Exploring programming in Thamil (not English) through Clojure

Elango Cheran
SF Bay Area Clojure Meetup
Oct. 2, 2014
• Disclaimer: Let me know if you see anything here that can be improved / corrected.

• Especially my macros!
Motivation

• Unicode is tricky

• Not so bad if your language is "Western"...

• ... and/or economic demand has already created solutions

• There isn't (yet) a lot of reusable libraries for Tamil language processing, etc.
Background - Unicode (1)

• Unicode
  • everything is a stream of 16-bit chars (aka codepoints)
  • all chars from all human languages won't overlap

• Encodings (UTF-8, etc.)
  • just a scheme (codec) to represent (serialize) the codepoints
  • UTF-8 is a "clever hack" that retro-actively supports ASCII
Background - Unicode (2)

• Characters vs. Letters
  • Western languages have 1 char <-> 1 letter
  • Many languages require 1+ chars -> 1 letter
  • The OS & fonts also need to support such languages
• Sorting, searching, substrings, word boundaries, etc. have to be re-thought
Background - Unicode (3)

- Letters vs. Glyphs vs. Ligatures...
  
  - These distinctions seem broken in the Unicode Tamil spec
  
  - Probably the result of letting India handle the specs for all languages spoken in its territory en masse
  
  - Ex: In the Tamil language, க (k) + க (i) = கி (ki)
    
    - In terms of Unicode codepoints, ka (ka) + i = ki (k)
Background - Unicode (4)

- Mac OS X, Win 2000-onwards support Unicode
- Java designed to support Unicode since beginning
  - All chars are 16 bits, not 8 bits
  - In Scala, Clojure, etc., you can use nearly any char in keywords/symbols
    - `val π = 3.1415926`
    - `(def π 3.1415926)`
Unicode in Java Code + Code==Data in Clojure

• In Clojure, why stop at user-made identifiers?

• Everything in code that isn't a literal value is a symbol

• clojure.core fns can be "renamed" easily enough:

  • (def ᵇ⁼Brandon interpose)
    ( ᵇ⁼Brandon "," ["one" "two" "three"])

• But you can't do so for macros and special forms -- they're not values

  • But of we can find a way to do for macros and special forms, we can "translate" Clojure entirely!
Goal, Challenges, Ideas

- **Goal**: Do a "translation" of Clojure from English to some other language -- fns, macros, special forms

- Macros are contagious
  - Once made, manipulating them requires macros
  - I'm a macro newb. Crap.

- Approach: create macros to wrap core macros & special forms
Hardest Part

- Hardest part: finding an editor (& terminal) that supports programming and Tamil properly
  - On Mac OS X, no less!
  - Tried: Emacs 24 for Mac OS X, Eclipse, jEdit, etc.
  - Eventually found: Aquamacs
- Issue: 2 codepoints per character requires support in each of: OS, font, app
2nd Hardest Part

• The macros were the 2nd hardest part!
  • In terms of time required (or lost)

• Here's what we want:
  • (defmacro டசிகு [\& body] `(if ~@body))
Iteration 2: abstract out the repetitive parts

At least for functions, we can sort of do that:

\begin{itemize}
\item (defmacro translate-fn [old-name new-name]
   \((\texttt{\texttt{\texttt{def}} \simold-name \simnew-name}))\))
\item (translate-form take \(\tau\overline{6}\))
\end{itemize}
Macro Time (2)

• Not much gain, as-is

• Could just as easily type `(def \(take\))`

• Although now we can separate data (symbols to translate) from code (macro wrapping special form call)
Macro Time (3)

• Iteration 2b: same thing for macros & special forms

  • Where we had a def-writing macro before, we need a [def]macro-writing macro now

  • (defmacro translate-form
      [old-name new-name]
      `(defmacro ~new-name
         [~'& body#
          ~(~'~old-name ~@body#))])

  (translate-form if  பதிவு)
Macro Time (4)

- Hacked together based on http://amalloy.hubpages.com/hub/Clojure-macro-writing-macros

- Okay, so now all we need to do is put our symbols for translation in a map and we're done, right?
Macro Time (5)

- Iteration 3: storing translation symbols in a map, then calling the macros on each map entry

- Old:
  (defmacro translate-fn
   [old-name new-name]
   `(def ~old-name ~new-name))

- New:
  (defmacro translate-fn-symbol
   [old-name new-name]
   `(def ~(eval new-name) ~(eval old-name)))
Macro Time (6)

- Iteration 3b: storing translation symbols in a map, ..., for macros

- Old:
  `(defmacro translate-form
   [old-name new-name]
   `(defmacro ~new-name
       [~'& body#
        ~(~'~old-name ~@body#))))

- New:
  `(defmacro translate-form-symbol
    [old-name new-name]
    `(defmacro ~(eval new-name)
        [~'& body#
         ~(~'(eval old-name) ~@body#))))
Macro Time (7)

- Iteration 3c: calling the new macros on a symbol map

- I tried doing this from a doseq, but it wasn't possible - "can't eval locals"

- (def test-map '{if தூத்தி})
  (doseq [[oldsym newsym] test-map] (translate-form-symbol oldsym newsym))

  ;=> CompilerException
  java.lang.UnsupportedOperationException: Can't eval locals
Macro Time (8)

- Solution? Yes... more macros

- `(defmacro translate-fns
  [symb-map]
  `(do
    ~@
    (for [[old-form# new-form#] (eval symb-map)]
      `(translate-fn-symbol '~old-form# '~new-form#))))`
Macro Time (9)

- (defmacro translate-forms
  [symb-map]
  `(do
     ~@
     (for [[old-form# new-form#] (eval symb-map)]
       `(translate-form-symbol '~old-form# '~new-form#))))

- I couldn't create these macros without the extra eval form inside

  - Can this be done more simply somehow?
Use 'em!

• (def fns-map '{
    take தேற
    drop தீற
    ...
  })

  (translate-fns fns-map)

• (def forms-map '{
    if அவ்வின்
    when அவ்வின் போன்று
    ...
  })

  (translate-forms forms-map)
Demo Time
Implications

• Can we do this for other languages?
  • Why not?
  • Unicode defines how an ordered stream of characters should be used to create glyphs
Exceptions

• The (ns ...) form at the top of each file can't be "translated" via macros
  • We need it to pull in the macros we need

• Keywords inside core fns/macros/forms
  • :require, :import, :as, :keys, ...

• Stacktraces (it's still Java underneath)

• Printing values of fns

• Reader macros

• Numerals
Is translating Clojure into other languages a good thing? (1)

• Pros:

  • Draw in underserved people worldwide into programming
    • Programming is hard enough, let alone if you need to learn a new language to do so, esp. the 1st or 2nd hardest to learn
    • Let new folks learn programming for the first time through Clojure, not OOPyness!

  • Draw in new demographics of diversity into Clojure community

  • Clojure is well-designed => few changes on pre-existing core forms over time => one of the few languages where translating would have practical value and reduced risks
Is translating Clojure into other languages a good thing? (2)

• Cons:
  • Lose advantage of network effects / fragmentation
    • Searching Google/Stack Overflow
    • Clojure community activity happens in English
  
• My overall take:
  • Just enable more people more ways to express their thoughts
  • The center of the computing world is still in the English-speaking world, hands down, so fragmentation unlikely

• What do you think?
Other Interesting Parts (1)

- Use of tries (prefix trees)
  - Split a word into letters, phonemes,...
  - Change between character encodings, transliteration schemes
- Tries implemented through nested maps
  - Attached a value to each sequence stored in trie
  - Enables to use trie like a map instead of like a set
Other Interesting Parts (1)

• A version of "index-of" for Clojure seqs, like indexOf for Strings

• Generalized fn to compare words basically applies a comparison predicate to seqs

• Does a seq elem-wise comparison fn exist?

• (defn word-before?
  [str1 str2]
  (loop [s1 (str->letters str1)
         s2 (str->letters str2)]
    (cond (not (seq s1)) (boolean (seq s2))
      (not (seq s2)) false
      (not= (first s1) (first s2)) (letter-before? (first s1) (first s2))
      :else (recur (rest s1) (rest s2)))))
Future Work

• Figure out how to use cljx -- compile to JVM and JS
• Figure out how to fit trie code into loom
• Create an Android keyboard input method using library using lein-droid