Write your own tiny programming system(s)

Why such a strange course topic?

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077

Research

Understanding & improving programming

 Programming languages, types and theory

 Interactive programming environments

 History and philosophy of computing

 Building tiny systems throughout!

Charles University

Many different tiny systems!

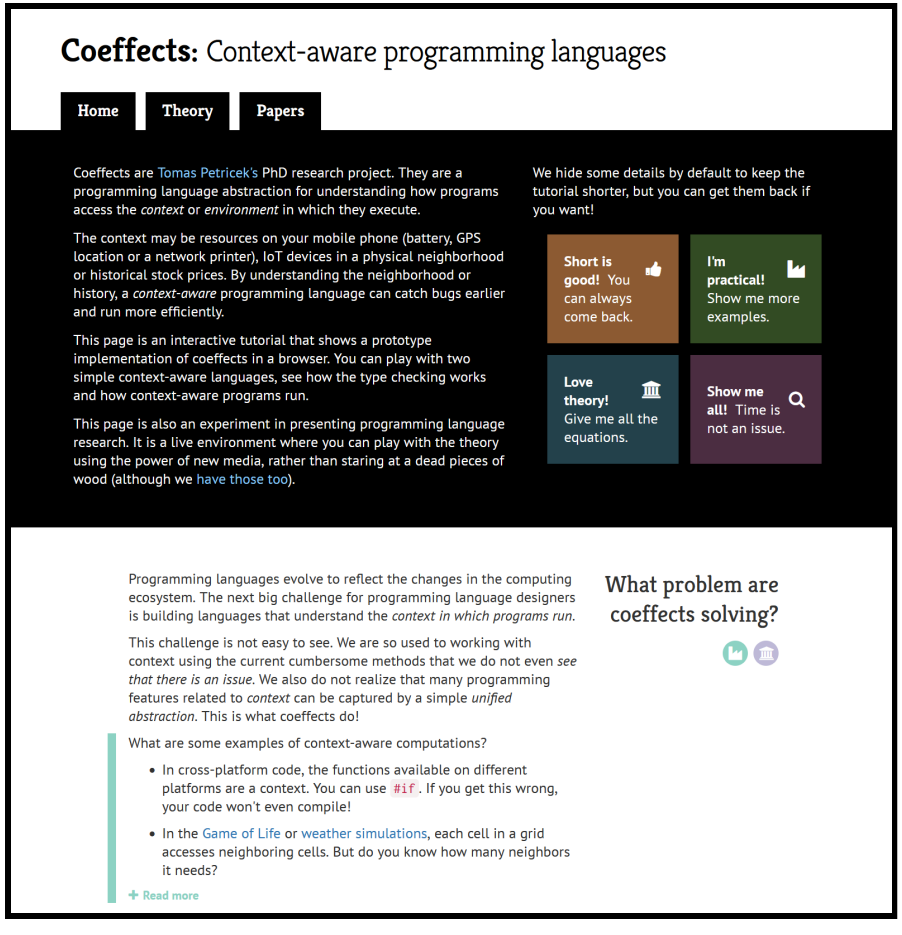
PhD, University of Cambridge
Context-aware programming languages

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
Principles of programming systems

Charles University, Prague
Human-computer interaction and history



Demo
Coeffects playground

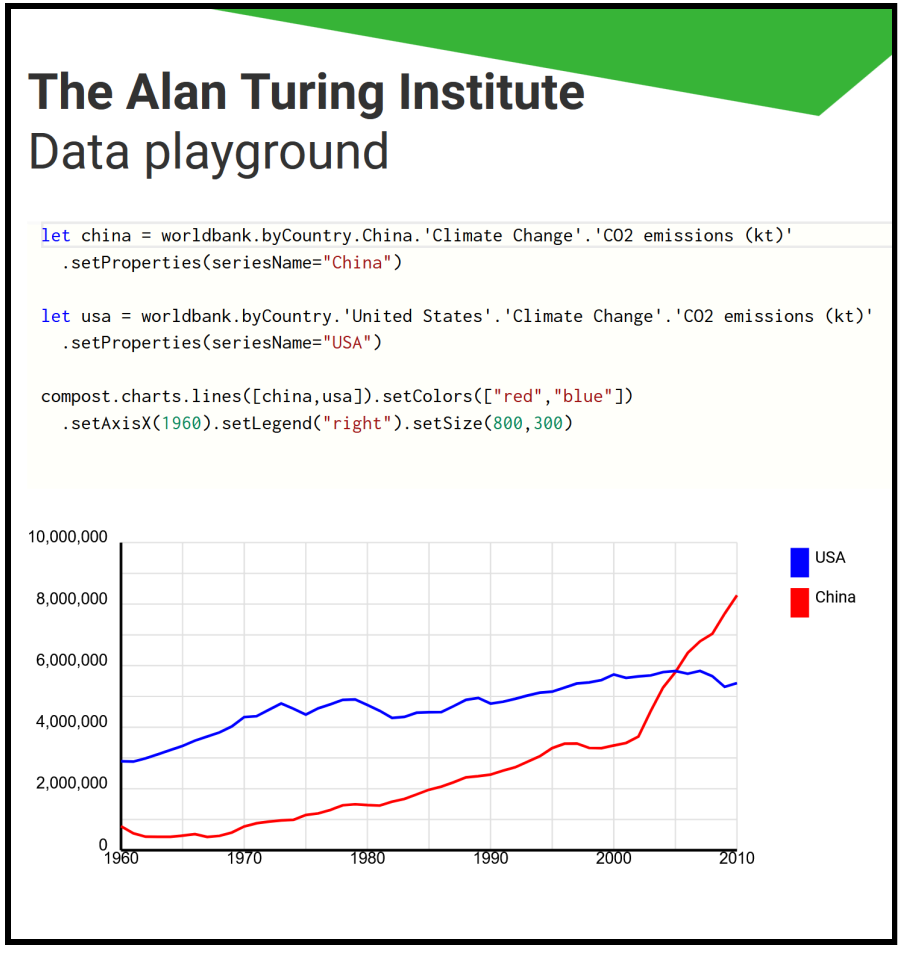
Needed "implementation" to finish my PhD!

How to show potential uses of theoretical work?

Tiny web demos of two potential applications

Charles University

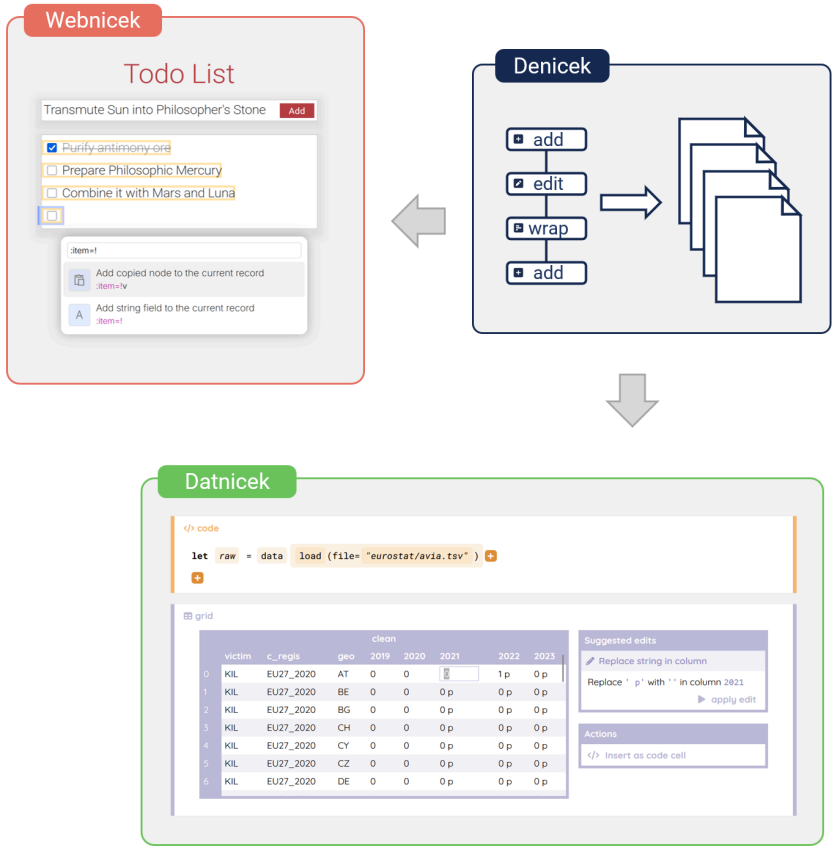
Demo
The Gamma project



Making programmatic data exploration accessible to non-programmers

Small typed language, but interaction is why it works!

Charles University



Demo

The Denicek project

Computational substrate for end-user programming

Making implementation of end-user programming experiences easier...

Charles University

Practical details

Course structure

- Videos to watch in advance
- Hands-on 3-hour labs
- Code skeleton with detailed comments

Doing the course

- Six different languages or systems
- Come to the labs to get help!
- Complete basic tasks for 4/6 systems

Charles University

PRG*PRG / Prague

Programming systems
& languages research

D3S at MFF CUNI
PRL at FIT CTU

Ask us about PhD &
post-doc opportunities!

https://prgprg.org

Write your own tiny programming system(s)

Tiny systems as a methodology

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077

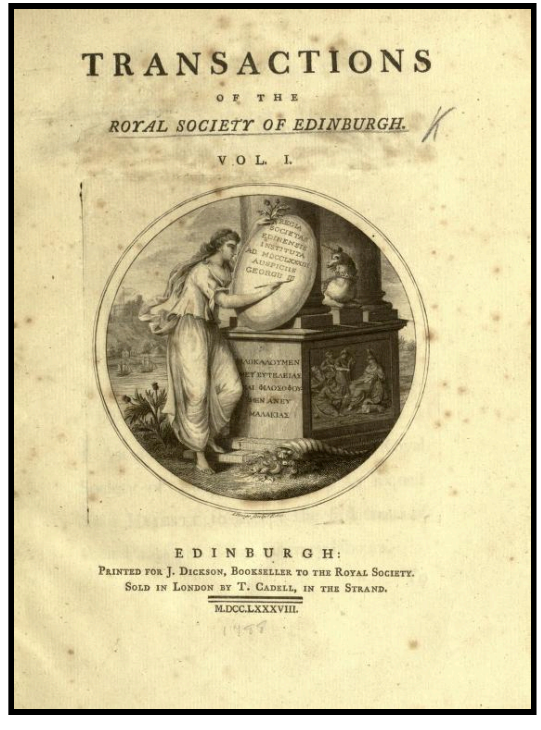
Academic research

What are we trying to study?

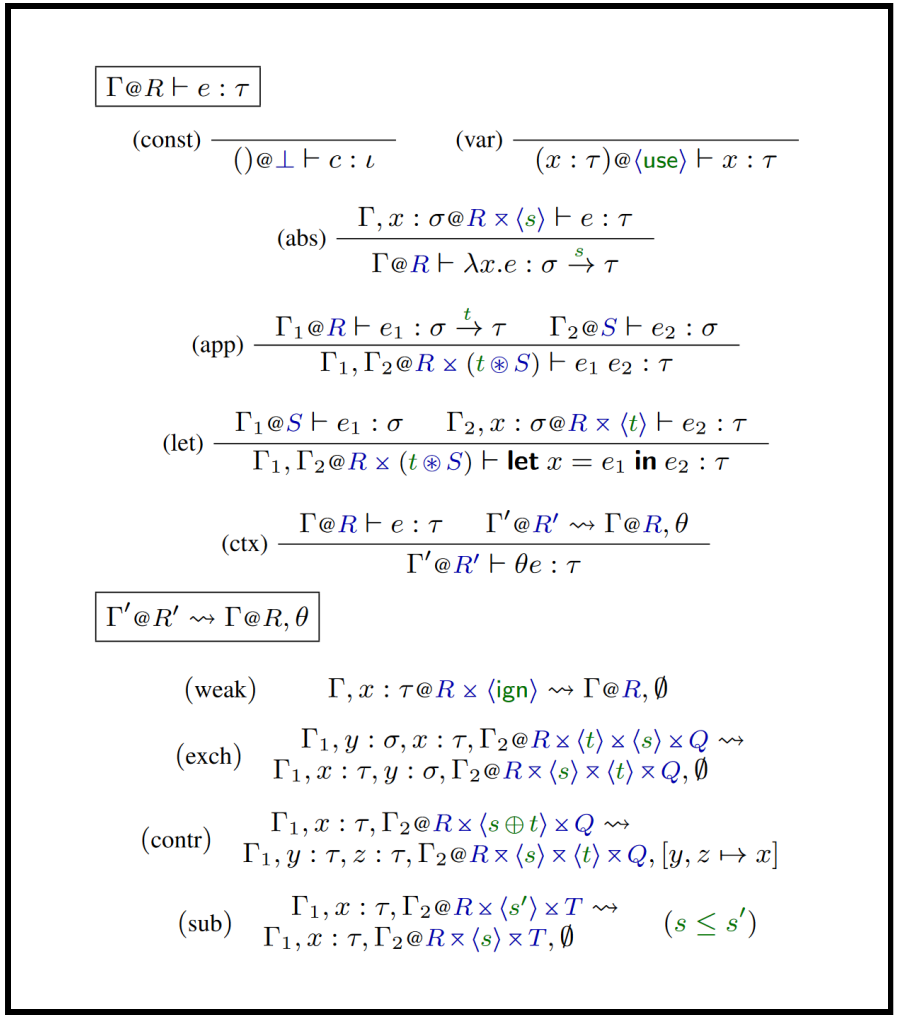
- Basic essential principles
- In isolation from other factors
- You have to ignore a lot!

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



$\boxed{\Gamma @ R \vdash e : \tau}$

$$(\text{const})\ \frac{}{() @ \bot \vdash c : \iota} \qquad (\text{var})\ \frac{}{(x : \tau) @ \langle \mathsf{use} \rangle \vdash x : \tau}$$

$$(\text{abs})\ \frac{\Gamma, x : \sigma @ R \times \langle s \rangle \vdash e : \tau}{\Gamma @ R \vdash \lambda x.e : \sigma \xrightarrow{s} \tau}$$

$$(\text{app})\ \frac{\Gamma_1 @ R \vdash e_1 : \sigma \xrightarrow{t} \tau \qquad \Gamma_2 @ S \vdash e_2 : \sigma}{\Gamma_1, \Gamma_2 @ R \times (t \circledast S) \vdash e_1\ e_2 : \tau}$$

$$(\text{let})\ \frac{\Gamma_1 @ S \vdash e_1 : \sigma \qquad \Gamma_2, x : \sigma @ R \times \langle t \rangle \vdash e_2 : \tau}{\Gamma_1, \Gamma_2 @ R \times (t \circledast S) \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau}$$

$$(\text{ctx})\ \frac{\Gamma @ R \vdash e : \tau \qquad \Gamma' @ R' \rightsquigarrow \Gamma @ R, \theta}{\Gamma' @ R' \vdash \theta e : \tau}$$

$\boxed{\Gamma' @ R' \rightsquigarrow \Gamma @ R, \theta}$

$$(\text{weak})\quad \Gamma, x : \tau @ R \times \langle \mathsf{ign} \rangle \rightsquigarrow \Gamma @ R, \emptyset$$

$$(\text{exch})\quad \begin{array}{l} \Gamma_1, y : \sigma, x : \tau, \Gamma_2 @ R \times \langle t \rangle \times \langle s \rangle \times Q \rightsquigarrow \\ \Gamma_1, x : \tau, y : \sigma, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, \emptyset \end{array}$$

$$(\text{contr})\quad \begin{array}{l} \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \oplus t \rangle \times Q \rightsquigarrow \\ \Gamma_1, y : \tau, z : \tau, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, [y, z \mapsto x] \end{array}$$

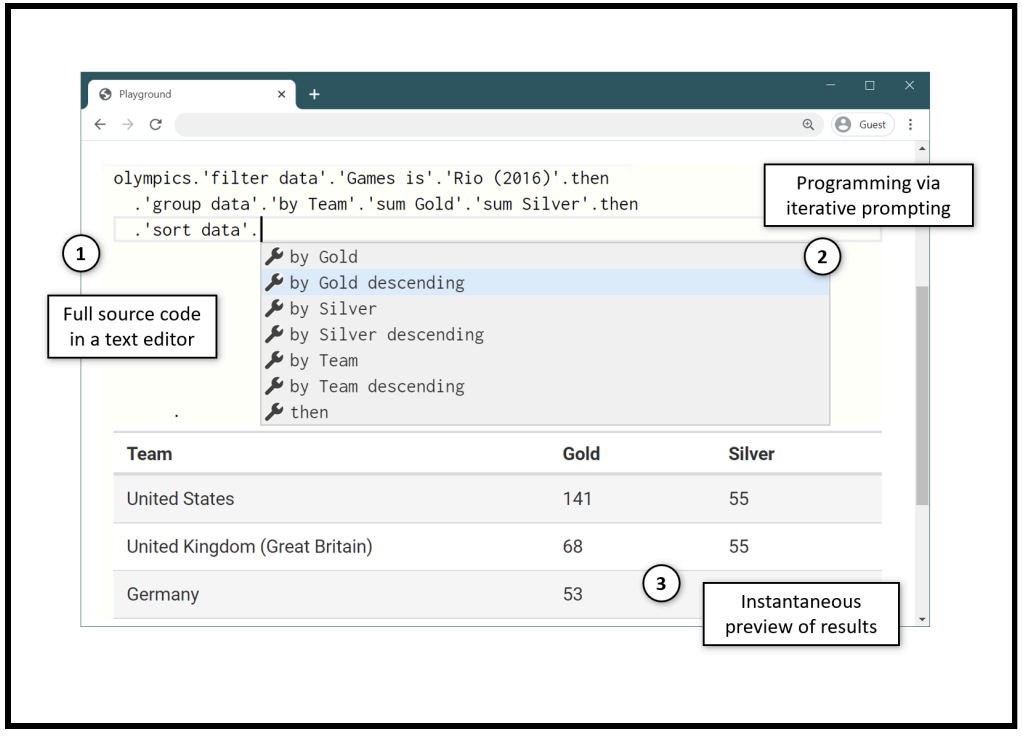
$$(\text{sub})\quad \begin{array}{l} \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s' \rangle \times T \rightsquigarrow \\ \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \rangle \times T, \emptyset \end{array} \qquad (s \leq s')$$

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

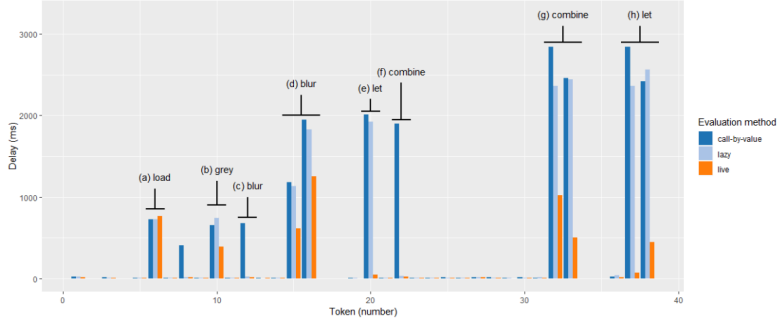


Figure 11 Time required to recompute the results of a sample program after individual tokens are added or modified for three different evaluations strategies.

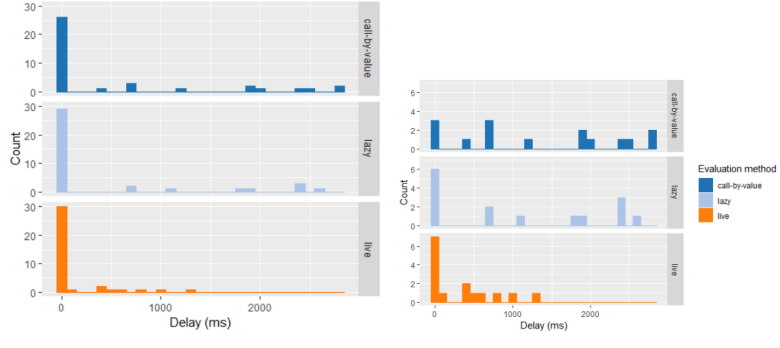


Figure 12 Distribution of delays incurred when updating previews. We show a histogram computed from all delays (left) and only from delays larger than 15 ms (right).

Charles University

Tiny systems

What is not covered?

- Syntax choices and writing parsers
- Compilation and JIT-based runtimes
- Formal semantics and correctness
- Supporting real-world use cases

Tiny systems

What can we study?

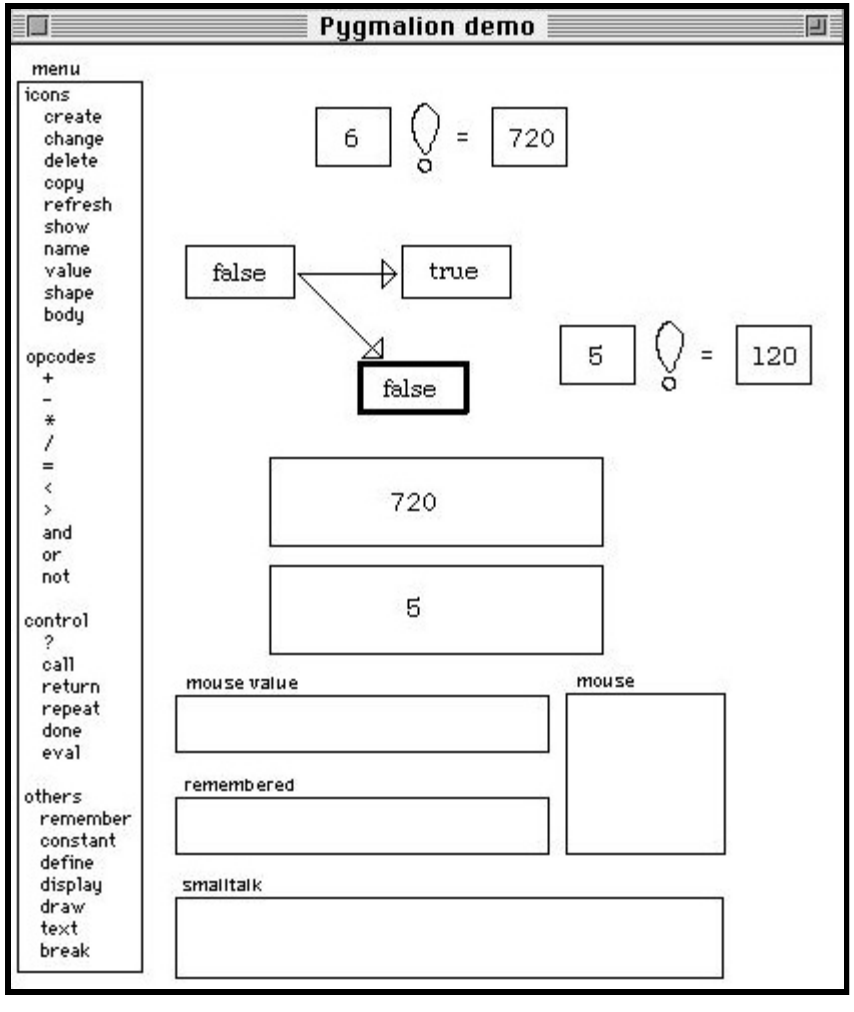

Can talk about stateful interactive systems


Implement key aspects of inner working


Reconstruct interesting past systems


But cannot be printed on 12 pages of A4

Charles University



Demo
Pygmalion

Why study 1908s programming systems?

No code programming!

First us of an "icon"!

Charles University

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!
Little may be enough...

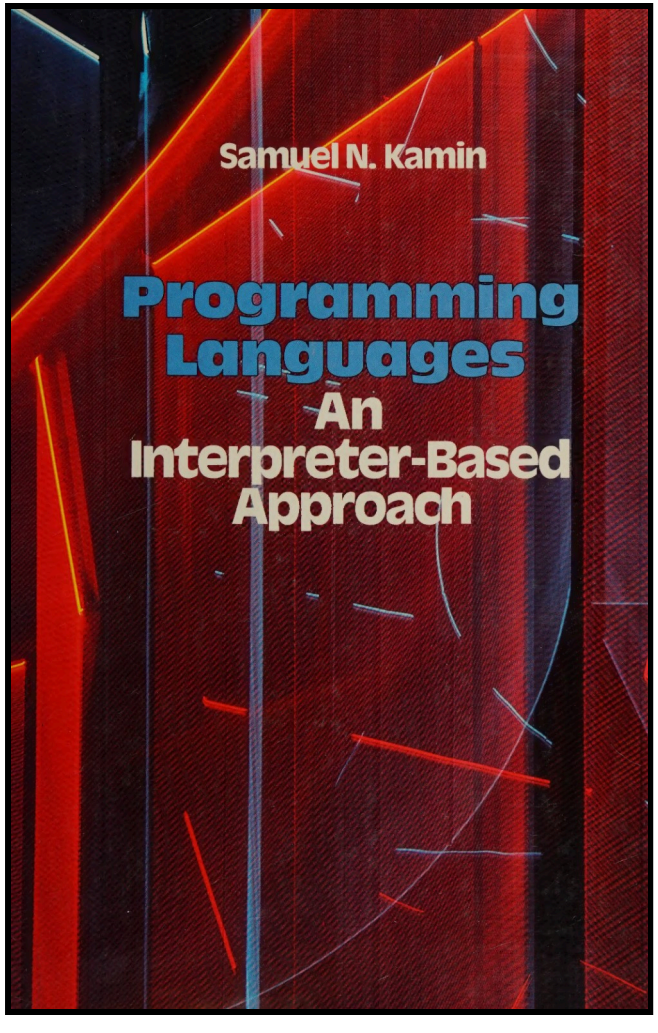
Research

Imagine new paradigms
End-user programming?

Focus on interaction
How exactly did it work

Ignore practical details
They can often wait!

Charles University



Teaching tiny systems

(Kamin, 1990)

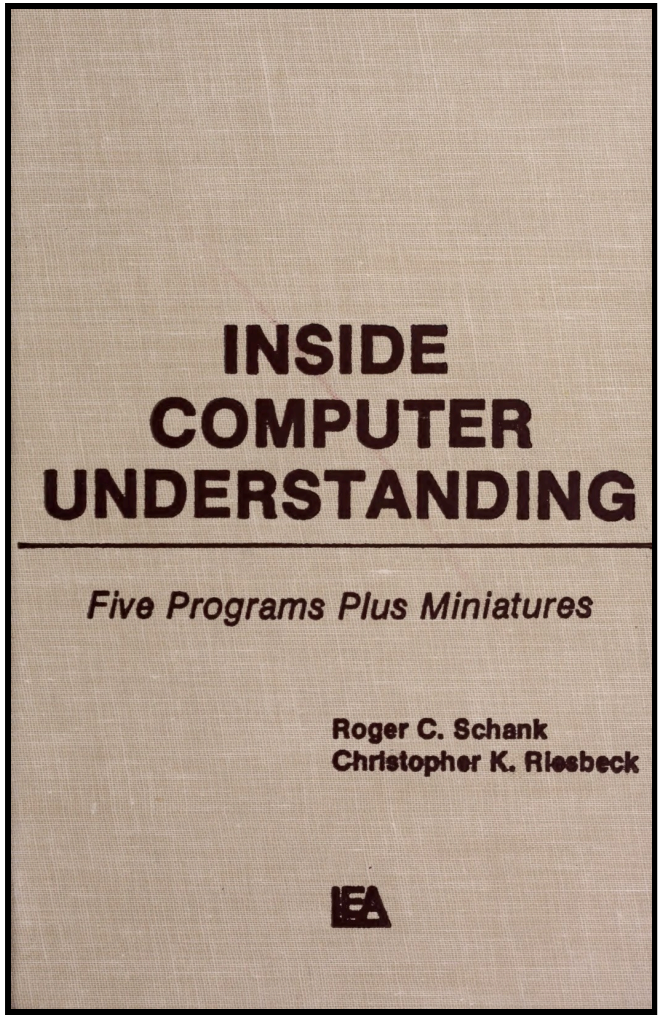
Used in multiple courses worldwide

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

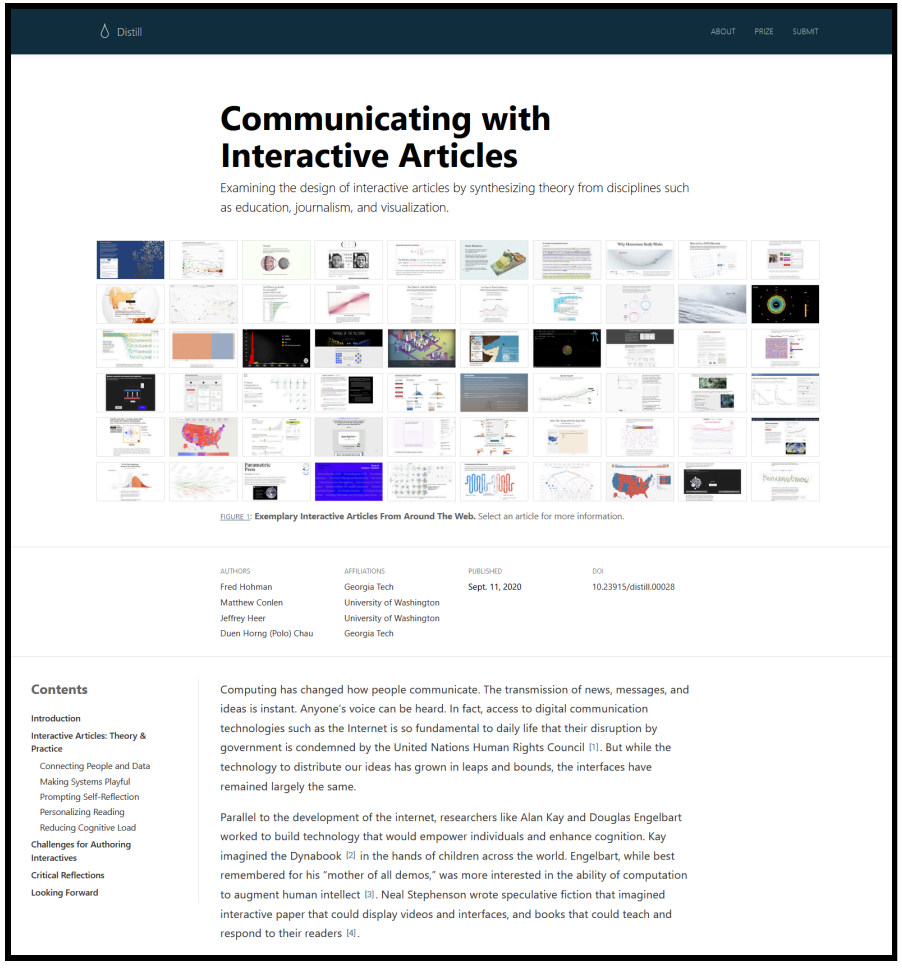
(Schank, Riesbeck, 1981)

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



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

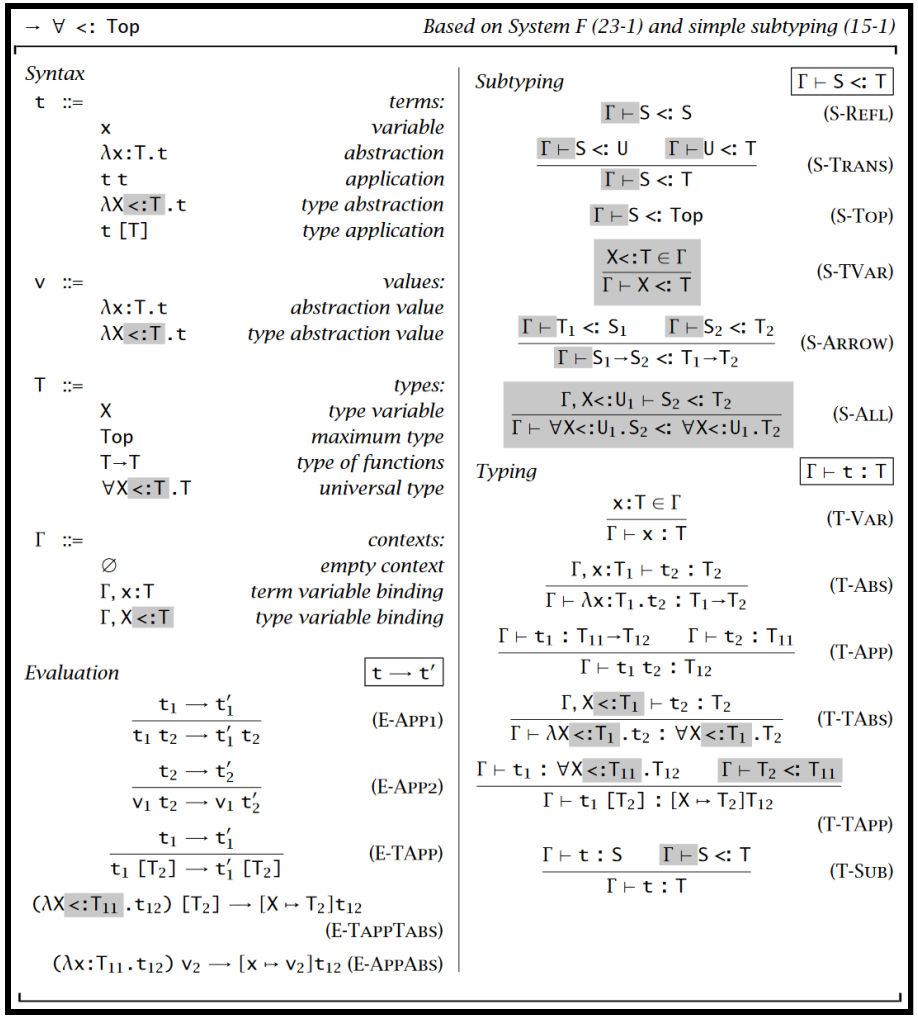
Charles University

Write your own tiny programming system(s)

Programming languages and systems

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077



→ ∀ <: 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 — $\Gamma \vdash t : T$

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

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

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

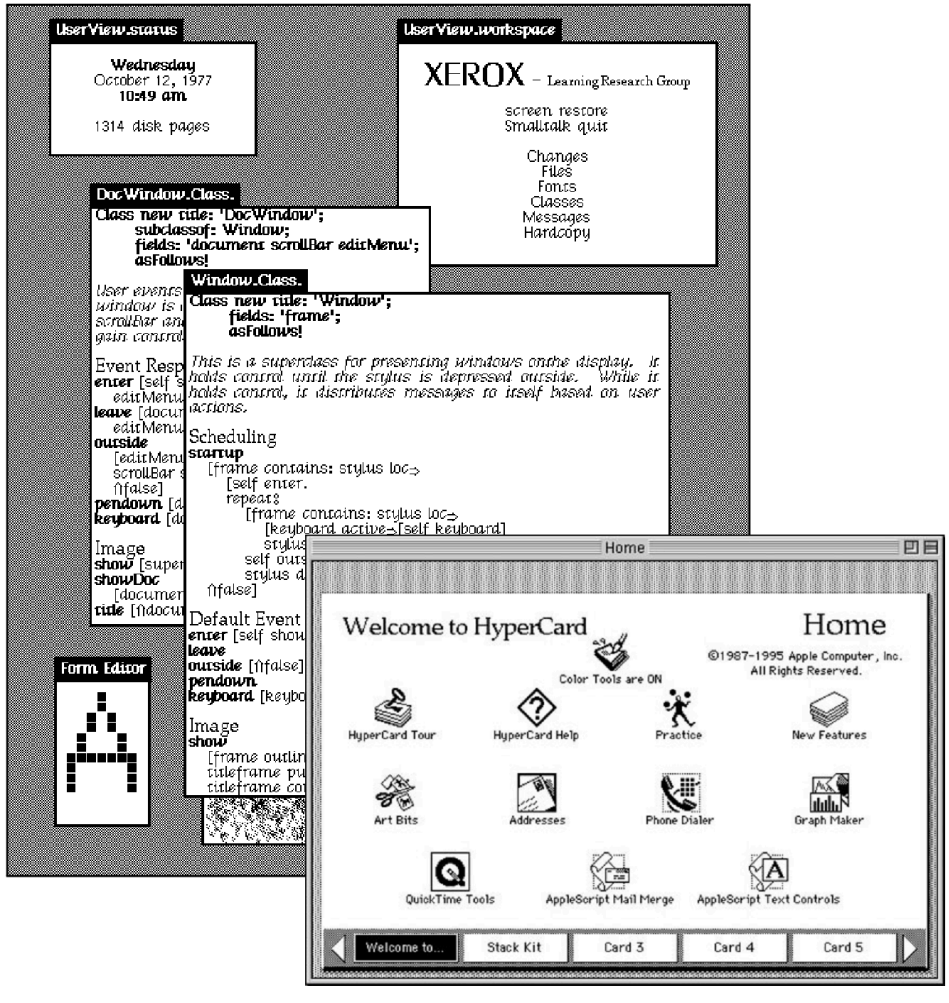
Programming Languages

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

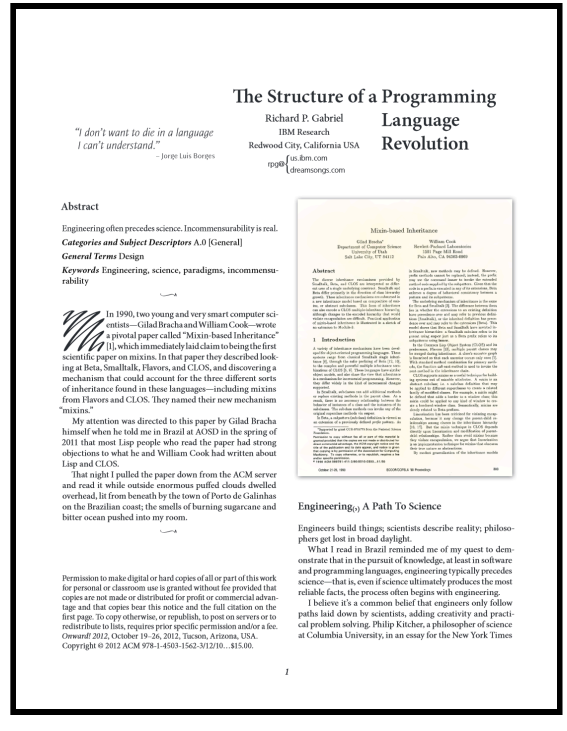
Paradigm shift in 1990s

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.

Onward! 2012, October 19–26, 2012, Tucson, Arizona, USA.

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

1

Charles University

Language paradigms

Functional programming

No mutable state, everything a function

Imperative programming

Mutable memory with pointers

Object-oriented programming

Everything an object, hides its own logic

Logic programming

Declare facts and use inference

Charles University

Demo

Logic programming in Prolog

System interaction

- Command line programming systems
  Code editor, compiler, build tools, etc.
- 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

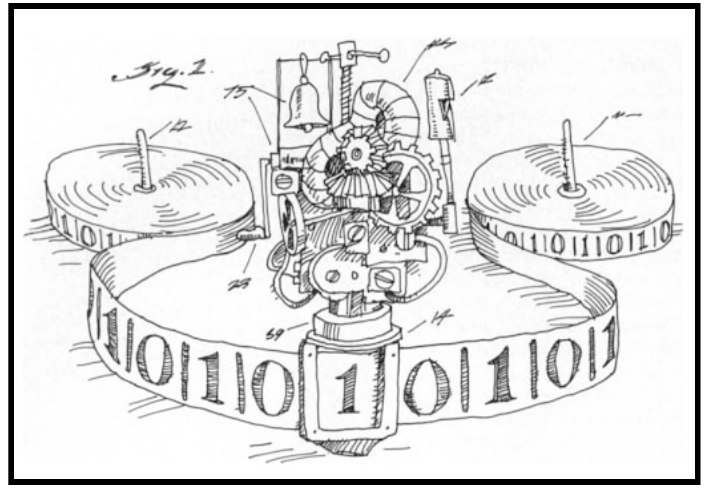
Demo

Object-orientation in Smalltalk

What really matters?

Static structure (program)

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



Dynamic structure (process)

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

Logic of evaluation (execution)

- How the dynamic state evolves?

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

Why write small interpreters instead?

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
```

Code can run!

A good way to explain the structures!

1) Functional data types for the static and dynamic structure

2) A function to model the evaluation logic

Charles University

Write your own tiny programming system(s)

A tase of the F# language

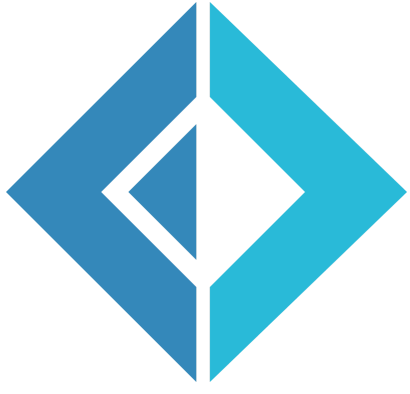
Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077

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

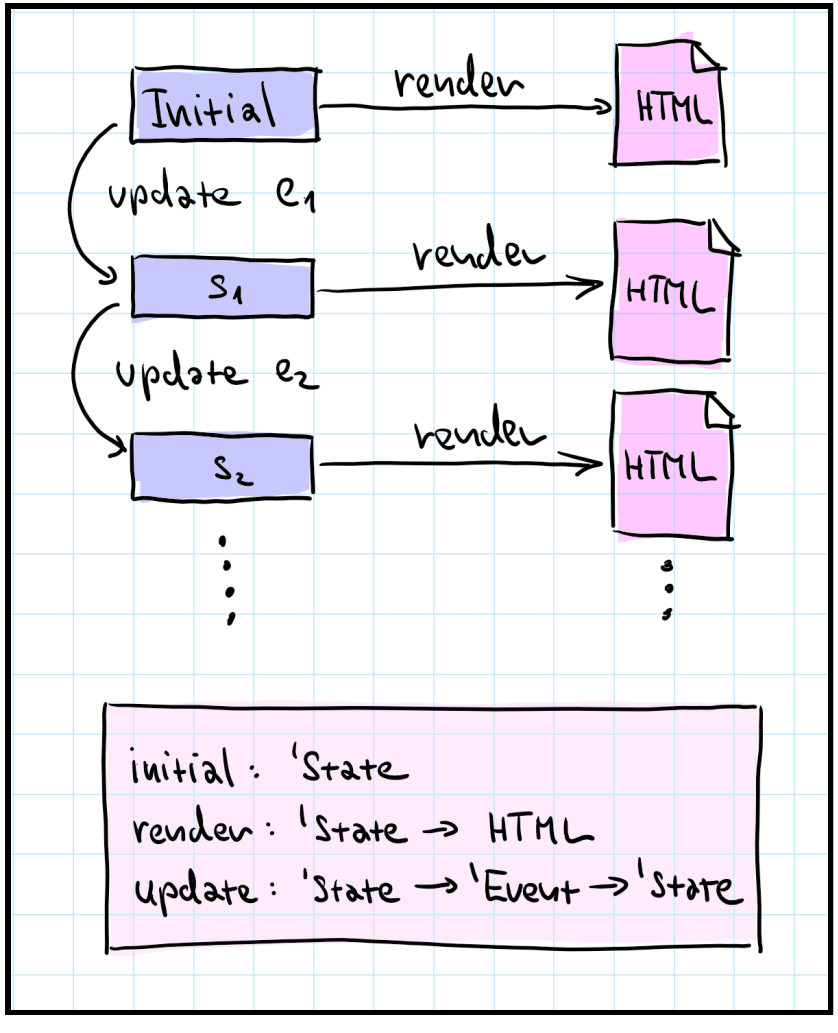

I like the language and can help you!

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

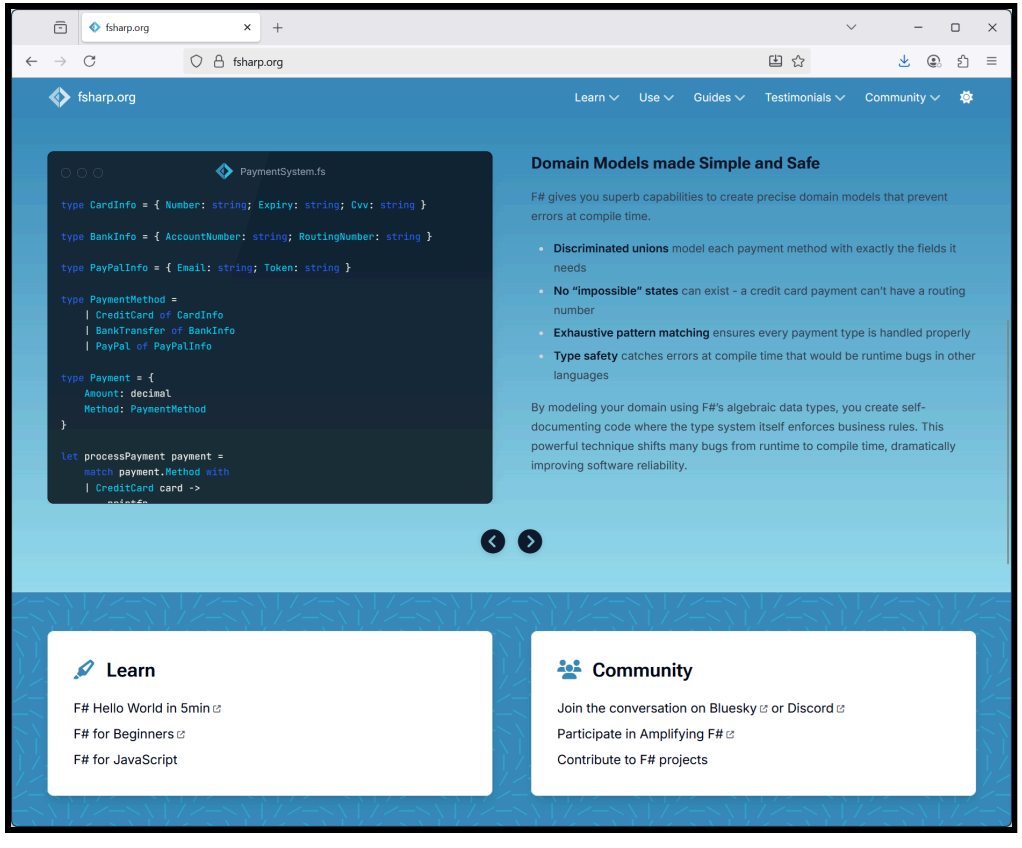
Charles University

Demo

Building a counter in F#

Demo

Building a TODO list in F#



More about F#

https://fsharp.org

We will only need a small part of the language!

I will introduce all constructs we will need as we go...

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