Lectures

TDDA69 Data and Program Structure Summary Cyrille Berger



Data structures only

Turing equivalent

Deterministic

Relational & logic

Constraint (logic

Concurrent

constraint

Lazy concurrent

constraint

programming

Oz, Alice, Curry

Logic and

More declarative

gramming

+ unification

+ by-need

nchron.

Lazy

functiona

programmin

Haskell

+ single assignment + synchronization

dataflow

programmin

Lazy

declarative

concurrent

orogrammin

Oz, Alice, Curry

Functional

Unnamed state (seq. or conc.)

(equality)

+ search

Prolog, SQL embeddings

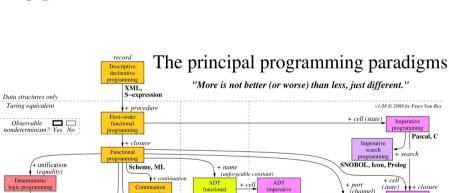
CLP, ILOG Solver

+ by-need synchronization

+ solver

+ thread

LIFE, AKL



functional

ogrammin

Haskell, ML, E

Nonmonotonic dataflow

programmin

programming

Functional reactiv

programming (FRF

Weak synchronous

programming

Strong synchronous

programming Esterel, Lustre, Signal

Nondet. state

FrTime, SL

choice

Oz, Alice, Curry, Excel, AKL, FGHC, FCP

+ synch. on partial termination

+ instantaneous computation

programming

Scheme, ML

Monotonic

programming

Declarative

concurrent

programming

+ thread (+ by-need

dataflow

+ thread

+ single a

"More is not better (or worse) than less, just different." v1.08 © 2008 by Peter Van Roy

programming

CLU, OCaml, Oz

Multi-agent dataflow

programming

Oz, Alice, AKL

Dataflow and

essage passing

+ port (channel)

Imperative

programming

(state) + closure

Sequential object-oriented

programming

Stateful

functional

programming

Concurrent

biect-oriente

programming Shared-state

concurrent

rogramming

Software

transaction

SOL embedding

Shared state

Less declarative

memory (STM

Java, OCaml

Smalltalk, Oz.

Java, Alice

+ log

+ thread

+ cell

(channel)

E in one vat

Erlang, AKI

+ local cell

+ thread

Event-loop

programming

Multi-agent

programming

Message-passing concurrent

programming

Active object

programming

Object-capability

programming

CSP. Occam.

E, Oz, Alice,

publish/subscribe

tuple space (Linda)

Message passing

Named state

Pascal, C

1Introduction

2Concepts and models of programming languages **3**Declarative Computation Model 4Declarative Programming Techniques 5Declarative Computation Implementation 6Declarative Concurrency 7Message Passing Concurrency 8Explicit State and Imperative Model 9Imperative Programming Techniques **10Imperative Programming Implementation** 11Shared-State Concurrency 12Relational Programming **13**Specialized Computation Models 14Macro 15Running natively and JIT **16**Garbage Collection 17Summary

2/34

Lecture content

- Summary
 - ^o Choosing an appropriate
 - **Programming Language Paradigm** The different types of interpreter
- GUI Programming

Summary

Do we need new programming languages?

New Concepts

in the early days, object orientation

- New problems and new infrastructure
 - ^o Multi-threading
 - ^o Distributed computing

•

Develop a better syntax

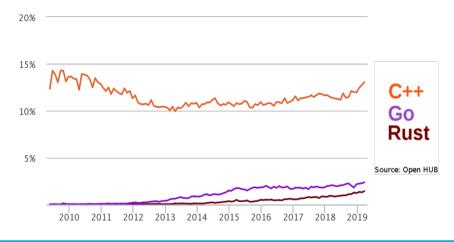
Motivation for creating Go

Rob Pike, Go creator: "A couple of years ago, several of us at Google became a little frustrated with the software development process, and particularly using C++ to write large server software. We found that the **binaries tended to be much too big**. They took **too long to compile**. And the language itself, which is pretty much the main system software language in the world right now, is **a very old language**. A lot of the **ideas and changes in hardware** that have come about in the last couple of decades haven't had **a chance to influence C++**."

Motivation for creating Rust

Graydon Hoare, Rust creator: "A lot of obvious good ideas, known and loved in other languages, haven't made it into widely-used systems languages, or are deployed in languages that have very poor (unsafe, concurrency-hostile) memory models. There were a lot of good competitors in the late 70s and early 80s in that space, and I wanted to revive some of their ideas and give them another go, on the theory that circumstances have changed: the internet is highly concurrent and highly securityconscious, so the design-tradeoffs that always favor C and C++ (for example) have been shifting."

Usage of C++ vs Go vs Rust



1.U LERVICE AND 9 / 34

PHP Criticism

PHP was not designed, but developed

- ^o Inconsistent naming of functions and order of their parameters Some function names were chosen to improve the distribution of hash values
- Rather than aborting with an error, PHP will try to guess the developer intent Problems with weak typing
- PHP compilation options, server configurations, applications configurations and global states can affect function behaviour
- $^{\rm o}$ Incoherent mix between functional and object-oriented programming $_{\rm o}$
- You need a vision and a design when developing a programming language!

Is it easier to change and fix existing languages?

- Backward-compatible changes
- Backward-incompatible changes
- ^o Some changes are too difficult
- ^o Introducing Unicode in Python and PHP
- ° Garbage collector in C++
- Those changes introduce long development time and long acceptance time
 - Python 3.0 was introduced in 2008
 - PHP 6 was started in 2006 and never released, PHP 7 released in December 2015

.	
\sim	

10/34

Design Considerations for a Programming Language

A programming language must be:

- predictable
 - $^\circ$ Source code is read more often than written, a human must be able to understand what he read
- consistent
- $^{\circ}$ Knowing part of a language should help learn other parts
- concise, simple and general
- reliable

I.U

- $^{\circ}$ Programming language are here to solve problem, not to introduce new one
- debuggable
- $^{\circ}$ Developers will inevability write *bugs*, they need all the help they can get to find them
- implementable
- $^{\circ}$ This reduce the number of bugs in the language implementation



What is the purpose of the new language?

- First question is, a new language, what for?
- ^o Querying knowledge?
- ^o Distributed numerical computation?
- ^o Writting drivers for an Operating System?
- ^o Writting web applications?
- Answering the Ultimate Question of Life, the Universe, and Everything
- A programming language for teaching about interpreters and programming models/paradigms

0

13 / 34

Design choices for a Programming Language

- Programming Paradigm
- Dynamic vs Static (Typing...)
- Low-level vs High-level
- Direct interpretation, Virtual Machine, JIT, Native Compilation...

14/34

Declarative

- Expresses logic of computation without control flow:
 - What should be computed and not how it should be computed.
- Examples: XML/HTML, antlr4/yacc, make/ants...

Choosing an appropriate Programming Language Paradigm

Functional

- Computation are treated as mathematical function
 vithout changing any internal state
- Examples: Lisp, Scheme, Haskell...

Logic Programming

- Based on Formal logic: expressing facts and rules
- Examples: Prolog

17 / 34

Imperative

- Express how computation are executed
 Describes computation in term of statements that change the internal state
- Examples: C/C++, Pascal, Java, Python, JavaScript...

Object-Oriented

- Based on the concept of *objects*, which are *data structures* containing *fields* and *methods*
 - Programs are designed by making objects interact with each others
- Examples: C++, Java, C#, Python, Ruby, JavaScript...

18/3

How to choose a programming paradigm?

- The choice depends on the goal
- ^o Querying knowledge?
 - \Rightarrow Declarative or logic (reasoning)
- Distributed numerical computation?
 ⇒ Functional
- ^o Writting drivers for an Operating System?
- \Rightarrow Imperative
- ^o Writting web applications?
 - \Rightarrow Object-Oriented
- The choice can be considered controversial!

21/34

The different types of interpreter

Conscequence for the users

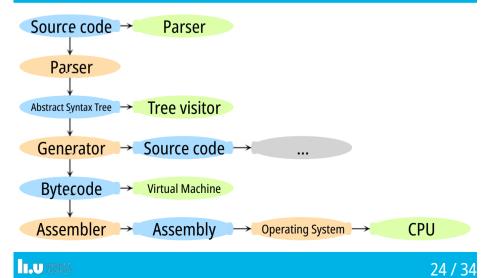
- ^o Expressivity, simplicity, readability
- Conscequence on the implementation

Conscequences of the choice

^o Different types of interpreters

22/34

How is a program interpreted?

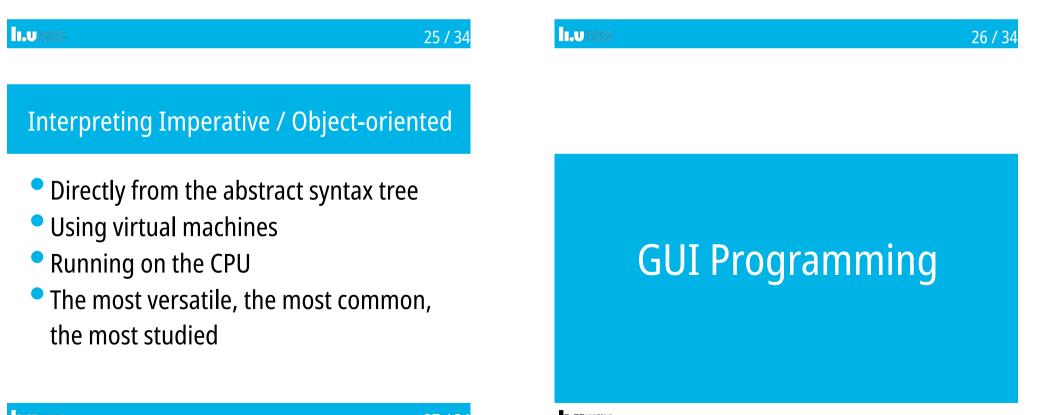


Interpreting Declarative and Logic

 Using a query executor (for SQL)
 Using a reasoning engine and unification (for Prolog)

Interpreting Functional

- Functions are evaluated in order
- Applicative vs Normal
- In normal order, the tree needs to be expanded, virtual machines are more difficult (might be impossible)



GUI Programming

- What model?
- Purely procedural
 - ^o Sequence of graphical command
 - ° Example: AWT, SWING... drawRectangle(0, 5, 10, 20)
- Purely declarative
 - $^{\circ}$ Choosen from a set of possibilities
 - ° Example: HTML... <rectangle geometry='0,5,10,20' />

29/34

Declarative approach

- Set of possible shapes for different attributes
- Limited expressiveness
- Easy match between data and user interface

Procedural approach

- Set of primitive operations combined in a program
- Unlimited expressiveness
- Harder to do formal manipulation (converting data to user interface)

Combining declarative/procedural approach

Examples: QML, React,

- Takes the best of both
- Declarativeness is used
 - $^{\circ}$ Describing static structure of a window
- ^o The type of widgets
- $^{\circ}$ The initial state of the widgets
- ° The resize behaviour
- Procedure is used
- Procedures executed when events occurs
 Handlers

30/3

Example



Conclusion

- Many problems, many solutions, many developers, many paradigms.
- A large program is likely to be a combination of paradigms
- Programming language tend to combine from different paradigms

33 / 34

34/34