

# Interactive Visualization and Minimization of Labelled Transition Systems

Felix Freiburger

Saarland University

Natural Sciences and Technology I

Computer Science

30.09.2014

Bachelor's Thesis

Reviewers:

Univ.-Prof. Dr.-Ing. Holger Hermanns

Univ.-Prof. Dr. Verena Wolf



# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                                  | <b>3</b>  |
| <b>2</b> | <b>Preliminaries</b>                                 | <b>4</b>  |
| 2.1      | pseuCo . . . . .                                     | 4         |
| 2.2      | CCS . . . . .  | 4         |
| 2.3      | Previous Work . . . . .                              | 4         |
| 2.4      | Explanation of Symbols and Technical Terms . . . . . | 4         |
| <b>3</b> | <b>Accessing pseuCo.com</b>                          | <b>5</b>  |
| 3.1      | Accessing the Online Version of pseuCo.com . . . . . | 5         |
| 3.2      | Building pseuCo.com from Source . . . . .            | 5         |
| <b>4</b> | <b>User Manual</b>                                   | <b>6</b>  |
| <b>5</b> | <b>Developer Manual</b>                              | <b>7</b>  |
| 5.1      | The Debugging Page . . . . .                         | 7         |
| 5.2      | Dependencies . . . . .                               | 7         |
| 5.3      | The Build Process . . . . .                          | 7         |
| 5.4      | Application Start-Up . . . . .                       | 8         |
| 5.5      | How pseuCo.com Stores and Handles Files . . . . .    | 9         |
| 5.6      | Offline Mode . . . . .                               | 10        |
| 5.7      | Tasks and the Background Worker . . . . .            | 13        |
| 5.8      | Editors, Translations and Actions . . . . .          | 16        |
| 5.9      | LTS Exploration . . . . .                            | 17        |
| 5.10     | LTS Visualization and Graph Layout . . . . .         | 18        |
| 5.11     | LTS Minimization . . . . .                           | 19        |
| 5.12     | The Server Component of pseuCo.com . . . . .         | 29        |
| 5.13     | A Build Server for pseuCo.com . . . . .              | 30        |
| <b>6</b> | <b>The Future of pseuCo.com</b>                      | <b>32</b> |
| 6.1      | Model Checking . . . . .                             | 32        |
| 6.2      | Additional Input Formats . . . . .                   | 32        |
| 6.3      | New Translations for Existing File Types . . . . .   | 32        |
|          | <b>Appendix A Definitions for LTS</b>                | <b>34</b> |
|          | <b>Appendix B Code Listings</b>                      | <b>35</b> |
|          | <b>Appendix C The Repository Structure</b>           | <b>47</b> |
|          | <b>Appendix D The Build Directory Structure</b>      | <b>49</b> |
|          | <b>Appendix E User Manual (digital printout)</b>     | <b>51</b> |
|          | <b>Glossary</b>                                      | <b>63</b> |
|          | <b>Bibliography</b>                                  | <b>65</b> |



# Introduction

Labelled transition systems (LTS) are a popular way of modelling the behaviour of reactive systems and processes. Consequently, there are many tools to create, view and analyse such systems. However, many of these tools are hard to set up or use or cannot work with large or infinite transition systems. This unnecessarily complicates the use of such systems, especially for teaching. There, complex set-up cannot be expected from students, and infinite systems are regularly analysed while exploring the limits of LTS and languages having LTS semantics.

For this thesis, a new application handling labelled transition systems was developed – **pseuCo.com**. Being a JavaScript-based web application, it offers a set-up-free experience, while still keeping many advantages of classic, native applications, like offline use.

In addition, **pseuCo.com** was designed to work with large or infinite transition systems as well as with small ones, enabling users to view and analyse them just as they would expect. Users can visualize any transition system right in their browser, with the ability to expand and collapse states as they wish. While automatic graph layout keeps the system easily readable even as states appear and disappear, users are free to rearrange states as they wish. Support for (partial) minimization – even of infinite systems – allows to get a faster overview over the behaviour modelled by a system.

**PseuCo.com** includes a compiler from **pseuCo** to CCS, and a mechanism to convert CCS terms to labelled transition systems. Both are based on **PseuCoCo**[1], and seamlessly integrated into the user interface.

**PseuCo.com** features an internal file system for supported file types, with extensive import and export capabilities. A link-based file sharing mechanism enables users to easily send **pseuCo.com**-files to any user, without requiring anything but an internet connection and a modern browser from the receiver.

# Preliminaries

## 2.1 pseuCo

PseuCo is a programming language designed to teach concurrent programming. It features a heavily simplified Java-like syntax, and has built-in support for message passing inspired by Go. PseuCo was created by Christian Eisentraut and Holger Hermanns at the chair for Dependable Systems and Software at Saarland University to be used in the mandatory Bachelor-level lecture “Nebenläufige Programmierung” (“Concurrent Programming”).

## 2.2 CCS

The Calculus of Communicating Systems (CCS) is a process calculus designed to model communicating systems. There are multiple versions of CCS. In this context, a pragmatic extension of CCS supporting value passing (of booleans, integers and strings), sequencing and termination is used[1]. The semantics of a CCS process is a Labelled Transition System (LTS).

## 2.3 Previous Work

For his Bachelor’s thesis[1], Sebastian Biewer developed a translational semantics for pseuCo, based on CCS. His work includes a JavaScript compiler from pseuCo programs to CCS terms, a parser for CCS and the CCS operational semantics, yielding transition systems as underlying foundational objects. This resulted in the open source software PseuCoCo, available at <https://github.com/Biewer/PseuCoCo>. It also contains a parser and type checker for pseuCo in JavaScript, written by Pascal Held.

However, PseuCoCo contains no full graphical user interface, and there are no ways to interact with the resulting transition system except for a simple tracing functionality.

## 2.4 Explanation of Symbols and Technical Terms

- `repository` refers to the path where you stored the source code repository of `pseuCo.com`.

In the electronic version, you can click on the remainder of the path to see the current version in the public online repository.

- `base URL` refers to the URL where you access `pseuCo.com` – either `http://pseuco.com/` or the URL your local webserver runs at.

In the electronic version, you can click on the remainder of the path to see the online version at `http://pseuco.com/`.

- `API base URL` refers to the path where the `pseuCo.com` server component can be reached. For the public instance, this is `http://pseuco.com/api/`.
- A short description of some technical terms can be found in the glossary on page 64.

# Accessing pseuCo.com

To access `pseuCo.com` you can either use the public online version or build it from source yourself.

## 3.1 Accessing the Online Version of `pseuCo.com`

Using a modern webbrowser, open `http://pseuco.com/`. Given a working internet connection, this will open the application and save a copy for later offline use. In the future, you can open this URL even without an internet connection to use the saved copy.

## 3.2 Building `pseuCo.com` from Source

The source code of `pseuCo.com` can be acquired using Git from its public repository, which is located at `http://git.fefrei.de/concurrent-programming-web.git/`.

`PseuCo.com` has many build- and run-time dependencies, most of which can be installed automatically. However, some have to be installed manually upfront:

1. a recent version of `node.js` from `http://nodejs.org/`, including `npm`, the Node Package Manager
2. the command-line interface of `grunt`, `grunt-cli`, which can be installed by running:

---

```
1 npm install -g grunt-cli
```

---

All other dependencies can be installed by `npm` and the default build script.

If `git`, `node` and `grunt-cli` are installed correctly, you can fetch and build `pseuCo.com` as follows:

---

```
1 git clone http://git.fefrei.de/concurrent-programming-web.git/
2 cd concurrent-programming-web
3 npm install
4 grunt
```

---

For file sharing and file templates, `pseuCo.com` needs a server component. While being in the directory `(repository)/server/`, it can be run with `node server.js`. The server will run on port 9128.

To open `pseuCo.com`, the directory `(repository)/build/` needs to be served by an HTTP server<sup>1</sup>.

By default, `pseuCo.com` tries to reach the `pseuCo.com` server at `(base URL)/api/`, relative to the URL it is running from. A reverse proxy can be used to forward request to this URL to the server component, as shown in Listing 3.1. Alternatively, for development, `pseuCo.com` can be configured to access the server component at another URL. For details, see section 5.1.

As soon as the HTTP server is running, accessing it with a modern browser will open the user interface.

Listing 3.1: A configuration file for using Apache as a reverse proxy

---

```
1 RewriteEngine On
2 RewriteRule (.*?) http://localhost:9128/$1 [P]
```

---

---

<sup>1</sup>Using the `file:///`-protocol will not work in most browsers due to security restrictions.

# User Manual

PseuCo.com features an interactive user manual that can be accessed at `base URL`/#/help. A digital printout of the manual can be found in Appendix E.

## Remark

The User Manual contains interactive elements. If possible, please read the online version instead of the digital printout version.

You can find the current public version of the user manual at <http://pseuco.com/#/help>.



# Developer Manual

## 5.1 The Debugging Page

The `pseuCo.com` UI has a hidden debugging page that allows quick access to some internal settings, located at `[base URL]/#/debug`.

## 5.2 Dependencies

`PseuCo.com` depends on several JavaScript libraries, both for the build system and for the actual application. Libraries are managed by npm and Bower.

A full list of libraries can be found in the corresponding configuration files: Listing 5.1 shows all dependencies that Bower installs, which are *runtime* dependencies. Similarly, Listing 5.2 shows the dependencies installed by npm. All of them are *build-time* dependencies, except for `PseuCoCo`.

The Ace text editor is an additional dependency. Since it is not available as a Bower package, it is downloaded and unzipped programatically by a build script.

## 5.3 The Build Process

`PseuCo.com` can be built from source using the included build mechanism – refer to section 3.2 for an explanation. In this section, you can find additional information on its internal functionality.

Running `npm install` installs all dependencies listed in Listing 5.2, including grunt and corresponding plug-ins. Afterwards, running `grunt` executes the `default` task defined in Listing B.1. This task performs the following actions:

1. Test the code with jshint to find common programming errors.
2. Call Bower to install all dependencies from Listing 5.1.

Listing 5.1: Dependency section from `bower.json`

---

```
18 "dependencies": {
19   "bootstrap": "~3.2.0",
20   "angular": "~1.2.23",
21   "angular-strap": "~2.0.5",
22   "angular-animate": "~1.2.23",
23   "angular-route": "~1.2.23",
24   "angular-motion": "~0.3.3",
25   "angular-sanitize": "~1.2.23",
26   "angularLocalStorage": "~0.1.7",
27   "underscore": "~1.7.0",
28   "momentjs": "~2.8.2",
29   "hammerjs": "~2.0.2",
30   "d3": "~3.4.11",
31   "bootstrap-additions": "~0.2.3",
32   "jquery": "~2.1.1",
33   "jquery.cookie": "~1.4.1"
```

---

Listing 5.2: Dependency section from `package.json`

```

9  "devDependencies": {
10     "grunt": "~0.4.5",
11     "grunt-bower-task": "~0.4.0",
12     "grunt-contrib-uglify": "~0.5.1",
13     "grunt-contrib-jshint": "~0.10.0",
14     "grunt-contrib-cssmin": "~0.10.0",
15     "grunt-contrib-copy": "~0.5.0",
16     "grunt-contrib-clean": "~0.6.0",
17     "grunt-contrib-concat": "~0.5.0",
18     "grunt-contrib-watch": "~0.6.1",
19     "grunt-peg": "~1.5.0",
20     "grunt-contrib-coffee": "~0.11.1",
21     "grunt-browserify": "~2.1.4",
22     "grunt-zip": "~0.16.0",
23     "grunt-curl": "~2.0.2",
24     "grunt-contrib-rename": "0.0.3"
25 },
26 "dependencies": {
27     "PseuCoCo": "git://github.com/Biewer/PseuCoCo.git#v0.6.26"
28 }

```

3. If the specified version of Ace is missing: download and unzip it.
4. Call PEG.js to create the parsers for all grammars.
5. Compile the CoffeeScript files from PseuCoCo.
6. Bundle up all JavaScript files from PseuCoCo to `pseucoco.js` using Browserify.
7. Concatenate all JavaScript files from the external libraries to a single file `lib.js`, and minimize it by removing unnecessary white space, resulting in `lib.min.js`.
8. Concatenate all main JavaScript files to `app.js`.
9. Concatenate and minify all CSS files from libraries to `lib.css`.
10. Concatenate and minify all CSS files from the main application to `app.css`.
11. Concatenate the JavaScript files needed for the background worker to `worker.js` – see section 5.7 for details.
12. Copy all the generated and static files to `repository/build/` to build the final directory structure the web server should present.

## 5.4 Application Start-Up

PseuCo.com heavily relies on AngularJS. All core functionality is implemented as AngularJS controllers, directives and factories.

### Remark

AngularJS heavily influences the design of JavaScript applications. Without a sound understanding of how AngularJS works, most of the code of `pseuCo.com` will be hard to understand.

The website of AngularJS has some small examples demonstrating the way AngularJS applications work as well as many useful resources in the “Learn” tab.

As in most AngularJS applications, the application start-up is controlled by the main HTML file. The core part of it is shown in Listing 5.3.

Listing 5.3: Simplified and shortened version of `index.html`

---

```

1 <!DOCTYPE html>
2 <html lang="en" data-ng-app="cp.app" manifest="offline.appcache">
3   <head>
4     <meta name="viewport" content="user-scalable=no, width=device-width, initial-
       ↳ scale=1, maximum-scale=1">
5   </head>
6   <body>
7     <div class="navbar navbar-inverse navbar-fixed-top" role="navigation" data-bs-
       ↳ navbar>
8       <!-- navigation bar elements -->
9     </div>
10    <div class="fill" id="mainFill">
11      <div class="container-fluid" id="mainContainer">
12        <div data-ng-view id="view"></div>
13      </div>
14    </div>
15    <script src="js/lib.min.js"></script>
16    <script src="js/app.js"></script>
17  </body>
18 </html>

```

---

Listing 5.4: Simplified and shortened route configuration from `app.js`

---

```

1 .config(function ($routeProvider) {
2   $routeProvider
3     .when('/landing', {
4       templateUrl: 'partials/landing.html'
5     })
6     .when('/files', {
7       controller: 'filesCtrl',
8       templateUrl: 'partials/files.html'
9     })
10    // ...
11    .otherwise({ redirectTo: '/' });
12 })

```

---

After loading the library and application code – which just *defines* the components, but does *not* actually *execute* any application code – AngularJS processes the HTML elements. The attribute `data-ng-app="cp.app"` on the `<html>` element causes the module `cp.app` to be loaded, which is defined in `(repository)/src/js/app.js`. This sets up some shared data, but most importantly, it configures `$routeProvider`, as shown in Listing 5.4.

This causes AngularJS to read the route in the URL, load the corresponding controller, and inject the corresponding partial into `<div data-ng-view id="view"></div>`. For example, if the user opens `(base URL)/#/files`, the partial `(repository)/src/partials/files.html` is injected, and `filesCtrl` (defined in `(repository)/src/js/ui/controllers/filesCtrl.js`) is instantiated.

## 5.5 How `pseuCo.com` Stores and Handles Files

In `pseuCo.com`, users can create and store files. These files are stored in the user's browser, using the HTML Web Storage API<sup>1</sup>.

File storage is managed by the factory `files` in the module `cp.files`, the definition of which can be found in `(repository)/src/js/files/files.js`. It uses the `angularLocalStorage` library. Upon creation every file is assigned a random, 12-digit id by which it can be referred to later on.

To allow listing all files without storing the data of all files in a single key (slowing down access), file metadata is stored separately from file contents: There is a single repository of file metadata, stored under

---

<sup>1</sup>see <http://www.whatwg.org/specs/web-apps/current-work/multipage/webstorage.html>

Listing 5.5: `fileTypes` declaration from `files.js`

---

```

3 .value('fileTypes', {
4   ccs: {
5     fullName: 'CCS',
6     allowCreation: true,
7     allowEmpty: true,
8     allowTemplate: true,
9     allowImport: false,
10    allowFork: true,
11    importableFilesStatement: "no formats for this file type"
12  },
13  pseuco: {
14    fullName: 'pseuCo',
15    allowCreation: true,
16    allowEmpty: true,
17    allowTemplate: true,
18    allowImport: false,
19    allowFork: true,
20    importableFilesStatement: "no formats for this file type"
21  },
22  lts: {
23    fullName: 'LTS',
24    allowCreation: true,
25    allowEmpty: false,
26    allowTemplate: false,
27    allowImport: true,
28    allowFork: false,
29    noWatch: true,
30    importableFilesStatement: "pseuCo.com-LTS-JSON-files and AUT-files"
31  }
32 })

```

---

the key `files`, and one key per file for the actual content.

In the same module, there is a configuration value called `fileTypes`, shown in Listing 5.5. For each allowed file type, it defines multiple properties which are explained in Figure 5.1.

## 5.6 Offline Mode

Albeit being a web application, `pseuCo.com` is available offline. This is achieved by using the HTML Application Cache<sup>2</sup>. It is configured using the manifest file `repository/src/offline.appcache`, shown in Listing 5.6. It contains a list of all files belonging to the application<sup>3</sup>. These files will be downloaded automatically by any browser supporting the application cache because the configuration file is referenced in `index.html`, as shown in Listing 5.3.

Additionally, this configuration file defines `base URL/api/` to be a `NETWORK` path. This ensures that the path is *not* cached (since it is used to communicate with the server component). Furthermore, it prevents the browser from blocking access to it based on a rule in the HTML Application Cache specification that tries to ensure applications behave the same in online and offline mode.

While this suffices to ensure `pseuCo.com` works offline, there are subtle details which make updates work properly.

As soon as the application cache received and stored a complete copy of the application, the server will never be contacted again, except for two possible reasons: To access a `NETWORK` path, or to update the manifest file. When the application is updated, the manifest file must change too, to ensure that existing users download the new version. To achieve this, the build process not only copies the manifest to the output directory, but appends the build date as a comment to the file, as shown in the `copy:appcache` section of Listing B.1.

---

<sup>2</sup>see <https://html.spec.whatwg.org/multipage/browsers.html#offline>

<sup>3</sup>`index.html` is not listed here, since the index file is included implicitly.

| property                 | type    | effect  |
|--------------------------|---------|---|
| fullName                 | string  | the human-readable name of the file type  |
| allowCreation            | boolean | if enabled, show a button to create this type of file on the “Files” tab                      |
| allowEmpty               | boolean | if enabled, allow to create an empty file of this type  |
| allowTemplate            | boolean | if enabled, show templates for this file type and allow creating a new file based on them     |
| allowImport              | boolean | if enabled, allow to import files of this type  |
| allowFork                | boolean | if enabled, allow the user to create a file of this type based on the result of a translation |
| importableFilesStatement | string  | human-readable string explaining which types of text files can be imported for this file type |

Figure 5.1: Explanation of the properties of a `fileType` definition

Listing 5.6: The configuration file for the application cache, `offline.appcache`

```

1 CACHE MANIFEST
2
3 version.txt
4 js/app.js
5 js/worker.js
6 js/lib.min.js
7 css/app.css
8 css/lib.css
9 partials/about.html
10 partials/ace.html
11 partials/actions.html
12 partials/backup.html
13 partials/debug.html
14 partials/edit.html
15 partials/error.html
16 partials/export.html
17 partials/fetch.html
18 partials/files.html
19 partials/help.html
20 partials/import.html
21 partials/landing.html
22 partials/lts.html
23 partials/newfile.html
24 partials/pseucojava.html
25 partials/share.html
26 partials/svg.html
27 partials/trace.html
28 fonts/glyphicons-halflings-regular.eot
29 fonts/glyphicons-halflings-regular.svg
30 fonts/glyphicons-halflings-regular.ttf
31 fonts/glyphicons-halflings-regular.woff
32 img/icon.png
33
34 NETWORK:
35 api/

```

An update has been downloaded. You can switch to the new version now.

Figure 5.2: Notification bar after the application cache was updated

Listing 5.7: A configuration file Apache to configure client-side caching

```
1 AddType text/cache-manifest .appcache
2
3 Header set Cache-Control "no-cache, must-revalidate"
4
5 <Files offline.appcache>
6     Header set Cache-Control "must-revalidate"
7 </Files>
```

When an update is available and the user opens `pseuCo.com`, the request is served from the cache. Afterwards, the browser requests the manifest file, determines that an update is available, and downloads the new version. To inform the user about this, `pseuCo.com` listens to events emitted by the browser to determine the update status, shows a progress bar during the download phase, and shows a notification bar (see Figure 5.2) offering to switch to the new version afterwards. This behaviour is implemented in `repository/src/js/app.js`.

It is important to ensure that the application cache never contains invalid versions of the application, because otherwise, the user's browser will only re-download the corrupted files when an update is released or the user manually clears the application cache<sup>4</sup>.

The HTML Application Cache has an integrated mechanism to ensure that if the application is updated on the server during the download phase, the download process (which would save a mixture of old and new files) fails safely: After the download finishes, the browser fetches the application manifest file again. If it differs from the initial version that started the download process, the downloaded files are assumed to be inconsistent and dropped.

Additionally, the browser makes sure that all requests are served with old versions from the cache, until the page reloads, which causes an *atomic* switch to the new version.

However, there is no integrated mechanism to ensure that all “downloaded” application files are actually fetched from the server, and not from a (possibly outdated) cache. Therefore, it is important to ensure this manually by correctly setting the corresponding HTTP headers. Listing 5.7 shows a suitable configuration file for Apache: It prevents all application files from being cached in the *normal* browser cache<sup>5</sup>, and makes sure that while the application manifest may be cached, the browser must check for updates on the server on any access.

#### Remark

Developing an application which uses the HTML Application Cache has a few possibly confusing differences from normal development:

1. You must change the manifest for changes to be picked up by the browser.
2. You must reload the page twice for changes to show up: Once to trigger the update, and a second time to switch to the new version.
3. Depending on the HTTP server and its configuration, deleting the manifest file does not stop this behaviour: The application cache is cleared only if the request for the manifest results in a **404 not found** error. Any redirects, server errors or other unexpected behaviours cause the browser to keep the application cache, and not download any updates<sup>a</sup>.

<sup>4</sup>Instructions on how to do so are available at `(base URL)/#/help#faq-blankscreen`.

<sup>5</sup>Doing so would be useless, since they will be stored in the – fully separate – HTML Application Cache anyway.

4. Clearing the normal browser cache will not ensure you see the most recent version, as in most browsers, the application cache needs to be deleted separately.
5. While you can change the API path on the debug page (see section 5.1), the browser will block access to any location not listed in the manifest.

To ease development, the build scripts normally *omit* the application manifest, disabling the offline mode. This causes a **404 not found** error for the manifest during testing, which is expected and causes no further issues.

To copy the application manifest and enable offline mode, execute the **appcache** build task, which is part of the **serverside** task the official build server runs.

<sup>a</sup>This is intended behaviour, and makes sure that users can keep using their offline applications when they are in a captive network that redirects all HTTP requests to a login page.

## 5.7 Tasks and the Background Worker

Normal JavaScript applications are single-threaded and use multiple threads only when accessing asynchronous browser APIs, for example for network access. This is fine, since most JavaScript applications only perform small actions, where the only significant delay is network access.

PseuCo.com however performs many long-running, CPU-intensive tasks. Without countermeasures, these would cause the UI to hang for excessive amounts of time. The only way to fully prevent this type of issue is by using multi-threading.

JavaScript was designed as a single-threaded scripting language, and has no memory model. Therefore, the only widely implemented mechanism of using multiple threads in JavaScript, Web Workers<sup>6</sup>, solely use a message-passing interface for communication between the threads.

The Web Worker API defines methods to start a worker (which runs a script file specified when the worker is initialized) and to exchange messages with it, but provides no further assistance to manage tasks that should be executed by the worker.

For pseuCo.com, a new framework was developed to simplify the definition and use of workers that process specific tasks submitted to them by AngularJS applications. It is contained in the **cp.tasks** module, and consists of the files in the `repository/src/js/tasks/` directory. This framework is intentionally kept general so it can be reused by other projects.

### 5.7.1 Features of the Tasks Framework

The tasks framework has the following responsibilities:

1. Manage start-up and termination of the workers.
2. Manage callbacks that hand back computation results and cancellation notifications.
3. Ensure that tasks are executed according to their priority, which can be specified when a task is submitted.

### 5.7.2 Defining a Worker

To define a worker, you need to build a JavaScript file containing everything the worker needs to execute, and place it at any path accessible to the application, preferably **js/worker.js**, relative to the base path of your application. This file must contain, in the following order:

1. all library code that the worker needs
2. a definition of the variable **workerData**, as specified below

<sup>6</sup>see <https://html.spec.whatwg.org/multipage/workers.html#workers>

3. a verbatim copy of the file `repository/src/js/tasks/backgroundWorker.js`

You can define multiple types of workers in one file. They are distinguished by a name, called **type**.

Each worker has one object storing its state. This object will be passed as **workerState** to all functions that may access it.

The **workerData** variable must be an object. Each property of this object defines a worker type (given by the property's key). The value of each such property must be an object with two properties: An initialization function **initialize** which will receive **workerState** as its argument (and may create arbitrary properties in the **workerState** object) and an object **tasks**. For each task the worker can carry out, **tasks** must contain a property whose key is the task name, and whose value is a function, the *task function*.

Each task function receives two arguments: **data** and **workerState**. The **data** is provided by the caller, and **workerState** is the object discussed above. The task function must return an object with at least the following two properties: A **boolean** value **taskCompleted**, and a serializable object **data**.

Each time a task function returns, **data** will be sent to the caller. If **taskCompleted** is **false**, the task will be scheduled to execute again. If the return value contains a property **newPriority** containing a number, the task's priority will be modified to match that number. A task function is free to modify the **data** object to store intermediate results.

To ensure that the worker has a chance to process other tasks with a higher priority, or detect that a task has been cancelled, task functions should regularly return intermediate results (if any) and **taskCompleted**: **false**, waiting to be called again by the framework.

You can find an example of a worker definition in `repository/src/js/worker.js`.

### 5.7.3 Using a Worker

To use a worker, developers can use the API given by the factory **taskManager**. It offers the following methods:

- **getRunningTaskCount** is a zero-argument function that returns the number of tasks in the execution queue.
- **requestWorker(type, crashCallback, sourcePath)** is a function that starts a worker.

**type** is a string indicating the type the worker should have (as specified in the worker definition).

**crashCallback(error)** is a function that will be called when the worker encounters an uncaught exception.

#### Remark

Do not use this mechanism for expected exceptions. Instead, let tasks normally return a value that indicates failure.

If a task crashes, **crashCallback(error)** and the **cancellationCallback()** of the corresponding task will be called.

**sourcePath** is an *optional* string indicating the path to the JavaScript file containing the worker. If it is omitted, **'js/worker.js'** is used.

The return value of this function is an object containing the following properties and methods:

- **id** is an integer identifying the worker instance that was started.
- **terminateWorker(finishCallback)** is a function that requests the worker to be terminated. This function must not be called if the worker is already terminating or has terminated. **finishCallback()** will be called once the worker has terminated.



#### Remark

This does not terminate the worker immediately, but waits for the current execution to return. Cancellation notifications are sent for all pending tasks that could not be executed any more.

This method will not forcefully terminate a worker that got stuck, and there is no method that will. Always ensure that all task functions in your worker terminate regularly.

- `submitTask(taskName, data, resultCallback, cancellationCallback, priority)` submits a task to the worker to be executed. This function must not be called if the worker is terminating or has terminated.

`taskName` is the name of the task to execute, as specified when the worker was defined.

`data` is a serializable object that will be sent to the worker and used to call the task function.

`resultCallback` is a function that processes results returned by the worker. It will be called as `resultCallback(data, taskCompleted)`, where `data` is the data the task function returned, and `taskCompleted` indicates whether the task has finished or is still computing.

`cancellationCallback` is a function that will be called if the task is cancelled for any reason.

`priority` is a number in the range 0-9, where 9 indicates highest and 0 lowest priority. Tasks with a higher priority are executed before any tasks with a lower priority.

The function returns a task id that can be passed to `cancelTask`.

- `cancelTask(taskId)` is a function that requests the specified task to be cancelled. This function must not be called if the worker is terminating or has terminated.

The task will be cancelled as soon as possible. If the task is currently running, it will be stopped the next time it returns. After `cancelTask` has been called, all results from this task will be silently dropped. This ensures that while a task might finish after you tried to cancel it, you can be guaranteed to never receive results from a task that you requested to be cancelled.

#### Remark

The tasks framework executes all callbacks in an AngularJS context: `$apply()` is called automatically, and exceptions are sent to `$exceptionHandler`. There is no need to call `$apply()` manually.

If you do not understand the previous paragraph, you can ignore it safely.

### 5.7.4 Usage of the Tasks Framework for `pseuCo.com`

The tasks framework powers all CPU-intensive computations on `pseuCo.com`. For all of the following, the computation is run by a task in the background worker, and only the results are sent to the main thread to be displayed in the UI:

- all translations (see section 5.8)
- LTS exploration (see section 5.9)
- LTS random tracing
- LTS export
- LTS minimization (see section 5.11)
- parsing of file imports

Listing 5.8: The configuration file `editorConfiguration.js`

---

```

1 angular.module('cp.ui').value('editorConfiguration', {
2   fileTypes: {
3     'ccs': {
4       editor: 'ace-text-editor',
5       translations: [
6         {
7           source: 'ccs',
8           target: 'lts',
9         },
10      ],
11      actions: []
12    },
13    'pseuco': {
14      editor: 'ace-text-editor',
15      translations: [
16        {
17          source: 'pseuco',
18          target: 'ccs',
19        },
20        {
21          source: 'ccs',
22          target: 'lts',
23        }
24      ],
25      actions: []
26    },
27    'lts': {
28      editor: 'lts-editor',
29      translations: [],
30      actions: [{
31        displayIcon: 'glyphicon glyphicon-road',
32        displayName: 'Random path',
33        action: 'traceLts'
34      }, {
35        displayIcon: 'glyphicon glyphicon-export',
36        displayName: 'Export LTS',
37        action: 'exportLts'
38      }
39    ]
40  },
41  fileActions: [
42    {
43      displayIcon: 'glyphicon glyphicon-share',
44      displayName: 'Share this file',
45      action: 'shareFile'
46    }
47  ]
48 });

```

---

## 5.8 Editors, Translations and Actions

The main features of `pseuCo.com` are `pseuCo` and `CCS` file editing, viewing `LTS`, the translations `pseuCo`  $\rightarrow$  `CCS` and `CCS`  $\rightarrow$  `LTS`, and actions like tracing, export or file sharing.

In the code, these features are referenced by three terms:

1. An *editor* is the component responsible for viewing and editing data of a specific file type.
2. A *translation* provides a way to convert data from one file type to another.
3. An *action* describes anything a user can actively initiate. Actions can be specific to a file type or apply to any file.

The interaction between editors, translations and actions is configured by the value `editorConfiguration`, defined in `(repository)/src/js/ui/editorConfiguration.js` and shown in Listing 5.8. More specifically, for each file type, it defines:

- the name of the editor directive that should be used for data of this file type,
- the translations that should be used when editing a file of this type, and
- the actions that apply to data of this file type.

Additionally, it defines actions that apply to files of all types.

#### Remark

There is one crucial difference between the **translations** and **actions** property:

- The **translations** property applies to *files* of a specific type. For example, when editing a `pseuCo` file, only the translations defined for the `pseuco` file type apply. Although there is a translation to `CCS`, no further translations *from* `CSS` occur, unless they are defined *in the* `pseuco` property.
- The **actions** property applies to *data* of the file type specified. For example, the actions defined for `lts` are available when editing a `CCS` file, since the `CCS`  $\rightarrow$  `LTS` translation makes `LTS` data available.

File editing is handled by the `fileEditCtrl` and the partial `edit.html`. `fileEditCtrl` is responsible for setting up the data structures for all editors and actions that are available. Given this data, the `editorManager` and `actionManager` directives are responsible for displaying the editors and action buttons.

There is one core difference between the data stored in files, and the data used by editors, translations and actions: While files store only the minimal amount of information needed to recreate everything that is visible to the user, the in-memory representation of data stores additional temporary information that is generated from the persisted data.

To separate these parts, the in-memory representation is a JavaScript object with two properties: The **core** property stores the part of data that will be persisted (and may have an arbitrary type). The **extended** property stores an object containing all additional in-memory-only data.

For example, this mechanism makes sure that when viewing a `pseuCo` file, the `CCS` term shown never has to be parsed: While the translation `pseuCo`  $\rightarrow$  `CCS` yields the `CCS` string (which is the **core** of the translation), in addition, it contains the parse tree in the **extended** part of the data. The `CCS`  $\rightarrow$  `LTS` translation can use this parse tree instead of the `CCS` string.

### 5.8.1 Translations in the Background

The actual translations are performed by the background worker so that the user interface stays responsive.

This process is initiated by `fileEditCtrl` which submits a **'translate'** task to the background worker. This causes the worker to compute all requested translations one-by-one, returning each translation as an intermediate result as it becomes available.

The parse trees, part of the **extended** data, never leave the background worker<sup>7</sup>. However, they are stored and used internally in the **extended** data in the worker.

## 5.9 LTS Exploration

The result of translations returning a `LTS` often is extremely large or even infinite, rendering the naive approach of computing the full transition system before returning it useless.

Therefore, translations to `LTS` are handled in a special way. The worker only computes the label of the initial state, and stores a function **generateTransitions** that can generate the transitions of this state.

<sup>7</sup>This is crucial: The parse trees are returned by the external library `PseuCoCo`, and are *not* serializable.

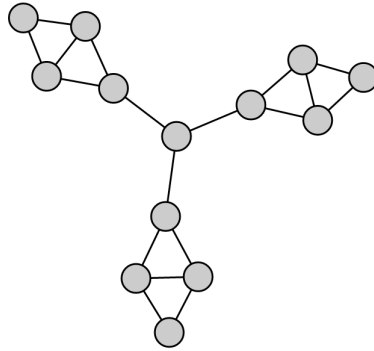


Figure 5.3: Graphic from <http://bl.ocks.org/mbostock/3750558>: D3.js’s built-in force layout

The resulting system is stored permanently in the worker. A version of it *without* `generateTransitions` is returned to the main thread, including a `dataId` that allows to reference the data stored in the worker.

The process of generating the transitions and remaining states is called “exploration”, controlled by the `ltsExplorer` service in the `cp.tools` package. It is defined in `repository/src/js/tools/ltsExplorer.js`. To start the exploration, it sends an `'exploreLts'` request to the worker, causing it to perform a breadth-first search through the transition system, computing each state’s transitions as they are needed. The transitions computed by this search are sent back to `ltsExplorer` in the main thread, where they are added to the local copy of the transition system, ready to be used by the user interface.

This exploration continues until all states are explored or a fixed number of states have been explored. In the latter case, exploration resumes if the user reaches a state with unexplored transitions, or explicitly requests exploration to resume in the UI.

## 5.10 LTS Visualization and Graph Layout

Since LTS are visualized as a graphs, rendering them involves graph layout. There are many well-known graph layout algorithms, but most of them cannot be used for `pseuCo.com` because of the restrictions and considerations that apply for this use case:

- The algorithm should not be overly CPU-intensive, since it is running in JavaScript, possibly on low-end devices.
- The algorithm must allow to add additional states and transitions afterwards, and refine the graph layout. This is because `pseuCo.com` allows users to view large or even infinite graphs by expanding states one by one.

When adding new states, existing states should not move significantly, to ensure the user still can easily recognize them.

- Since `pseuCo.com` is an interactive application (and is even running on touch-enabled devices in many cases), allowing the user to influence the graph layout is both feasible and desirable.

To provide the best graph layout possible in this specific application, `pseuCo.com` uses force-based graph layout. The implementation is based on D3.js’s force layout<sup>8</sup>, but adds some tricks to apply the algorithm to transition systems.

LTS visualization, including graph layout, is implemented in the `ltsEditor` directive, defined in `repository/src/js/ui/directives/ltsEditor.js`.

D3.js’s force layout successfully layouts graphs as shown in Figure 5.3. It does so by applying three kinds of forces:

- A simulated electrical charge of nodes ensures sufficient distance between the nodes.

<sup>8</sup>see <https://github.com/mbostock/d3/wiki/Force-Layout>, demonstration: <http://bl.ocks.org/mbostock/929623>

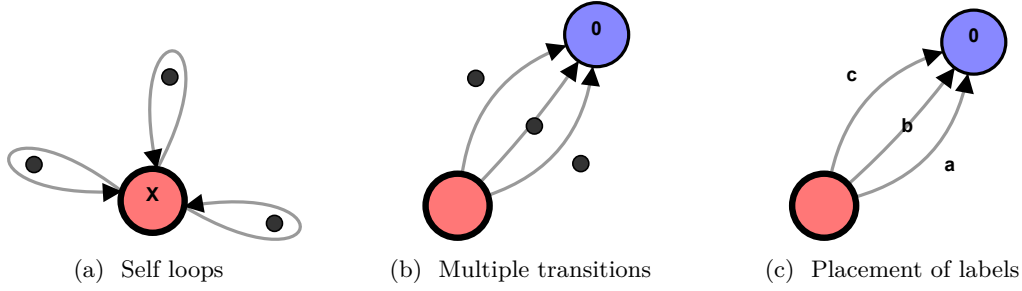


Figure 5.4: Advantages of using control nodes (indicated as dark dots, usually invisible)

- Virtual springs along the edges try to keep them at a specified target length.
- Virtual gravity centres the whole graph.

Additionally, there is a temperature mechanism, slowly increasing a virtual friction-like effect until all nodes have stopped moving.

This approach does not suffice for LTS, which have some differences compared to simple graphs:

1. In LTS, states may have self loops.
2. In LTS, edges are directed, and states can be connected by edges in both directions simultaneously.
3. In LTS, edges are labelled, and states can be connected by arbitrarily many edges, differing in their label.

To account for these changes, `pseuCo.com` introduces one additional, invisible “control node” per transition. This node behaves just like the visible nodes representing states, and serves as the control point for a Bézier curve representing the transition.

This change has three main effects:

1. Self loops automatically repel each other, so they point in different directions and don’t overlap, as shown in Figure 5.4a.
2. Multiple transitions between two states bend slightly, so all of them are visible, as shown in Figure 5.4b.
3. Since the control nodes repel each other, they indicate places where short labels can be placed without overlapping, as shown in Figure 5.4c.

This approach often produces decent results, like the one shown in Figure 5.5. In some cases, nodes get “stuck” in a suboptimal position between other nodes, as shown in Figure 5.6. In these cases, users can drag-and-drop nodes to move them manually. This works well in practice: Because of the virtual springs, just dragging one misplaced node moves its neighbours as well, often fully untangling the graph with only a single manual intervention.

## 5.11 LTS Minimization

`PseuCo.com` can minimize labelled transition systems, computing the smallest transition system that still is observation congruent (see Definition 6) to the original transition system.

There are many well-known minimization algorithms[2][4][5], but they assume to be run on fully-explored transition systems. `PseuCo.com`, however, has the goal to provide users with tools that work as far as possible with extremely large or even infinite transition systems, requiring modification to the classic approach.

The design goals of such a minimization feature are:

- If the system is fully explored, return the (*unique* [2]) minimal observation congruent system.

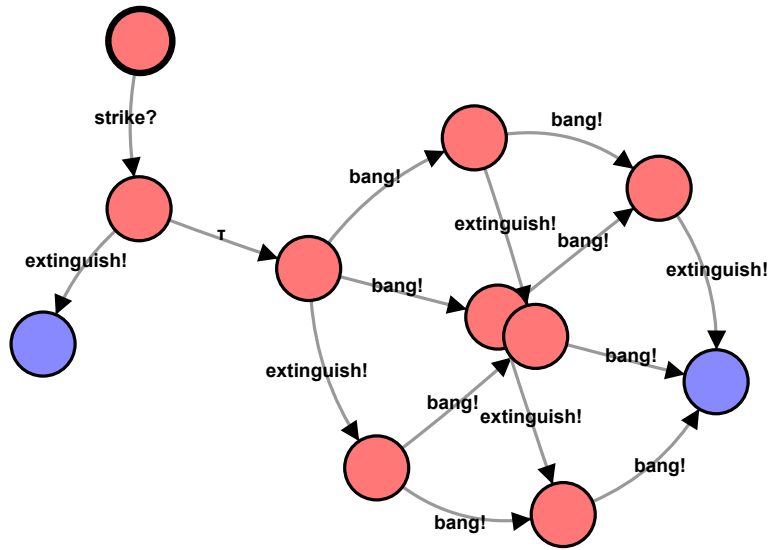


Figure 5.5: A result of graph layout with control nodes

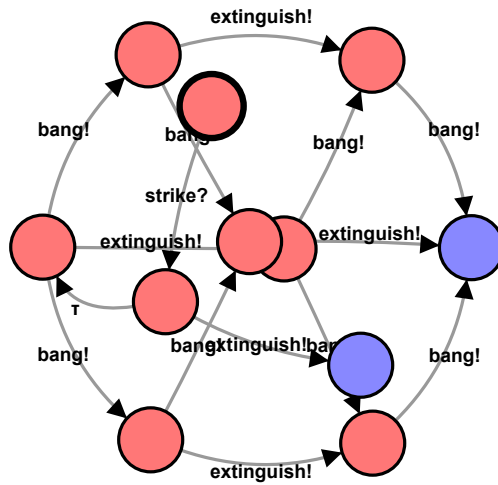


Figure 5.6: Graph layout with control nodes sometime produces bad results

- If the system is not fully explored, provide a safe approximation of the minimal system: If in doubt, assume states cannot be unified.
- Pre-compute all data that would be needed for full minimization, so it can be used if the user requests minimization before exploration finishes.

The resulting algorithm is highly similar to the one described in [2], and works in four distinct phases:

1. Weak transitions are computed.
2. The relational coarsest partitioning is computed, using the algorithm from [4]. Since weak transitions have been added to the transition system, the result is *weakly* bisimilar[2].
3. The algorithm from [2] to minimize the number of transitions is run.
4. To ensure observation congruence, a  $\tau$  self loop is added to the initial state if necessary.

However, the actual algorithm contains subtle differences to this simplified version to cope with partially unexplored systems and to increase performance. The details are given in the following sections.

### 5.11.1 Computing Weak Transitions

The minimization algorithm needs access to the weak transitions in the transition system. They are pre-computed by an exploration mechanism (“weak exploration”), similar to the mechanism described in section 5.9.

Weak exploration runs in parallel to the normal exploration, but is slightly delayed. This ensures that a user can quickly see the initial fragment of the transition system, but as soon as exploration got ahead of the user, weak transitions are computed as fast as possible to allow minimization to run.

The core function of this algorithm, `weakProcessState()`, is called automatically in a breadth-first manner by the exploration mechanism – its only remaining responsibility is to update the weak transition information to incorporate the transitions starting in the state it processes.

To allow for a simpler and faster implementation, weak  $\tau$ -transitions are stored as `weakConnections`, separate from all other transitions which are stored in `weakTransitions`. In both cases, transitions are stored in both their start and end state.

Let  $LTS = (S, \rightarrow, s_0)$  be a transition system and  $S_{\text{reach}}$  be the set of reachable states. After having been run in a breadth-first manner for all reachable states, `weakProcessState()` guarantees the following:

1. For each state  $s \in S_{\text{reach}}$ , `states[s].weakConnections.pre` is an array representing the following set:

$$\left\{ s' \in S_{\text{reach}} \mid s' \xRightarrow{\tau} s \right\}$$

2. For each state  $s \in S_{\text{reach}}$ , `states[s].weakConnections.post` is an array representing the following set:

$$\left\{ s' \in S_{\text{reach}} \mid s \xRightarrow{\tau} s' \right\}$$

3. For each state  $s \in S_{\text{reach}}$ , `states[s].weakTransitions.outgoing` is an array representing the following set:

$$\left\{ (s, \alpha, s') \mid \alpha \neq \tau \wedge s \xRightarrow{\alpha} s' \wedge s' \in S_{\text{reach}} \right\}$$

4. For each state  $s \in S_{\text{reach}}$ , `states[s].weakTransitions.incoming` is an array representing the following set:

$$\left\{ (s', \alpha, s) \mid \alpha \neq \tau \wedge s' \xRightarrow{\alpha} s \wedge s' \in S_{\text{reach}} \right\}$$

Listing 5.9 shows a simplified PseudoCode version of `weakProcessState`, while Listing B.2 shows the actual JavaScript implementation. The behaviour of this procedure is illustrated in Figure 5.7. The PseudoCode version assumes that the initial state is contained in its own `weakConnections.pre` and `weakConnections.post` sets by default.

When **weakProcessState** is called on a state, it iterates over all outgoing (normal) transitions  $s_1 \xrightarrow{\alpha} s_2$ . After adding the transition  $s_2 \xRightarrow{\tau} s_2$  to the corresponding **weakConnections** sets, the algorithm distinguishes between  $\tau$  transitions and non- $\tau$  transitions:

- If the transition is *not* a  $\tau$ -transition ( $\alpha \neq \tau$ ), the algorithm adds all weak transitions  $s_3 \xRightarrow{\alpha} s_4$  it can find *using the already computed data*:  $s_3$  (**prestate**) iterates over all states where  $s_3 \xRightarrow{\tau} s_1$  is already known (stored in the set **weakConnections.pre** of  $s_1$ ), and  $s_4$  (**poststate**) iterates over all states where  $s_2 \xRightarrow{\tau} s_4$  is already known (stored in the set **weakConnections.post** of  $s_2$ ).

For example, in Figure 5.7c, the transition  $B \xrightarrow{a} C$  is processed. This adds the transition  $A \xRightarrow{a} C$  (since  $A \xRightarrow{\tau} B$  and  $C \xRightarrow{\tau} C$ ) and  $B \xrightarrow{a} C$  (since  $B \xRightarrow{\tau} B$  and  $C \xRightarrow{\tau} C$ ). However, no weak transitions ending in state  $D$  are found, since the weak connection  $C \xRightarrow{\tau} D$  has not been found yet.

- If the transition *is* a  $\tau$ -transition ( $\alpha = \tau$ ), the algorithm first adds all new weak connections  $s_3 \xRightarrow{\tau} s_4$ :  $s_3$  (**prestate**) iterates over all states where  $s_3 \xRightarrow{\tau} s_1$  is already known (stored in the set **weakConnections.pre** of  $s_1$ ), and  $s_4$  (**poststate**) iterates over all states where  $s_2 \xRightarrow{\tau} s_4$  is already known (stored in the set **weakConnections.post** of  $s_2$ ).

Since adding a weak connection might give rise to new weak transitions (by prolonging already computed ones), for every weak connection  $s_3 \xRightarrow{\tau} s_4$  the algorithm adds, it searches for every already computed weak transition  $s_5 \xRightarrow{\beta} s_3$  or  $s_4 \xRightarrow{\beta} s_6$  (by iterating  $s_5$  over **weakTransitions.incoming** of  $s_3$  and  $s_6$  over **weakTransitions.outgoing** of  $s_4$ ), and adds the new transition  $s_5 \xRightarrow{\beta} s_4$  or  $s_3 \xRightarrow{\beta} s_6$ , respectively.

#### Remark

The second part of this search (extending the *beginning* of weak transitions) is relevant in cyclical graphs only. Figure 5.8d shows an example where this search yields a result.

For example, in Figure 5.7d, the weak connection  $C \xRightarrow{\tau} D$  is added since  $C \xRightarrow{\tau} C$  and  $D \xRightarrow{\tau} D$ . This causes two searches:

1. A search for all weak transitions ending in  $C$  finds  $A \xRightarrow{a} C$  and  $B \xRightarrow{a} C$ . This causes the weak transitions  $A \xRightarrow{a} D$  and  $B \xRightarrow{a} D$  to be added.
2. A search for all weak transitions starting in  $D$  does not find any transitions.

## Correctness of Weak Transition Computation

Correctness of this algorithm – assuming it runs to completion – can be shown in two steps:

**Lemma 1.** *The sets in **weakConnections** are computed correctly, as specified in subsection 5.11.1.*

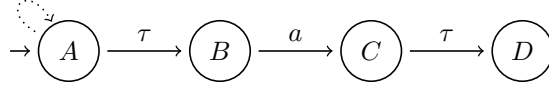
*Proof. Soundness* can be shown by induction over the program execution: The initial state (only the initial state has a  $\tau$  self loop) is sound. Whenever a transition  $s \xRightarrow{\tau} t$  is added to **weakConnections**, it either has the form  $s \xRightarrow{\tau} s$  (which is always a valid weak transition), or it is added as the combination of two weak transitions  $s \xRightarrow{\tau} s'$  and  $s' \xRightarrow{\tau} t$  that have been added before, both sound by induction.

For *completeness* we need to show that every weak  $\tau$ -transition  $s \xRightarrow{\tau} t$  is added to **weakConnections** during weak exploration if  $s$  and  $t$  are both reachable.

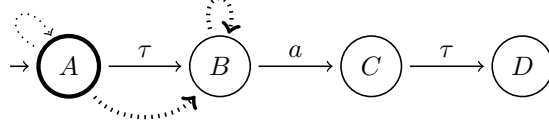
Each such transition is by definition based on a sequence of  $n \in \mathbb{N}$   $\tau$ -transitions  $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_{n-1} \xrightarrow{\tau} t$  with  $n - 1$  intermediate states. We use induction to show the following version of the claim: For any  $n \in \mathbb{N}$ , any weak transition  $s \xRightarrow{\tau} t$  based on a sequence of  $n$  intermediate states has been added to **weakConnections** after **weakExploreState** has been called with  $s$  and the intermediate states  $s_1, s_2, \dots, s_{n-1}$ .

Let  $n \in \mathbb{N}$  be a fixed natural number, and the claim be valid for any  $m \in \mathbb{N}$  with  $m < n$ . Case distinction on  $n$ :

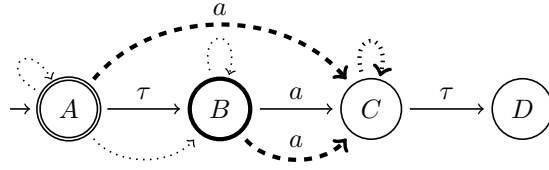




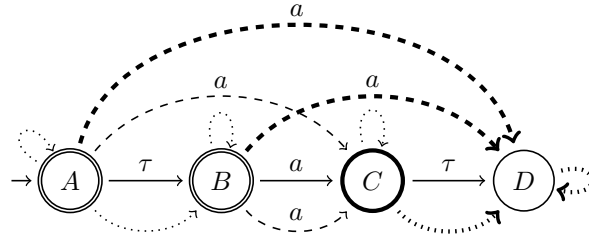
(a) The original transition system



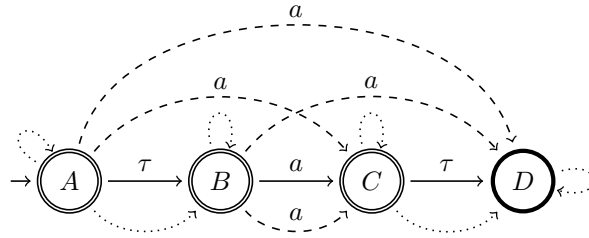
(b) `weakProcessState(A)`



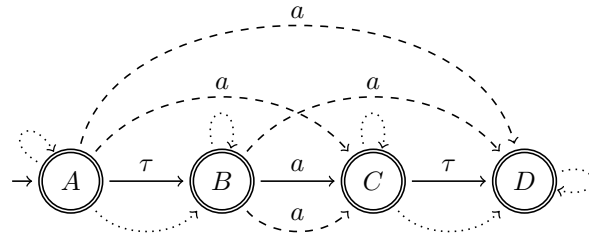
(c) `weakProcessState(B)`



(d) `weakProcessState(C)`



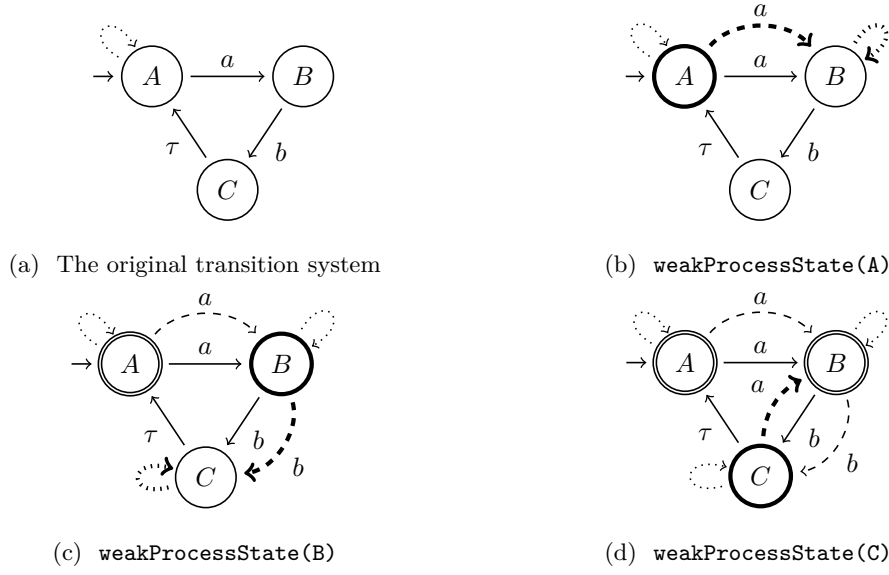
(e) `weakProcessState(D)`



(f) Result of weak exploration

Dashed arrows represent an element of **weakTransitions**, while dotted arrows represent an element of **weakConnections**. States with double borders are already weakly explored. Thick lines indicate the state that is currently being processed, and the weak transitions and connections that are added as a result.

Figure 5.7: How weak exploration processes a transition system



For an explanation of the different types of lines, see Figure 5.7.

Figure 5.8: How weak exploration processes a *cyclic* transition system

Listing 5.9: PseudoCode for `weakProcessState()`

---

```

1 procedure weakProcessState(state):
2   for each transition from state:
3     add transition.target to its own weakConnections.pre and weakConnections.post
4     ↳ sets
5
6   if transition is a tau-transition:
7     for each prestate in state.weakConnections.pre, poststate in transition:
8       ↳ target.weakConnections.post:
9         add poststate to the set prestate.weakConnections.post
10        add prestate to the set poststate.weakConnections.pre.add
11
12    if poststate and prestate have not been in these sets before:
13      // add new transitions that consist of an old transition and the
14      ↳ newly added connection
15      for each transition in prestate.weakTransitions.incoming:
16        add a longer version of transition ending in poststate to both
17        ↳ corresponding sets
18      end for
19      for each transition in poststate.weakTransitions.outgoing:
20        add a longer version of transition starting in prestate to both
21        ↳ corresponding sets
22      end for
23    end if
24  end for
25
26  else: // transition is not a tau-transition
27    for each prestate in state.weakConnections.pre, poststate in transition:
28      ↳ target.weakConnections.post:
29        add prestate to the set poststate.weakConnections.pre
30        add poststate to the set prestate.weakConnections.post
31    end for
32  end if
33 end for
34
35  remember that state has been weak explored
36 end procedure

```

---

1.  $n = 0$

This means that  $s = t$ , and the transition has the form  $s \xRightarrow{\tau} s$ . Since  $s$  must be reachable, it is either initial or has an incoming transition from a reachable state  $s'$ . If  $s$  is initial,  $s \xRightarrow{\tau} s$  is in **weakConnections** by default. Otherwise, since  $s'$  is reachable, **weakProcessState** must have been called on it (based on the breadth-first order), causing  $s \xRightarrow{\tau} s$  to be added to **weakConnections**.

2.  $n > 0$

This means that the chain  $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_{n-1} \xrightarrow{\tau} t$  contains at least one intermediate state. We know that **weakProcessState** must be called on  $s, s_1, s_2, \dots, s_{n-1}$  since they are all reachable ( $s$  is reachable by assumption, all others are because they are reachable from  $s$ ). Let  $s'$  be the one where **weakProcessState** is called *last*, and  $s''$  be the next one in the chain.

We know that **weakProcessState** has been called on all other states. Since the weak transitions  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  are based on chains shorter than  $n$ , we know by induction they already must have been added to the corresponding **weakConnections** sets.

Then, **weakProcessState** must find  $s \xRightarrow{\tau} t$  during its search and add it.

All cases have been considered. ■

**Lemma 2.** *The sets in **weakTransitions** are computed correctly, as specified in subsection 5.11.1.*

*Proof.* *Soundness* can be shown by induction over the program execution: The initial state (no weak transitions) is sound. Whenever any transition  $s \xRightarrow{\alpha} t$  is added to **weakTransitions**, it is caused by any of the following combinations:

- $s \xRightarrow{\tau} s' \xrightarrow{\alpha} s'' \xRightarrow{\tau} t$
- $s \xRightarrow{\alpha} s' \xRightarrow{\tau} t$
- $s \xRightarrow{\tau} s' \xRightarrow{\alpha} t$

All of these must be valid by induction, and cause  $s \xRightarrow{\alpha} t$  to be a valid weak transition by definition.

For *completeness* we need to show that every weak transition  $s \xRightarrow{\alpha} t$  with  $\alpha \neq \tau$  is actually added to **weakTransitions** during weak exploration if  $s$  and  $t$  are both reachable. This is shown similarly as in the proof of Lemma 1, by induction on the length of the chain of transitions  $s \xrightarrow{\tau} \dots \xrightarrow{\alpha} \dots \xrightarrow{\tau} t$  backing the weak transition.

By definition each such transition can be decomposed into a sequence of transitions  $s \xRightarrow{\tau} s' \xrightarrow{\alpha} s'' \xRightarrow{\tau} t$ . We know that **weakProcessState** must be called on  $s, s'$  and  $s''$  since they are all reachable ( $s$  is reachable by assumption,  $s'$  and  $s''$  because they are reachable from  $s$ ).

Case distinction on whether  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  have both been discovered before **weakProcessState** searches for weak transitions based on  $s' \xrightarrow{\alpha} s''$ .

1. Assume  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  have both been discovered.

Then, this call to **weakProcessState** will add  $s \xrightarrow{\alpha} t$ .

This serves as the base case for the induction, with  $s = s'$  and  $s'' = t$ .

2. Assume not both  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  have been discovered yet. This is only possible if  $s \neq s'$  or  $s'' \neq t$ , otherwise, both  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  would have been added before – during creation of the LTS for initial states, when  $s'$  was discovered by weak exploration (which must have occurred beforehand, due to the breadth-first order), or at the beginning of the iteration in **weakProcessState**.

When **weakProcessState** is called on  $s'$ , the weak transition  $s' \xRightarrow{\alpha} s''$  is added, because  $s' \xRightarrow{\tau} s'$  has been added before (because  $s'$  is initial, or was discovered during the breadth-first traversal) and  $s'' \xRightarrow{\tau} s''$  is added by this run of **weakProcessState**.

Further case distinction on the order in which  $s \xRightarrow{\tau} s'$  and  $s'' \xRightarrow{\tau} t$  are discovered:

(a) Assume  $s \xRightarrow{\tau} s'$  is discovered after  $s'' \xRightarrow{\tau} t$ .

When  $s \xRightarrow{\tau} s'$  is added, the resulting iteration will use  $s' \xRightarrow{\alpha} t$ , added by then by induction, to add  $s \xRightarrow{\alpha} t$ .

Listing 5.10: PseudoCode for `stateIsFullyWeakExplored()`

---

```

1 procedure stateIsWeakExplored(state):
2   return whether state has been weak explored
3 end procedure
4
5 procedure stateIsWeakConnectionExplored(state):
6   if the state has been marked as weak connection explored: return true
7
8   if stateIsWeakExplored(state) && every state in weakConnections.post fulfills
9     ↳ stateIsWeakExplored:
10    mark this state as weak connection explored
11    return true
12   else:
13    return false
14   end if
15 end procedure
16
17 procedure stateIsFullyWeakExplored(state):
18   if (weak) exploration of the LTS is finished already: return true
19   if state has been marked as fully weak explored: return true
20
21   if not stateIsWeakConnectionExplored(state): return false
22
23   if (every target state of a transition in state.weakTransitions.outgoing fulfills
24     ↳ stateIsWeakConnectionExplored):
25    mark state as fully weak explored
26    return true
27   else:
28    return false
29   end if
30 end procedure

```

---

(b) Assume  $s'' \xrightarrow{\tau} t$  is discovered after  $s \xrightarrow{\tau} s'$ .

When  $s'' \xrightarrow{\tau} t$  is added, the resulting iteration will use  $s \xrightarrow{\alpha} s''$ , added by then by induction, to add  $s \xrightarrow{\alpha} t$ .

All cases have been considered. ■

### When Weak Exploration Did Not Finish

If the breadth-first traversal of the graph is interrupted at any point, the initial fragment of the graph is likely to already contain all weak transitions, allowing minimization to be run on it.

For any state, all weak transitions must have been found if weak exploration has run far enough that all paths starting in this state must pass at least two non- $\tau$  transitions before reaching a state that has not been visited by weak exploration yet.

Whether this is the case can be determined by the algorithm given in Listing B.3, a PseudoCode version is given in Listing 5.10. If this algorithm returns **true**, the sets **weakConnections** and **weakTransitions** will not be changed further during weak exploration.

**PseuCo.com** makes direct use of this property. As soon as the user requests the minimization result, weak exploration is paused, and the system enters the partitioning phase. While the resulting system may not be minimal, this ensures a prompt response to the user's request, since weak exploration has the highest computational cost in the minimization algorithm.

### 5.11.2 Partitioning, Transition Minimization & Generation of the Resulting System

The actual partitioning is similar to the algorithm in [4], an alternative description of which can be found as the “naive version” of the relational coarsest partitioning algorithm in [5]. There is one key difference: Initially, not all states start in one partition. Instead, states which are not fully weakly explored (as

explained in subsection 5.11.1) are placed in a separate one-state partition each, while all fully weakly explored states start in a single, separate partition.

This ensures that states cannot be falsely considered equal because weak transitions have not been fully computed yet. Consequently, this means that the resulting system may not be minimal. However, if weak exploration finished, all states start in the same partition, and the result is guaranteed to be minimal.

The full implementation of the minimization algorithm (as a background worker task) can be found in Listing B.4. A simplified PseudoCode version of it is shown in Listing 5.11.

**To ensure the resulting system has the minimal number of transitions,** the algorithm must remove superfluous transitions. For example, in the transition system in Figure 5.9, the transition  $X \xrightarrow{a} Z$  can be removed, because  $X \xrightarrow{a} Y \xrightarrow{\tau} Z$ , and therefore  $X \xRightarrow{a} Z$ .

[2] describes an algorithm to solve this issue, which removes a transition  $s_1 \xrightarrow{a} s_2$  if there is a state  $s_3$  such that:

$$(s_1 \xrightarrow{a} s_3 \wedge s_3 \xrightarrow{\tau} s_2) \vee (s_1 \xrightarrow{\tau} s_3 \wedge s_3 \xrightarrow{a} s_2) \quad (5.1)$$

This approach only works in a transition system which is *saturated*, meaning that  $s_1 \xRightarrow{\alpha} s_2$  implies  $s_1 \xrightarrow{\alpha} s_2$  for any states  $s_1, s_2$  and for any action  $\alpha$ .

`PseuCo.com` uses an adapted version of this algorithm, integrated into the construction of the new transition system. To avoid having to compute the weak transitions, it argues over the (weak) transitions in the *original* transition system.

Let  $LTS = (S, \rightarrow, s_0)$  be the initial transition system. Let  $S'$  be the set of states of the new, minimized transition system, and  $\rho : S \rightarrow S'$  a function mapping old states to the corresponding new states. For any weak transition  $s_1 \xRightarrow{\alpha} s_2$  in the old transition system,  $s'_1 := \rho(s_1)$  and  $s'_2 := \rho(s_2)$  are computed, and the corresponding transition  $s'_1 \xrightarrow{\alpha} s'_2$  is added to the transition system unless any of the following conditions hold:

1.  $\alpha = \tau \wedge \rho(s_1) = \rho(s_2)$

This prevents  $\tau$  self loops from being added to the result.

2. There are  $s_a, s_b, s_c, s_d$ , such that all of the following conditions hold:

- (a)  $s_a \xrightarrow{\tau} s_b \xrightarrow{\alpha} s_c \xrightarrow{\tau} s_d$

- (b)  $\rho(s_a) = s'_1$

- (c)  $\neg(\alpha = \tau \wedge \rho(s_b) = \rho(s_c))$

- (d)  $\rho(s_d) = s'_2$

- (e)  $\rho(s_a) \neq \rho(s_b) \vee \rho(s_c) \neq \rho(s_d)$

This condition describes the same idea as Equation 5.1, but is easier to compute: It only reasons over weak transitions in the original system (which have already been computed for the minimization).

Condition 2a ensures that the states build a path as in Equation 5.1. Condition 2b ensures that the starting state  $s_1$  actually corresponds to the correct starting state in the minimized system. Condition 2c ensures that  $s_b \xrightarrow{\alpha} s_c$  does not correspond to a  $\tau$  self loop in the new system, which could be removed. Condition 2d ensures that the ending state corresponds to the correct ending state in the new system. Finally, condition 2e ensures that either  $s_a \xrightarrow{\tau} s_b$  or  $s_c \xrightarrow{\tau} s_d$  (or both) correspond to an actual  $\tau$  step in the minimized LTS.

While in `pseuCo.com` weak transitions do not have to be fully computed to start a minimization, leaving out weak transitions can only make this condition false-negative, resulting in a system with *too many* transitions. As explained in the beginning of section 5.11, this is an accepted behaviour. When all weak transitions have been computed, minimization is guaranteed to be complete.

Correctness of this approach follows from the correctness of the corresponding algorithm in [2]. The only differences are:

- This approach reduces looking at weak transitions in the new, minimized system to looking at weak transitions in the original system.

Listing 5.11: PseudoCode for LTS minimization

---

```

1 procedure getSplit(partition, splitter):
2   if partition ∩ splitter ≠ ∅ && partition \ splitter ≠ ∅:
3     return p1: partition ∩ splitter ≠ ∅, p2: partition \ splitter ≠ ∅
4   else:
5     return false
6   end if:
7 end procedure
8
9 procedure minimizeLts(states, initialState):
10  var initialPartition = ∅
11  var partitions = {}
12
13  // set up initial partition
14  for each reachable state in states: // remove unreachable states
15    if stateIsFullyWeakExplored(state):
16      initialPartition.add(state)
17    else:
18      partitions.add({state})
19    end if
20  end for
21
22  // add initialPartition to partitions if it is not empty
23  if (initialPartition.length > 0): partitions.add(initialPartition)
24
25  // split partitions
26  repeat until partitions does not change anymore:
27    for all pairs of partitions (partition1, partition2):
28      if partition1 was already split in this iteration: continue
29
30      for all transition labels (including tau):
31        // compute using weakConnections for tau, and weakTransitions otherwise:
32        var splitter = set of all states having a transition with this label to a
33          ↳ state in partition2
34
35        if getSplit(partition1, splitter) returns a split:
36          replace partition1 by the partitions getSplit returned
37          continue with the next pair
38        end if
39      end for
40    end for
41  end repeat
42
43  var newStates = ∅
44  for each partition in partitions:
45    add a new state for it to newStates
46
47    for each outgoing weak transition or tau-connection from a state in partition:
48      if it is a tau-connection to itself: continue // tau self loops are
49        ↳ superfluous
50      build a corresponding transition to the new state
51
52      if (transition can be reproduced by following a longer chain of weak
53        ↳ connections and transitions):
54        ignore the new transition // would be superfluous
55      else:
56        add the transition to newState
57      end if
58    end for
59  end for
60
61  var newInitialState = the state corresponding to the partition with initialState
62
63  if there is a (strong) tau-transition from initialState to another state that ended
64    ↳ up in the same partition:
65    add a tau self loop to newInitialState
66  end if
67
68  return newStates, newInitialState
69 end procedure

```

---

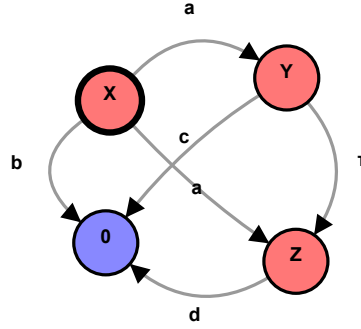


Figure 5.9: While having the minimal number of states, this system is not minimal in the number of transitions.

- Instead of only removing a transition in the cases given in Equation 5.1, this algorithm additionally allows a transition  $s_1 \xrightarrow{a} s_2$  to be removed if there are states  $s_3, s_4$  such that  $s_1 \xrightarrow{\tau} s_3 \xrightarrow{a} s_4 \xrightarrow{\tau} s_2$ .

Correctness follows by saturation:  $s_1 \xrightarrow{\tau} s_3 \xrightarrow{a} s_4$  implies  $s_1 \xrightarrow{a} s_4$ , allowing the transition to be removed based on the first disjunct in Equation 5.1.

**To ensure the result is observation congruent** (in addition to being weakly bisimilar), a  $\tau$  self loop is added to the new initial state if and only if the initial system contained a strong  $\tau$ -transition from its initial state to another state *which ends up in the partition that contains the initial state*.

## 5.12 The Server Component of `pseuCo.com`

`PseuCo.com` mostly is a JavaScript-application, running client-side in the browser. This only requires a HTTP server to deliver the (static) application files. However, there are two features that use an active server component:

1. To create new files, users can choose start with “template files” that can be updated without updating the whole application.
2. Users can create short links to files. Opening such a link opens the `pseuCo.com` user interface and directly navigates to the file that was shared.

By default, `pseuCo.com` communicates with the server at `[base URL]/api/`. The server is implemented to listen at port 9128, and `[repository]/server/server.htaccess` configures Apache to forward the requests to `[base URL]/api/` to this port, acting as a reverse proxy.

The server is implemented in JavaScript, and designed to be run with `node.js`. All server code is located in `[repository]/server/server.js`.

The server implements the following API calls:

1. `[API base URL]/paste/add` allows to submit files and retrieve a sharing URL for it. Requests to this path must be POST requests, having a JSON-encoded body, containing an object with the following properties:
  - (a) `file` must be an object representing the file to be shared. It must have the properties `type` and `content` and may have a `name`.
  - (b) `sharingAgreementVersion` must be a number, representing the version number of the legal notice the user agreed to before uploading the file. The server will reject uploads not having a recent enough `sharingAgreementVersion`.
  - (c) `temporary` is an optional value. If it evaluates to `true`, the server will only store the uploaded file for a short amount of time.

The server will store the submitted file – either permanently in the `repository`/server/data/paste/ directory, or in memory for temporary shares – and return a JSON-encoded response. It has the following properties:

- (a) `id` is the identifier of the file, as chosen by the server.
  - (b) `url` is a URL which can be given to other users to view this file.
  - (c) `temporary: true` will be set for temporary shares. Otherwise, this property is omitted.
2. `API base URL`/paste/get allows to get files that have been submitted to the server. Requests to this path must be GET requests, and have one parameter `id`, which is the identifier for the shared file.

The server will then return a JSON object having the following properties:

- (a) `file` will be the file object that was submitted to the server when the share was created.
  - (b) `temporary: true` will be given for temporary shares. Otherwise, this property is omitted.
3. `API base URL`/templates/get allows to get access to the templates that are stored on the server for a specific file type. Requests to this path must be GET requests, and have a single parameter `type`: a string representing the file type for which the templates are requested.

The server will return a JSON object having an array `templates` as its single property. It is an array of templates, each being an object having the following properties:

- (a) `name` is a name for the template
- (b) `description` is a description string for the template, which may contain basic HTML formatting including links.
- (c) `content` is the content of the template. If the user selects the template, `content` should be used to initialize the contents of the file that is created.

The server features a banning system to prevent malicious users from causing excessive system load on the server.

The server sets `Cache-Control` headers to ensure that the responses it gives may not be cached, except for successful calls to `API base URL`/paste/get.

Since the clients interacting with this server are sandboxed inside browsers, the server has to conform to the Cross-Origin Resource Sharing standard<sup>9</sup> to ensure clients are allowed access to it:

1. To ensure not just clients running in the same domain can access the server, it sends the header `Access-Control-Allow-Origin: *`.
2. When a client sends an `OPTIONS` request, the server responds with `Access-Control-Allow-Methods: GET, POST`, `Access-Control-Max-Age: 3600`, and mirrors all values listed in the `Access-Control-Request-Headers` header of the request in the `Access-Control-Allow-Headers` header of its response. This indicates to the client's browser that it is allowed to send arbitrary `GET` and `POST` requests.

## 5.13 A Build Server for `pseuCo.com`

`PseuCo.com` can be built on a remote server – usually on the server that hosts the repository and serves the resulting page publicly. The exemplary script in Listing 5.12 automates this process. It accepts the branch to build as its single command line argument and performs the following actions:

1. If the branch specified is `master` – which contains the public server part – stop the server.
2. Check out the specified branch from the Git repository.
3. Install missing npm packages, and remove superfluous ones.

---

<sup>9</sup>see <http://www.w3.org/TR/access-control/>



Listing 5.12: A build script for server-side building

---

```
1 #!/bin/bash
2
3 GIT_REPO=$HOME/repositories/concurrent-programming-web.git/
4 CHECKOUT_DIR=/var/www/virtual/fefrei/apps/concurrent-programming-web/$1/
5
6 echo Building branch $1.
7 echo Preparing...
8 mkdir -p $CHECKOUT_DIR
9
10 if [ $1 == "master" ]
11 then
12     echo Stopping server...
13     svc -d /home/fefrei/service/cpw-server/
14 fi
15
16 echo Checking out ...
17 GIT_WORK_TREE=$CHECKOUT_DIR git checkout -f --quiet $1
18 GIT_VERSION='git rev-parse $1'
19 cd $CHECKOUT_DIR
20
21 echo Installing NPM packages...
22 npm install
23
24 echo Pruning NPM packages...
25 npm prune
26
27 if [ $1 == "master" ]
28 then
29     echo Starting server...
30     svc -u /home/fefrei/service/cpw-server/
31 fi
32
33 echo Writing version information...
34 echo $1 [${GIT_VERSION}] > version.txt
35
36 echo Deploying...
37 grunt serverside
38
39 exit
```

---

4. Start the server if it was stopped earlier.
5. Run the `serverside` task of the build script, which performs the normal build and generates a configuration file for the HTML Application Cache – see section 5.6.

# The Future of `pseuCo.com`

`PseuCo.com` has been designed to be easily extensible, featuring many self-contained, reusable modules. This section points out some possible extensions, and evaluates how well the existing infrastructure supports them.

## 6.1 Model Checking

A useful potential extension is model checking, allowing users to verify their transition systems against a LTL[6] or CTL[3] property.

The infrastructure of `pseuCo.com` is well-suited for this extension:

1. The user interface is prepared for this addition: Model checking could be implemented as an action (see section 5.8), similar to tracing and LTS export.
2. Model checking is computationally expensive. With the tasks API (see section 5.7), there already is an extensive infrastructure for background computation.
3. `PseuCo.com` supports on-demand exploration of large or infinite transition systems. This integrates well with model checking, which often does not require access to the full transition system, for example for evaluating existentially quantified properties.

## 6.2 Additional Input Formats

Another potential kind of extension is the addition of another input language (similar to the current support for `pseuCo`).

The infrastructure of `pseuCo.com` is well-suited for this type of extension:

1. File management and editing is parametrized, allowing to easily add new file types. Just declaring the new file type in `repository/src/js/files/files.js` and configuring the translations and actions in `repository/src/js/ui/editorConfiguration.js` is enough to add the user interface elements to create, manage and edit files of this new file type.

The additional translations can be implemented in the background worker (see section 5.7).

2. While LTS exploration currently evaluates CCS terms (since the only way to create LTS files – apart from importing them – is the translation  $\text{CCS} \rightarrow \text{LTS}$ ), this process is encapsulated in the `generateTransitions` function. A translation from any other input to LTS can easily create transition systems that work with the current code by returning transition systems with a new `generateTransitions` function.

## 6.3 New Translations for Existing File Types

`PseuCo.com` can be extended by new translations for existing file types – for example a translation from `pseuCo` code to Petri nets.

Depending on the kind of integration, the `pseuCo.com` infrastructure would need to be extended:

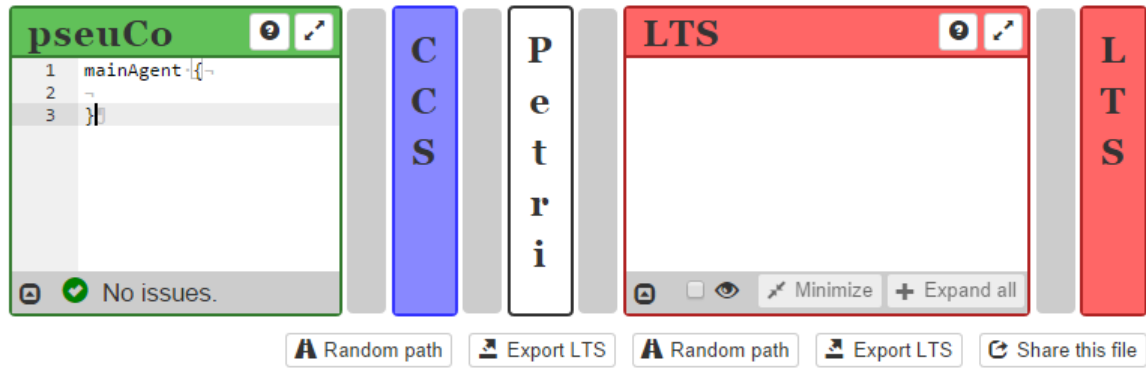


Figure 6.1: Adding additional translations doesn’t scale and causes confusing UI

1. As long as the new translations are just an extension to the existing ones, the current infrastructure suffices.
2. There is no support for the user to select a “translation chain”. If in addition to the old  $\text{pseuCo} \rightarrow \text{CCS} \rightarrow \text{LTS}$  chain, a new translation chain  $\text{pseuCo} \rightarrow \text{Petri} \rightarrow \text{LTS}$  was be added, the file editing interface for  $\text{pseuCo}$  files would look as illustrated in Figure 6.1.

Adding such a selection capability requires changes to the editor configuration specification format in `repository/src/js/ui/editorConfiguration.js` and to the controller `fileEditCtrl`.

# Appendix A

## Definitions for LTS

In the following, let  $Act$  be an arbitrary set with  $\tau \in Act$ .  $\tau$  is called *internal action*.

**Definition 1** (LTS). A *Labelled Transition System (LTS)* over  $Act$  is a triple  $(S, \rightarrow, s_0)$  where  $S$  is a set of states,  $s_0 \in S$  the initial state, and  $\rightarrow \subseteq S \times Act \times S$  the transition relation.

We will write  $s \xrightarrow{\alpha} t$  as a shorthand for  $(s, \alpha, t) \in \rightarrow$ .

**Definition 2** ( $\rightsquigarrow$ ). For any LTS  $(S, \rightarrow, s_0)$  over  $Act$ ,  $s, s' \in S$  and  $\sigma \in Act^*$ , let  $s \rightsquigarrow^\sigma s'$  be defined inductively on  $\sigma$ :

$$\begin{aligned} s \rightsquigarrow^\varepsilon s' &: \Leftrightarrow s = s' \\ s \rightsquigarrow^{\sigma' \alpha} s' &: \Leftrightarrow \exists s'' : s \rightsquigarrow^{\sigma'} s'' \xrightarrow{\alpha} s' \end{aligned}$$

**Definition 3** (Weak Transition ( $\Rightarrow$ )). For any transition system  $LTS = (S, \rightarrow, s_0)$ , the *weak transition relation*  $\Rightarrow$  is defined as follows (where  $a \in Act \setminus \{\tau\}$ ):

$$\begin{aligned} s \xRightarrow{\tau} s' &: \Leftrightarrow \exists n \in \mathbb{N} : s \rightsquigarrow^{\tau^n} s' \\ s \xRightarrow{a} s' &: \Leftrightarrow \exists s'', s''' \in S : s \xRightarrow{\tau} s'' \xrightarrow{a} s''' \xRightarrow{\tau} s' \end{aligned}$$

**Definition 4** (Weak Bisimulation). For two transition systems  $LTS_s = (S, \rightarrow_s, s_0)$  and  $LTS_t = (T, \rightarrow_t, t_0)$  over the same set  $Act$ , a relation  $\mathcal{R} \subseteq S \times T$  is called *weak bisimulation* if for any  $(s, t) \in \mathcal{R}$  and  $\alpha \in Act$ , both of the following conditions hold:

- If  $s \xRightarrow{\alpha} s'$ , then there is a  $t' \in T$  such that  $t \xRightarrow{\alpha} t'$  and  $(s', t') \in \mathcal{R}$ .
- If  $t \xRightarrow{\alpha} t'$ , then there is a  $s' \in S$  such that  $s \xRightarrow{\alpha} s'$  and  $(s', t') \in \mathcal{R}$ .

**Definition 5** (Weakly Bisimilar ( $\approx$ )). Two transition systems  $LTS_s = (S, \rightarrow_s, s_0)$  and  $LTS_t = (T, \rightarrow_t, t_0)$  over the same set  $Act$  are called *weakly bisimilar* ( $LTS_s \approx LTS_t$ ) if there is a weak bisimulation  $\mathcal{R}$  such that  $(s_0, t_0) \in \mathcal{R}$  holds.

**Definition 6** (Observation Congruent ( $\cong$ )). Two transition systems  $LTS_s = (S, \rightarrow_s, s_0)$  and  $LTS_t = (T, \rightarrow_t, t_0)$  over the same set  $Act$  are called *observation congruent* ( $LTS_s \cong LTS_t$ ) if both of the following conditions hold:

- If  $s_0 \xrightarrow{\alpha} s'$ , then there are  $n, m \in \mathbb{N}$  and  $t' \in T$  such that  $t_0 \xRightarrow{\tau^n \alpha \tau^m} t'$  and  $(S, \rightarrow_s, s') \approx (T, \rightarrow_t, t')$ .
- If  $t_0 \xrightarrow{\alpha} t'$ , then there are  $n, m \in \mathbb{N}$  and  $s' \in S$  such that  $s_0 \xRightarrow{\tau^n \alpha \tau^m} s'$  and  $(S, \rightarrow_s, s') \approx (T, \rightarrow_t, t')$ .

# Appendix B

## Code Listings

Listing B.1: Gruntfile.js, the configuration file for grunt

```
1 module.exports = function(grunt) {
2     "use strict";
3
4     var aceVersion = '1.1.7';
5
6     grunt.initConfig({
7         pkg: grunt.file.readJSON('package.json'),
8         curl: {
9             ace: {
10                 src: 'https://github.com/ajaxorg/ace-builds/archive/v' + aceVersion + '.zip',
11                 dest: 'managed_components/ace/ace-' + aceVersion + '.zip'
12             },
13         },
14         unzip: {
15             ace: {
16                 src: 'managed_components/ace/ace-' + aceVersion + '.zip',
17                 dest: 'managed_components/ace/build',
18                 router: function (filepath) {
19                     return filepath.split('/').splice(1).join('/');
20                 }
21             },
22         },
23         bower: {
24             install: {
25                 options: {
26                     targetDir: 'managed_components/bower/',
27                     cleanTargetDir: true
28                 }
29             },
30         },
31         uglify: {
32             options: {
33                 mangle: false,
34                 sourceMap: true
35             },
36             app: {
37                 files: {
38                     'temporary_files/build/js/app.min.js': ['temporary_files/build/js/app.js']
39                 },
40             },
41             lib: {
42                 files: {
43                     'temporary_files/build/js/lib.min.js': ['temporary_files/build/js/lib.js']
44                 },
45             },
46             worker: {
47                 files: {
```

```

48         'temporary_files/build/js/worker.min.js': ['temporary_files/build/js/
49             ↳ worker.js']
50     }
51 },
52 concat: {
53     app: {
54         src: [
55             'src/js/libs.js',
56             'src/js/app.js',
57             'src/js/ui/ui.js',
58             'src/js/files/files.js',
59             'src/js/actions/actions.js',
60             'src/js/tasks/tasks.js',
61             'src/js/api/api.js',
62             'src/js/tools/tools.js',
63             'src/js/**/*.js',
64             '!src/js/pre-pseucoco.js', // helper for browserify build
65             'src/js/post-pseucoco.js', // helper for browserify build
66             '!src/js/pseucoco.js', // must be build with browserify
67             'src/js/worker.js', // must be executed as Web Worker
68             '!src/js/tasks/backgroundWorker.js' // must be executed as Web Worker
69         ],
70         dest: 'temporary_files/build/js/app.js',
71         nonull: true,
72         options: {
73             stripBanners: true,
74             banner: '/*!_@license_(c)_Copyright_2014_Felix_Freiberger._Contains_
↳ PseuCoCo,_(c)_Copyright_2013,_2014_Sebastian_Biewer._Details_in_
↳ ReadMe.md_or_on_the_About-tab.\n'
75         }
76     },
77     lib: {
78         src: [
79             'managed_components/bower/jquery/jquery.js',
80             'managed_components/bower/angular/angular.js',
81             'managed_components/bower/**/*.js',
82             'managed_components/ace/build/src/ace.js'
83         ],
84         dest: 'temporary_files/build/js/lib.js',
85         nonull: true
86     },
87     CCS: {
88         src: [
89             'temporary_files/CCSParser.js',
90             'temporary_files/CCS_no_parser.js'
91         ],
92         dest: 'temporary_files/CCS.js',
93         nonull: true
94     },
95     PseuCo: {
96         src: [
97             'temporary_files/PseuCoParser.js',
98             'temporary_files/PseuCo_no_parser.js'
99         ],
100         dest: 'temporary_files/PseuCo.js',
101         nonull: true
102     },
103     worker: {
104         src: [
105             'src/js/pre-pseucoco.js',
106             'temporary_files/pseucoco.js',
107             'src/js/post-pseucoco.js',
108             'managed_components/bower/underscore/underscore.js',
109             'temporary_files/AUTParser.js',
110             'src/js/worker.js',
111             'src/js/tasks/backgroundWorker.js'
112         ],
113         dest: 'temporary_files/build/js/worker.js',
114         nonull: true
115     }
116 },
117 jshint: {

```

```

118     options: {
119         jshintsrc: true
120     },
121     all: ['src/js/**/*.js']
122 },
123 cssmin: {
124     combine: {
125         files: {
126             'temporary_files/build/css/app.css': ['src/css/**/*.css'],
127             'temporary_files/build/css/lib.css': ['managed_components/bower/**/*.css']
128         }
129     },
130 },
131 copy: {
132     partials: {
133         expand: true,
134         flatten: true,
135         src: 'src/partials/*',
136         dest: 'temporary_files/build/partials/'
137     },
138     index: {
139         src: 'src/index.html',
140         dest: 'temporary_files/build/index.html'
141     },
142     htaccess: {
143         src: 'src/.htaccess',
144         dest: 'temporary_files/build/.htaccess'
145     },
146     serverhtaccess: {
147         src: 'server/server.htaccess',
148         dest: 'temporary_files/build/api/.htaccess'
149     },
150     appcache: {
151         src: 'src/offline.appcache',
152         dest: 'build/offline.appcache',
153         options: {
154             process: function (content) {
155                 return content + '\n\n#_ ' + (new Date()).toString() + '\n';
156             }
157         }
158     },
159     fonts: {
160         expand: true,
161         flatten: true,
162         src: 'managed_components/bower/bootstrap/glyphicons-halflings-regular.*',
163         dest: 'temporary_files/build/fonts/'
164     },
165     images: {
166         expand: true,
167         flatten: true,
168         src: 'src/img/*',
169         dest: 'temporary_files/build/img/'
170     },
171     version: {
172         src: 'version.txt',
173         dest: 'temporary_files/build/version.txt'
174     }
175 },
176 clean: {
177     tmpbuild: ['temporary_files/build/'],
178     build: ['build/'],
179     oldbuild: ['temporary_files/old_build'],
180     tmp: ['temporary_files/'],
181     managed: ['managed_components/'],
182     ace: ['managed_components/ace/']
183 },
184 watch: {
185     js: {
186         files: 'src/js/**/*.js',
187         tasks: ['test', 'clean:tmpbuild', 'rename:unbuild', 'concat:app', 'concat
188             ↳ :worker', 'rename:build']
189     }
190 }

```

```

189     html: {
190         files: 'src/**/*.html',
191         tasks: ['clean:tmpbuild', 'rename:unbuild', 'copy:partials', 'copy:index',
192             ↳ , 'rename:build']
193     },
194     css: {
195         files: 'src/**/*.css',
196         tasks: ['clean:tmpbuild', 'rename:unbuild', 'cssmin', 'rename:build']
197     },
198     parsers: {
199         files: 'src/js/parsers/*.pegjs',
200         tasks: ['clean:tmpbuild', 'rename:unbuild', 'peg:AUT', 'concat:worker', '
201             ↳ rename:build']
202     }
203 },
204 peg: {
205     CCS: {
206         src: 'node_modules/PseuCoCo/CCSParser.pegjs',
207         dest: 'temporary_files/CCSParser.js',
208         options: {
209             exportVar: 'CCSParser'
210         }
211     },
212     PseuCo: {
213         src: 'node_modules/PseuCoCo/PseuCoParser.pegjs',
214         dest: 'temporary_files/PseuCoParser.js',
215         options: {
216             exportVar: 'PseuCoParser'
217         }
218     },
219     AUT: {
220         src: 'src/js/parsers/AUT.pegjs',
221         dest: 'temporary_files/AUTParser.js',
222         options: {
223             exportVar: 'AUTParser'
224         }
225     }
226 },
227 coffee: {
228     CCS: {
229         options: {
230             bare: true
231         },
232         files: {
233             'temporary_files/CCS_no_parser.js': ['node_modules/PseuCoCo/CCS.
234             ↳ coffee', 'node_modules/PseuCoCo/CCSRules.coffee', 'node_modules/
235             ↳ PseuCoCo/CCS+Traces.coffee', 'node_modules/PseuCoCo/CCSExecutor.
236             ↳ coffee', 'node_modules/PseuCoCo/CCSExport.coffee']
237         }
238     },
239     PseuCo: {
240         options: {
241             bare: true
242         },
243         files: {
244             'temporary_files/PseuCo_no_parser.js': ['node_modules/PseuCoCo/PseuCo
245             ↳ .coffee', 'node_modules/PseuCoCo/PCType.coffee', 'node_modules/
246             ↳ PseuCoCo/PCEnvironment.coffee', 'node_modules/PseuCoCo/PCExport.
247             ↳ coffee']
248         }
249     },
250     CCSCompiler: {
251         options: {
252             bare: true
253         },
254         files: {
255             'temporary_files/CCSCompiler.js': ['node_modules/PseuCoCo/PCCCompiler
256             ↳ .coffee', 'node_modules/PseuCoCo/PCCProcessFrame.coffee', '
257             ↳ node_modules/PseuCoCo/PCCProgramController.coffee', 'node_modules/
258             ↳ PseuCoCo/PCCContainer.coffee', 'node_modules/PseuCoCo/
259             ↳ PCCCompilerStack.coffee', 'node_modules/PseuCoCo/PseuCo+Compiler.
260             ↳ coffee', 'node_modules/PseuCoCo/PCCExecutor.coffee', 'node_modules
261             ↳ /PseuCoCo/PCCExport.coffee']
262         }
263     }
264 }

```



```

248     }
249   }
250 },
251 browserify: {
252   PseuCoCo: {
253     src: 'src/js/pseucoco.js',
254     dest: 'temporary_files/pseucoco.js',
255   },
256   options: {
257     browserifyOptions: {
258       paths: ['temporary_files/']
259     }
260   },
261 },
262 rename: {
263   build: {
264     files: [
265       {
266         src: 'build',
267         dest: 'temporary_files/old_build'
268       },
269       {
270         src: 'temporary_files/build',
271         dest: 'build'
272       }
273     ],
274   },
275   unbuild: {
276     files: [
277       {
278         src: 'build',
279         dest: 'temporary_files/build'
280       }
281     ],
282   }
283 }
284 });
285
286 grunt.loadNpmTasks('grunt-bower-task');
287 grunt.loadNpmTasks('grunt-contrib-uglify');
288 grunt.loadNpmTasks('grunt-contrib-cssmin');
289 grunt.loadNpmTasks('grunt-contrib-jshint');
290 grunt.loadNpmTasks('grunt-contrib-copy');
291 grunt.loadNpmTasks('grunt-contrib-clean');
292 grunt.loadNpmTasks('grunt-contrib-concat');
293 grunt.loadNpmTasks('grunt-contrib-watch');
294 grunt.loadNpmTasks('grunt-contrib-coffee');
295 grunt.loadNpmTasks('grunt-peg');
296 grunt.loadNpmTasks('grunt-browserify');
297 grunt.loadNpmTasks('grunt-zip');
298 grunt.loadNpmTasks('grunt-curl');
299 grunt.loadNpmTasks('grunt-contrib-rename');
300
301 grunt.registerTask('get-ace', 'Get and unzip the Ace text editor.', function() {
302   if (!grunt.file.exists('managed_components/ace/ace-' + aceVersion + '.zip')) {
303     grunt.task.run(['clean:ace', 'curl:ace', 'unzip:ace']);
304   }
305 });
306
307 grunt.registerTask('test', ['jshint']);
308 grunt.registerTask('build', ['peg', 'coffee', 'concat:app', 'concat:lib', 'concat:CCS',
309   'concat:PseuCo', 'browserify', 'concat:worker', 'uglify:lib', 'cssmin', 'copy:
310   partials', 'copy:fonts', 'copy:images', 'copy:htaccess', 'copy:serverhtaccess',
311   'copy:index', 'copy:version', 'clean:oldbuild', 'rename:build']);
312 grunt.registerTask('appcache', ['copy:appcache']);
313 grunt.registerTask('deploy', ['bower:install', 'get-ace', 'build']);
314 grunt.registerTask('serverside', ['deploy', 'appcache']);
315 grunt.registerTask('default', ['test', 'deploy']);

```

---

Listing B.2: weakProcessState() from worker.js

```

284 var weakProcessState = function (ltsData, stateName) {

```

```

285     if (!stateName) stateName = ltsData.extended.explorationState.weakFrontier.shift
286         ↳ (); // draw next state
287
288     var states = ltsData.core.states;
289     var state = states[stateName];
290
291     var insertConnection = function (set, value) {
292         if (!_.contains(set, value)) {
293             set.push(value);
294             return true;
295         }
296     };
297
298     var insertTransition = function (set, transition) {
299         if (!_.some(set, function (t) {
300             return t.source === transition.source && t.target === transition.target
301                 ↳ && t.label === transition.label;
302         }))) {
303             set.push(transition);
304             return true;
305         }
306     };
307
308     var createTransition = function (preStateName, postStateName, label) {
309         var weakTransition = {
310             source: preStateName,
311             target: postStateName,
312             label: label
313         };
314
315         insertTransition(states[preStateName].weakTransitions.outgoing,
316             ↳ weakTransition);
317         insertTransition(states[postStateName].weakTransitions.incoming,
318             ↳ weakTransition);
319     };
320
321     _.each(state.transitions, function (transition, index) {
322         // insert self-connection to target state
323         var targetState = ltsData.core.states[transition.target];
324         insertConnection(targetState.weakConnections.pre, transition.target);
325         insertConnection(targetState.weakConnections.post, transition.target);
326
327         if (transition.weak) {
328             // weak transition
329
330             // update weak connections
331             _.each(state.weakConnections.pre, function (preStateName) {
332                 _.each(targetState.weakConnections.post, function (postStateName) {
333                     var wasNew = insertConnection(states[preStateName].
334                         ↳ weakConnections.post, postStateName);
335                     insertConnection(states[postStateName].weakConnections.pre,
336                         ↳ preStateName);
337
338                     if (wasNew) {
339                         // we discovered a new connection
340                         // this might allow new transitions (longer than existing
341                             ↳ ones)
342
343                         _.each(states[preStateName].weakTransitions.incoming,
344                             ↳ function (t) {
345                             createTransition(t.source, postStateName, t.label);
346                         });
347                         _.each(states[postStateName].weakTransitions.outgoing,
348                             ↳ function (t) {
349                             createTransition(preStateName, t.target, t.label);
350                         });
351                     }
352                 });
353             });
354         } else {
355             // strong transition
356
357             // create weak transitions

```

```

349         .each(state.weakConnections.pre, function (preStateName) {
350             .each(targetState.weakConnections.post, function (postStateName) {
351                 createTransition(preStateName, postStateName, transition.label);
352             });
353         });
354     }
355 });
356
357 state.weakExplored = true;
358 };

```

---

Listing B.3: stateIsFullyWeakExplored() and utility functions from worker.js

```

361 var stateIsWeakExplored = function (ltsData, stateName) { // true if a state has been
362     ↳ weak explored
363     return ltsData.core.states[stateName].weakExplored;
364 };
365
366 var stateIsWeakConnectionExplored = function (ltsData, stateName) { // if true, no
367     ↳ more states will be added to weakConnections
368     var state = ltsData.core.states[stateName];
369     if (state.weakConnectionExplored) return true;
370
371     if(stateIsWeakExplored(ltsData, stateName) && .every(state.weakConnections.post,
372     ↳ function (postStateName) {
373         return stateIsWeakExplored(ltsData, postStateName);
374     })) {
375         state.weakConnectionExplored = true;
376         return true;
377     } else {
378         return false;
379     }
380 };
381
382 var stateIsFullyWeakExplored = function (ltsData, stateName) { // if true, no more
383     ↳ weak transitions or connections will be added to this state
384     var states = ltsData.core.states;
385     var state = states[stateName];
386
387     if (ltsData.extended.explorationState.explorationFinished) return true;
388     if (state.fullyWeakExplored) return true;
389     if (!stateIsWeakConnectionExplored(ltsData, stateName)) return false;
390
391     if (.every(state.weakTransitions.outgoing, function (transition) {
392     return stateIsWeakConnectionExplored(ltsData, transition.target);
393     })) {
394         state.fullyWeakExplored = true;
395         return true;
396     }
397
398     return false;
399 };

```

---

Listing B.4: Task function of the minimizeLts task from worker.js

```

739 minimizeLts: function (data, workerState) {
740     var ltsData = workerState.data[data.dataId];
741
742     if (!ltsData) {
743         return {
744             data: null,
745             taskCompleted: true
746         };
747     }
748
749     enhanceIfNeeded(ltsData);
750
751     var states = ltsData.core.states;
752
753     var initialBlock = [];
754     var blocks = [];

```

```

755
756     var stateToBlockMap = {};
757
758     var notFullyWeaklyExploredStates = [];
759
760     _.each(states, function (state, stateName) {
761         if (_.contains(ltsData.extended.explorationState.explored,
762             ↳ stateName) || _.contains(ltsData.extended.explorationState.
763             ↳ frontier, stateName)) {
764             if (stateIsFullyWeakExplored(ltsData, stateName)) {
765                 initialBlock.push(stateName);
766                 stateToBlockMap[stateName] = initialBlock;
767             } else {
768                 // unexplored state
769                 var newBlock = [stateName]; // this state may not be
770                 ↳ grouped
771                 blocks.push(newBlock);
772                 stateToBlockMap[stateName] = newBlock;
773                 notFullyWeaklyExploredStates.push(stateName);
774             }
775         } else {
776             // unreachable state - ignore
777         }
778     });
779
780     if (initialBlock.length > 0) blocks.push(initialBlock);
781
782     var done = false;
783
784     var buildToDo = function () {
785         var todo = [];
786
787         _.each(blocks, function (block1) {
788             _.each(blocks, function (block2) {
789                 todo.push({ block1: block1, block2: block2 });
790             });
791         });
792
793         return todo;
794     };
795
796     var getIncomingActionPredecessors = function (block) { // gets a map
797         ↳ of all strong actions to predecessor states
798         if (block.incomingActionPredecessors) return block.
799             ↳ incomingActionPredecessors;
800
801         var result = {};
802
803         _.each(block, function (stateName) {
804             _.each(states[stateName].weakTransitions.incoming, function (
805             ↳ transition) {
806                 var label = transition.label;
807                 var preList = result[label];
808                 if (!preList) { preList = []; result[label] = preList; }
809                 preList.push(transition.source);
810             });
811         });
812
813         block.incomingActionPredecessors = result;
814         return result;
815     };
816
817     var getWeakPredecessors = function (block) {
818         return _.flatten(_.map(block, function (stateName) {
819             var state = states[stateName];
820             return state.weakConnections.pre;
821         })), true);
822     };
823
824     var getSplit = function (block, splitter) {
825         var intersection = _.intersection(block, splitter);
826         if (!_.isEmpty(intersection)) {
827             var difference = _.difference(block, splitter);

```

```

822         if (!_.isEmpty(difference)) {
823             return { block1: intersection, block2: difference };
824         }
825     }
826
827     return false;
828 };
829
830 var performSplit = function (block, split) {
831     var index = blocks.indexOf(block);
832     if (index > -1) blocks.splice(index, 1);
833     else throw "trying to split a nonexisting block";
834
835     blocks.push(split.block1);
836     blocks.push(split.block2);
837
838     var assignBlock = function (block) {
839         return function (stateName) {
840             stateToBlockMap[stateName] = block;
841         };
842     };
843
844     _.each(split.block1, assignBlock(split.block1));
845     _.each(split.block2, assignBlock(split.block2));
846 };
847
848 // partition
849 while (!done) {
850     done = true;
851
852     var todo = buildToDo();
853     todoloop: while (!_.isEmpty(todo)) {
854         var task = todo.shift();
855
856         var block1 = task.block1;
857         if (!_.contains(blocks, block1)) continue; // this block was
            ↳ split already
858
859         var block2 = task.block2;
860
861         var weakPredecessors = getWeakPredecessors(block2);
862         var weakSplit = getSplit(block1, weakPredecessors);
863
864         if (weakSplit) {
865             done = false;
866             performSplit(block1, weakSplit);
867         } else {
868             var incomingActionPredecessors2 =
            ↳ getIncomingActionPredecessors(block2);
869
870             for (var action in incomingActionPredecessors2) {
871                 var predecessors = incomingActionPredecessors2[action]
            ↳ ];
872                 var split = getSplit(block1, predecessors);
873                 if (split) {
874                     done = false;
875                     performSplit(block1, split);
876                     break todoloop;
877                 }
878             }
879         }
880     }
881 }
882
883 // partitioning completed
884 var newStates = {};
885 var nextStateLabel = 0;
886
887 var blockToStateInfo = {};
888 _.each(blocks, function (block) {
889     blockToStateInfo[block] = {
890         label: '' + nextStateLabel++,

```

```

891         weakConnectionPostBlocks: _.uniq(_.flatten(_.map(block,
892             ↳ function (stateName) {
893                 return _.map(states[stateName].weakConnections.post,
894                     ↳ function (postStateName) {
895                         return stateToBlockMap[postStateName];
896                     });
897             })), true)) // blocks that can be reached by only tau steps
898     });
899     var oldToNewLabel = function (oldStateName) {
900         return blockToStateInfo[stateToBlockMap[oldStateName]].label;
901     };
902     _.each(blocks, function (block) {
903         var thisBlockLabel = blockToStateInfo[block].label;
904
905         var outgoingOldTransitions = _.flatten(_.map(block, function (
906             ↳ stateName) {
907                 return states[stateName].weakTransitions.outgoing;
908             })), true);
909
910         var outgoingNewTransitions = _.map(outgoingOldTransitions,
911             ↳ function (transition) {
912                 // this weak transition points to an old state - build a new
913                 ↳ one, pointing to a new state
914                 return {
915                     label: transition.label,
916                     target: oldToNewLabel(transition.target)
917                 };
918             });
919
920         // remove duplicates
921         var uniqueOutgoingNewTransitions = _.flatten(_.map(_.groupBy(
922             ↳ outgoingNewTransitions, 'label'), function (
923                 ↳ transitionsWithCommonLabel) {
924                     return _.uniq(transitionsWithCommonLabel, 'target');
925                 })), true);
926
927         var weaklyReachibleNewStates = _.uniq(_.flatten(_.map(block,
928             ↳ function (stateName) {
929                 return _.map(states[stateName].weakConnections.post,
930                     ↳ oldToNewLabel);
931             })), true));
932
933         _.each(weaklyReachibleNewStates, function (targetStateName) {
934             if (targetStateName !== thisBlockLabel) {
935                 // targetStateName is reachable with a weak transition -
936                 ↳ add that
937                 uniqueOutgoingNewTransitions.push({
938                     target: targetStateName,
939                     weak: true
940                 });
941             }
942         });
943
944         var filteredUniqueOutgoingNewTransitions = _.filter(
945             ↳ uniqueOutgoingNewTransitions, function (transitionToTest) {
946                 // check if the transition is needed (or can be replaced with
947                 ↳ a transition of a weak post state)
948                 return !_some(blockToStateInfo[block].
949                     ↳ weakConnectionPostBlocks, function (
950                         ↳ weakConnectionPostBlock) {
951                     var weaklyOutgoingMatchingTransitions = _.flatten(_.map(
952                         ↳ weakConnectionPostBlock, function (
953                             ↳ weakConnectionPostState) {
954                             return _.filter(states[weakConnectionPostState].
955                                 ↳ transitions, function (transition) {
956                                     return transition.label === transitionToTest.
957                                         ↳ label && transition.weak === transitionToTest.
958                                         ↳ weak;
959                                 });
960                         });
961                     });
962                 });
963             });
964     });

```

```

945
946 // weaklyOutgoingMatchingTransitions are all transitions
947 // that might make transitionToTest superfluous
948
949 return _.some(weaklyOutgoingMatchingTransitions, function
950   (weaklyOutgoingMatchingTransition) {
951     if (weaklyOutgoingMatchingTransition.weak &&
952         stateToBlockMap[weaklyOutgoingMatchingTransition.
953           target] === weakConnectionPostBlock) return false;
954     // prevent tau steps from replacing themselves as
955     // part of tau*
956     return _.some(states[weaklyOutgoingMatchingTransition
957       .target].weakConnections.post, function (
958         stateAfterOutgoingWeakMatchingTransition) {
959           if (oldToNewLabel(
960             stateAfterOutgoingWeakMatchingTransition) !==
961             transitionToTest.target) return false; // we
962             // ended up in the wrong state - no possible
963             // replacement
964
965             // we found a matching route: tau* ->
966             // weaklyOutgoingMatchingTransition -> tau*
967             // we still must check that this is longer than
968             // transitionToTest, otherwise, each transition
969             // would remove itself
970             return block !== weakConnectionPostBlock ||
971             transitionToTest.target !== oldToNewLabel(
972               weaklyOutgoingMatchingTransition.target);
973           });
974         });
975     });
976   });
977
978   newStates[thisBlockLabel] = {
979     transitions: filteredUniqueOutgoingNewTransitions
980   };
981 });
982
983 var newInitialState = oldToNewLabel(ltsData.core.initialState);
984
985 // add weak self-loop if needed for observation congruence
986 var blockWithInitialState = stateToBlockMap[ltsData.core.initialState
987   ];
988 if (_.some(states[ltsData.core.initialState].transitions, function (
989   transition) {
990   return transition.weak && stateToBlockMap[transition.target] ===
991     blockWithInitialState;
992   }
993 )) {
994   newStates[newInitialState].transitions.push({
995     target: newInitialState,
996     weak: true
997   });
998 }
999
1000 var dataId = workerState.nextDataId++;
1001
1002 var newLtsData = {
1003   core: {
1004     initialState: newInitialState,
1005     states: newStates
1006   },
1007   extended: {
1008     dataId: dataId
1009   }
1010 };
1011
1012 var generateRelayingTransitionGenerator = function (state,
1013   underlyingStateName) {
1014   return function () {
1015     var underlyingState = states[underlyingStateName];
1016     if (!underlyingState.transitions) blindExploreState(data.
1017       dataId, ltsData, underlyingStateName);
1018   };
1019 }

```

```

996         var underlyingTansitions = underlyingState.transitions;
997
998         state.transitions = state.transitionsFromMinimization;
999         if (!state.transitions) state.transitions = [];
1000
1001         _.each(underlyingTansitions, function (underlyingTransition)
1002         {
1003             if (!stateToBlockMap[underlyingTransition.target]) {
1004                 var newStateLabel = 'U-' + nextStateLabel++;
1005
1006                 var newState = JSON.parse(enhancedStateTemplate);
1007                 newState.generateTransitions =
1008                     ↳ generateRelayingTransitionGenerator(newState,
1009                     ↳ underlyingTransition.target);
1010
1011                 newStates[newStateLabel] = newState;
1012
1013                 var newBlock = [newState];
1014                 stateToBlockMap[underlyingTransition.target] =
1015                     ↳ newBlock;
1016                 blockToStateInfo[newBlock] = { label: newStateLabel
1017                     ↳ };
1018             } else {
1019                 if (_.some(state.transitions, function (transition) {
1020                     return transition.label === underlyingTransition.
1021                         ↳ label && transition.weak ===
1022                         ↳ underlyingTransition.weak && transition.target
1023                         ↳ === oldToNewLabel(underlyingTransition.target
1024                         ↳ );
1025                 ))) {
1026                     return; // transition already exists - ignore
1027                 }
1028             }
1029
1030             state.transitions.push ({
1031                 label: underlyingTransition.label,
1032                 weak: underlyingTransition.weak,
1033                 target: oldToNewLabel(underlyingTransition.target)
1034             });
1035         });
1036     });
1037
1038     var safeDataCopy = JSON.parse(JSON.stringify(newLtsData));
1039
1040     // restore explore functionality for not fully explored states
1041     _.each(notFullyWeaklyExploredStates, function (originalStateName) {
1042         var state = newStates[oldToNewLabel(originalStateName)];
1043         state.generateTransitions = generateRelayingTransitionGenerator(
1044             ↳ state, originalStateName);
1045
1046         // hide transitions from view - state needs to be explored
1047         state.transitionsFromMinimization = state.transitions;
1048         state.transitions = undefined;
1049     });
1050
1051     workerState.data[dataId] = newLtsData;
1052
1053     return {
1054         data: {
1055             command: 'minimized',
1056             data: safeDataCopy
1057         },
1058         taskCompleted: true
1059     };
1060 }

```

---



## Appendix C

# The Repository Structure

In the source code repository of `pseuCo.com`, you can find a multitude of files and directories. Some of them are generated by the package managers and the build script, and not included in the repository directly.

Here is an overview over this directory structure:

- 📁 `.git` — hidden data directory from Git
- 📁 `bower_components` — managed by Bower, contains all files from the dependencies shown in Listing 5.1
- 📁 `build` — contains the resulting directory structure the web server should present – see Appendix D
- 📁 `managed_components` — contains files managed automatically by the build scripts
  - 📁 `ace` — contains the downloaded archive and unzipped version of Ace
  - 📁 `bower` — contains only the needed files from the Bower dependencies shown in Listing 5.1
- 📁 `node_modules` — managed by npm, contains all files from the dependencies shown in Listing 5.2
- 📁 `server` — contains the files needed for the server component – see section 5.12
  - 📁 `data` — the data directory for the server component
    - 📁 `paste` — files that have been submitted to the paste service from users
    - 📁 `templates` — template files that will be offered to users, separated by file type
      - 📄 `ccs.json`
      - 📄 `lts.json`
      - 📄 `pseuco.json`
  - 📄 `server.htaccess` — configuration file for Apache to direct incoming HTTP requests to the `pseuCo.com` server running at port 9128
  - 📄 `server.js` — the server code
- 📁 `src`
  - 📁 `css` — all CSS files needed by `pseuCo.com` (CSS definitions can be in any CSS file in this folder – the separation in different files is purely for convenience.)
  - 📁 `img` — contains images used in `pseuCo.com`
  - 📁 `js` — contains all main JavaScript files
  - 📁 `partials` — contains partials
  - 📄 `.htaccess` — configuration file for Apache, setting headers to disable the normal client-side caching (in favour of the Application Cache – see section 5.6)

- `index.html` — main HTML file, describing the core parts of the UI (Additional UI elements (partials) are injected during runtime.)
- `offline.appcache` — configuration file for the Application Cache – see section 5.6
- 📁 `temporary_files` — location for temporary files needed during the build process
- `.gitattributes` — configuration file for Git, ensuring line endings are normalized
- `.gitignore` — configuration files for Git, ensuring generated files are not committed to the repository by accident
- `.jshintrc` — configuration file for jshint
- `bower.json` — configuration file for Bower, listing dependencies
- `COPYING` — license text
- `Gruntfile.js` — configuration file for grunt, describing the build process – see section 5.3
- `package.json` — configuration file for npm, describing this package and the dependencies
- `ReadMe.md`
- `version.txt` — contains the string `testing` to allow `pseuCo.com` to identify itself as having been built from source (For the official builds, this file is overwritten with the string displayed at `base URL/#/about.`)

## Appendix D

# The Build Directory Structure

This section describes the final directory structure, ready to be served by Apache or any other HTTP server. It is set up by the build scripts in `repository/build/`.

- 📁 **api** — virtual path, to which the server component should respond (The folder contains a `.htaccess` file to configure Apache to redirect requests to it.)
- 📁 **css**
  - 📄 `app.css` — CSS styles from the main application
  - 📄 `lib.css` — CSS styles from external libraries
- 📁 **fonts** — contains fonts containing Glyphicon symbols
- 📁 **img**
  - 📄 `icon.png` — the `pseuCo.com` icon, PNG version
  - 📄 `icon.svg` — the `pseuCo.com` icon, SVG version
- 📁 **js**
  - 📄 `app.js` — JavaScript code from the main application
  - 📄 `lib.js` — JavaScript code from external libraries
  - 📄 `lib.min.js` — JavaScript code from external libraries, minified
  - 📄 `lib.min.js.map` — source map for `lib.js`, for debugging
  - 📄 `worker.js` — JavaScript code for the background worker – see section 5.7
- 📁 **partials**
  - 📄 `about.html` — UI fragment: “About” tab
  - 📄 `ace.html` — UI fragment for including Ace
  - 📄 `actions.html` — UI fragment: action buttons
  - 📄 `backup.html` — UI fragment: “Backup” tab
  - 📄 `debug.html` — UI fragment: the hidden “Debug” tab – see section 5.1
  - 📄 `edit.html` — UI fragment: file viewing and editing
  - 📄 `error.html` — UI fragment: displays critical errors
  - 📄 `export.html` — UI fragment: LTS file export
  - 📄 `fetch.html` — UI fragment: message shown while a shared file downloads
  - 📄 `files.html` — UI fragment: “Files” tab
  - 📄 `help.html` — UI fragment: “Help” tab

- `import.html` — UI fragment: file import
- `landing.html` — UI fragment: main “pseuCo.com” tab
- `lts.html` — UI fragment: the LTS viewer
- `newfile.html` — UI fragment: new file creation
- `pseucojava.html` — UI fragment: the classical pseuCo-Java compiler
- `share.html` — UI fragment: file upload for sharing
- `svg.html` — UI fragment: result of SVG export
- `trace.html` — UI fragment: random trace through LTS
- `.htaccess` — configuration file for Apache – see section 5.6
- `index.html` — main HTML file
- `offline.appcache` — configuration file for the Application Cache – see section 5.6
- `version.txt` — describes the version of `pseuCo.com` – either “testing” if built from source locally, or a branch name and commit identifier if built on the official server

## Appendix E

# User Manual (digital printout)

| Remark   |
|--|
| The documentation of the pseuCo language contains work by Kathrin Stark and Holger Hermanns. |

# pseuCo.com User Manual

## Overview

With this application, you can easily create and share CCS and pseuCo files and convert them to Labelled Transition Systems. You can also minimize and analyze such transition systems.

The following sections will help you to use pseuCo.com.

## Requirements

PseuCo.com is designed to run in any modern browser. The most important requirements are:

- **JavaScript** must be enabled.
- **LocalStorage** must be available. This is the default in most browsers. If you receive an error message that LocalStorage it is not available, make sure *private browsing* is disabled, and try enabling *cookies* – some browsers silently block LocalStorage if cookies are disabled.
- **Cookies** are required to switch to the beta version.


PseuCo.com has been tested in recent versions of Chrome, Firefox and Internet Explorer.


## Managing Files


Everything you create in this application is saved automatically within your browser. If you go back to pseuCo.com (<http://pseuco.com/>) on the same device, you will see all files you created in the Files tab.

If you did not enter a file name, you might still be able to find your file in the “Unnamed Files” section. However, such files are deleted automatically after a few days. To keep a file permanently, you must give it a name. To do so, open it and enter a name into the box at the top of the page.

Your files can be deleted by your browser in some cases, e.g. if you “clear private data” or use “private mode”. To keep your files safe, you should save a backup regularly.

If you want to create a copy of a file, click the  clone icon next to its name on the Files tab. An unnamed copy of the file will open. To save it permanently, give it a name as usual.

To delete a named file, click the yellow  delete button next to it. If the file was named, the name will be removed, causing the file to be deleted after a few days. If you accidentally deleted a named file, open it from the “Unnamed Files” section, and re-enter a file name.

To delete an unnamed file immediately, click the red  delete button next to it. The file will be deleted immediately. **Watch out:** There is no way to undo this.

## Backing Up and Restoring Your Files

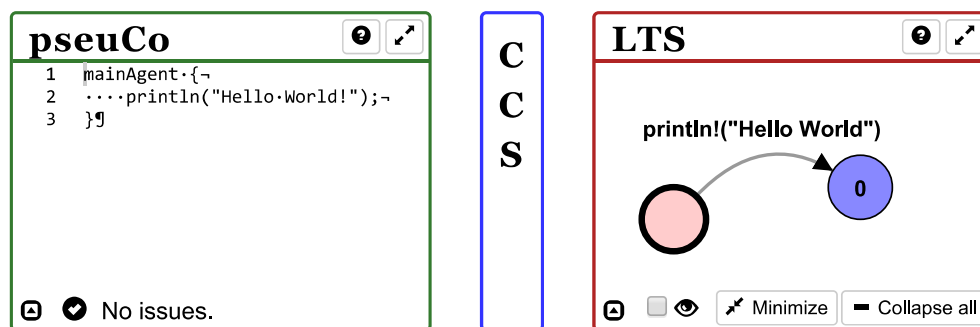
To backup your files, open the Backup tab. In the “Backup” section, you will see a text box containing all your files. Select all text by clicking into it, then copy it. Paste the contents into an empty text file and store it.

Please note that unnamed files will not be included in the backup.

To restore a backup, open the Backup tab. Copy the backup data from the text file you stored earlier, and paste it into the text box in the “Restore” section. If your backup is valid, a button will appear below the text box. Read the instructions, and click the button to start the restore process.

## Editing Files

If you open a file for editing, you will see one or multiple windows, each containing an editor. It will look like this:



What you see above isn't a static image – it's a live demo. Feel free to experiment with the windows. Changes you make are ignored and will be lost when you leave this page.

You can resize these windows horizontally by dragging the grey separator bar. In addition, every window has a button in the top right corner, which will maximize the window.

If a window shrinks too much, it will be reduced to a thin vertical stripe. To show the contents again, drag a divider to make room for it, or click on the stripe to quickly maximize it.

Only the leftmost window is editable – it contains your file. All other windows show data generated from that file, for example the LTS graph. If you want to edit such generated data, check if the title has an edit icon: If it has, you can click it to create a new file based on the content below.

To get more information about a specific editor, click the help icon on the corresponding title bar.

While editing a file you will see some buttons, called “action buttons”, on the bottom of the page. You can find more information on them below, or in the section for the corresponding file type.

## File Sharing

You can easily share any open file by clicking the “ Share this file” action button. After accepting the conditions and clicking the upload button, you will be given a unique link. You can pass this link on to anyone you want to share this file with.


Any user opening a sharing link will see a read-only view of the shared file. The file can be saved and made editable by clicking the blue notification bar.

## Importing Files


You can import Labelled Transition Systems into pseuCo.com. Simply open the import page for transition systems, and paste data in a supported file format. See the import page for details on which formats are supported.

In the rare case that the data you pasted can be interpreted in various formats, you will see multiple import buttons. Make sure you select the correct one so your data will be interpreted correctly.

## Distraction-Free Editing

To edit files with as few distractions as possible, click the  maximize icon in the navigation bar. The UI will shrink, hiding elements that are less important.

**Hint:** You can also enable full screen mode in your browser. On desktop browsers, this usually happens when you press `F11` or `⌘F`.

To leave this mode, click the  restore icon in the top-right corner. The mode will also be disabled automatically when you navigate to another page.

## File Types

### Labelled Transition Systems

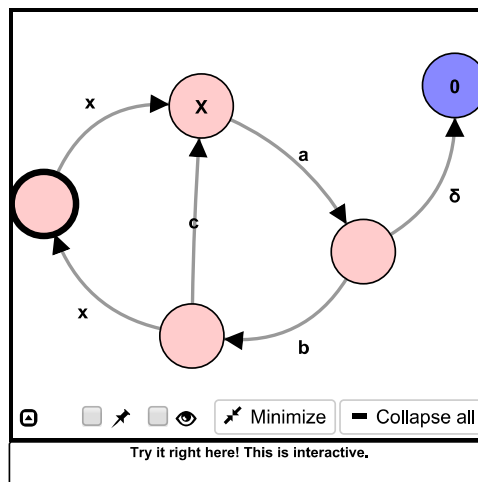
LTS graphs can be huge, so pseuCo.com cannot display them completely. Therefore, you can interactively expand just the parts of the graph you are interested in.

When you see a new LTS graph, you will only be shown the initial state. If you click on this state, it will *expand*, revealing all successor states. The color of a state will turn into a lighter red to tell you that this state is fully expanded. If a state turns blue, it is *terminal*, meaning that there are no successors.

To save space, you can collapse nodes you are no longer interested in by clicking on them.

The application tries to show you LTS graphs as clearly arranged as possible – but this may not always work automatically. If the graph gets tangled up, you can drag the nodes to rearrange them.



Labels for states will be shown on the states themselves only if they are extremely short. Otherwise, hover your mouse over a state or touch the state to show the label in the grey tool bar on the bottom.





## Other Modes for Exploring LTS Graphs

For large graphs, there are alternative modes to help you retain control. They can be enabled by ticking the appropriate checkboxes on the grey tool bar, or in the settings panel you can reveal by clicking the tool bar.


-  **Pin dragged nodes** helps you to arrange complex graphs if the automatic arrangement produces unacceptable results. If you enable this option, any state you drag will stick to the position where you dropped it. To release a state, disable this option and drag the state again.
-  **Focus on single node** helps you by automatically collapsing nodes. With this setting enabled, only one state can be expanded fully, all other states will collapse to only show you the paths leading to the focused state.

## Additional Display Options


Clicking the grey tool bar reveals a panel with additional options. It allows you to:

- **Compress weak steps** to save space: Weak transitions will be shorter and have no visible label.
- **Export as SVG graphic:** This will show SVG code. Copy it and save it to an empty text file and save it with the file extension `.svg` to get a vector graphic export of the current LTS view.

## LTS Minimization

For any transition system, pseuCom.com can compute the minimal observation congruent transition system for you. Start this process by clicking the  Minimize button in the tool bar.

Fully minimizing the system only works if the transition system has been fully explored. You will see a notice in the upper left corner of the LTS viewer if exploration is not finished yet. You can start the minimization process at any time, but the resulting system may be too large.

If you view a minimized transition system, click the  Restore button at any time to return to the original transition system.

## Actions

You can perform additional actions on the transition system using the corresponding “action buttons”.

### Random Path

Clicking this button shows a random path through the transition system. You can generate new random paths by clicking “Restart”.

### Export LTS

Clicking this button allows you to export the transition system in a number of formats. You can view details about a format by selecting it in the list.

Please note that infinite systems cannot be exported – if you try to do so, the export process will get stuck. Exporting large systems can take a while.

Some export formats fail to describe certain transition systems. Please read the details before proceeding.

## pseuCo

You can find information about the history of pseuCo on the landing page.

The following sections give you an overview over pseuCo's syntax and semantics.

### Getting Started

In pseuCo, execution starts in the `mainAgent` :

```
mainAgent {  
    // execution starts here  
}
```

Each statement must end with a semicolon.

With `println`, you can output strings and integer values.

```
println(7);  
println(7 + ": Primzahl.");
```

You can declare variables **locally** or **globally**. Initialization is optional.

```
int x = 5;  
mainAgent {  
    bool y; // defaults to false  
    y = true;  
    x = 4;  
}
```

Variables may be declared at most once.

Procedure definitions need a return value and types for **all** arguments:

```
int calculateSomething(int input1, bool input2) {  
    // do something  
    return 5;  
}
```

Procedures that do not return anything have the return type `void`.

### Expressions, Conditionals and Loops

A conditional looks like this:

```
if (<condition>) {  
    // then case  
} else {  
    // else case  
}
```

The `else` case is optional.

In conditions, the operators `!` (negation), `==` (equality test), `&&` (logical and) and `||` (logical or) are supported.

There is also an in-line conditional expression:

```
println(n > 5 ? "viel" : "wenig");
```

Both `while` and `for` loops are available:

```
while (<expression>) {  
    // repeats until the condition is false  
}  
  
for (<initializer>; <condition>; <action>) {  
    // repeats until the condition is false  
}
```

## Agents

Agents can be declared with the `agent` keyword:

```
agent a1 = calculateSomething(input1, input2);  
start a1;  
start calculateSomething(input1, input2); // anonymous agent
```

Agents do not need to be started immediately when they are declared, and may be anonymous. If an agent is not, waiting for its termination is achieved by the `join` keyword:

```
join a1;
```

## Structures

The `struct` keyword allows declaring structures that can encapsulate data. It is also possible to define procedures in them, which usually operate on the encapsulated data. Only methods are accessible from outside the structure.

```
struct Count {  
    int a = 0;  
    int plusPlus() {  
        a++;  
        return a;  
    }  
}
```

Instances of a structure are declared just like variables, but using the name of the structure. The available methods can then be accessed using the `.` notation.

```
Count s;  
println(s.plusPlus());
```

## Call-By-Value / Call-By-Reference

When evaluating procedure calls, arguments with one of the primitive types `bool`, `int` and `string` are initialized with **copies** of the passed value (Call-By-Value).

For arrays, struct, monitor, mutex and channels, a reference to the **same** object is used (Call-By-Reference).

```
void p(int i, int[1] a) {
    i = 42;
    a[0] = 1337;
}

mainAgent {
    int i;
    int[1] a;

    println(i + " " + a[0]); // prints "0 1337"
}
```

## Message Passing

PseuCo features channels for communication between agents.

### Channels

Channels can be synchronous or asynchronous, and have any of the primitive types `bool`, `int` or `string`. Asynchronous channels are declared with a fixed capacity.

```
boolchan chan1; // synchronous, type bool
intchan7 chan1; // asynchronous, type int
```

Unless full, channels can accept values from a sender. Unless empty, channels can emit values to a receiver, using a first-in first-out policy:

```
intchan7 chan2;
chan2 <! 7; // send 7
chan2 <! 8; // and send 8
<? chan2; // receive 7
int x = <? chan2; // now x will have value 8
```

Attempting to receive from an empty channel blocks the receiver, attempting to send to a full channel blocks the sender. Receiving is destructive.

Arrays come in handy for declaring multiple channels:

```
intchan100[4] chan;
start anAgent(chan[0], chan[1]);
```

When declaring a procedure, generic channel types without capacity must be used:

```
intchan5 c;
void p(intchan c) {}
p(c); // valid procedure call
```

Such procedures will work with any synchronous or asynchronous channel.

### Select-Case Statements

A `case` clause consists of the keyword `case`, followed by a receive expression (`<?`) or a send expression (`<!`), a `:` and a statement.

A `select-case` statement consists of one or more `case` clauses, followed by an optional `default` clause.

```
select {
  case chan1 <! a: {
    // several statements in brackets
  }
  case b = <? chan2:
    // or just one
  case <? chan3:
    // receive and forget
  default:
    // always enabled
}
```

## Shared Memory

In `pseuCo`, several agents may share access to some variables. If of primitive type, such variables need to be declared globally, while instances of structures can also be shared by passing them as arguments (owed to `Call-by-Reference`).

If an agent can write to some variable while at least one other agent can read or write the same variable, we are facing a **data race**. Programs with data races are considered **incorrect**.

## Explicit Locking

The keyword `mutex` is used to declare a **lock** (or **mutex**). Locks are used to coordinate access to data by preventing concurrent access, preventing data races. Locks can be locked with `lock` and unlocked with `unlock`:

```
mutex m; // declaration
int i = 5;

void dec() {
  lock m; // give it to me
  i--; // no data race
  unlock m; // done
}
```

There is no built-in correspondence between a lock and the data guarded by that lock. It is the programmers responsibility to make sure every access to the guarded data is protected correctly.

## Implicit Locking

The keyword `monitor` instead of `struct` declares a **monitor** – a structure that makes use of an **implicit** lock: There is a built-in mechanism that locks this lock **before** any monitor method call is effectuated and unlocks it only **after** the method's `return` has happened. This ensures that the encapsulated data is properly guarded against data races:

```

monitor Count {
    int a = 0;
    int plusPlus() {
        a++;
        return a;
    }
}

```

Monitor methods may need to wait for specific **conditions** concerning the data encapsulated by the monitor before effectuating certain operations on them. Conditions are declared with the keyword `condition`. The waiting is done by `waitForCondition` which only proceeds once the condition is satisfied. Internally the implicit lock of the monitor is unlocked so as to allow others to proceed and operate on the encapsulated data, so that eventually the condition can become satisfied. Agents can indicate that a condition is now satisfied using `signal` or `signalAll`. This wakes up one or all of the agents waiting on the condition, respectively. On re-acquiring the lock, these agents only proceed if that condition is indeed satisfied, otherwise, they will wait again.

```

monitor Count {
    int i;
    condition notNull with (i >> 0);

    void inc() {
        i++;

        // now someone can do a dec()
        signal notNull;
    }

    void dec() {
        waitForCondition notNull;
        i--;
    }
}

```

## CCS

CCS is a process calculus to model concurrent systems. PseuCo.com uses a pragmatic extension of CCS with value passing, sequencing and termination.

If  $P$  and  $Q$  are valid CCS processes, then the following are valid CCS processes as well:

| Process     | Semantics   |
|-------------|---|
| $\emptyset$ | cannot perform any action   |
| $1$         | terminated process – emits $\delta$ , behaves like $\emptyset$ afterwards |
| $a.P$       | performs action $a$ , behaves like $P$ afterwards                         |
| $P + Q$     | non-deterministic choice between $P$ and $Q$                              |

|                              |   |
|------------------------------|---|
| $P \mid Q$                   | concurrent execution of $P$ and $Q$   |
| $P \setminus \{a, b, c\}$    | behaves like $P$ except that actions $a$ , $b$ , $c$ are <b>not allowed</b>         |
| $P \setminus \{*, a, b, c\}$ | behaves like $P$ except that <b>only</b> actions $a$ , $b$ , $c$ are <b>allowed</b> |
| $P; Q$                       | behaves like $P$ until it terminates, then performs a $\tau$ -transition to $Q$     |
| $X$                          | behaves like $P$ if $X$ was defined before as $X := P$                              |

A CCS action is any combination of letters starting with a lower-case letter. Some of them have a special meaning:

| Input Sequence | Action           | Meaning                                       |
|----------------|------------------|---|
| $i$            | $\tau$ (tau)     | represents in internal, non-observable action |
| $e$            | $\delta$ (delta) | signals that a process has terminated         |

CCS actions may contain a  $!$  or  $?$ . If both counterparts can be executed at the same time, they can synchronize, resulting in an internal  $\tau$  transition. After  $!$ , a sending expression can be given, the value of which will be bound to the input variable given after  $?$ . Input variables may be bound to a range, defined as  $\text{range } R := \emptyset..0$ , with  $:R$ . Ranges of string length can be specified, too:  $\$. \$\$$  allows strings of length 1 to 3.

A process name is any combination of letters starting with an upper-case letter.

Process definitions may contain an arbitrary number of variables in square brackets, which must be given when a process is called.

To get examples for valid CCS terms, create a new CCS file and select a template.

## Offline Mode

The application is available offline automatically if supported by your browser. Just reopen pseuCo.com (<http://pseuco.com/>) to access it.

## Frequently Asked Questions

### My LTS graph shows only one state, but I'm sure there must be some transitions!

The graph is not shown automatically, since graphs can be too huge to render in a readable way. Click on a state to *expand* it, and you will see all successors of the state. Click here for more information.

### In the LTS graph, I can see the states, but transitions aren't drawn completely: I just see their label text, but no arrow!

You're using Internet Explorer, right?

This is a known bug that Microsoft has not fixed yet. Please, tell Microsoft you can reproduce this bug here (<https://connect.microsoft.com/IE/feedback/details/781964/>). Then, tick this box:

☐ do not show arrowheads

After you've done that, the arrows will be rendered without the arrowhead (unluckily, this means you cannot see the direction of the arrow) – but at least, they will be rendered at all.

If Microsoft fixes this bug in a future version of Internet Explorer, uncheck the box above to render the arrowheads again.

If you do not want to wait, try to use another (<https://www.google.com/intl/de/chrome/browser/>) browser (<http://firefox.com/>).

### **On my mobile device, the text editor has some issues, for example, the backspace key is not working reliably.**

We are sorry – the text editor we are using has issues with soft keyboards in some browsers. You can try switching to a different browser.

Please report this issue and provide additional details about your device, browser, and how to replicate this issue, so we can try to fix it.

In the meantime, you can try ticking this box:

☐ do not use the advanced text editor

If you do so, you will not see the normal text editor, but a simple text box. This should work without issues, but you will see no error messages in the editor.

### **On one of my devices, the application just shows a blank screen.**

This can occur if a corrupted version of the application is downloaded. Please contact us with details so we can try to fix this issue.

In the meantime, try to clear the *application cache*, which is different from the normal cache. Here is how you can do that in some modern browsers:

| Browser           | Steps   |
|-------------------|---|
| Chrome            | open the URL <code>chrome://appcache-internals</code> → find this application → Remove  |
| Firefox           | Menu button → Options → Advanced → Network → find the application in the list on the bottom → Remove  |
| Internet Explorer | Gear icon → Internet Options → Settings (in the section Browsing history → Caches and databases → find the application in the list → Delete |
| Safari on iOS     | Settings → Safari → Advanced → Website Data → find the application in the list on the bottom → Edit → Delete icon → Delete                  |



# Glossary

## Ace

a JavaScript-based text editor – see <http://ace.c9.io/> 7, 8, 47, 49

## AngularJS

a rich JavaScript framework that helps building dynamic HTML-based applications by enhancing HTML – see <https://angularjs.org/> 8, 9, 13, 15, 63

## Apache

a HTTP server – see <http://apache.org/> 5, 12, 29, 47, 49, 50

## Bézier curve

a curved described by several (usually four) so-called control points, defining not only the start and end point, but also points the curve will bend towards – see [https://en.wikipedia.org/wiki/B%C3%A9zier\\_curve](https://en.wikipedia.org/wiki/B%C3%A9zier_curve) 19

## Bower

a package manager for web applications – see <http://bower.io/> 7, 47, 48

## Browserify

a JavaScript tool to bundle JavaScript-files with their npm dependencies to be runnable in a browser environment – see <http://browserify.org/> 8

## CoffeeScript

a programming language that cross-compiles to JavaScript – see <http://coffeescript.org/> 8

## controller

refers to application code that provides the functionality behind a user interface and manages the data model 8, 9

## CSS

a language to describe the appearance of HTML elements 8, 47, 49

## D3.js

a JavaScript library for visualizations based on SVG elements – see <http://d3js.org/> 18

## directive

an AngularJS-specific term for a HTML extension: an element that provides a part of the user interface, consisting of both HTML code describing the UI, and JavaScript code providing the functionality 8, 17

## factory

an AngularJS-specific term for a singleton: a component that provides a specific function, and must be initialized once before it can be used arbitrarily often 8

**Git**

a version control system – see <http://git-scm.com/> 5, 30, 47, 48

**grunt**

a JavaScript-based build system – see <http://gruntjs.com/> 5, 7, 35, 48

**jshint**

a tool to find common issues in JavaScript code – see <http://jshint.com/> 7, 48

**node.js**

a command-line JavaScript interpreter – see <http://nodejs.org/> 29

**npm**

the Node Package Manager, part of `node.js` – see <https://www.npmjs.org/> 5, 7, 30, 47, 48, 63

**partial**

fragment of an HTML file defining a part of the UI 9, 17, 47, 48

**PEG.js**

a JavaScript-based parser generator to generate JavaScript parsers for arbitrary languages – see <http://pegjs.majda.cz/> 8

**PseuCoCo**

a compiler from `pseuCo` to CCS code – see section 2.3 3, 4, 7, 8, 17

# Bibliography

- [1] Sebastian Biewer. A compiler for pseuCo to CCS, 2013.
- [2] Jaana Eloranta. Minimizing the number of transitions with respect to observation equivalence. *BIT*, 31(4):576–590, 1991.
- [3] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
- [4] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 228–240. ACM, 1983.
- [5] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.
- [6] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.



# Legal Notes

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

## Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

## Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

## Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 30.09.2014

---

Felix Freiburger