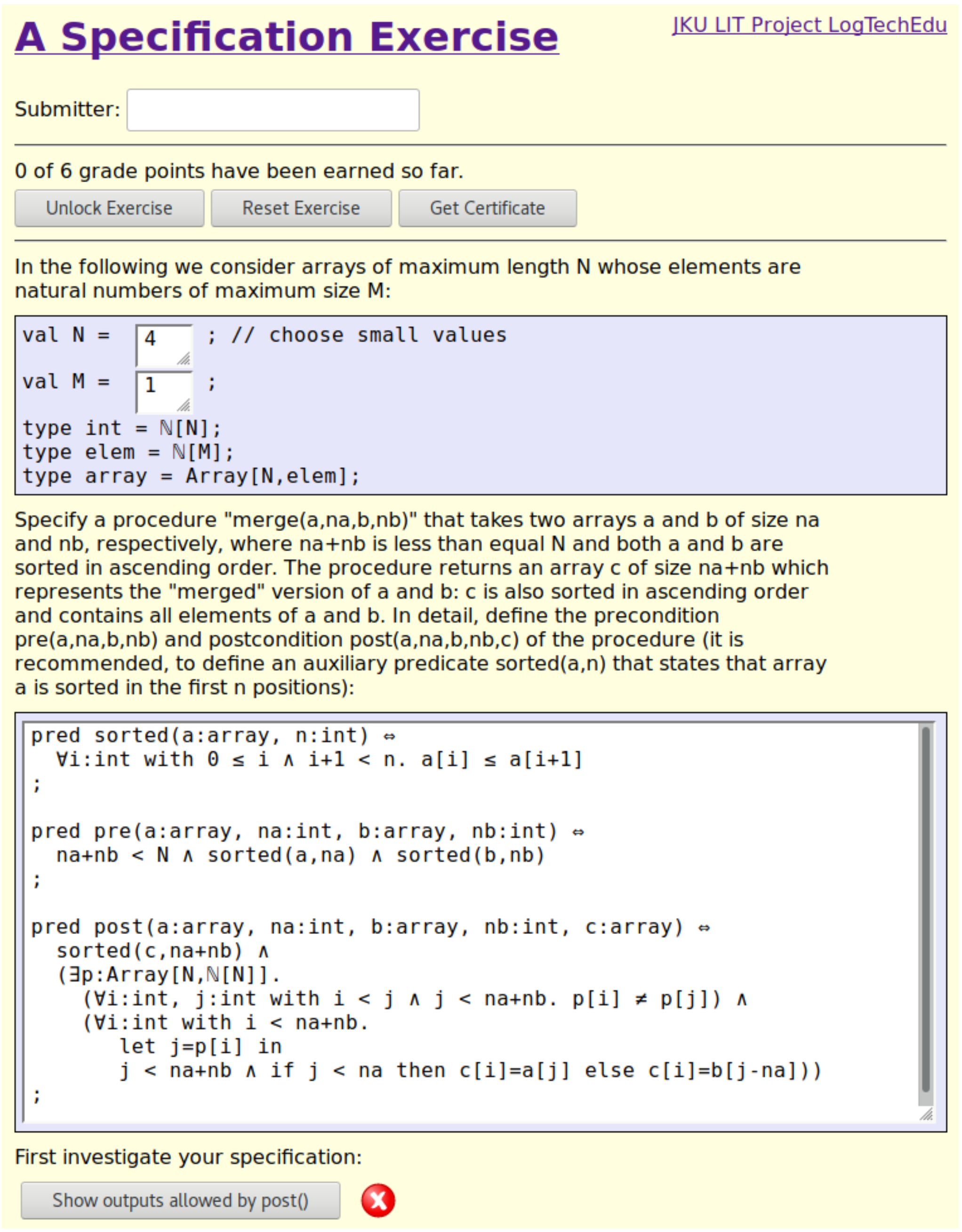
Write a procedure "merge(a,na,b,nb)" that takes two arrays a and b of size na and nb, respectively, where na+nb is less than equal N and both a and b are sorted in ascending order. The procedure returns an array c of size na+nb which represents the "merged" version of a and b: c is also sorted in ascending order and contains all elements of a and b. Formally, the procedure must satisfy the following specification represented by precondition pre(a,na,b,nb) and postcondition post(a,na,b,nb,c):

```
pred sorted(a:array, n:int) ⇔
  ∀i:int with 0 ≤ i ∧ i+1 < n. a[i] ≤ a[i+1];

pred pre(a:array, na:int, b:array, nb:int) ⇔
  na+nb < N ∧ sorted(a,na) ∧ sorted(b,nb);

pred post(a:array, na:int, b:array, nb:int, c:array)
  requires na+nb < N;
⇔ sorted(c,na+nb) ∧
  (∃p:Array[N,ℕ[N]].
    (∀i:int, j:int with i < j ∧ j < na+nb. p[i] ≠ p[j]) ∧
    (∀i:int with i < na+nb.
       let j=p[i] in
       j < na+nb ∧ if j < na then c[i]=a[j] else c[i]=b[j-na]));
```

Give here the body of merge(a,na,b,nb):

```
/*@ <webex id="program" type="riscal"
      header="A Programming Exercise"
      href="https://www.risc.jku.at/research/formal/software/RISCAL">
@*/

/*@ <display>
In the following we consider arrays of maximum length N whose elements are
natural numbers of maximum size M:
</display> @*/

//@ <public>
val N = /*@<input id="N" cols="2">@*/4/*@</input>@*/; // choose small values
val M = /*@<input id="M" cols="2">@*/2/*@</input>@*/;
type int = ℕ[N];
type elem = ℕ[M];
type array = Array[N,elem];
/*@ <display>
Write a procedure "merge(a,na,b,nb)" that takes two arrays a and b of size na
and nb, respectively, where na+nb is less than equal N and both a and b are
sorted in ascending order. The procedure returns an array c of size na+nb which
represents the "merged" version of a and b: c is also sorted in ascending order
and contains all elements of a and b. Formally, the procedure must satisfy the
following specification represented by precondition pre(a,na,b,nb) and
postcondition post(a,na,b,nb,c):
</display> @*/
pred sorted(a:array, n:int) ⇔
  ∀i:int with 0 ≤ i ∧ i+1 < n. a[i] ≤ a[i+1];

pred pre(a:array, na:int, b:array, nb:int) ⇔
  na+nb < N ∧ sorted(a,na) ∧ sorted(b,nb);

pred post(a:array, na:int, b:array, nb:int, c:array)
  requires na+nb < N;
⇔ sorted(c,na+nb) ∧
  (∃p:Array[N,ℕ[N]].
    (∀i:int, j:int with i < j ∧ j < na+nb. p[i] ≠ p[j]) ∧
    (∀i:int with i < na+nb.
       let j=p[i] in
       j < na+nb ∧ if j < na then c[i]=a[j] else c[i]=b[j-na]));
/*@ <display>
Give here the body of merge(a,na,b,nb):
</display>
@*/
proc merge(a:array, na:int, b:array, nb:int): array
//@ <private>
  requires pre(a,na,b,nb);
  ensures post(a,na,b,nb,result);
//@ <public>
{
  //@ <input id="body" cols="70" rows="16">
var c:array = Array[N,elem](0);

var ia:int := 0; var ib:int := 0;
for var i:int = 0; i<na+nb; i:=i+1 do
{
  if ia < na ∧ (ib = nb ∨ a[ia] ≤ b[ib]) then
  {
    c[i] := a[ia]; ia := ia+1;
  }
  else
  {
    c[i] := b[ib]; ib := ib+1;
  }
}

return c;//@ </input>


}
//@ <private>
//@<display>Check your implementation with respect to the specification:</display>
/*@ <button id="run" label="Check merge(a,na,b,nb)" action="merge">
    <argument value="-opt-si"> <argument option="silent">
</button>
@*/
//@<display></display>
//@ <option id="silent" label="Silent Execution:" type="bool">
```

A.5 A Specification Exercise



A Specification Exercise

JKU LIT Project LogTechEdu

Submitter:

0 of 6 grade points have been earned so far.

In the following we consider arrays of maximum length N whose elements are natural numbers of maximum size M:

```
val N = 4 ; // choose small values
val M = 1 ;
type int = ℕ[N];
type elem = ℕ[M];
type array = Array[N,elem];
```

Specify a procedure "merge(a,na,b,nb)" that takes two arrays a and b of size na and nb, respectively, where na+nb is less than equal N and both a and b are sorted in ascending order. The procedure returns an array c of size na+nb which represents the "merged" version of a and b: c is also sorted in ascending order and contains all elements of a and b. In detail, define the precondition pre(a,na,b,nb) and postcondition post(a,na,b,nb,c) of the procedure (it is recommended, to define an auxiliary predicate sorted(a,n) that states that array a is sorted in the first n positions):

```
pred sorted(a:array, n:int) ⇔
  ∀i:int with 0 ≤ i ∧ i+1 < n. a[i] ≤ a[i+1]
;

pred pre(a:array, na:int, b:array, nb:int) ⇔
  na+nb < N ∧ sorted(a,na) ∧ sorted(b,nb)
;

pred post(a:array, na:int, b:array, nb:int, c:array) ⇔
  sorted(c,na+nb) ∧
  (∃p:Array[N,ℕ[N]].
    (∀i:int, j:int with i < j ∧ j < na+nb. p[i] ≠ p[j]) ∧
    (∀i:int with i < na+nb.
       let j=p[i] in
       j < na+nb ∧ if j < na then c[i]=a[j] else c[i]=b[j-na]))
;
```

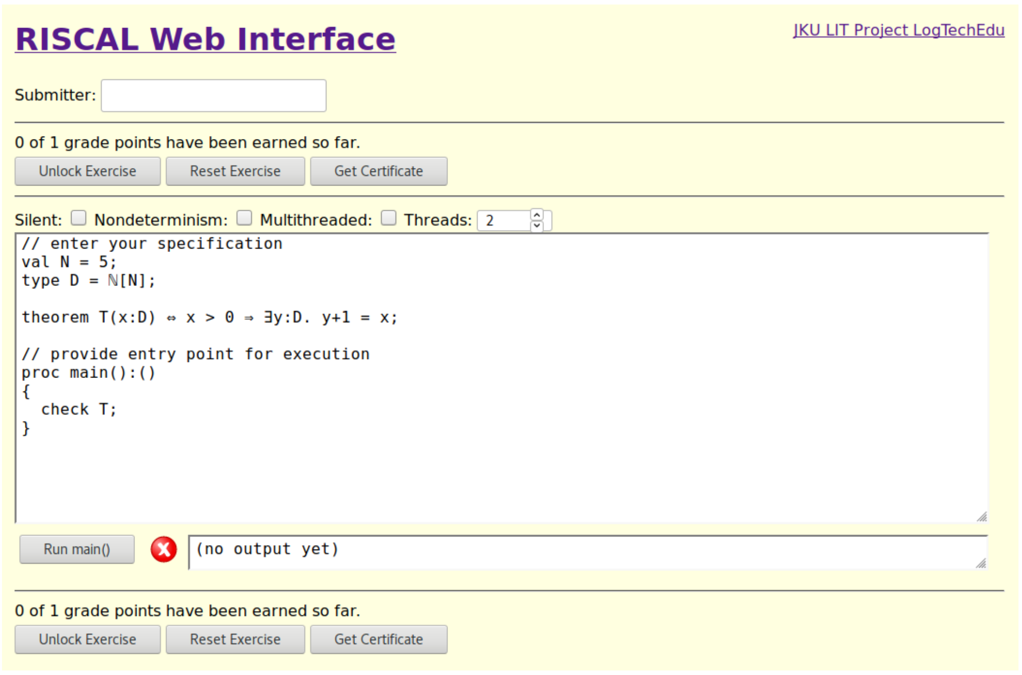
First investigate your specification:

```
/*@ <webex id="specification" type="riscal"
      header="A Specification Exercise"
      href="https://www.risc.jku.at/research/formal/software/RISCAL">
@*/

/*@ <display>
In the following we consider arrays of maximum length N whose elements are
natural numbers of maximum size M:
</display> @*/

//@ <public>
val N = /*@<input id="N" cols="2">@*/4/*@</input>@*/; // choose small values
val M = /*@<input id="M" cols="2">@*/1/*@</input>@*/;
type int = ℕ[N];
type elem = ℕ[M];
type array = Array[N,elem];
```

```
/*@ <display>
Specify a procedure "merge(a,na,b,nb)" that takes two arrays a and b of size na
and nb, respectively, where na+nb is less than equal N and both a and b are
sorted in ascending order. The procedure returns an array c of size na+nb which
represents the "merged" version of a and b: c is also sorted in ascending order
and contains all elements of a and b. In detail, define the precondition
pre(a,na,b,nb) and postcondition post(a,na,b,nb,c) of the procedure (it is
recommended, to define an auxiliary predicate sorted(a,n) that states that array
a is sorted in the first n positions):
</display> @*/
//@ <input id="spec" cols="70" rows="16">
pred sorted(a:array, n:int) ⇔
  ∀i:int with 0 ≤ i ∧ i+1 < n. a[i] ≤ a[i+1]
;

pred pre(a:array, na:int, b:array, nb:int) ⇔
  na+nb < N ∧ sorted(a,na) ∧ sorted(b,nb)
;

pred post(a:array, na:int, b:array, nb:int, c:array) ⇔
  sorted(c,na+nb) ∧
  (∃p:Array[N,ℕ[N]].
    (∀i:int, j:int with i < j ∧ j < na+nb. p[i] ≠ p[j]) ∧
    (∀i:int with i < na+nb.
       let j=p[i] in
       j < na+nb ∧ if j < na then c[i]=a[j] else c[i]=b[j-na]))
;
//@ </input>
//@ <private>
fun result(a:array, na:int, b:array, nb:int):array
  requires pre(a,na,b,nb);
= choose c:array with post(a,na,b,nb,c);

theorem satisfiable(a:array, na:int, b:array, nb:int)
  requires pre(a,na,b,nb);
⇔ ∃c:array. post(a,na,b,nb,c);

theorem nottrivial()
⇔ ∃a:array, na:int, b:array, nb:int with pre(a,na,b,nb).
    ∃c:array. ¬post(a,na,b,nb,c);

theorem unique(a:array, na:int, b:array, nb:int)
  requires pre(a,na,b,nb);
⇔ ∀c1:array with post(a,na,b,nb,c1).
  ∀c2:array with post(a,na,b,nb,c2).
    ∀i:int with 0 ≤ i ∧ i < na+nb.
      c1[i] = c2[i];

//@<display>First investigate your specification:</display>
/*@ <button id="result" label="Show outputs allowed by post()"
      action="result">
      <argument value="-opt-si"> <argument value="0">
      <argument value="-opt-nd"> <argument value="1">
</button>@*/

//@<display>Now further validate your specification:</display>
/*@ <button id="satisfiable" label="Is post() satisfiable?"
      action="satisfiable">
      <argument value="-opt-si"> <argument value="1">
</button>
@*/

//@<display></display>
/*@ <button id="nottrivial" label="Is post() not trivial?"
      action="nottrivial">
      <argument value="-opt-si"> <argument value="1">
</button>
@*/

/*@<display>
Now check whether post() define the first na+nb elements of the result array
```

```
uniquely (the remaining elements are not of interest):
</display>@*/

/*@ <button id="unique" label="Does post() uniquely define result?"
      action="unique">
      <argument value="-opt-si"> <argument value="1">
      <argument value="-opt-mt"> <argument value="1">
      <argument value="-opt-tr"> <argument value="4">
</button>
@*/

proc merge(a:array, na:int, b:array, nb:int): array
{
var c:array = Array[N,elem](0);

var ia:int := 0; var ib:int := 0;
for var i:int = 0; i<na+nb; i:=i+1 do
{
  if ia < na ∧ (ib = nb ∨ a[ia] ≤ b[ib]) then
  {
    c[i] := a[ia]; ia := ia+1;
  }
  else
  {
    c[i] := b[ib]; ib := ib+1;
  }
}

return c;
}

theorem notTooStrong(a:array, na:int, b:array, nb:int)
  requires pre(a,na,b,nb);
⇔ post(a, na, b, nb, merge(a, na, b, nb));

theorem notTooWeak(a:array, na:int, b:array, nb:int, c:array)
  requires pre(a,na,b,nb) ∧ post(a, na, b, nb, c);
⇔  let c0 = merge(a, na, b, nb) in
    ∀i:int with 0 ≤ i ∧ i < na+nb.
      c[i] = c0[i];

/*@<display>
Finally check whether your specification specifies merge(a,na,b,nb) adequately:
</display>@*/
/*@
<button id="notTooStrong" label="Is specification not too strong?" action="notTooStrong">
    <argument value="-opt-si"> <argument value="1">
</button>
@*/
//@<display></display>
/*@
<button id="notTooWeak" label="Is specification not too weak?" action="notTooWeak">
    <argument value="-opt-si"> <argument value="1">
    <argument value="-opt-mt"> <argument value="1">
    <argument value="-opt-tr"> <argument value="4">
</button>
@*/
```

A.6 RISCAL Web Interface



RISCAL Web Interface

JKU LIT Project LogTechEdu

Submitter:

0 of 1 grade points have been earned so far.

Silent: ☐ Nondeterminism: ☐ Multithreaded: ☐ Threads: 2

```
// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}
```

```
(no output yet)
```

0 of 1 grade points have been earned so far.

```
/*@ <webex id="RISCAL" type="riscal"
       header="RISCAL Web Interface"
       href="https://www.risc.jku.at/research/formal/software/RISCAL">
@*/

//@ <option id="silent" label="Silent:" type="bool">
//@ <option id="nondet" label="Nondeterminism:" type="bool">
//@ <option id="multi" label="Multithreaded:" type="bool">
//@ <option id="threads" label="Threads:" type="int" min="2" max="4">

//@ <input id="in" cols="98" rows="15">
// enter your specification
val N = 5;
type D = ℕ[N];

theorem T(x:D) ⇔ x > 0 ⇒ ∃y:D. y+1 = x;

// provide entry point for execution
proc main():()
{
  check T;
}
//@ </input>

/*@ <button id="run" label="Run main()" action="main">
    <argument value="-opt-si"> <argument option="silent">
    <argument value="-opt-nd"> <argument option="nondet">
    <argument value="-opt-mt"> <argument option="multi">
    <argument value="-opt-tr"> <argument option="threads">
    </button>
@*/
```