Lisp in Summer Projects Submission

Submission Date	2013-10-01 15:49:23
Full Name	Thomas Greve Kristensen
Country	United Kingdom
Project Name	Propaganda
Type of software	library
General category	library
LISP dialect	Clojure
GitHub URL	https://github.com/tgk/propaganda
Did you start this project?	Yes, all the code is written by me
Project Description	I want to describe my project in this form.
Purpose	This Clojure library is an implementation of the propagator computational model proposed by Gerald Jay Sussman and Alexey Radul in the paper "The Art of the Propagator". This implementation extends upon the original work, as it contains not only an implementation of the original concepts, but also an implementation which does not rely on synchronisation mechanisms on the runtime platform, allowing it to run on the JVM as well as in an internet browser.
Function	Using the library, programmers can express their problem domain in a style similar to, but often simpler than, logic programming. Constraints and values are added to a system representing the problem domain. As more and more information is added to the system, values in the system become more precise. Information flows back and forth, allowing seemingly un-coupled values in the system to be refined automatically.
Motivation	There has been an industrial paradigm switch from object oriented programming towards functional style programming, especially with the advance of Clojure and Racket. On top of these, declarative programming has had a revival, especially

through cKanren and core.logic. Propagators represent a novel method of declarative programming that, for some problems, is easier to describe than logic programming. Until now, propagators have only been available in scheme. With the implementation of this library, propagators will be available through Clojure, and therefore available to run on the JVM and in internet browsers.

Audience

The audience is Clojure programmers with an interest in declarative programming.

Methodology

Propagators represent an interesting computational model in which cells hold on to values. The values are updated by propagators.

A cell is much like a variable, in that it holds on to a value over time. Initially, all cells have no value, but as the program runs, cells can take on values when instructed to do so, either directly by the programmer or by propagators.

Propagators attach themselves to one or more cells. Whenever the value of a cell is changed, the propagator is informed, and it can then choose to take some form of action, typically updating other cells.

Propagation can happen in both directions: if we specify a propagator ensuring cell B is the square of cell A, and a propagator ensuring cell A is the square-root of cell B, we can derive cell B from cell A or cell A from B, depending on what information is available.

As a part of the project, I have created a short illustration of these concepts. It can be found at http://tgk.github.io/propaganda/.

The original implementation of propagators used scheme variables and coordination mechanisms to propagate values through cells. This translated fairly well to Clojure, where cell values could be kept in refs that were updates in dosync statements. When values need to be merged, a globally bound "merge" function is used, as proposed in the original paper.

Not all of these STM mechanism are available in ClojureScript, and a second strategy is therefore included in the library. This strategy uses an immutable map representing the system which can be extended with constraints and values. Such as system contains a "merge" function, that is no longer globally bound, and can very from system to system.

https://github.com/tgk/propaganda/blob/master/doc/stm_vs_system.md

The original implementation used an implementation of generic operators available in scheme. This was not available in Clojure, and this library therefore contains an implementation of generic operators. The enclosed implementation of generic operator designed for this library, but is generally usable.

A goal of the library was to be extensible. Using the generic operator implementation it is possible to extend the propagator model with new datatypes. As a part of this project, a tutorial briefly outlining how to extend the library with support for sets has been made available at;

https://github.com/tgk/propaganda/blob/master/doc/set_datatype.md

Conclusion

The library contains a complete implementation of the propagator strategy outlined in the original paper. Furthermore, it extends the propagator concept to also be

	usable on a system with weaker STM mechanisms than those found in the scheme runtime. The library is usable from both Clojure and ClojureScript, making it possible to run declarative programs utilising the library both on the JVM and in ClojureScript. The original article went into the illustration of some examples that have not been covered in the example section of the project. One of these is the maintenance of multiple conflicting world views, which in the original article were used to express an algorithm for solving Dinesman's multiple-dwelling problem. This requires a quite complicated algorithm to be implemented. The immutable datastructure solution referenced earlier makes it possible to retain old systems, and therefore effectively maintain conflicting world views to solve the multiple-dwelling problem. Doing so would be interesting project. It would not require any extension to the library as it is now, but it would simple be an application of it. As the library is extensible with new datatypes in cells, it would be interesting to see it being used with datastructures available from the target platform, such as classes representing dates and date ranges.
Build Instructions	The library can be built with the commands lein deps lein uberjar The resulting jar file contains an implementation that can be used from both Clojure and ClojureScript.
Test Instructions	Invoking lein test will run all the automated tests. Apart from these, the "example" folder contains instructive examples of how the library can be used.
Execution Instructions	The library can be included as a maven or leinigen dependency, as described in the README. The "examples" folder contains examples that can be executed from either the REPL started in the project with "lein repl". The following repository contains a complete example for setting up an example in the browser: https://github.com/tgk/cljs-propaganda-example
Describe any bugs or caveats	No known bugs. The caveat of declarative programming models in general is that it can be difficult to reason about how results are arrived at. Although the library supports some help in the form of what is called "supported values", it can still be fairly difficult to reason about for users not used to declarative programming.

Screen shots	KEEP CALM AND PROPAGATE ON propaganda.png
Official	I have read rules and have abided by them. I am 18 years of age or older. I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.