Exploratory Programming for Formal Concept Analysis
An Introduction to conexp-clj

Daniel Borchmann
TU Dresden
May 24, 2013

Main Question

Why another FCA tool?
Main Question
Why another FCA tool? Many tools which can do things fast and well!

But what if you want to do something else?
What if you want to process your results further on?
What if you want to do something from which you are not completely sure?
Main Question

Why another FCA tool? Many tools which can do things fast and well!

The “Problem”

But what if you want to do something else?
What if you want to process your results further on?
What if you want to do something from which you are not completely sure of?

Solution

Need flexible “FCA scripting”

Hard to achieve with available tools

conexp-clj
Main Question
Why another FCA tool? Many tools which can do things fast and well!

The “Problem”
- But what if you want to do something else?
Main Question
Why another FCA tool? Many tools which can do things fast and well!

The “Problem”
- But what if you want to do something else?
- What if you want to process your results further on?
Main Question

Why another FCA tool? Many tools which can do things fast and well!

The “Problem”

- But what if you want to do something else?
- What if you want to process your results further on?
- What if you want to do something from which you are *not completely sure of*?
Motivation

Main Question
Why another FCA tool? Many tools which can do things fast and well!

The “Problem”

- But what if you want to do something else?
- What if you want to process your results further on?
- What if you want to do something from which you are not completely sure of?

Solution

- Need flexible “FCA scripting”
Main Question
Why another FCA tool? Many tools which can do things fast and well!

The “Problem”
- But what if you want to do something else?
- What if you want to process your results further on?
- What if you want to do something from which you are not completely sure of?

Solution
- Need flexible “FCA scripting”
- Hard to achieve with available tools
Main Question

Why another FCA tool? Many tools which can do things fast and well!

The “Problem”

- But what if you want to do something else?
- What if you want to process your results further on?
- What if you want to do something from which you are not completely sure of?

Solution

- Need flexible “FCA scripting”
- Hard to achieve with available tools
- conexp-clj!
What `conexp-clj` is good for

- Flexible tool to try out new ideas in FCA
- Suitable for exploratory programming, i.e. trying out new algorithms to see if they are correct and how they behave
- Compute non-trivial examples (pedagogical or otherwise)

What `conexp-clj` is not good for

- High performance computations
- Data-intense computations
- GUI enthusiasts
What `conexp-clj` is good for

- flexible tool to try out new ideas in FCA
### What conexp-clj is good for

- **flexible tool to try out new ideas in FCA**
- **suitable for exploratory programming**, i.e. trying out new algorithms to see if they are correct and how they behave
What conexp-clj is good for

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
### What conexp-clj is good for

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
- *FCA scripting*
What `conexp-clj` is good for

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
- *FCA scripting*

What `conexp-clj` is *not* good for

- High performance computations
- Data-intense computations
- GUI enthusiasts
**What conexp-clj is good for**

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
- *FCA scripting*

**What conexp-clj is *not* good for**

- High performance computations
### What conexp-clj is good for

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
- *FCA scripting*

### What conexp-clj is *not* good for

- High performance computations
- Data-intense computations
**What conexp-clj is good for**

- flexible tool to try out new ideas in FCA
- suitable for *exploratory programming*, i.e. trying out new algorithms to see if they are correct and how they behave
- compute non-trivial examples (pedagogical or otherwise)
- *FCA scripting*

**What conexp-clj is not good for**

- High performance computations
- Data-intense computations
- GUI enthusiasts
Main Features of conexp-clj (Overview)

- basic operations on formal contexts
- relational algebra with formal contexts
- transparent IO for formal and many-valued contexts
- scaling for many-valued contexts
- implicational theory and basic attribute exploration
- computing Luxenburger-bases and iceberg concept sets
- lattice layouts and lattice IO (some...)
- a bit of fuzzy-FCA
- interface for Java
- interface for sage
Implementation
Implementation

- implemented in Clojure, a Lisp dialect running on the JVM
Implementation

- implemented in Clojure, a Lisp dialect running on the JVM
- highly portable (JVM)
Implementation

- implemented in Clojure, a Lisp dialect running on the JVM
- highly portable (JVM)
- highly flexible (Lisp)
**Implementation**

- implemented in Clojure, a Lisp dialect running on the JVM
- highly portable (JVM)
- highly flexible (Lisp)
- transparent access to all Java functionality
Implementation

- implemented in Clojure, a Lisp dialect running on the JVM
- highly portable (JVM)
- highly flexible (Lisp)
- transparent access to all Java functionality
- compiled
Prerequisites

Java 1.6 or higher (JRE sufficient)

Download and Installation

Go to conexp-clj’s website: http://github.com/exot/conexp-clj

Move to How to Run

Download one of the .zip files and unpack them where you want

Running

Run ./bin/conexp-clj for a simple (yet sufficient!) command-line interface

Run ./bin/conexp-clj --gui for a “convenient” (but mostly broken) graphical user interface
Prerequisites

- Java 1.6 or higher (JRE sufficient)
Basic Principles

Installing and Running

Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

Go to conexp-clj's website: http://github.com/exot/conexp-clj

Move to How to Run

Download one of the .zip files and unpack them where you want

Running

Run ./bin/conexp-clj for a simple (yet sufficient!) command-line interface

Run ./bin/conexp-clj --gui for a "convenient" (but mostly broken) graphical user interface
Basic Principles

Installing and Running

Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

- Go to conexp-clj’s website: http://github.com/exot/conexp-clj

Run ./bin/conexp-clj for a simple (yet sufficient!) command-line interface

Run ./bin/conexp-clj --gui for a “convenient” (but mostly broken) graphical user interface
Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

- Go to conexp-clj’s website: http://github.com/exot/conexp-clj
- Move to How to Run
Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

- Go to conexp-clj’s website: http://github.com/exot/conexp-clj
- Move to *How to Run*
- Download one of the .zip files and unpack them where you want
Prerequisites
- Java 1.6 or higher (JRE sufficient)

Download and Installation
- Go to conexp-clj’s website: http://github.com/exot/conexp-clj
- Move to How to Run
- Download one of the .zip files and unpack them where you want

Running
Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

- Go to conexp-clj’s website: http://github.com/exot/conexp-clj
- Move to How to Run
- Download one of the .zip files and unpack them where you want

Running

- Run ./bin/conexp-clj for a simple (yet sufficient!) command-line interface
Prerequisites

- Java 1.6 or higher (JRE sufficient)

Download and Installation

- Go to conexp-clj’s website: http://github.com/exot/conexp-clj
- Move to *How to Run*
- Download one of the .zip files and unpack them where you want

Running

- Run "./bin/conexp-clj" for a simple (yet sufficient!) command-line interface
- Run "./bin/conexp-clj --gui" for a “convenient” (but mostly broken) graphical user interface
Code

user=>
user=> 1
Code

user=> 1
1
user=>

user=>
Code

user=> 1
1
user=> (+ 1 2)
3
user=> (javax.swing.JOptionPane/showMessageDialog nil "Wow!"
nil
user=> Daniel Borchmann (TU Dresden)
Code

```
user=> 1
1
user=> (+ 1 2)
3
user=>
```
Code

user=> 1
1
user=> (+ 1 2)
3
user=> (make-context #{1 2 3} #{0 1 2} <=)

Basic Principles

Principle Workflow

Code

user=> 1
1
user=> (+ 1 2)
3
user=> (make-context #{1 2 3} #{0 1 2} <=)
    |0 1 2
--+------
 1 |. x x
 2 |. . x
 3 |. ..
user=>

Daniel Borchmann  (TU Dresden)
Code

user=> 1
1
user=> (+ 1 2)
3
user=> (make-context #{1 2 3} #{0 1 2} <=)
   |0 1 2
--+-----
  1 |. x x
  2 |. . x
  3 |. . 
user=> (javax.swing.JOptionPane/showMessageDialog nil "Wow!")
Code

user=> 1
1
user=> (+ 1 2)
3
user=> (make-context #{1 2 3} #{0 1 2} <=)
   |0 1 2
--+-----
  1 |. x x
 2 |. . x
 3 |. . |
user=> (javax.swing.JOptionPane/showMessageDialog nil "Wow!")
nil
user=>
Example
Example
live!
Code (Functions)

user=>
(defn f [x] (+ x 3))
#'user/f
user=>
(f 5)
8
user=>
(def f (fn [x] (+ x 3)))
Code (Functional Programming)
user=>
(reduce + [1 2 3 4 5])
15
user=>
(reduce * (range 1 10))
362880
user=>
(map f [4 5 6])
(7 8 9)
user=>
(filter odd? [1 2 3 4 5 6])
(1 3 5)
Code (Functions)

user=> (defn f [x] (+ x 3))
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f
user=>

user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=> (map f [4 5 6])
(7 8 9)
user=> (filter odd? [1 2 3 4 5 6])
(1 3 5)
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=>
Code (Functions)

user=> (defn f [x] (+ x 3))

#’user/f

user=> (f 5)

8

user=> (def f (fn [x] (+ x 3)))
Code (Functions)

```clojure
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

Code (Functional Programming)

```clojure
user=>
```
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f

user=> (f 5)
8

user=> (def f (fn [x] (+ x 3)))

Code (Functional Programming)

user=> (reduce + [1 2 3 4 5])
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))

Code (Functional Programming)

user=> (reduce + [1 2 3 4 5])
15
user=>
Code (Functions)

```clojure
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

Code (Functional Programming)

```clojure
user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
```
Code (Functions)

user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))

Code (Functional Programming)

user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=>
## Code (Functions)

```clojure
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

## Code (Functional Programming)

```clojure
user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=> (map f [4 5 6])
```

---

Daniel Borchmann (TU Dresden) conexp-clj May 24, 2013 9 / 14
**Code (Functions)**

```clojure
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

**Code (Functional Programming)**

```clojure
user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=> (map f [4 5 6])
(7 8 9)
```

Daniel Borchmann (TU Dresden)
### Code (Functions)

```clojure
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

### Code (Functional Programming)

```clojure
user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=> (map f [4 5 6])
(7 8 9)
user=> (filter odd? [1 2 3 4 5 6])
```
**Code (Functions)**

```
user=> (defn f [x] (+ x 3))
#'user/f
user=> (f 5)
8
user=> (def f (fn [x] (+ x 3)))
```

**Code (Functional Programming)**

```
user=> (reduce + [1 2 3 4 5])
15
user=> (reduce * (range 1 10))
362880
user=> (map f [4 5 6])
(7 8 9)
user=> (filter odd? [1 2 3 4 5 6])
(1 3 5)
```
Task

Is there a correlation between the number of intents and the number of pseudo intents of a formal context?
Task

Is there a correlation between the number of intents and the number of pseudo intents of a formal context?
(def points
  (map (fn [_]
     (let [ctx (reduce-context (random-context (rand-int 2048)
       (list (count (intents ctx))
       (count (pseudo-intents ctx)))))
     (range 1 1000))))

Daniel Borchmann (TU Dresden)
Code

user=>
Code

```clojure
user=> (doc make-context)
```
Code

user=> (doc make-context)

--------------

conexp.main/make-context
([objects attributes incidence])

   Standard constructor for contexts. Takes a sequence of
   objects,
   a sequence of attributes and either a set of pairs or
   function of
   ...
   nil

user=>
Code

user=> (doc make-context)

----------

conexp.main/make-context
([objects attributes incidence])

   Standard constructor for contexts. Takes a sequence of objects, a sequence of attributes and either a set of pairs or function of

...

nil

user=> (find-doc "formal_context")
Code

user=> (doc make-context)
-------------
conexp.main/make-context
([objects attributes incidence])
  Standard constructor for contexts. Takes a sequence of objects,
  a sequence of attributes and either a set of pairs or function of
... nil
user=> (find-doc "formal_context")
-------------
conexp.fca.implications/proper-premises-by-hypertrans
...
conexp.fca.implications/proper-premises-for-attribute
...
user=>
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation

Alternate Reality
- Reimplementation in Guile (Scheme, Python, Lua, ...)
- C backend for better performance
- Retain flexibility, but increase speed

Daniel Borchmann (TU Dresden)
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation
- Alternate Reality
  - Reimplementation in Guile (Scheme, Python, Lua, ...)
  - C backend for better performance
  - Retain flexibility, but increase speed

Daniel Borchmann (TU Dresden)
The Future

- A better GUI
- Java backend for more performance
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation

Alternate Reality

- Reimplementation in Guile (Scheme, Python, Lua, ...)
- C backend for better performance
- Retain flexibility, but increase speed
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation

Alternate Reality

- Reimplementation in Guile (Scheme, Python, Lua, ...)

Daniel Borchmann (TU Dresden)
conexp-clj
May 24, 2013
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation

Alternate Reality

- Reimplementation in Guile (Scheme, Python, Lua, ...)
- C backend for better performance
The Future

- A better GUI
- Java backend for more performance
- More flexible IO system
- More documentation

Alternate Reality

- Reimplementation in Guile (Scheme, Python, Lua, ...)
- C backend for better performance
- Retain flexibility, but increase speed
Exercises!
Thank You!