EDAA40

Discrete Structures in Computer Science

Administrivia

Jörn W. Janneck, Dept. of Computer Science, Lund University

learnings: notation, concepts, skills

basic concepts, terminology/language & notation for math foundations and discrete structures relevant to CS

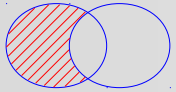
some hands-on experience with simple structures

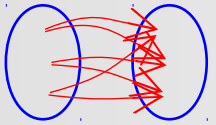
Clojure

overview

$R = \{x : x \notin x\}$

sets $\supset$ relations $\supset$ functions



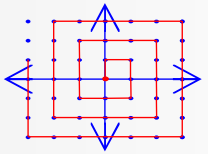


$\heartsuit \subseteq P \times Q$

$f : A \longrightarrow B$

$A \hookrightarrow B$

investigate →

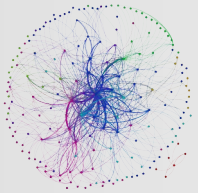


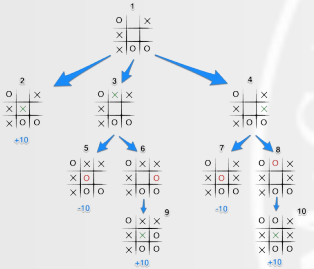
infinity

working with infinite
(or arbitrarily large) stuff ↓

$\cup$

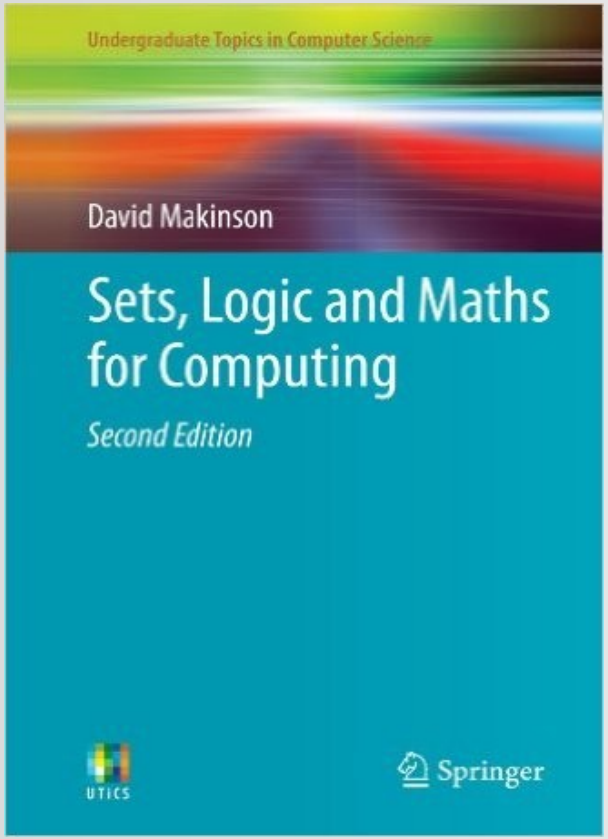
graphs $\supset$ trees





definition, construction,
recursion, induction
(also: proofs, logic)

textbook



Sets, Logic and Maths for Computing (SLAM)

Collecting Things Together: Sets

Comparing Things: Relations

Associating One Item with Another: Functions

Recycling Outputs as Inputs: Induction and Recursion

Counting Things: Combinatorics

Weighing the Odds: Probability

Squirrel Math: Trees

Yea and Nay: Propositional Logic

Something About Everything: Quantificational Logic

Just Supposing: Proof and Consequence

course components

Lectures

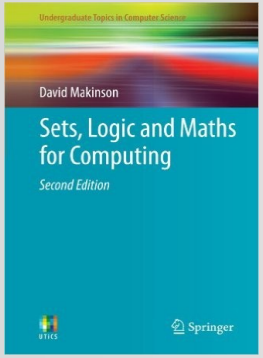
strongly recommended

- basic source of new material

Exercises

strongly recommended

- recap of material in lectures
- diagnosis → seminars, Piazza



- background material, more extensive discussion, alternative explanations, examples
- additional exercises (not always with solutions → seminars, Piazza)

Seminars

optional

- free form: bring questions from lectures, labs, exercises
- I usually bring material, too

Programming Contest

optional

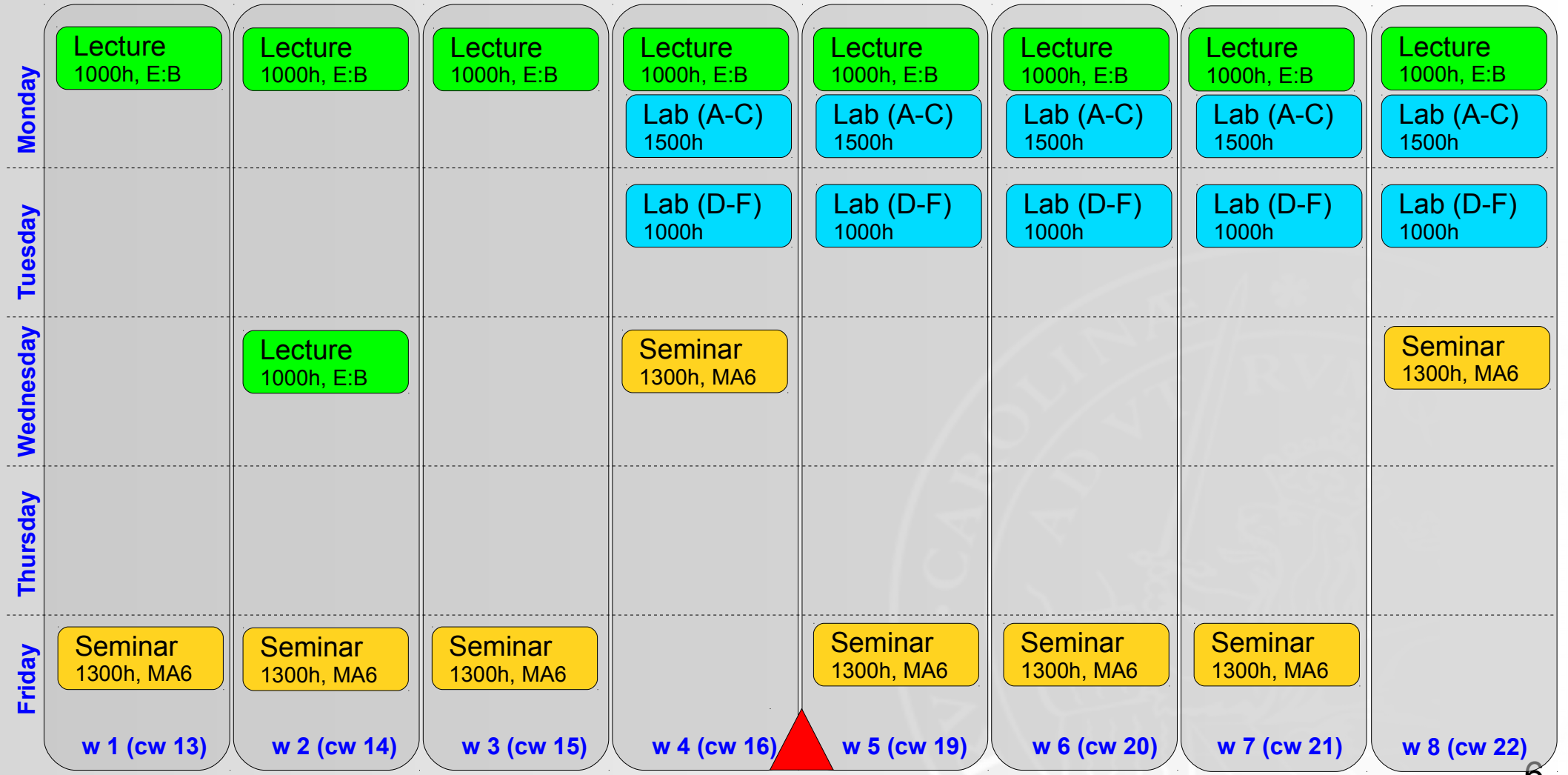
- intended to get you “into” the Clojure language
- win points for the exam!

Labs

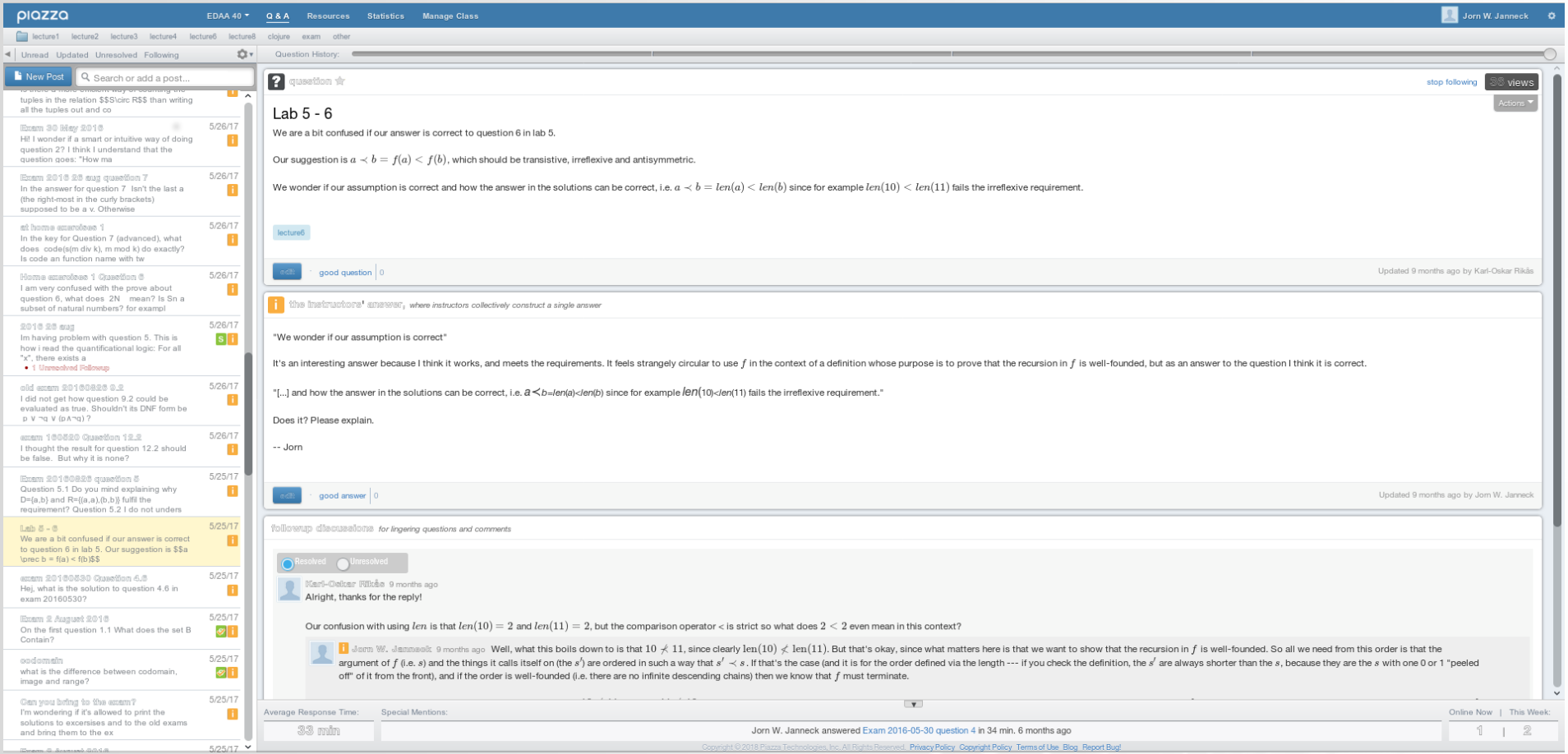
required

- must be passed for credits
- putting math into practice

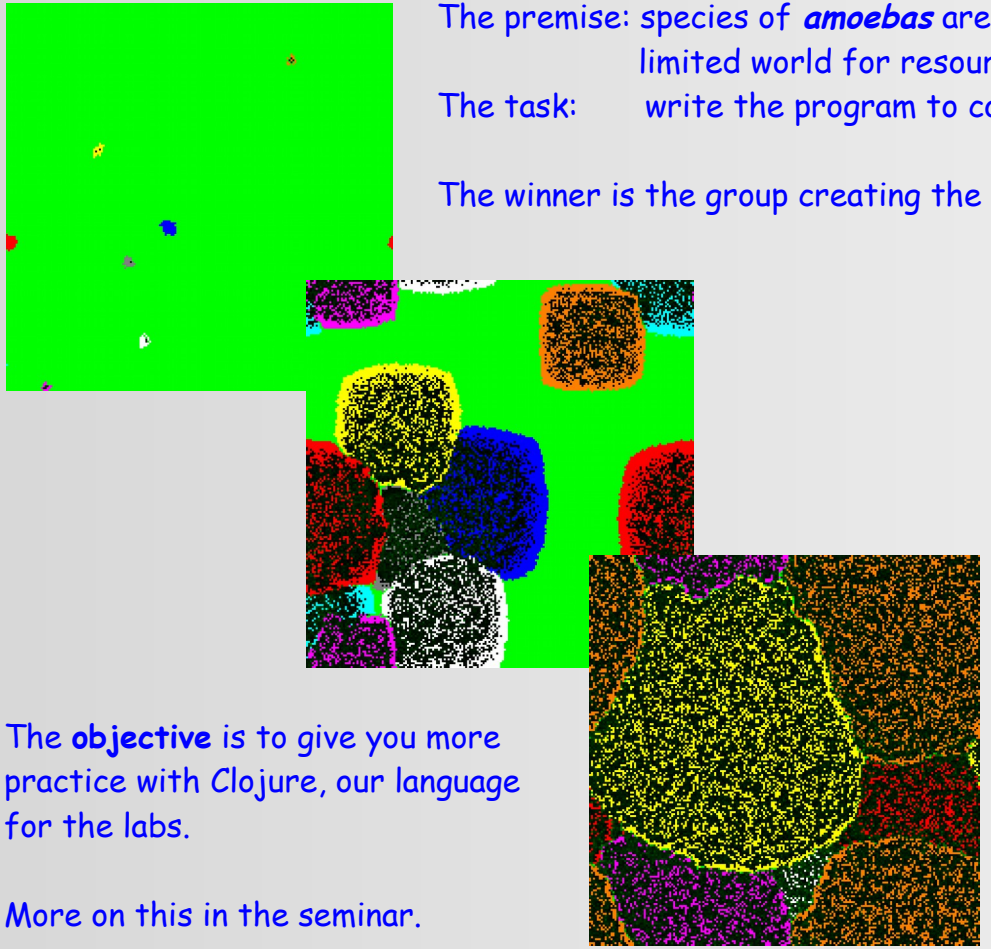
structure and schedule



questions, feedback: Piazza



programming contest (experimental)



The premise: species of ***amoebas*** are competing in a limited world for resources.

The task: write the program to control your species.

The winner is the group creating the most successful amoebas.

```
(defn create-mindless-divider
    [division-prob]

    (fn [energy health species env data]

        (if (< energy (+ MinDivideEnergy (/ (- MaxAmoebaEnergy MinDivideEnergy) 2)))
            {:cmd :rest}
            (if (< (rand) division-prob)
                {:cmd :divide, :dir (rand-int 8)}
                {:cmd :move, :dir (rand-int 8)}
            )
        )
    )
)
```

```
(defn create-slightlybrainy
    [low-energy divide-energy]


    (fn [energy health species env data]
        (let
            [
                do-move (fn []
                            (let                                        ;; otherwise we gotta move...
                                [
                                    empty-nb     (empty-neighbors env)
                                    by-fuel      (sections-by-fuel empty-nb env)
                                ]

                                (if (empty? empty-nb)
                                    {:cmd :rest}                ;; hunker down, we can't move --- FIXME: perhaps we should hit someone?
                                    {:cmd :move :dir (last by-fuel)}    ;; move toward the most fuel
                                )
                            )
                        )
                do-fuel (fn []
                            (if (< MaxFuelingEnergy (:fuel (env Here)))     ;; are we *at* a McDonald's?
                                {:cmd :rest}                                ;; chomp chomp
                                (do-move)
                            )
                        )
                do-hit  (fn []
                            (let
                                [h  (hostiles species Neighbors env)]

                                (if (empty? h)      ;; nobody to hit?
                                    (do-fuel)       ;; eat
                                    {:cmd :hit :dir (Neighbor-To-Dir (rand-nth h))}
                                )
                            )
                        )
                do-div  (fn [empty-nb]
                            {:cmd :divide :dir (rand-nth empty-nb)}
                        )
            ]

            (cond
                (< energy low-energy)           ;; need some chow?
                    (do-fuel)
                (< divide-energy energy)                ;; parenthood!
                    (let
                        [empty-nb   (empty-neighbors env)]

                        (if (empty? empty-nb)       ;; nowhere to put that crib?
                            (do-hit)                ;; then screw parenthood, hit someone
                            (do-div empty-nb)       ;; oooh, look, it's... an amoeba :-(
                        )
                    )
                (hostiles species Neighbors env)            ;; someone looking at us funny?
                    (do-hit)                    ;; whack 'em
                :else
                    (do-fuel)                   ;; let's eat some more
            )
        )
    )
)
```


The **objective** is to give you more practice with Clojure, our language for the labs.

More on this in the seminar.

sidebars

sidebars contain "non-essential" material

i.e.

nothing introduced in a sidebar
will be required in an exam

BUT it may help understand other material,
or put it in context

sidebar

exam

1. It's "all math", so no Clojure programming tasks.

2. It's "open book", so notes, books, printed material, etc. are allowed.

3. No electronic and communication devices of any kind.

4. Stuff that might be tested includes anything discussed in the lectures (except "sidebar" material).