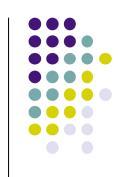
Paradigms of this course (1)



- We defined a paradigm as an approach to programming a computer based on a coherent set of principles or a mathematical theory
 - Different theories of computing result in different paradigms
 (λ calculus, π calculus, first-order logic, Hoare logic, ...)
 - Each theory highlights a different way of programming!
 - Programming is truly a new discipline that is not covered by traditional mathematical theories
- In this course we have covered five important paradigms
 - Why do we need more than one paradigm?
 - Because solving a problem is much easier when done in the right paradigm!

Paradigms of this course (2)



Functional programming

Functions and recursion
Higher-order programming
Single-assignment variables

Object-oriented programming

+ state (cells)

Data abstraction Polymorphism

Inheritance

- We have seen five paradigms
- These paradigms have very different ways of thinking, but their kernel languages are very similar!

Deterministic dataflow

No race conditions
Concurrency transparency
Streams and agents

+ ports (named streams)

+ concurrency (threads)

Multi-agent dataflow

Deterministic dataflow Nondeterminism where needed

Active objects

Object-oriented programming Multi-agent dataflow

Kernel language of the functional paradigm



- <v> ::= <number> | | <record>
- <number> ::= <int> | <float>
- o cedure> ::= proc {\$ <x>₁ ... <x>_n} <s> end
- <record>, ::= <lit>(<f>1:<x>1 ... <f>n:<x>n)

Kernel language of the object-oriented paradigm



Functional paradigm

Extension with cells and exceptions

- <number> ::= <int> | <float>
- o cedure> ::= proc {\$ <x>1 ... <x>n} <s> end
- <record>, ::= <lit>(<f>1:<x>1 ... <f>n:<x>n)

Kernel language of deterministic dataflow



```
• \langle s \rangle ::= \begin{vmatrix} skip \\ | \langle s \rangle_1 \langle s \rangle_2 \\ | local \langle x \rangle | in \langle s \rangle \end{vmatrix} Functional paradigm \langle x \rangle_1 = \langle x \rangle_2 \\ | \langle x \rangle_2 = \langle x \rangle_2 \\ | \langle x \rangle_1 = \langle x \rangle_2 \\ | \langle x \rangle_1 = \langle x \rangle_2 \\ | \langle x \rangle_1 = \langle x \rangle_2 \end{vmatrix} Functional paradigm \langle x \rangle_1 = \langle x \rangle_2  Functional paradigm \langle x \rangle_1 = \langle x \rangle_2  Functional paradigm \langle x \rangle_1 = \langle x \rangle_2  Functional paradigm \langle x \rangle_1 = \langle x \rangle_2  Functional paradigm \langle x \rangle_1 = \langle x \rangle_2 = \langle x \rangle_2  Functional paradigm \langle x \rangle_1 = \langle x \rangle_2 = \langle x \rangle_2
```

- <v> ::= <number> | | <record>
- <number> ::= <int> | <float>
- o cedure> ::= proc {\$ <x>1 ... <x>n} <s> end
- <record>, ::= <lit>(<f>1:<x>1 ... <f>n:<x>n)

Kernel language of multi-agent dataflow



```
• <s> ::= skip

| <s>1 <s>2

| local <x> in <s> end

| <x>1 =<x>2

| <x>=<v>

| <x> =<v>

| <x> then <s>1 </br>
<br/>
<y'
                            if <x> then <s><sub>1</sub> else <s><sub>2</sub> end
                           | \{ \langle \chi \rangle \langle y \rangle_1 \dots \langle y \rangle_n \}
                           case <x> of  then <s>1 else <s>2 end
                           thread <s> end
                           | {NewPort <x> <y>}
                           | {Send <x> <y>}
```

Functional paradigms

Extension with threads

Extension with ports

- <v> ::= <number> | | <record>
- <number> ::= <int> | <float>
- o cedure> ::= proc {\$ <x>₁ ... <x>_n} <s> end
- <record>, ::= <lit>(<f>1:<x>1 ... <f>n:<x>n)

Kernel language of active objects



- Active objects combine the abilities of objectoriented programming with multi-agent dataflow
 - The behavior of each active object is determined by a class, and active objects communicate by message passing
 - The kernel language is the union of the two kernel languages of these paradigms
 - I leave it as an exercise for you to write it down!