Steps towards teaching the Clojure programming language in an introductory CS class

Elena Machkasova, Stephen J Adams, Joe Einertson

University of Minnesota, Morris

Trends in Functional Programming in Education (TFPIE) 2013.

Outline

What is Clojure?

- Clojure is a LISP.
- Developed by Rich Hickey, released in 2007, rapidly gaining popularity.
- Designed to support concurrency.
- Provides multiple immutable persistent data structures (lists, vectors, hash maps, sets, etc.).
- Runs on the JVM, fully integrated with Java.
- Provides REPL (Read Eval Print Loop).

Why the popularity?

- Elegant.
- Efficient (fast bytecode, efficient implementation of data structures).
- Convenient and safe efficient multi-threading.
- Integrates with Java.

Why Clojure in intro CS courses?

- It's a real-life language done well.
- Introduces multiple data structures; abstraction vs implementation.
- Can be used in later courses (concurrency, interoperability with Java, purely functional data structures).
- Can be easily parallelize on multiple cores (no locking, only a tiny change to the program).
- Has a large friendly community (online resources, google groups, open source projects, meetups) - easy to continue on your own.
- Rapidly increasing demand in industry.

Clojure at UMM

- UMM (University of Minnesota, Morris) is an undergrad-only liberal arts campus of UMN, has a small, very active CS department.
- Included Clojure in upper-division courses (concurrency, functional programming).
- Introductory course focuses on problem solving and key concepts, e.g. abstraction, recursion.
- Current project: use Clojure in introductory class.

Technical challenges of teaching Clojure to beginners

A need for a beginner-friendly development environment:

- Text editor.
- Project manager.
- Error handling.
- Some functions behave unexpectedly for beginners.

Development environment: text editor

Currently there are very few options for a text editor for beginner programmers.

What doesn't work:

- Emacs, vim (too complicated for beginners).
- Eclipse plugin Counterclockwise (too large).
- Clojure-specific text editors: Clooj, Catnip (too unstable).

What we would like eventually:

- Light Table: a text editor based on functions, not files; instant evaluation, etc. Still in development.

What we are using:

- jEdit with LISP/Clojure plugins.

Development environment: project setup

Clojure projects are managed by a tool called `leiningen`. We need to include beginner-friendly error handling and functions. Program code can be written in a file or typed into REPL.

We are developing a `leiningen` plugin for creating and running student projects (work in progress).

Error handling

- Clojure error messages are Java exceptions.
- Come with many lines of stack trace.
- Refer to Java types. For example, `(cons 2 3)` causes: `IllegalArgumentException Don't know how to create ISeq from: java.lang.Long`
- We use `try/catch` to catch exceptions and transform them.
- We "filter" stack trace, leaving only student's code.
- We replace types with beginner-friendly ones and rephrase error messages.

Error handling: examples

Code: `(5 6)`

Original:

```
java.lang.ClassCastException: java.lang.Long cannot
be cast to clojure.lang.IFn
```

Transformed:

```
Error: Attempted to use a number, but a function was
expected.
```

Code: `([1 3 2] 5)` (trying to access an element at index 5 in a 3-element vector).

Original:

```
java.lang.IndexOutOfBoundsException
```

Transformed:

```
Error: An index in a sequence is out of bounds
```

Error handling: work in progress

Current work in progress:

- Provide error handling for code typed in REPL.
- Handle compilation errors.
- Developing leiningen plugin to run all student code (file and REPL) inside try/catch.
- Provide hints and examples for error messages ("perhaps you swapped the order of arguments?")

Collections vs sequence abstraction

Clojure collections (lists, vectors, hash maps, sets, etc.):

- ...are stored in a way that optimizes their intended use.
    - lists have constant access time to the beginning and linear to the end.
    - vectors are shallow trees, provide logarithmic access to any position.
- ...have a few functions specialized to a collection type, e.g. `conj` that returns a collection of the same type with a new element added.
    - lists: `(conj '(2 3 1) 4)` results in a list `(4 2 3 1)`.
    - vectors: `(conj [2 3 1] 4)` results in a vector `[2 3 1 4]`.

The difference in behavior is likely to be confusing to beginners.

Collections vs sequence abstraction (cont.)

Sequences are an abstraction for a number of elements (possibly infinite) in a specific order.

- Most Clojure functions work on sequences and return sequences (e.g. `map`).
- It is easier for beginners to program in a collection-independent (i.e. abstract) way.
- We provide several functions that work in a collection-independent way. They return sequences (look like lists):
    - `(add-first '(2 3 1) 4)` results in a sequence `(4 2 3 1)`.
    - `(add-first [2 3 1] 4)` results in a sequence `(4 2 3 1)`.
    - `(add-last '(2 3 1) 4)` results in a sequence `(2 3 1 4)`.
    - `(add-last [2 3 1] 4)` results in a sequence `(2 3 1 4)`.

Collections vs sequence abstraction (example)

Define a function to reverse a sequence, using `reduce` (`fold`).
Note: `defn` = define function, `fn` = anonymous function (lambda), `'()` = empty list, `[]` = empty vector.

```
;; works because conj adds to the beginning of a list
(defn my-reverse [coll]
    (reduce (fn [c x] (conj c x))  '() coll))

;; doesn't work because conj adds at the end of a vector
(defn my-reverse [coll]
    (reduce (fn [c x] (conj c x)) [] coll))

;; abstract approach (works with a list or a vector)
(defn my-reverse [coll]
    (reduce (fn [c x] (add-first x c)) '() coll))
```

Abstraction-based teaching approach

- Students will see both collection-specific and collection-independent functions.
- Collection-independent functions allow focus on problem-solving, make things easier.
- Different collections will be introduced slowly, as needed.
- Understanding the differences between implementation details (collections) and abstraction (sequence) can carry on to Data Structures and Software Development.

Benefits and challenges of teaching Clojure in intro classes

Benefits:

- Clojure has a rich collection of data structures.
- Based on abstraction, teaches good programming skills.
- Used in industry and has a well-developed friendly community.
- Provides opportunities for parallelization.

Challenges:

- Development of beginner-friendly development environment.
- Handling error messages.
- Developing approaches to teaching that present the strengths of Clojure without confusing beginners.
- Developing beginner-friendly documentation and examples.

Acknowledgments and selected references

Selected references:

- Richard Brown, Elizabeth Shoop, et al: Strategies for preparing computer science students for the multicore world. ITiCSE working group reports, 2010.
- Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi: How to design programs. MIT Press, 2001.
- Rich Hickey: The Clojure programming language. Symposium on Dynamic languages, 2008.
- Simon Thompson, Steve Hill: Functional programming through the curriculum. Functional Programming Languages in Education, 1995.

The authors thank Jon Anthony, Brian Goslinga, Nic McPhee, and Simon Hawkin for helpful discussion and Max Magnuson and Paul Schliep for help in testing.