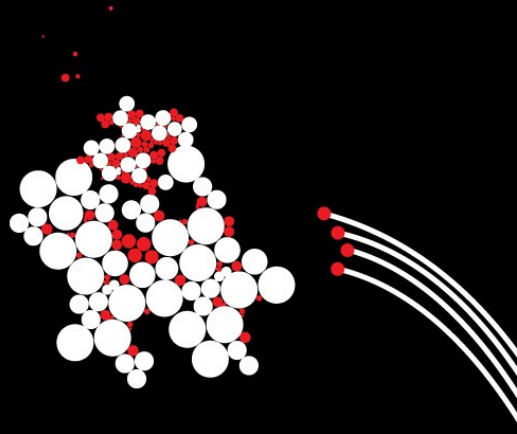
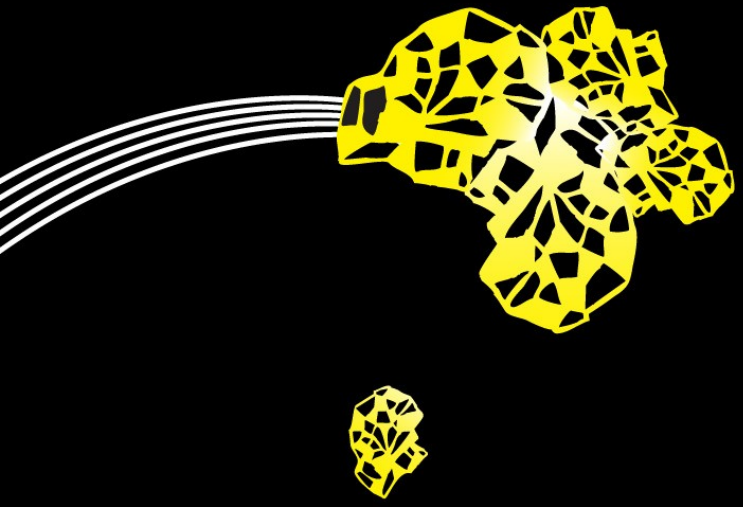
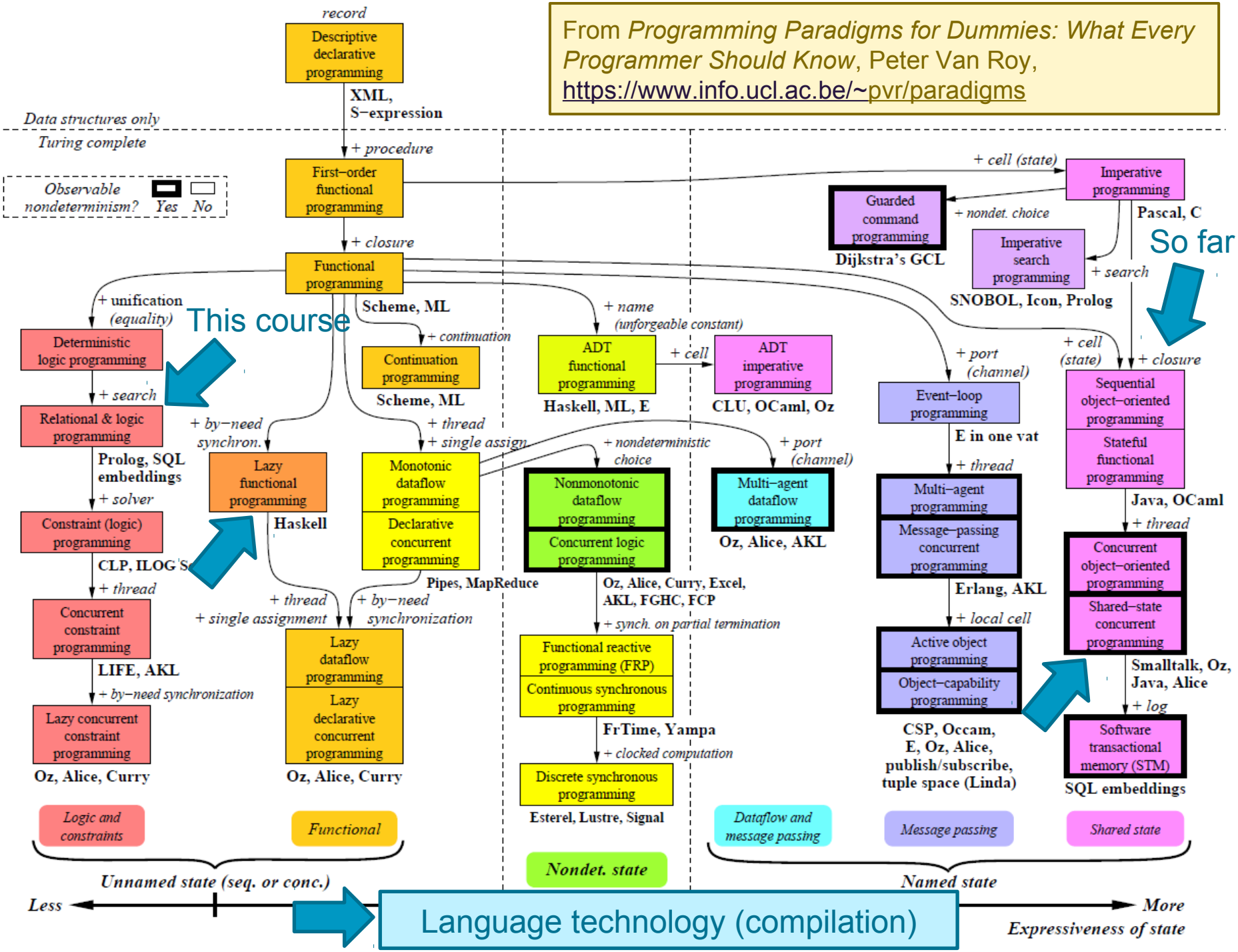
UNIVERSITY OF TWENTE.

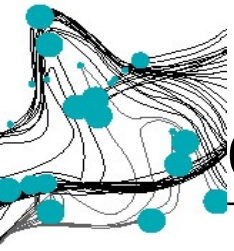


MODULE 8: PROGRAMMING PARADIGMS SETUP AND INTRODUCTION

23 APRIL 2018




From Programming Paradigms for Dummies: What Every Programmer Should Know, Peter Van Roy, https://www.info.ucl.ac.be/~pvr/paradigms
record
Descriptive declarative programming
XML, S−expression
Data structures only
Turing complete
+ procedure
First−order functional programming
Observable nondeterminism? Yes No
+ closure
Functional programming
+ cell (state)
Imperative programming
Pascal, C
Guarded command programming
Dijkstra's GCL
+ nondet. choice
Imperative search programming
SNOBOL, Icon, Prolog
+ search
So far
This course
+ unification (equality)
Scheme, ML
+ name (unforgeable constant)
+ cell (state)
+ closure
Deterministic logic programming
+ search
Relational & logic programming
Prolog, SQL embeddings
+ solver
Constraint (logic) programming
CLP, ILOG S
+ thread
Concurrent constraint programming
LIFE, AKL
+ by−need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ continuation
Continuation programming
Scheme, ML
ADT functional programming
Haskell, ML, E
+ cell
ADT imperative programming
CLU, OCaml, Oz
+ port (channel)
Event−loop programming
E in one vat
+ thread
Multi−agent programming
Message−passing concurrent programming
Erlang, AKL
+ local cell
Active object programming
Object−capability programming
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
Sequential object−oriented programming
Stateful functional programming
Java, OCaml
+ thread
Concurrent object−oriented programming
Shared−state concurrent programming
Smalltalk, Oz, Java, Alice
+ log
Software transactional memory (STM)
SQL embeddings
+ by−need synchron.
Lazy functional programming
Haskell
+ thread
+ single assign
Monotonic dataflow programming
Declarative concurrent programming
Pipes, MapReduce
+ thread
+ single assignment
+ by−need synchronization
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
+ nondeterministic choice
Nonmonotonic dataflow programming
Concurrent logic programming
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
+ synch. on partial termination
Functional reactive programming (FRP)
Continuous synchronous programming
FrTime, Yampa
+ clocked computation
Discrete synchronous programming
Esterel, Lustre, Signal
+ port (channel)
Multi−agent dataflow programming
Oz, Alice, AKL
Logic and constraints
Functional
Dataflow and message passing
Message passing
Shared state
Nondet. state
Unnamed state (seq. or conc.)
Named state
Less
More
Expressiveness of state
Language technology (compilation)



CONTENTS

- **CP**: Concurrent Programming
  - Language: Java, Rust, Haskell
  - Especially: Shared-memory concurrency (threads)

Marieke Huisman

- **FP**: Functional Programming
  - Language: Haskell
  - Lazy evaluation
  - Stand-alone: **FP**

Marco Gerards

- **LP**: Logic Programming
  - Language: Prolog

Jaco van de Pol

- **CC**: Compiler Construction
  - Language: Java + Antlr 4

Arend Rensink

(LOGICAL) BLOCKS RATHER THAN (TEMPORAL) WEEKS

From Blackboard:

Week						Block	Date	Assessment
16	23-Apr b1:d1	24-Apr b1:d2	25-Apr b1:d3	26-Apr b1:d4	27-Apr King's Day	1		
17	30-Apr b1:d5	1-May b2:d1	2-May b2:d2	3-May b2:d3	4-May b2:d4	1/2	3-May	Hard deadline b1
18	7-May b2:d5	8-May b3:d1	9-May b3:d2	10-May Ascension	11-May Bridging	2/3		
19	14-May b3:d3	15-May b3:d4	16-May b3:d5	17-May b4:d1	18-May b4:d2	3/4	14-May	Hard deadline b2
20	21-May Whit Monday	22-May b4:d3	23-May b4:d4	24-May b4:d5	25-May b4:d6	4	22-May 25-May	Hard deadline b3 CC (homework 1)
21	28-May b5:d1	29-May b5:d2	30-May b5:d3	31-May b5:d4	1-Jun b5:d5	5	30-May 1-Jun	Hard deadline b4 LP (project)
22	4-Jun b6:d1	5-Jun b6:d2	6-Jun b6:d3	7-Jun b6:d4	8-Jun b6:d5	6	6-Jun 11-Jun	Hard deadline b5 FP (project)
23	11-Jun b7:d1	12-Jun b7:d2	13-Jun b7:d3	14-Jun b7:d4	15-Jun b7:d5	7	13-Jun 15-Jun	Hard deadline b6 FP (test)
24	18-Jun b8:d1	19-Jun b8:d2	20-Jun b8:d3	21-Jun b8:d4	22-Jun b8:d5	8	18-Jun 22-Jun	CC (homework 2) CP (test)
25	25-Jun b9:d1	26-Jun b9:d2	27-Jun b9:d3	28-Jun b9:d4	29-Jun b9:d5	9	2-Jul	FP (resit)
26	2-Jul b10:d1	3-Jul b10:d2	4-Jul b10:d3	5-Jul b10:d4	6-Jul b10:d5	10	4-Jul 6-Jul	CP (resit) Project

Why? Maintainability!

3-May b2:d3	These signify ultimate sign-off deadlines for lab exercises of the previous block
25-May b4:d4	These signify tests or hand-in dates for projects

ASSESSMENT

- From Blackboard:

Test	Kind	I/G	Weight	Min.	Block:Day
FP exam	written exam	I	15%	5.0 [1]	7:5 (resit: 10:1)
LP/FP projects	project	G	15%	5.5 [2]	6:1/7:1
CP exam	written exam	I	20%	5.0 [1]	8:5 (resit: 10:3)
CC homework	take-home exam	I	20%	5.5 [3]	5:1 & 8:1
Integrating project	project	G	30%	5.5	10:5

[1] In addition, the average of the FP and CP exams must be at least 5,5
[2] Calculated as the weighted average of an LP project (25%) and an FP project (75%)
[3] Calculated as the average of two separate take-home exams

- Note: lab exercises to be signed off weekly
  - To be done in pairs (as is the integrating project)
  - Mandatory & *in time* (end of block)
  - Not part of the grade

HARD RULES

1. Laboratory and project to be done in pairs
   - Also if you are a repeat student
   - I can pair up students; ask!
   - At sign-off time, both students have to be there
2. Laboratory blocks to be finished in time
   - Failing to do so may result in exclusion from remainder of module
3. All parts of the module to be done again for repeat students
   - It would be unclever to just break out last year's solutions!

- We (as teachers) do *not* grant exceptions to 3
  - If your personal situation merits an exception, ask the Examination Board
  - Candidate Board Inter-Active *may be* ground for exception
  - Pandora is *no* ground for exception

CULTURAL CONTEXT

- Attending scheduled sessions
  - There are no obligations to attend (in contrast to signing off; previous slide!)
  - With the exception of Logic Programming, lectures are recorded
  - Attendance implies participation
- Getting in touch
  - When you have questions, ask them!
  - Teachers and TAs expect and welcome this; use the resources you have!
- Plagiarism and fraud
  - All work you show and hand in should be *your own*
  - It is proof that *you* have learned the material
  - See intro to module guide (Blackboard → Course information)
- (In)appropriate attitudes and behaviour
  - All cultures and genders are respected and welcomed here
  - Words spoken in jest may be taken in offense
  - Do *not* let your discontent simmer below the surface

CONTENT: FUNCTIONAL PROGRAMMING

- What does the following Java program do?

```
public int[] seriesUp(int n) {
    int[] result = new int[n*(n+1) /
2];
    int pos = 0;
    for (int i = 1; i <= n+1; i++)
        for (int j = 1; j < i; j++)
            result[pos++] = j;
    return result;
}
```

  - It returns the sequence [1, 1,2, 1,2,3, …, 1,2,3,…,n]

- Ten weeks from now, you will be able to write the equivalent Haskell:

```
seriesUp :: Int -> [Int]
seriesUp n = concat (map oneToN
[1..n])
    where oneToN n = [1..n]
```

Source: *An Example of the Beauty of Haskell. Gregor Ulm*

CONTENT: LOGIC PROGRAMMING

- How do you write a Sudoku solver?
- Ten weeks from now, you'll (almost) be able to write this Prolog:

```
:- use_module(library(clpfd)).

sudoku(Rows) :- // Rows is list of lists
    append(Rows, Vs), Vs ins 1..9, // All elements in range
1..9
    maplist(all_distinct, Rows),  // All rows distinct
    transpose(Rows, Columns),
    maplist(all_distinct, Columns), // All colums distinct
    Rows = [A,B,C,D,E,F,G,H,I],
    blocks(A, B, C), blocks(D, E, F), blocks(G, H, I),
    maplist(label, Rows). // Collapse ranges to single values

blocks([], [], []). // First three (combined) recursively
distinct
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
    all_distinct([A,B,C,D,E,F,G,H,I]),
    blocks(Bs1, Bs2, Bs3).
```

Source: *Prolog Sudoku Solver Explained, Manuel Rotta*

CONTENT: CONCURRENT PROGRAMMING

- What could go wrong in the following Java program?

```
public class NoVisibility {
  private static boolean ready;
  private static int number;

  private static class ReaderThread extends
Thread {
    public void run() {
      while (!ready)
        Thread.yield();
      System.out.println(number);
    }
  }

  public static void main(String[] args
    new ReaderThread().start();
    number = 42;
    ready = true;
  }
}
```

May fail to terminate

May print 0

Ten weeks from now, you'll
- Understand this
- Know how to avoid this

Source: *Java Concurrency in Practice, Brian Goetz Tim Peierls et al.*

CONTENT: COMPILER CONSTRUCTION

- How do you make a computer understand human-readable text?
  - Correctly parse and execute `2 + x * 3` and `(2 + x) * 3` and `2 * x + 3`

```
grammar Expr;

prog : stat+ ;

stat : expr LINE
     | ID '=' expr LINE
     | LINE ;

expr : expr ( '*' | '/' ) expr
     | expr ( '+' | '-' ) expr
     | INT
     | ID
     | '(' expr ')' ;

ID   : [a-zA-Z]+ ;// match identifiers
INT  : [0-9]+ ;    // match integers
LINE : '\r'? '\n' ;    // line ending =
separator
WS   : [ \t]+ -> skip ;// toss out whitespace
```

Ten weeks from now, you'll be able to
- write this grammar
- generate executing code

Source: [The Definitive Antlr4 Reference, Terence Parr]

CONTENT: PROJECT

- Bringing it all together [except for LP]
  1. Define your own programming language
  2. Include concurrency features
  3. Compile to hardware emulator in Haskell
  4. Extend hardware emulator with own “machine instructions”
- Performed in pairs
  - No loners!
- Schedule
  - Starts in Block 7
  - Hand in on last day of quarter
  - Rules for late delivery apply (-1 grade point for every extra day)