NPRG077

Write your own tiny programming system(s)!

Tomáš Petříček, 309 (3rd floor)
petricek@d3s.mff.cuni.cz
https://tomasp.net | @tomaspetricek
https://d3s.mff.cuni.cz/teaching/nprg077

Introduction

Why such a strange course?

Charles University

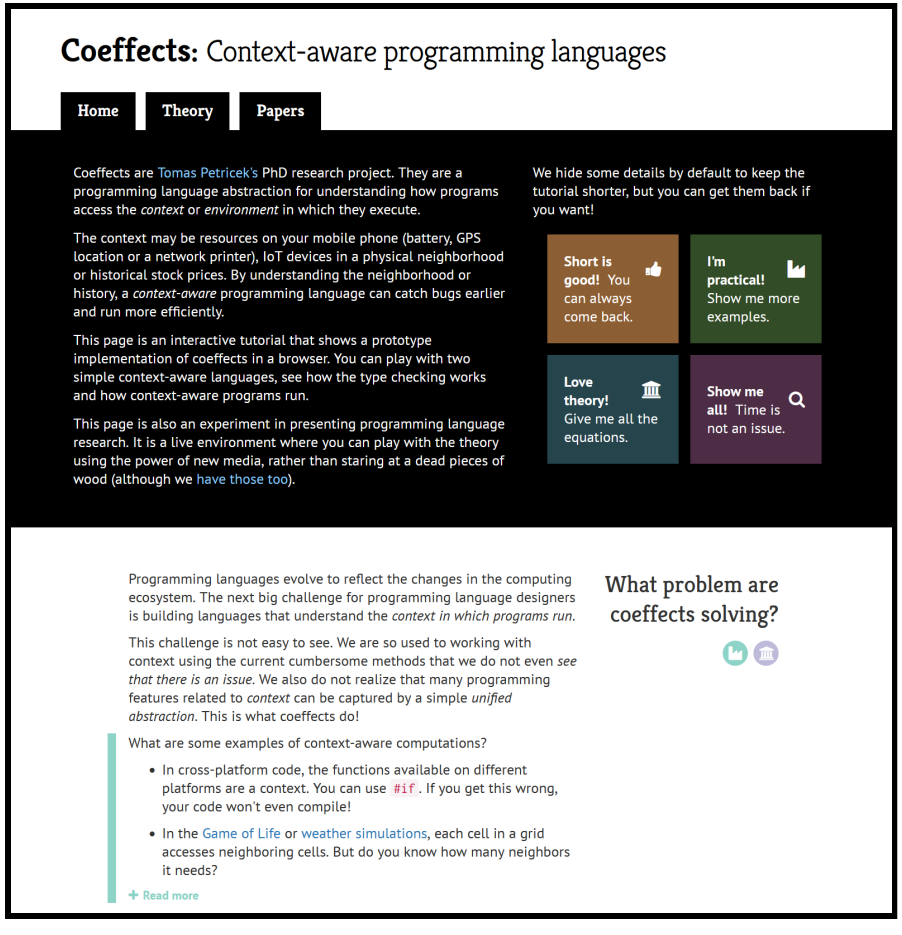
Where I'm coming from?

PhD, University of Cambridge
Context-aware programming languages

Microsoft Research Cambridge
F# and applied functional programming

The Alan Turing Institute, London
Expert and non-expert tools for data science

University of Kent, Canterbury
History and programming systems



Demo

Coeffects playground

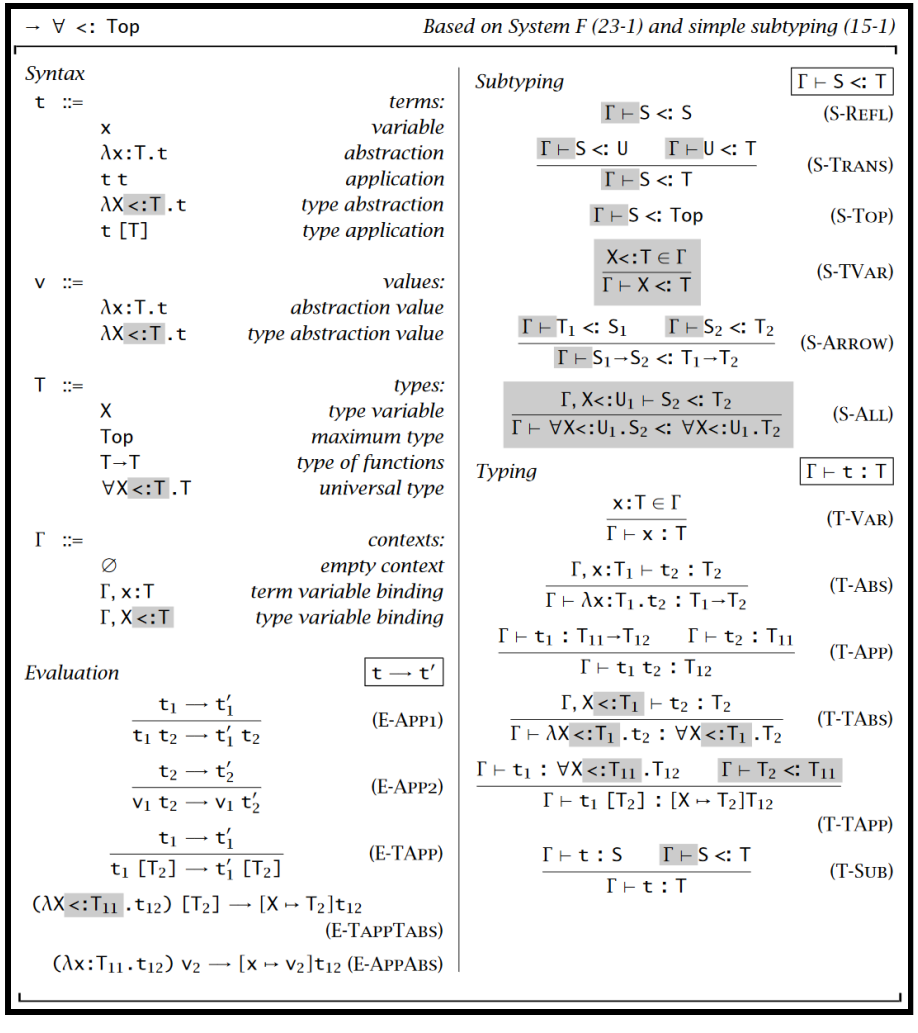
Did this to get my PhD...

How to show potential uses of theoretical work?

Tiny type system running in the web browser

Tiny demos of two potential applications

Charles University



→ ∀ <: Top — *Based on System F (23-1) and simple subtyping (15-1)*

Syntax

t ::=		*terms:*
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	λX<:T.t	*type abstraction*
	t [T]	*type application*
v ::=		*values:*
	λx:T.t	*abstraction value*
	λX<:T.t	*type abstraction value*
T ::=		*types:*
	X	*type variable*
	Top	*maximum type*
	T→T	*type of functions*
	∀X<:T.T	*universal type*
Γ ::=		*contexts:*
	∅	*empty context*
	Γ, x:T	*term variable binding*
	Γ, X<:T	*type variable binding*

Evaluation — $t \longrightarrow t'$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}\ \text{(E-App1)} \qquad \frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}\ \text{(E-App2)} \qquad \frac{t_1 \longrightarrow t_1'}{t_1\ [T_2] \longrightarrow t_1'\ [T_2]}\ \text{(E-TApp)}$$

$$(\lambda X{<:}T_{11}.t_{12})\ [T_2] \longrightarrow [X \mapsto T_2]t_{12}\ \text{(E-TappTabs)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12}\ \text{(E-AppAbs)}$$

Subtyping — $\Gamma \vdash S <: T$

$$\Gamma \vdash S <: S\ \text{(S-Refl)} \qquad \frac{\Gamma \vdash S <: U \quad \Gamma \vdash U <: T}{\Gamma \vdash S <: T}\ \text{(S-Trans)} \qquad \Gamma \vdash S <: \text{Top}\ \text{(S-Top)}$$

$$\frac{X{<:}T \in \Gamma}{\Gamma \vdash X <: T}\ \text{(S-TVar)} \qquad \frac{\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2}\ \text{(S-Arrow)}$$

$$\frac{\Gamma, X{<:}U_1 \vdash S_2 <: T_2}{\Gamma \vdash \forall X{<:}U_1.S_2 <: \forall X{<:}U_1.T_2}\ \text{(S-All)}$$

Typing — $\Gamma \vdash t : T$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T}\ \text{(T-Var)} \qquad \frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2}\ \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}}\ \text{(T-App)} \qquad \frac{\Gamma, X{<:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda X{<:}T_1.t_2 : \forall X{<:}T_1.T_2}\ \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash t_1 : \forall X{<:}T_{11}.T_{12} \quad \Gamma \vdash T_2 <: T_{11}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}}\ \text{(T-TApp)} \qquad \frac{\Gamma \vdash t : S \quad \Gamma \vdash S <: T}{\Gamma \vdash t : T}\ \text{(T-Sub)}$$

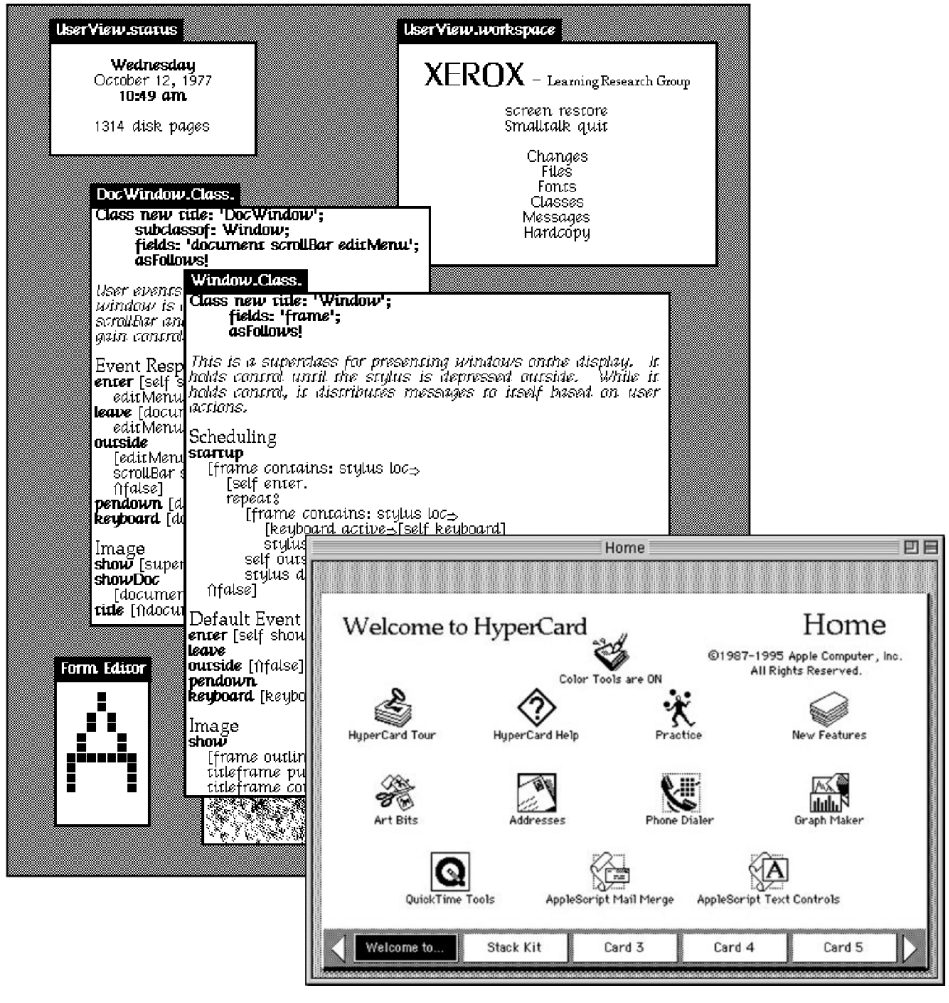
Programming Languages

Programming is writing code

Formal semantics, implementation, paradigms, types

We know how to study this!

Charles University



Programming Systems

Interacting with a stateful system

Feedback, liveness, interactive user interfaces

But how do we study this?

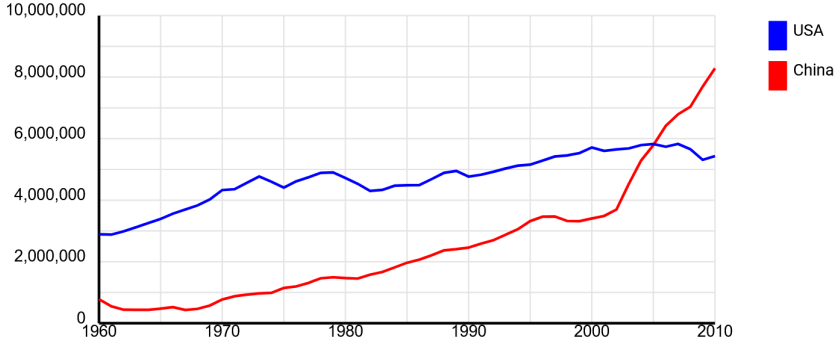
Charles University

The Alan Turing Institute
Data playground

```
let china = worldbank.byCountry.China.'Climate Change'.'CO2 emissions (kt)'
  .setProperties(seriesName="China")

let usa = worldbank.byCountry.'United States'.'Climate Change'.'CO2 emissions (kt)'
  .setProperties(seriesName="USA")

compost.charts.lines([china,usa]).setColors(["red","blue"])
  .setAxisX(1960).setLegend("right").setSize(800,300)
```



Demo

The Gamma project

Making programmatic data exploration accessible to non-programmers

From language to system

Small typed language

Interaction is the key.
This is why it works!

Charles University

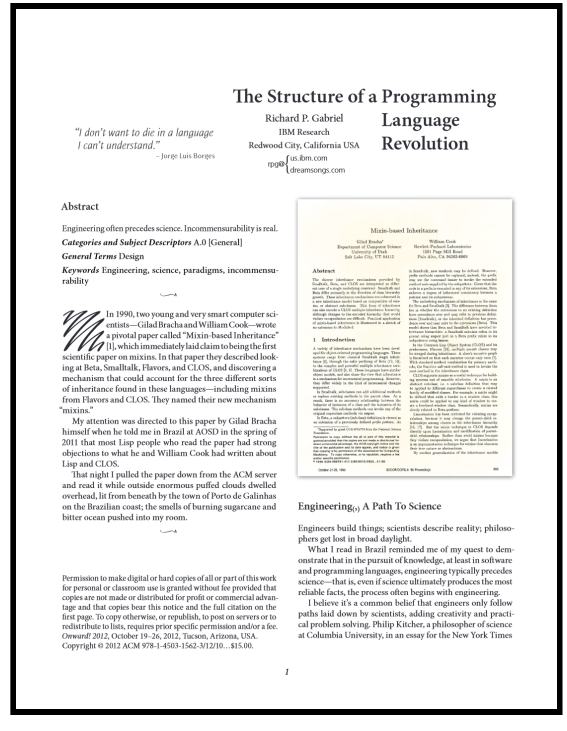
Paradigm shift in 1990s

From systems to languages

- From running system to code
- From state & interaction to semantics
- Incommensurable ways of thinking!

History of science matters!

- How did we get where we are?
- What ideas got lost along the way?
- How to recover them?



The Structure of a Programming Language Revolution

Richard P. Gabriel
IBM Research
Redwood City, California USA
rpg@{us.ibm.com, dreamsongs.com}

"I don't want to die in a language I can't understand."
– Jorge Luis Borges

Abstract

Engineering often precedes science. Incommensurability is real.

Categories and Subject Descriptors A.0 [General]

General Terms Design

Keywords Engineering, science, paradigms, incommensurability

In 1990, two young and very smart computer scientists—Gilad Bracha and William Cook—wrote a pivotal paper called "Mixin-based Inheritance" [1], which immediately laid claim to being the first scientific paper on mixins. In that paper they described looking at Beta, Smalltalk, Flavors, and CLOS, and discovering a mechanism that could account for the three different sorts of inheritance found in these languages—including mixins from Flavors and CLOS. They named their new mechanism "mixins."

My attention was directed to this paper by Gilad Bracha himself when he told me in Brazil at AOSD in the spring of 2011 that most Lisp people who read the paper had strong objections to what he and William Cook had written about Lisp and CLOS.

That night I pulled the paper down from the ACM server and read it while outside enormous puffed clouds dwelled overhead, lit from beneath by the town of Porto de Galinhas on the Brazilian coast; the smells of burning sugarcane and bitter ocean pushed into my room.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Onward! 2012, October 19–26, 2012, Tucson, Arizona, USA.
Copyright © 2012 ACM 978-1-4503-1562-3/12/10…$15.00.

Mixin-based Inheritance

Gilad Bracha
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

William Cook
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304-0969

Abstract

1 Introduction

Engineering$_{(0)}$ A Path To Science

Engineers build things; scientists describe reality; philosophers get lost in broad daylight.

What I read in Brazil reminded me of my quest to demonstrate that in the pursuit of knowledge, at least in software and programming languages, engineering typically precedes science—that is, even if science ultimately produces the most reliable facts, the process often begins with engineering.

I believe it's a common belief that engineers only follow paths laid down by scientists, adding creativity and practical problem solving. Philip Kitcher, a philosopher of science at Columbia University, in an essay for the New York Times

Charles University

Research

What do I work on today?

 History and philosophy of computing

 Programming languages, types and theory

 Interactive programming environments

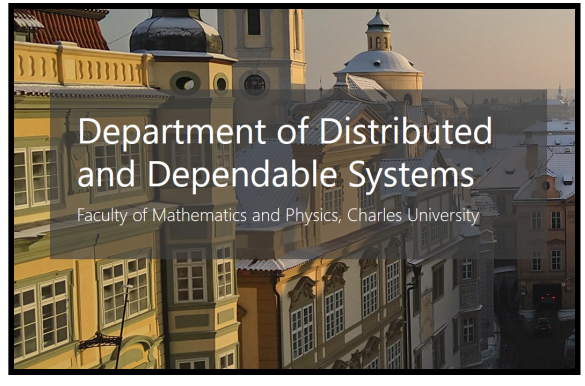
 Will artificial intelligence make me obsolete?

Charles University

Programming languages at D3S

Growing group of great people

- Jan Vitek (via Northeastern)
- Aleksander Boruch-Gruszecki
- Also talking to PRL-PRG at CTU!



Growing number of activities!

- Programming languages reading group
- New courses (NSWI182, NPRG075, NPRG077)
- PL topics at the regular D3S seminar

Charles University

D3S

PhD in programming languages, systems and tools

Can we make programming faster, better, safer, easier and more fun?
Join our group to work on data science languages or pursue your own ideas!

≠ Theory and type systems?

Add types to libraries never designed to have them like **data visualization in R?**

Usability and interactivity?

Find ways of creating programs that are **accessible also to non-programmers?**

Learning from past systems?

Think about **interactive and stateful programming environments**, not languages?

Funded positions available!

Making data scripting safer and more with **salary in the range 35k-40k/month**

Get in touch to find out more and discuss your own research ideas!

Tomas Petricek (Malá strana, S309), email: petricek@d3s.mff.cuni.cz

Starting points

Writing tiny systems

Two uses of tiny systems

Education

Best way to learn?
Write it on your own!

Understand principles
As well as subtle details

I hope you'll have fun!
Doing more with less?

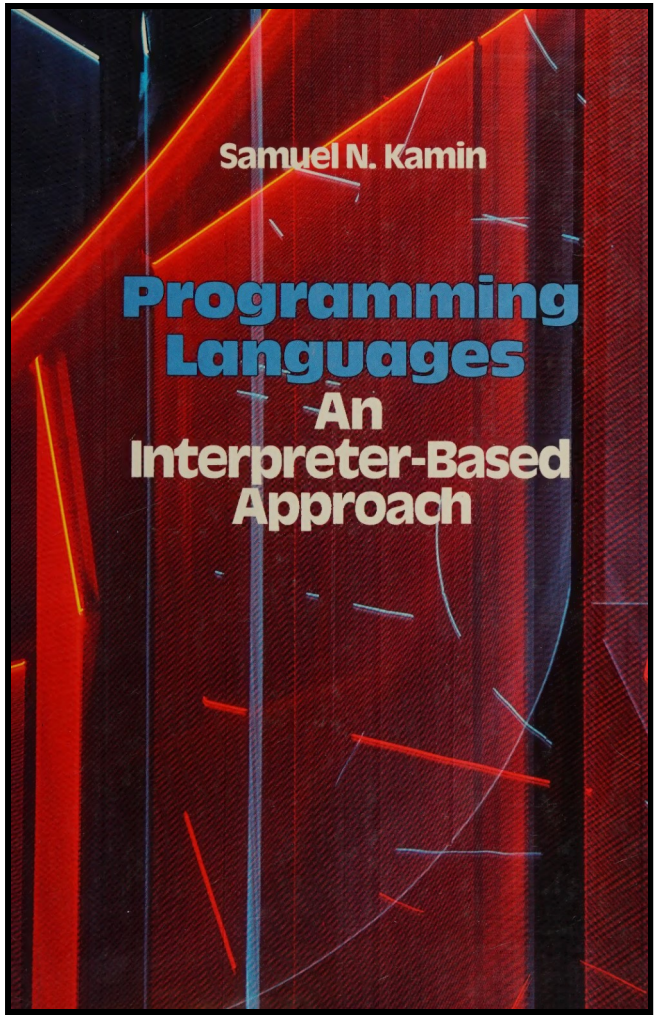
Research

Imagine new paradigms
Variable names

Focus on interaction
How exactly did it work

Ignore practical details
New mode of interaction

Charles University



Teaching tiny systems

(Kamin, 1990)

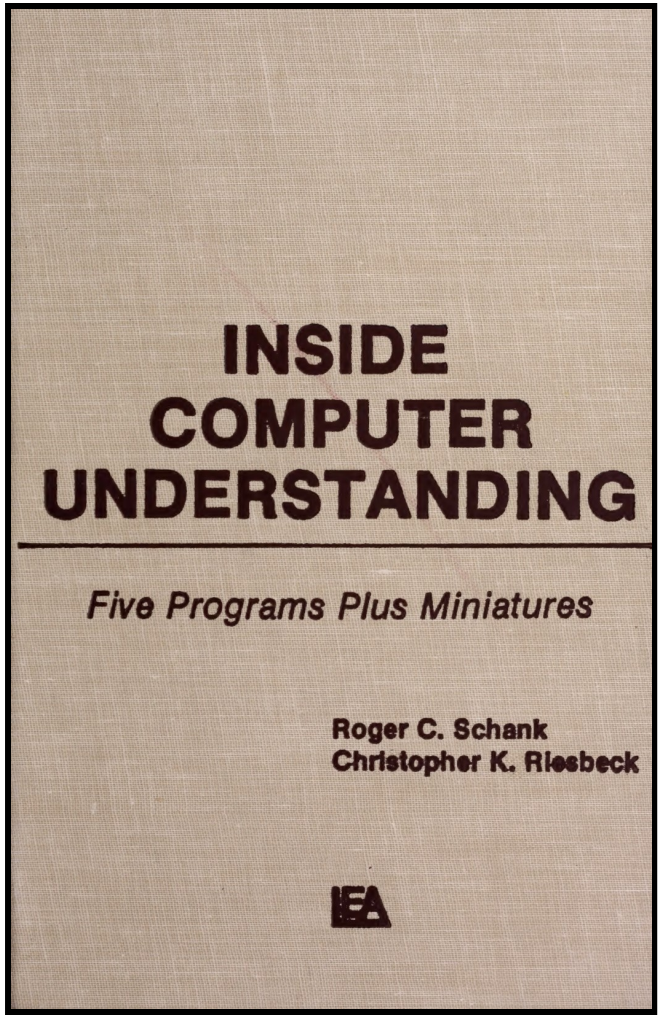
Used in multiple courses worldwide

Examples in Pascal

Languages covered are APL, Clu, LISP, Prolog, Smalltalk, Scheme, SASL

Not always focused on the key aspect

Charles University



Tiny systems and AI

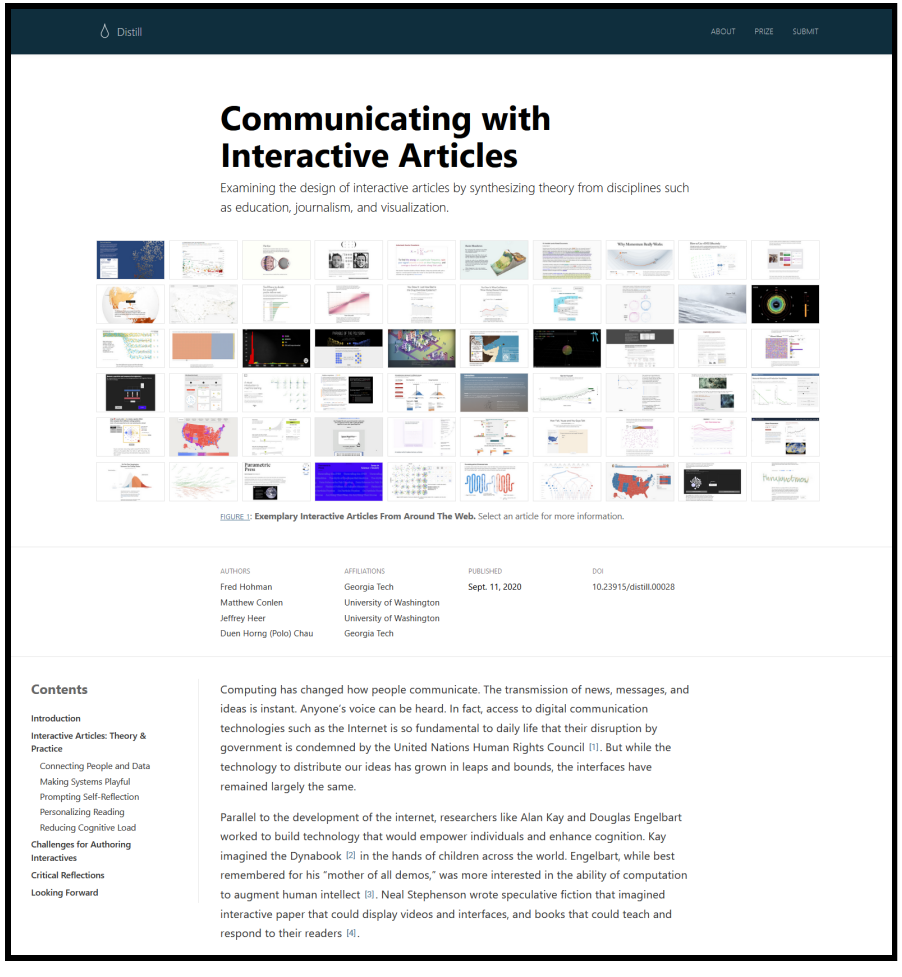
(Schank, Riesbeck, 1981)

Miniature implementations of 5 Yale AI lab programs

Faster, more efficient, easier to understand, modify and extend

"Miniatures, demos and artworks" by Warren Sack

Charles University



Distill

ABOUT PRIZE SUBMIT

Communicating with Interactive Articles

Examining the design of interactive articles by synthesizing theory from disciplines such as education, journalism, and visualization.

FIGURE 1: **Exemplary Interactive Articles From Around The Web.** Select an article for more information.

AUTHORS
Fred Hohman
Matthew Conlen
Jeffrey Heer
Duen Horng (Polo) Chau

AFFILIATIONS
Georgia Tech
University of Washington
University of Washington
Georgia Tech

PUBLISHED
Sept. 11, 2020

DOI
10.23915/distill.00028

Contents

Computing has changed how people communicate. The transmission of news, messages, and ideas is instant. Anyone's voice can be heard. In fact, access to digital communication technologies such as the Internet is so fundamental to daily life that their disruption by government is condemned by the United Nations Human Rights Council [1]. But while the technology to distribute our ideas has grown in leaps and bounds, the interfaces have remained largely the same.

Parallel to the development of the internet, researchers like Alan Kay and Douglas Engelbart worked to build technology that would empower individuals and enhance cognition. Kay imagined the Dynabook [2] in the hands of children across the world. Engelbart, while best remembered for his "mother of all demos," was more interested in the ability of computation to augment human intellect [3]. Neal Stephenson wrote speculative fiction that imagined interactive paper that could display videos and interfaces, and books that could teach and respond to their readers [4].

Tiny systems and ML

(Distill, 2016-2021)

Five affordances of interactive articles

Connecting people & data
Making systems playful
Prompting self-reflection
Personalizing reading
Reducing cognitive load

Charles University

Programming models

Learning by implementing

Programming models

Language paradigms

 Functional programming

 Imperative programming

 Object-oriented programming

 Logic programming

Charles University

Programming models

System interaction

Image-based programming model

Programming system is always running

Interactive and live programming

System provides continuous feedback

Incremental or reactive evaluation

Recompute on edit or when new data come

Charles University

Demo

Logic programming in Prolog

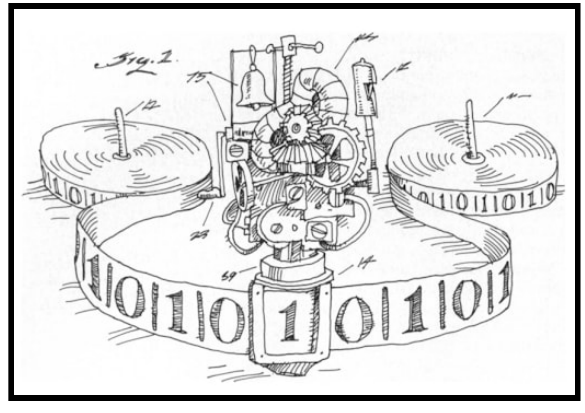
Demo

Object-orientation in Smalltalk

What really matters?

Static structure

- Source code of the program
- What you have at the start



Dynamic structure

- Runtime data structures
- What else do you need to run

Logic of evaluation

- How the dynamic state evolves?

Charles University

```
(* A term like 'father(william, X)'
   consists of predicate 'father',
   atom 'william' and variable 'X' *)
type Term =
  | Atom of string
  | Variable of string
  | Predicate of string * Term list

(* A rule 'head(...) :- body.' *)
type Rule =
  { Head : Term
    Body : Term list }

(* A program is a list of rules *)
type Program = Rule list
```

Why interpreters?

A good way to explain the structures!

Functional data types for the static and dynamic structure

A function to model the evaluation logic

Charles University

$$\text{(deref)} \quad \langle !\ell, s\rangle \longrightarrow \langle n, s\rangle \quad \text{if } \ell \in \mathrm{dom}(s) \text{ and } s(\ell) = n$$

$$\text{(assign1)} \quad \langle \ell := n, s\rangle \longrightarrow \langle \textbf{skip}, s + \{\ell \mapsto n\}\rangle \quad \text{if } \ell \in \mathrm{dom}(s)$$

$$\text{(assign2)} \quad \frac{\langle e, s\rangle \longrightarrow \langle e', s'\rangle}{\langle \ell := e, s\rangle \longrightarrow \langle \ell := e', s'\rangle}$$

$$\text{(seq1)} \quad \langle \textbf{skip}; e_2, s\rangle \longrightarrow \langle e_2, s\rangle$$

$$\text{(seq2)} \quad \frac{\langle e_1, s\rangle \longrightarrow \langle e_1', s'\rangle}{\langle e_1; e_2, s\rangle \longrightarrow \langle e_1'; e_2, s'\rangle}$$

$$\text{(if1)} \quad \langle \textbf{if } \textbf{true } \textbf{then } e_2 \textbf{ else } e_3, s\rangle \longrightarrow \langle e_2, s\rangle$$

$$\text{(if2)} \quad \langle \textbf{if } \textbf{false } \textbf{then } e_2 \textbf{ else } e_3, s\rangle \longrightarrow \langle e_3, s\rangle$$

$$\text{(if3)} \quad \frac{\langle e_1, s\rangle \longrightarrow \langle e_1', s'\rangle}{\langle \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3, s\rangle \longrightarrow \langle \textbf{if } e_1' \textbf{ then } e_2 \textbf{ else } e_3, s'\rangle}$$

Operational semantics

Standard approach to programming language theory

Equations vs. Code

Code actually runs!
Easier to write?

Charles University

Course scope

What is not covered?


Syntax choices and writing parsers


Compilation and JIT-based runtimes


Formal semantics and correctness


Supporting real-world use cases

Charles University

Tiny systems

Programming systems research

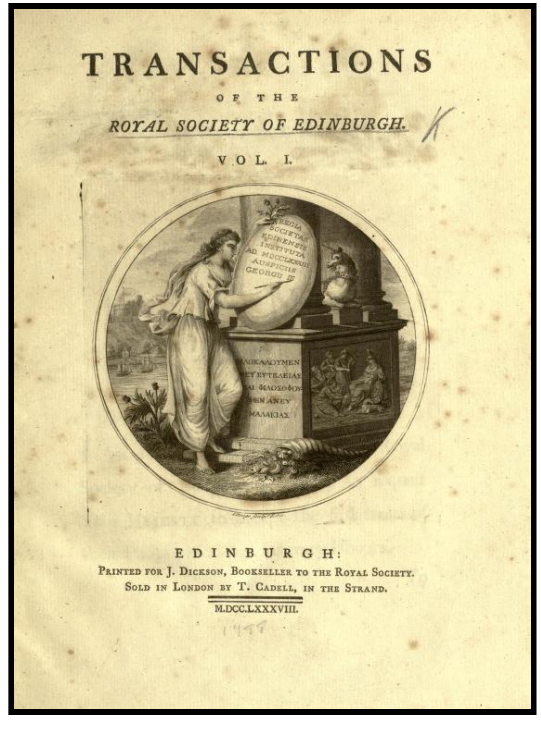
Academic research

What are we trying to study?

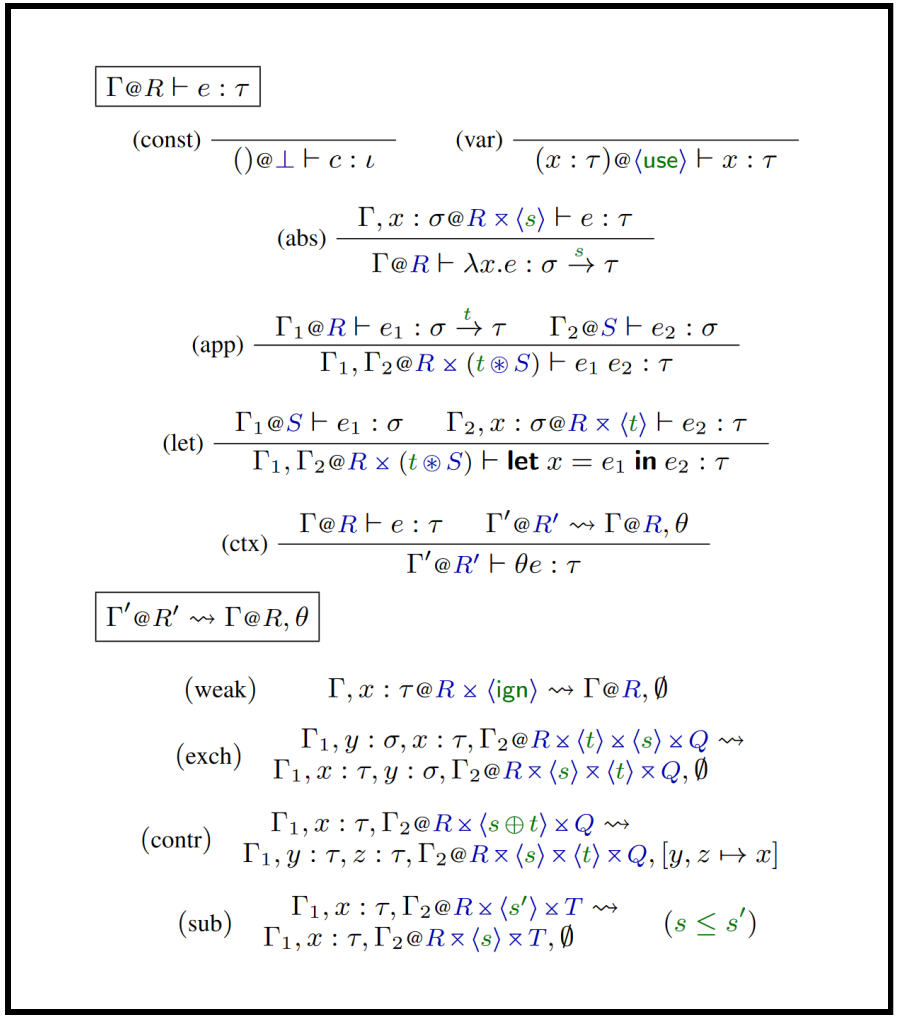
- Basic essential principles
- In isolation from other factors
- You have to ignore a lot!

What to ignore in programming?

- Efficient implementation?
- Wide-spread user adoption?
- User interface of editor tools?


TRANSACTIONS
OF THE
ROYAL SOCIETY OF EDINBURGH.
VOL. I.
EDINBURGH:
PRINTED FOR J. DICKSON, BOOKSELLER TO THE ROYAL SOCIETY.
SOLD IN LONDON BY T. CADELL, IN THE STRAND.
M.DCC.LXXXVIII.

Charles University

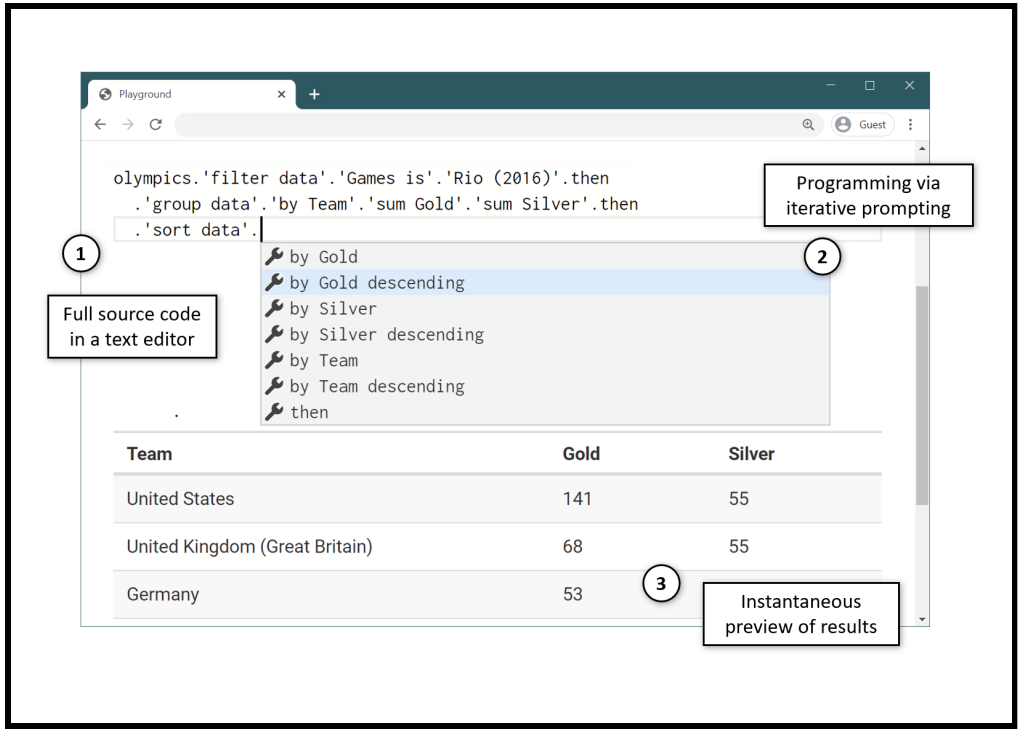


Programming language theory

Ignore implementation and practical features

Prove that the core idea is formally sound

Charles University



Human-computer interaction (HCI)

Ignore inner working and implementation

Show that users can actually use it and how

Charles University

Performance evaluation

Ignore usability and design implications

Show that you can do better than a baseline

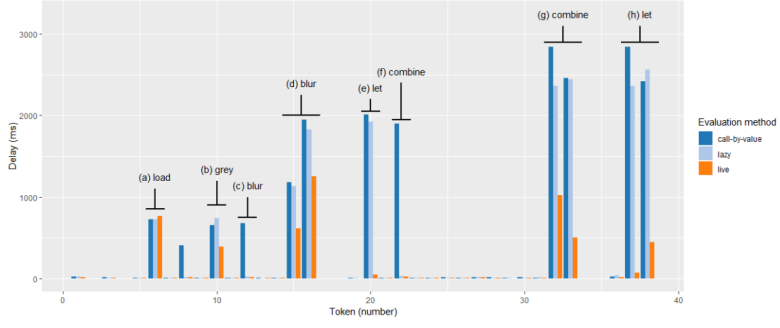


Figure 11 Time required to recompute the results of a sample program after individual tokens are added or modified for three different evaluations strategies.

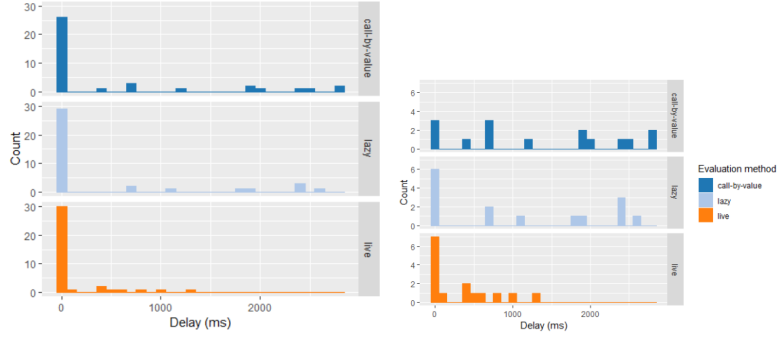


Figure 12 Distribution of delays incurred when updating previews. We show a histogram computed from all delays (left) and only from delays larger than 15 ms (right).

Charles University

Tiny systems

What can we study?

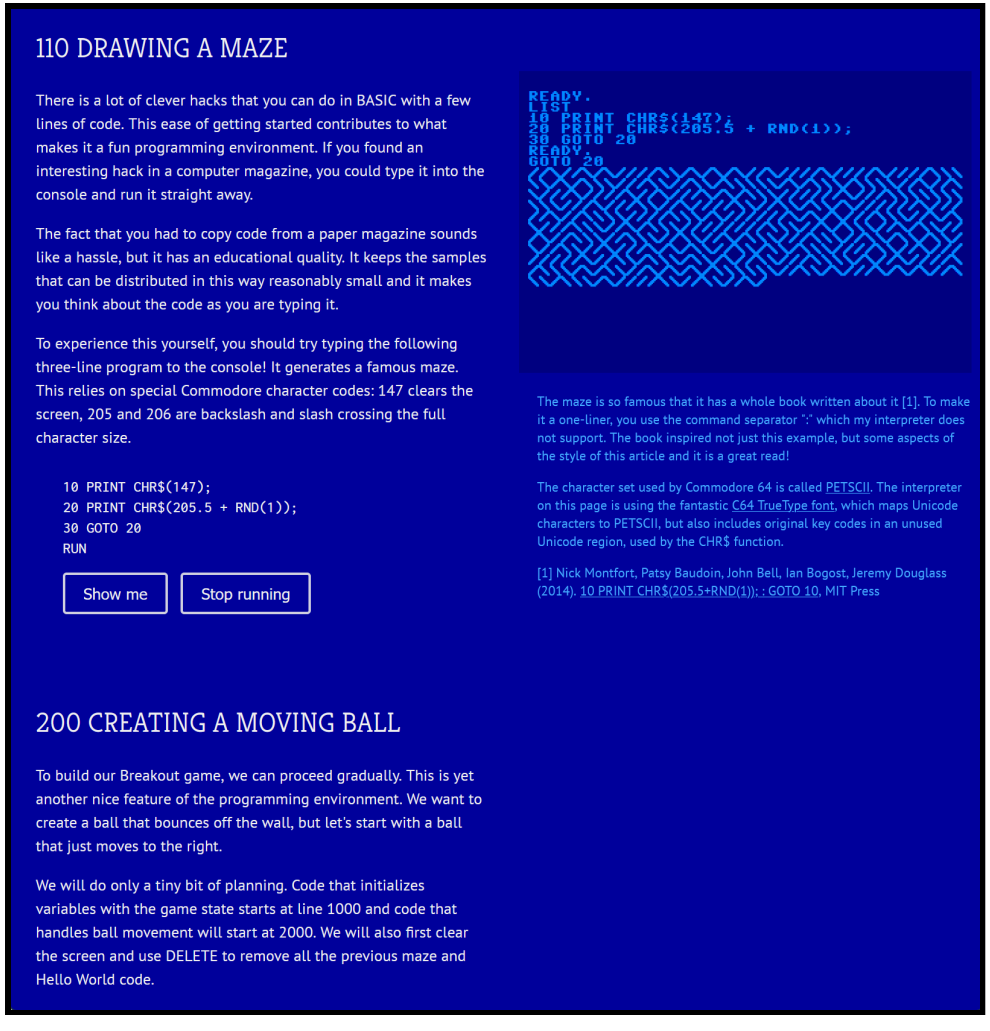

Can talk about stateful interactive systems


Implement key aspects of inner working


Reconstruct interesting past systems


But cannot be printed on 12 pages of A4

Charles University



Demo

C64 BASIC

Why study universally disliked programming language?

Somehow allowed everyone to program!

Interesting mode of interaction!

Charles University

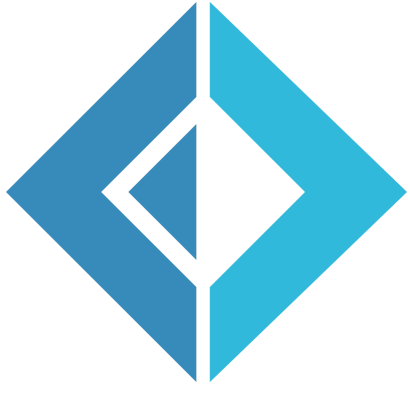
Course background

Getting started with F#

The F# programming language

What is F# about?

- Functional-first based on OCaml
- Great interop with .NET and JS
- Open-source (MIT) with team in Prague!



Who uses F# for what?

- Consultancies for full-stack web dev
- Finance and insurance companies for modelling
- TU Kaiserslautern for systems biology
- Success stories like Jet.com

Charles University

Why F#?

Building tiny programming systems


Algebraic data types for structure modelling


Mostly functional is great for logic


Runs everywhere & has nice tools

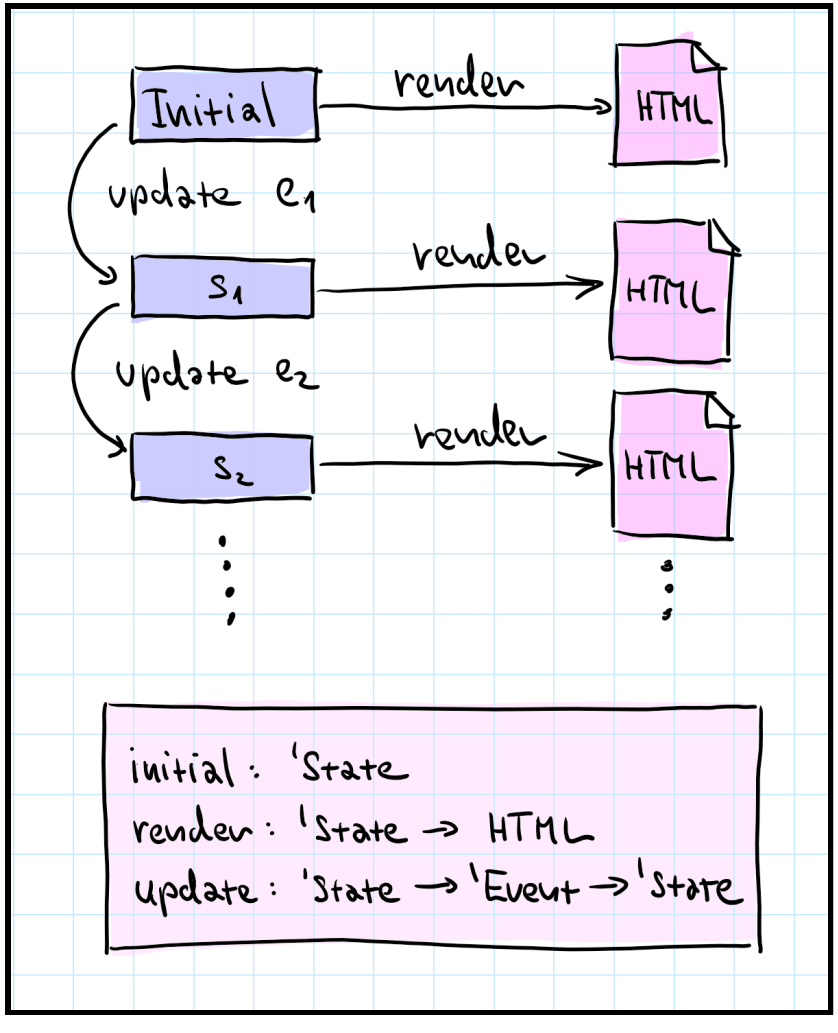

I like the language and can help you!

Charles University

Demo

First look at F#

Elmish architecture



Functional interactive user interface development

Types for application `State` and user `Event`

Functions to `render` and `update` state

Charles University

Demo

Building a TODO list in F#

Closing

Write your own tiny system

Practical details

Course structure

- Videos + bi-weekly hands-on labs
- Watch before & finish after!
- Remote possible - email me
- Check the schedule on course web site!

To get the credits

- Active participation in the labs
- Awarded based on a git repo
- Complete basic tasks for 4/6 systems

Charles University

Conclusions

Write your own tiny programming system(s)!

- Learn interesting programming models!
- Nice programming research methodology
- We have projects and PhD positions available :-)

Tomáš Petříček, 309 (3rd floor)
petricek@d3s.mff.cuni.cz
https://tomasp.net | @tomaspetricek
https://d3s.mff.cuni.cz/teaching/nprg075

Charles University

References

Tiny system examples

- Coeffects: Context-aware programming languages
- The Gamma: Democratizing data science
- The Lost Ways of Programming: Commodore 64 BASIC

Starting points

- Ingalls, D. (2020). The Smalltalk Zoo: Smalltalk-78 (NoteTaker)
- Hohman, F. et al. (2020). Communicating with Interactive Articles
- Schank, R. C., Riesbeck, C. K. (1981). Inside Computer Understanding Five Programs Plus Miniatures
- Kamin, S. (1990) Programming languages: an interpreter-based approach. Addison-Wesley.
- Kamin, S. (1990) PLIBA source code mirror on GitHub
- Sack. W. (2020). Miniatures, Demos and Artworks: Three Kinds of Computer Program, Their Uses and Abuses

Charles University