Use of Clojure in Undergraduate CS Curriculum

Elena Machkasova

University of Minnesota, Morris

Boston Clojure meetup, November 8, 2012.

Outline

What is University of Minnesota, Morris?

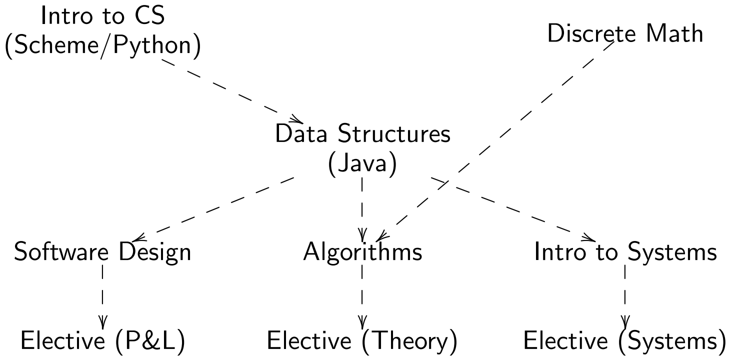
UMM: small public undergraduate liberal arts college in rural MN.

- a campus of UMN, 3.5 hrs away from Twin Cities.
- town of 5000 people, amidst corn fields and cows.
- $<$ 2000 students (15-20 CS majors per year)

Why UMM?

- Functional language in intro class (Scheme).
- Combination of in-depth theory with new technologies (emphasis on testing, group work support).
- Openness to new technologies and ideas.
- Close connections and collaboration between faculty, students, and alums.
- Small group of dedicated faculty.

Structure of CS program (partial)

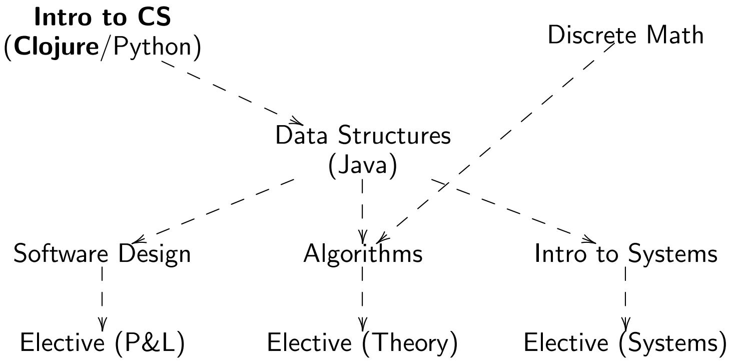


Clojure in CS at UMM

Timeline:

- 2010: Brian Goslinga (UMM CS'11) mentioned Clojure to faculty.
- Spring 2011: *Programming for Parallel Architecture* class explored Clojure, Erlang, and Hadoop.
- Spring 2011: A one-semester project on improving Clojure error messages.
- Spring 2012: Used Clojure in *Programming Languages* class: as an example of a Lisp and as a language for concurrency.
- Stephen Adams (UMM CS'12): a one-semester project on integrating Clojure and Java, with the idea of integrating Clojure into Data Structure.
- Spring 2012: Decision to change our intro class to Clojure.

Where the new Clojure class belongs



Clojure in CS at UMM

Timeline (cont.):

- Fall 2012: a directed study (Joe Einertson, UMM CS'13) on developing a Clojure programming environment for novice students.
- Spring 2013: a directed study on creating sample exercises and improving the environment.
- Fall 2013: the first offering of the new class.
- ...Possibly incorporating Clojure into Data Structures and/or Software Design? Web development? Concurrency?
- ...Possibly a textbook?

The introductory class

Problem Solving and Algorithm Development.

4 credits (roughly 3 in-class hours, no allocated lab time)

Course description: Introduction to problem solving approaches, major programming paradigms, hardware, software, and data representations. Study of the functional programming paradigm, concentrating on recursion and inductively-defined data structures. Simple searching and sorting algorithms. Prerequisites: none.

Why a functional language in an intro CS class?

Why teach a functional language in an intro class if mostly imperative languages are used later?

- Better understanding of recursion (useful for recursive data structures, e.g. binary trees, and recursive algorithms, e.g. sorting)
- Focus on functions.
- Focus on abstraction, generalization, and modularity.

Current intro class at UMM

Currently use How to Design Programs (HtDP) curriculum developed by Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi.

- 35+ students.
- Language: Racket (a version of Scheme).
- Environment: DrRacket. Features several language levels (Beginner → Advanced), libraries for incorporating graphics and interaction.
- Online textbook, series of exercises.
- A large scale open-ended group exercise: develop a game. Interactive, somewhat competitive. Allows students to explore the language and practice/develop design techniques.

Why Clojure in an intro CS class?

Benefits from switching to Clojure:

- Used in industry.
- Interoperability with Java.
- Concurrency features.
- Efficiency.
- *More?*

Challenges in introducing Clojure

Challenges:

- HtDP framework is hard to beat.
- HtDP framework is hard not to repeat - how do we teach Clojure and not Racket?
- Purely functional vs state?
- How much concurrency do we include?
- Environment for new students (convenience, tracing a program, graphics).
- Error messages!!!
- Using off-the-shelf libraries and tools.

Libraries and tools

What we are planning to use :

- Turtle graphics.
- Seesaw (Clojure abstraction over Java Swing) .
- Testing (still to be decided), use of pre- and post-conditions.

What we aren't planning to use in the intro class:

- web dev, including ClojureScript. Introducing web dev concepts would take too much time and distract from the educational goals of the class.

Thanks to Jon Anthony, Brian Goslinga, and Stephen Adams for helpful suggestions and discussions!

Programming environment

Goals:

- Cross-platform (Linux in the lab, Windows, Mac).
- Easy for novice programmers.
- Access to REPL, easy/fast reloading of student code.
- Some way to trace the program.
- The ability to pre-package some libraries and abstract over project management.
- Reasonably easy maintenance (version updates, library updates, etc).

Programming environment (cont.)

Possible setups:

- jEdit (with ClojureShell and LispIndent plugins) + leiningen. *How do we reload code fast?*
- Light Table IDE - a kickstarter project, under development

We are not considering:

- Eclipse with Counterclockwise (too many unclear options, too slow),
- emacs or vim (too confusing to new students).

Tracing a program

Options for step-by-step execution:

- dotrace and such,
- `slingshot` - enhances exceptions (can throw any object; could be used for reporting program execution),
- `spyscope` - a library designed for debugging single- and multi-threaded applications.
- Light Table does some tracing automagically.

Still need to look more into it, depends on the choice of IDE.

Error messages: unreadable and unhelpful for new programmers

Passing a number instead of a function: `(map 2 [1 2 3])`

```
(Exception in thread "main" java.lang.ClassCastException: java.lang.Long cannot
be cast to clojure.lang.IFn
        at clojure.core$map$fn__4182.invoke(core.clj:2469)
        at clojure.lang.LazySeq.sval(LazySeq.java:42)
        at clojure.lang.LazySeq.seq(LazySeq.java:60)
        at clojure.lang.RT.seq(RT.java:474)
        at clojure.core$seq.invoke(core.clj:133)
        at clojure.core$print_sequential.invoke(core_print.clj:46)
        at clojure.core$fn__5378.invoke(core_print.clj:143)
        at clojure.lang.MultiFn.invoke(MultiFn.java:231)
        at clojure.core$pr_on.invoke(core.clj:3306)
        at clojure.core$pr.invoke(core.clj:3318)
        at clojure.lang.AFn.applyToHelper(AFn.java:161)
        at clojure.lang.RestFn.applyTo(RestFn.java:132)
        at clojure.core$apply.invoke(core.clj:614)
        at clojure.core$prn.doInvoke(core.clj:3351)
        at clojure.lang.RestFn.invoke(RestFn.java:408)
        at clojure.main$eval_opt.invoke(main.clj:299)
       ...
        at clojure.lang.RestFn.invoke(RestFn.java:421)
        at clojure.lang.Var.invoke(Var.java:419)
        at clojure.lang.AFn.applyToHelper(AFn.java:163)
        at clojure.lang.Var.applyTo(Var.java:532)
        at clojure.main.main(main.java:37)
```

Improving error messages

Our approach to improving error messages:

- Filtering the stack.
- Wrapping common functions (e.g. map) into pre-conditions with meaningful naming.
- Dictionary of common messages.

Work in progress, by Joe Einertson (UMM CS'13) and myself. The pre-conditions idea is due to Joe.

Filtering the stack

- `clj-stacktrace` library gives access to stack trace info.
- Filter out everything that starts with `clojure` and `java`. More fine-grained and customizable filtering would be preferred eventually. *Any pointers on converting between namespaces and package names?*
- surround all student code with `try/catch`, catch exceptions, pass through a filtering/prettifying function.

Pre-conditions for error messages

Wrap functions in `clojure.core` in our function with a pre-condition:

```
(ns corefns.core
(:refer-clojure :exclude [map])) ; can't overwrite core

(def is-function? fn?)
(def is-collection? coll?)

(defn map [argument1 argument2]
{:pre[(is-function? argument1)(is-collection? argument2)]}
  (clojure.core/map argument1 argument2))
```

Sample error messages

Wrong number of arguments: `(map [1 2 3])`

```
ERROR: Wrong number of args (1) passed to: core$map
Possible causes:
        intro.core/-main (core.clj line 45)
        user/eval5273 (NO_SOURCE_FILE line 1)
```

Wrong type: `(map 2 [1 2 3])`

```
ERROR: Assert failed: (is-function? argument1)
Possible causes:
        corefns.core/map (core.clj line 9)
        intro.core/-main (core.clj line 44)
        user/eval5273 (NO_SOURCE_FILE line 1)
```

Dictionary of common messages

Messages are still confusing: `(2 + 3)`

```
ERROR: java.lang.Long cannot be cast to clojure.lang.IFn
Possible causes:
        intro.core/-main (core.clj line 44)
        user/eval5273 (NO_SOURCE_FILE line 1)
```

Possible meaning: you are using a number in place of a function or an operation.
Ideally would like to remind about the prefix notation. Work for the Directed Study class in the Spring.

Clojure is not Racket

What Clojure-specific features do we teach?

- Pre- and post-conditions; how do they coexist with testing? What testing framework?
- How much of state do we show and explain? atoms, refs, STM, etc? To which degree do we abstract over state?
- Concurrency: how much of automagical parallelism (pmap and friends) can we show/use?
- How much of Java interops do we show/use?

Conflicting goals: focus on nice clean functional code vs learning the real useful language. Can we do both?

Introductory class

Future work for the introductory class:

- Finalize the environment, fill in the gaps (error message dictionary, etc.).
- Examples, exercises, notes.
- How much state, how much concurrency?
- Fall 2013: teach the class.
- Assessment, modify notes, perhaps a textbook?

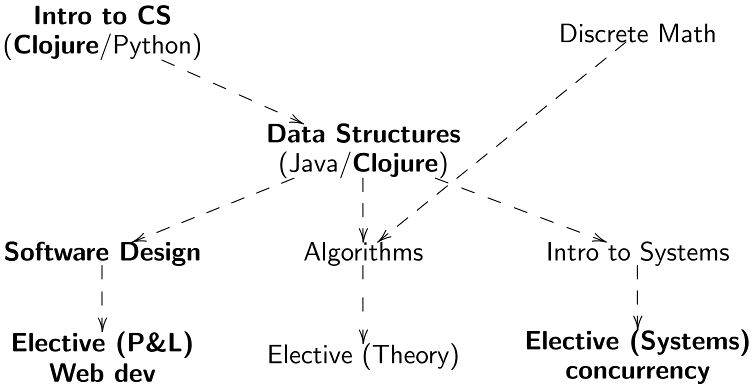
Other classes

Possible other classes to incorporate Clojure:

- *Data Structures:* currently Java, mutable structures (for the most part). Possibly: functional structures, mix of Java and Clojure.
- *Software Design and Development:* abstracting over Swing with Seesaw, web dev stuff (ClojureScript, web framework).
- *Web programming courses:* see above.
- *Parallel/distributed algorithms and systems:* `pmap` and friends.

Challenge: not all students would go through Clojure intro class.

Other classes that could include Clojure



Any suggestions?

- What is a fast way of reloading a file in REPL? Restarting the JVM takes forever. Currently we: remove the namespace, reload the file, then do `in-ns`.
- Are there tools for converting between namespaces and package/class names?

Discussion

Questions? Suggestions? Comments? Critique?