A concepts-based approach for teaching programming

SIGCSE 2004
Birds of a Feather Session
March 4, 2004

Peter Van Roy
Université catholique de Louvain
Louvain-la-Neuve, Belgium
pvr@info.ucl.ac.be
http://www.info.ucl.ac.be/~pvr/

The problems of teaching programming

- For our purposes, let us define "programming" broadly as the activity that starts with a specification and leads to its solution on a computer
  - This includes designing a program and coding it in a language
- How can we teach programming without being tied down by the limitations of existing tools and languages?
  - Example: concurrency is both complicated and expensive in Java, so Java-taught programmers get the mistaken impression that it is always so.
  - Example: data abstraction is limited in pure object-oriented languages to a single style, the "object style", so programmers don't realize that there are many other styles, e.g., the "abstract data type" style, each with its own trade-offs.
- How can we teach programming as a unified discipline with a scientific foundation?
  - Not as a set of disjoint paradigms

Our solution: A concepts-based approach

- We start with a small language containing just a few programming concepts
  - We show how to program and reason in this language
- We then add concepts one by one to remove limitations in expressiveness
- In this way we cover all major programming paradigms
  - We show how they are related and how and when to use them together
- Similar approaches have been used before, notably by Abelson & Sussman in SICP
  - We apply it both broader and deeper: we cover more paradigms and we have a simple formal semantics for all concepts
  - We have especially good coverage of concurrent programming

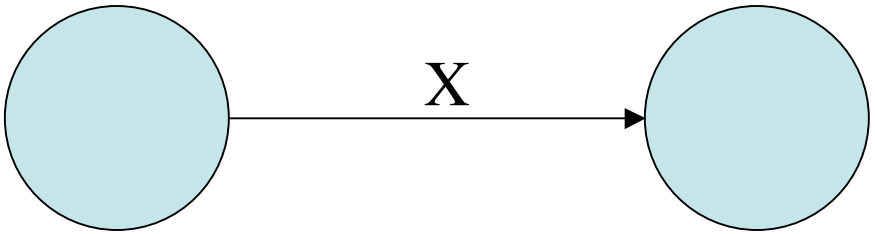
Realizing the approach

- We draw on more than a decade of research in language design and implementation by an international group, the Mozart Consortium
  - We have a software system, Mozart, that can run all the examples
  - We have a simple formal semantics for all the paradigms
- We have been writing a textbook for four years and teaching with a draft for three and a half years
  - The draft has been used in ten universities around the world
  - The textbook is now available for the first time at SIGCSE 2004 from MIT Press: “Concepts, Techniques, and Models of Computer Programming”, by Peter Van Roy and Seif Haridi
- We make available complete course materials for several courses based on the approach

Some courses

- Here are two ways we have taught with the approach
- Single course (Datalogi II at KTH, CS2104 at NUS, second year)
  - Start with functional programming
  - Give declarative techniques and higher-order programming
  - Add concurrency: gives dataflow programming
  - Add communication channel: gives multi-agent programming
- Two course sequence (at UCL, second and third years)
  - First course: similar to the SICP approach (LINF1251)
    - Start with functional programming
    - Give declarative techniques and higher-order programming
    - Add state: lets us cover techniques for data abstraction, such as OOP
    - Explain components and objects
  - Second course: focus on concurrency (INGI2131)
    - Give refresher on functional programming
    - Add concurrency: gives dataflow programming
    - Add communication channel: gives multi-agent programming
    - Add state: gives locks, monitors, and transactions
- Let us give an example to illustrate dataflow concurrency …

Stream communication with dataflow concurrency (1)

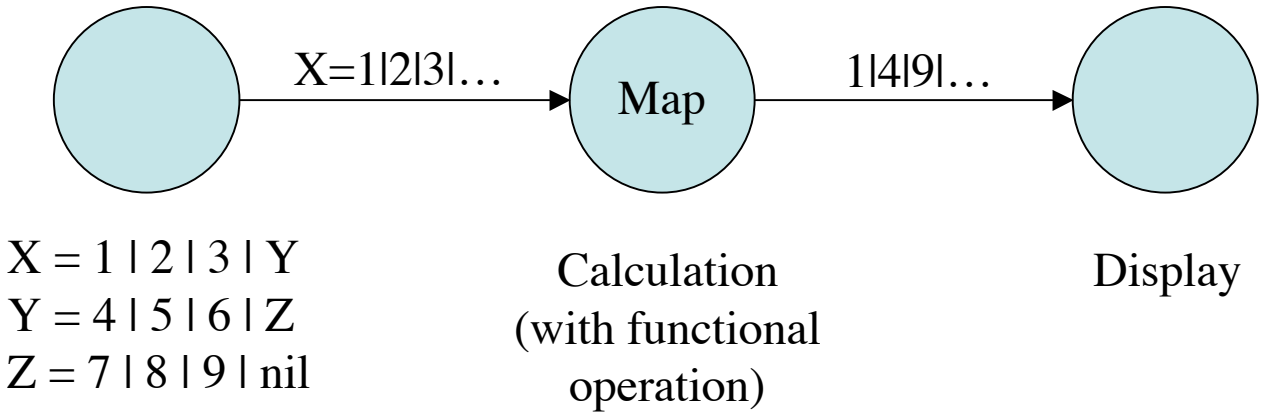


X = all | roads | Y
Y = lead | to | Z
Z = alexandria | nil

Display
(with Browse tool)

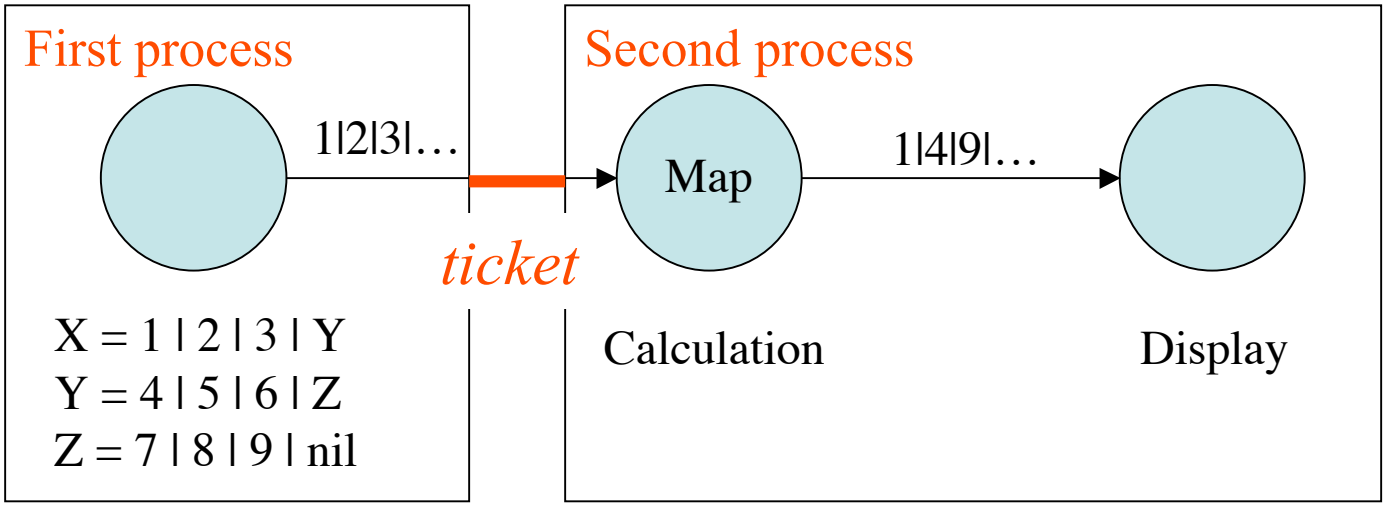
- There are two threads
- The first thread creates the stream X incrementally
- The second thread displays it using dataflow
- Transmission is asynchronous (like a pipe)

Stream communication with dataflow concurrency (2)



- There are three threads
- The first thread creates a stream of data
- The second thread does a calculation
- The third thread displays the results

Stream communication with dataflow concurrency (3)



- Exactly the same thing, but *distributed*
- The processes connect through a *ticket*
  - A ticket is a reference that can exist outside of a process (since it is coded as an Ascii string)
- Except for the ticket, the program is unchanged

Other courses

- We also cover these other paradigms
  - Distributed programming (see dataflow example)
  - Lazy (demand-driven) programming
  - Relational programming
  - Constraint programming
  - Logic programming (deterministic and nondeterministic)
  - Concurrent logic programming
  - Graphical user interface programming
- All of these paradigms fit naturally with the rest
  - They are all covered in the textbook

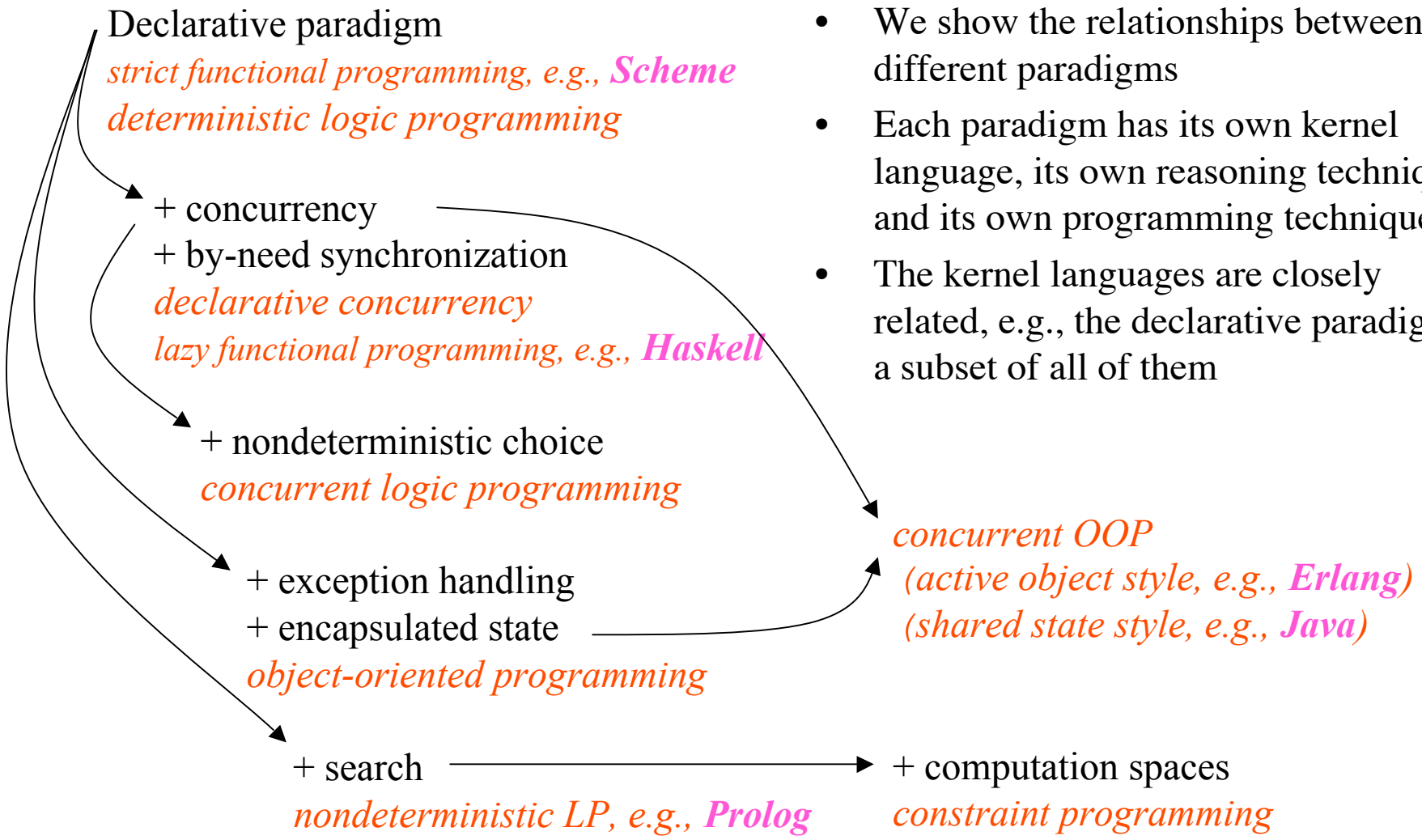
The exaggerated importance of object-oriented programming

- Consider for example the task of building robust telecommunications systems
  - Ericsson has developed an extremely reliable ATM switch (the AXD 301) using a message-passing architecture
  - The important concepts are isolation, concurrency, and higher-order programming
  - Not used: inheritance, classes and methods, UML diagrams, and monitors
- We find that inheritance especially is overused with respect to other techniques such as composition
- Our approach is agnostic with respect to object-oriented programming. We place it in the wider context of data abstraction and concurrent programming.

Semantics

- It's important to put programming on a solid foundation. Otherwise, students will have muddled thinking for the rest of their careers.
- We propose a flexible approach, where more or less semantics can be given depending on taste and the course goals
- Semantics can be taught at three levels:
  - Informal presentation of the formal semantics. Give an outline of an abstract machine. Explain the concepts of execution stack and environment. This can explain last call optimization and memory management (including garbage collection).
  - Complete formal semantics using an abstract machine. The semantics is at the service of programming: it is as simple as possible without sacrificing rigor or coverage. Simple reasoning techniques such as invariant assertions can be used in both declarative and procedural programming.
  - Structural operational semantics. This is the most concise way to give the semantics of a practical language. Other approaches (axiomatic, denotational, and logical) are introduced for the paradigms in which they work the best.

Programming languages and paradigms



- We show the relationships between the different paradigms
- Each paradigm has its own kernel language, its own reasoning techniques, and its own programming techniques
- The kernel languages are closely related, e.g., the declarative paradigm is a subset of all of them

Conclusions

- We have presented an approach for teaching programming that is based on programming language concepts
  - This covers all major programming paradigms; they are placed in a wider framework and we show why and how to use them together
  - The approach is based on more than a decade of research in language design and implementation by the Mozart Consortium (see http://www.mozart-oz.org for information and downloads)
- We have been teaching with this approach for more than three years and we have written a textbook now published by MIT Press
  - If you are interested in trying out the approach, we will be happy to help
  - See http://www.info.ucl.ac.be/people/PVR/book.html
- Special note: the Second International Conference on Mozart/Oz will be held on October 7-8, 2004 in Charleroi, Belgium
  - There will be special emphasis on the use of Oz and Mozart for education
  - See http://www.cetic.be/moz2004 for more information