CL-INSTANS

Instructions for Installation and Use

Authors: Esko Nuutila, Mikko Rinne, ...

Document History:

Date:	Ver:	Description:	By:
5.12.2013	0.0.1	First version created	Mikko
8.12.2013	0.0.2	Added description on installing and running	Esko
11.12.2013	0.0.3	Added some notes to explain that only SBCL and the "configure"-script are needed. Added the list of commands from "bin/instans"	Mikko
19.3.2014	0.0.4	Updates to the latest version. Added comments on configuration scripts side effects. Went through the command line parameters and added descriptions.	Mikko, Esko
22.5.2014	0.0.5	Documented the aggregate library as an example. Removed the -i input parameter.	Mikko
23.5.2014	0.0.6	Added some guidelines on memory management in queries.	Mikko

Table of Contents:

Environment

The current version of INSTANS has been developed using Steel Bank Common Lisp (SBCL). The code is likely to work also in other Common Lisp environments, but has not been tested in any others yet.

After installing INSTANS it can be run from the command line (shell) either by using SBCL or directly by giving command line arguments listing the Sparql rule and RDF triples files to be used.

The system has been developed and tested in an OS X (Mac) environment. It is should be straightforward to operate also in other *nix-like environments. Windows is likely to be a bit more exotic.

Common Lisp Installation

Note: In OS X everything else except SBCL is currently installed - if needed - with the `configure` script explained below, so the other components are only listed here for reference.

Steel Bank Common Lisp: We are currently using v. 1.1.16, but v. 1.1.12 and higher are expected to work.

Steel Bank Mac OS X Installation: SBCL is available on Brew. Once Brew is installed, SBCL can be installed from the shell with:

```
$ brew install sbcl
```

Brew is highly recommended for keeping up-to-date with future SBCL releases!

ASDF: Included in the latest SBCL:s. If using another platform you may need to install it separately.

Quicklisp: A truly excellent library manager, must-have also for other reasons than INSTANS. Installation instructions are here, but if you don't have it yet, the configuration script below will install it.

Downloading, configuring and building INSTANS on OS X (Mac)

1. Clone the repository from github.

```
    git clone https://github.com/aaltodsg/instans.git
```

2. Change to directory `instans`

```
cd instans
```

3. Run the configuration and build script

```
./configure
```

It initializes the system:

- Checks that your Lisp configuration is suitable for INSTANS
- Downloads and installs quicklisp.lisp, if it is not installed yet
- Creates a custom SBCL core that contains INSTANS
- Downloads Sparql test cases from w3.org, if not downloaded yet.
- Downloads JQuery DataTables that are used to present the Sparql test case coverage.

The `./configure` script assumes that `curl`, `tar`, and `unzip` are installed in your system.

Additional options:

```
./configure --clean
```

Re-compiles everything, i.e. cleans any previous installations of INSTANS. WARNING: The --clean option also re-installs quicklisp, removing e.g. quicklisp-slime-helper in the process. Use with caution, if you already have a custom installation and like your settings!

Other tools

If you have any plans for developing in Common Lisp, three other components are highly recommended:

1. A good Emacs. We are using Aquamacs on OS X.
2. Slime: Creates the Lisp development environment on emacs. It is available also on quicklisp, which makes updating easier.
3. Quicklisp-slime-helper: "Makes it easy to use Slime from quicklisp"

Some configuration is needed. The following slime-related lines in your "~/.emacs" file do the trick:

```
------------ .emacs ---------- .emacs ------------
(load "~/quicklisp/slime-helper.el") ; your quicklisp directory
(add-to-list 'load-path "~/Dev/Lisp/Slime/")  ; your SLIME directory
(add-to-list 'load-path "~/Dev/Lisp/Slime/contrib")  ; your SLIME directory
(setq inferior-lisp-program "/usr/local/bin/sbcl") ; your Lisp system path
(require 'slime)
(require 'slime-autoloads)
(slime-setup '(slime-fancy slime-asdf))
(slime-setup '(slime-repl slime-editing-commands slime-scratch slime-asdf))
```

```
(add-hook 'slime-mode-hook
        (lambda ()
           (unless (slime-connected-p)
             (save-excursion (slime)))))
-----------------------------------------------------------
```

Using INSTANS from the command-line

INSTANS executable can be found in directory "instans/bin". Use the command:

```
      $ instans [ arguments ]
```

to run INSTANS on command line. Running `instans` with no arguments prints the following list:

```
Usage: instans [ -h | --help | -v | --version | { <configuration-option> } ]

General options:
-h | --help                             Print this message and exit.
-v | --version                          Print version information and exit.

Configuration options:
-n <string>   | --name <string>         Use <string> as the name of the system.
-d <dir>      | --directory <dir>       Use <dir> as the prefix for file lookup.
                                        You can use a file or an URL as <dir>
-b <url>      | --base <url>            Use <url> as the base.
-g <graph>    | --graph <graph>         If <graph> is 'default' add the following
                                        inputs to the default graph. If it is <url>
                                        add them to the graph named by <url>
-r <rules>    | --rules <rules>         Use rules in <rules>.
-t <input>    | --triples <input>       Read input based on <input>.
-o <output>   | --output <output>       Write output based on <output>.
-f <commands> | --file <commands>       Read options from <commands>.
--triple-processing-policy <policy>     See the documentation.
--rule-instance-removal-policy <policy> See the documentation.
--rete-html-page-dir <dir>              Create an HTML page about the Rete network.
--queue-execution-policy <policy>       See the documentation.
-e            | --execute               Execute the system. This is done by default at
the end of processing all arguments.
--debug <phases>                        See the documentation.
--verbose <phases>                      Same as --debug.

See the documentation for description of the parameters of the options above.
```

Command line specifiers (-d, -g, -n or -b) persist until replaced by a new instance of the same specifier.

Directory (-d or --directory) [Status: Implemented]

Use <dir> as the prefix for all file or URI lookups (until changed with another -d option). The default is the current directory. Both a file system name and URL are valid.

Base (-b or --base) [Status: Implemented]

Use <url> as the base for triple input parsing & IRI expansion (as specified in section 5 of RFC 3986). The default value is "http://".

Examples:

- "-b file:///." is the current directory of the file system.

Graph (-g or --graph) [Status: Implemented]

If <graph> is 'default' or not specified, add the following inputs to the default graph. If it is <url> add them to the graph named <url>. Multiple -g parameters can be used in one command line to input to multiple graphs.

One file can currently only contain input to one graph.

Rules (-r or --rules) [Status: Implemented]

Use SPARQL rules in <rules>. Valid rules can be either in the local file system or URI:s, in which case the file is fetched from the specified address. Expansion is based on the prefix specified with -d.

Input (-t, --triples) [Status: Implemented]

Read triple input from <input>. Valid inputs can be either in the local file system or URI:s, in which case the file is fetched from the specified address. Expansion is based on the prefix specified with -d.

Input from a Unix pipe can be read using "-t /dev/stdin". E.g.:

```
$ cat mytriples.ttl | instans -r myqueries.rq -t /dev/stdin
```

Currently only turtle serialization is supported.

Output (-o or --output) [Status: File support implemented]

Write output to <output>. Currently <output> can only be a file. If not specified, the results are sent to the standard output.

Future additions are to include definition of an output format and separation of SELECT and CONSTRUCT outputs.

Read options from <file or url> (-f or --file) [Status: Implemented]

The file may contain the same options as the command line. The file must contain the LONG names of the options WITHOUT the "--" prefix. E.g. option "--rules rulefile.rq" would become "rules rulefile.rq".

Execute the system (-e or --execute) [Status: Implemented]

By default the Rete-network is executed at the end of processing all arguments. The -e parameter can be used to request earlier execution of Rete.

Define system name (-n or --name) [Status: Implemented]

Shows as a "key" when executing the system and as the name of the (GraphViz) graph in the HTML representation of a generated Rete network.

Rete HTML Visualization (--rete-html-page-dir) [Status: Implemented]

Creates an HTML page of the constructed Rete network to the specified directory.

Triple input policy (--triple-input-policy)

Specifies how many triples are processed at once from the input. The default is single-triple.

- single-triple: Process one triple at a time
- triples-block: Processing takes place every time a non-terminal according to Turtle grammar rule 6 is identified.
- all: Reads the whole input before doing any processing

Implemented in: src/rete/interpreter/lisp: process-triple-input

Triple processing operations (--triple-processing-operations)

A ':'-separated list of (add|remove|execute). The default is add:execute.

- add: Add the triples, which have been read based on the triple input policy, to the Rete engine.
- remove: Remove the triples, which have been read based on the triple input policy, from the Rete engine.
- execute: Execute the rule instances based on the queue execution policy.

Rule Instance Removal Policy (--rule-instance-removal-policy)

Relates to the rule instance execution queue processing. A ':'-separated list of (remove|...).

- Remove: Removes a rule instance from the queue without executing, if the rule condition is not satisfied anymore.
- Keep: Keeps the rule in the queue despite the original condition not being satisfied.

Implemented in: src/rete/interpreter/lisp: rete-remove-rule-instance

Queue Execution Policy (--queue-execution-policy)

A ':'-separated list of (first|snapshot|repeat-first|repeat-snapshot). The default is "repeat-first".

src/rete/interpreter/lisp: execute-rules

- first: Takes the first rule from the queue and executes it.
- repeat-first: Repeats "first".
- snapshot: Removes all rules from the current queue and processes them. Only takes new rules after the current snapshot has been processed.
- repeat-snapshot: Repeats "snapshot".

Verbose output for debugging (--debug or --verbose)

Print out internal information of the Rete engine for debugging purposes. <phases> can be "true" or "false".

Running tests

The package comes with a few test scripts, which can be used to verify that the installation is working correctly. All tests are in directory "instans/tests".

Syntax tests (SPARQL Compliance Test Set)

Tests INSTANS against the SPARQL test set available from http://www.w3.org/2009/sparql/docs/tests/. Run from ".../instans/tests" with

$ make syntax-tests

The results are written to the "syntax-test-results" subdirectory and HTML-formatted in "index.html".

Note: The current testing procedure cannot identify tests, which have been "removed" late in the process. Therefore the compilation of results also includes tests, which are no longer compliant with the latest syntax.

Aggregate Library

Some examples for computing aggregate values (sum, average, min, max) over time windows are available in directory "instans/tests/input/aggr-window". A test can be run by entering:

```
$ ./aggr-test.sh
```

The related files are:

- aggr-lib.rq: The SPARQL "library" to compute aggregates
- ex_windef.ttl: Window parameters defined in a turtle file
- ex_events.ttl: Some sample events for window trials (xsd:dateTime timestamp)
- ex_int_events.ttl: Sample events with an integer timestamp
- aggr-test.sh: The sample command line to run the example

The library and parameters are documented in the aggr-lib.rq and ex_windef.ttl files. Not everything is practical to parametrize and therefore users are encouraged to directly modify an own copy of aggr-lib.rq e.g. to match more complex input event formats or to format the output as needed.

Additionally the same folder contains a sample of using the SPARQL built-in AVERAGE aggregate for computing averages. The files are aggr-builtin.rq and aggr-builtin.ttl. The example can be executed with:

```
$ instans -r aggr-builtin.rq -t aggr-builtin.ttl
```

Close Friends (a.k.a Friends Nearby)

This case is described in:
Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing Heterogeneous RDF Events with Standing SPARQL Update Rules. In: Meersman, R., Dillon, T. (eds.) OTM 2012 Conferences, Part II. pp. 793–802. Springer-Verlag (2012)

And in this simulation report:
Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: INSTANS Comparison with C-SPARQL on Close Friends. Available online:
http://www.cs.hut.fi/~mjrinne/sw/instans/INSTANS-Simulations-March2012.pdf

At the time of writing the reports the tests were executed with INSTANS v. 0.1 available from the project homepage. To run Close Friends with CL-INSTANS (directory "instans/tests"):

1) With 100 location events: $ make fnb100

2) With 1000 location events: $ make fnb1000

The query results are compared with an earlier execution and the success or failure (based on similarity) is reported.

Using INSTANS with SBCL on command-line

These instructions are for running INSTANS with SBCL from the shell, and do not require or use emacs or slime. The easiest way to start SBCL with INSTANS installed is to use

```
instans/bin/sbcl-instans [ sbcl-arguments ]
```

You can use SBCL command line arguments to load Lisp files (`--load filename)` or evaluate expressions (`--eval expr`) before you enter the read-eval-print-loop. See the documentation of SBCL for all available arguments.

Using INSTANS with Emacs and Slime

TBA.

Working with INSTANS

Changes from INSTANS 0.1 & 0.2:

1) Multiple queries and rules in a single file are separated by a semicolon (";").

(INSTANS 0.1 & 0.2: dot (".") was used as the separator. Semicolon is more compliant with SPARQL.)

2) The graph for each triple input can be specified with a command line parameter.

(INSTANS 0.1 & 0.2: A comment line in the beginning of a triple file was used to specify the graph)

3) Turtle input is functional also in triple input files.

(INSTANS 0.1 & 0.2: Turtle parser was only used for SPARQL. Input files had to be converted to ntriples.)

4) Timed events are not yet available

(INSTANS 0.2: Supported timed events, i.e. events programmed with SPARQL to trigger at certain times.)

Implementation Status

A table of what is or isn't supported should be added here.

Not implemented yet

CONSTRUCT, ASK, DESCRIBE Queries

Currently the only form of getting output is SELECT queries, which format the output as CSV.

SERVICE

No service query support yet.

Memory Management

In stream processors based on time windows and stream-to-relation operators memory management for the end user is pretty simple: Once queries over a window have been processed, the data of the window is deleted by the platform. Not so simple in forward-chaining rule processors such as Rete: Any data that comes in, stays by default in the Rete until deleted. The bonus is that it is possible to compare very old and very new elements in the stream - independent of any time windows. The downside is that memory management is largely left to the user.

In the future there will be a parameter to select, whether incoming data should be deleted automatically after it has been processed by all the queries referencing that data. Once working, this parameter should be sufficient for cleaning up any incoming data.

For intermediate data generated in the queries, one approach is to use the following type of rule:

```
DELETE {
  GRAPH <http://processed_input> {
    ?event1 a ep:EventObject ;
            :hasWindowTime ?windowTime1
  }
}
WHERE {
  GRAPH <http://processed_input> {
    ?event1 a ep:EventObject ;
            :hasWindowTime ?windowTime1
  }
  FILTER EXISTS { GRAPH <http://processed_input> {
      ?event2 a ep:EventObject ;
              :hasWindowTime ?windowTime2 .
      FILTER (?windowTime1 < ?windowTime2)
  }  }
}
```

This rule keeps only the newest matching event in memory and deletes all older ones. Remember to include the complete event information for event1 both in WHERE {} and DELETE {}.

Note: This is also a useful strategy for cases, where multiple queries or rules are expecting the same event. If any of the queries processing the event delete it, the other queries may never receive the event. The rule shown above lets all related queries do their processing before cleanup.