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

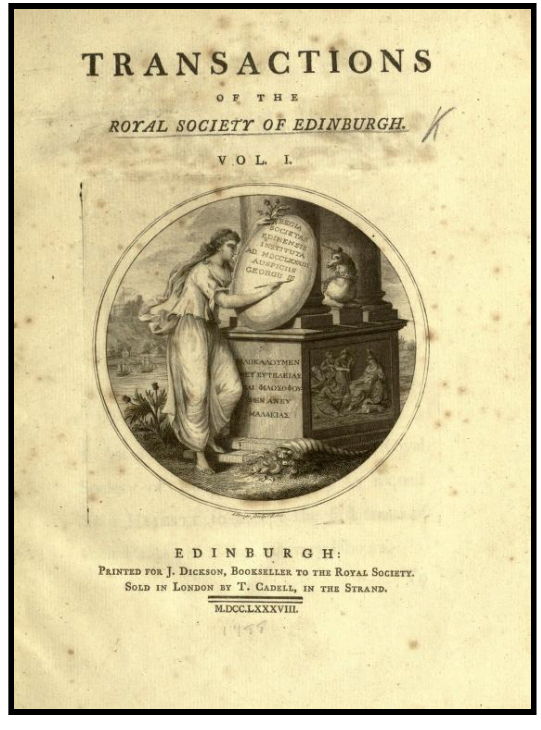
Academic research

What are we trying to study?

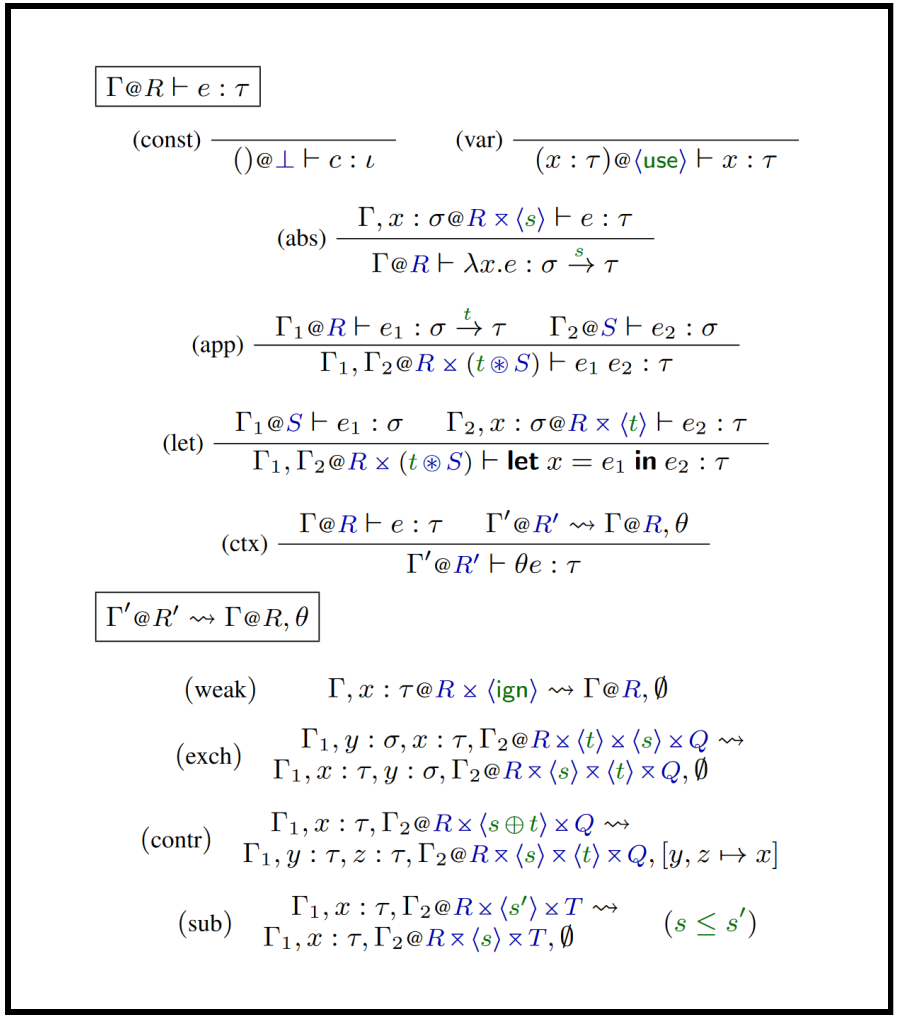
- Basic essential principles
- In isolation from other factors
- You have to ignore a lot!

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



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

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

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

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

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

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

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

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

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

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

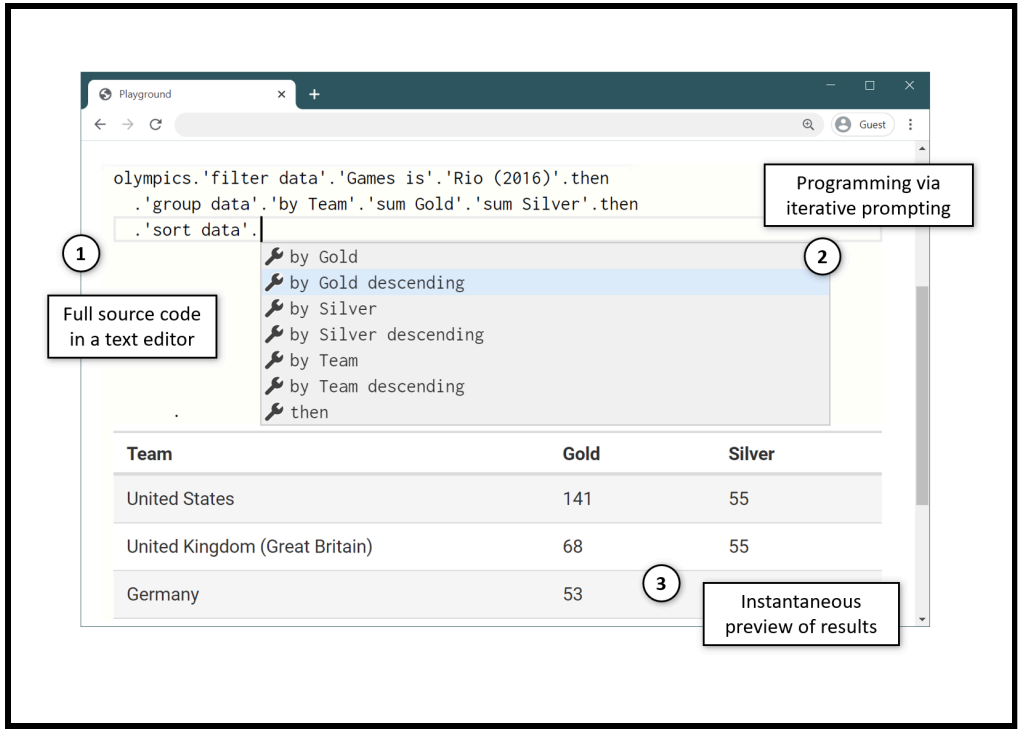
$$(\text{sub})\quad \begin{array}{l} \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s' \rangle \times T \rightsquigarrow \\ \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \rangle \times T, \emptyset \end{array} \qquad (s \leq s')$$

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

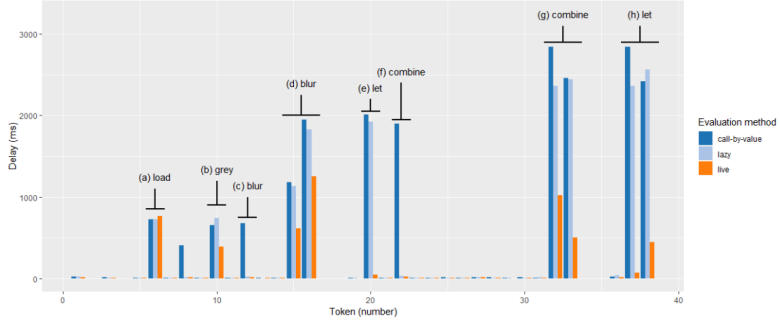


Figure 11 Time required to recompute the results of a sample program after individual tokens are added or modified for three different evaluations strategies.

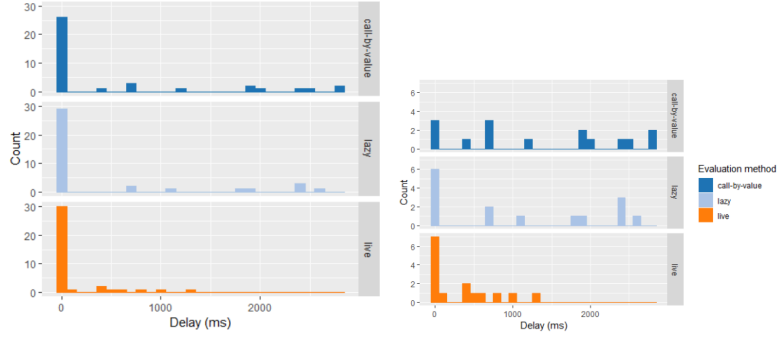


Figure 12 Distribution of delays incurred when updating previews. We show a histogram computed from all delays (left) and only from delays larger than 15 ms (right).

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?

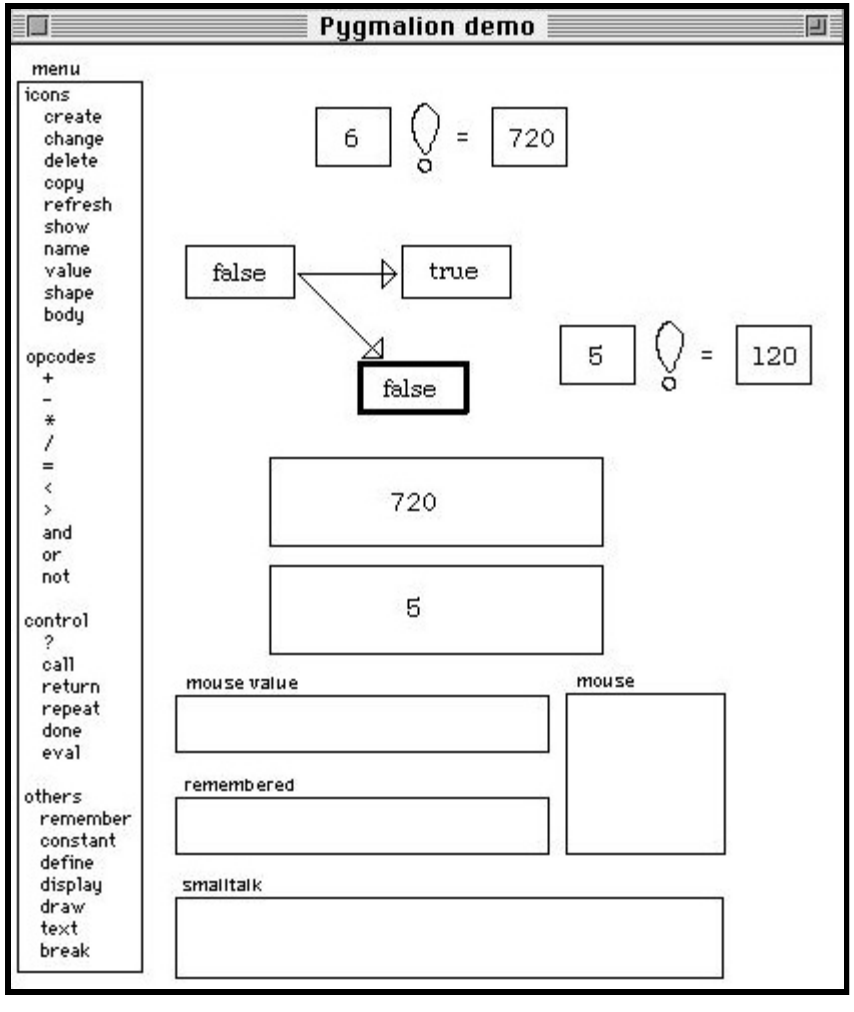

Can talk about stateful interactive systems


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...

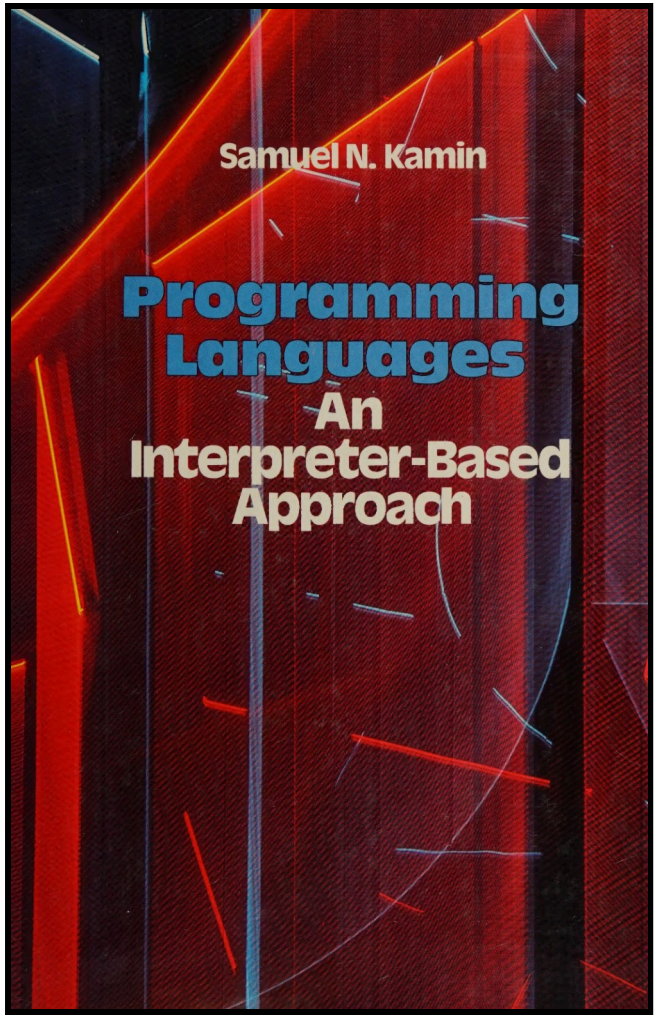
Research

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)

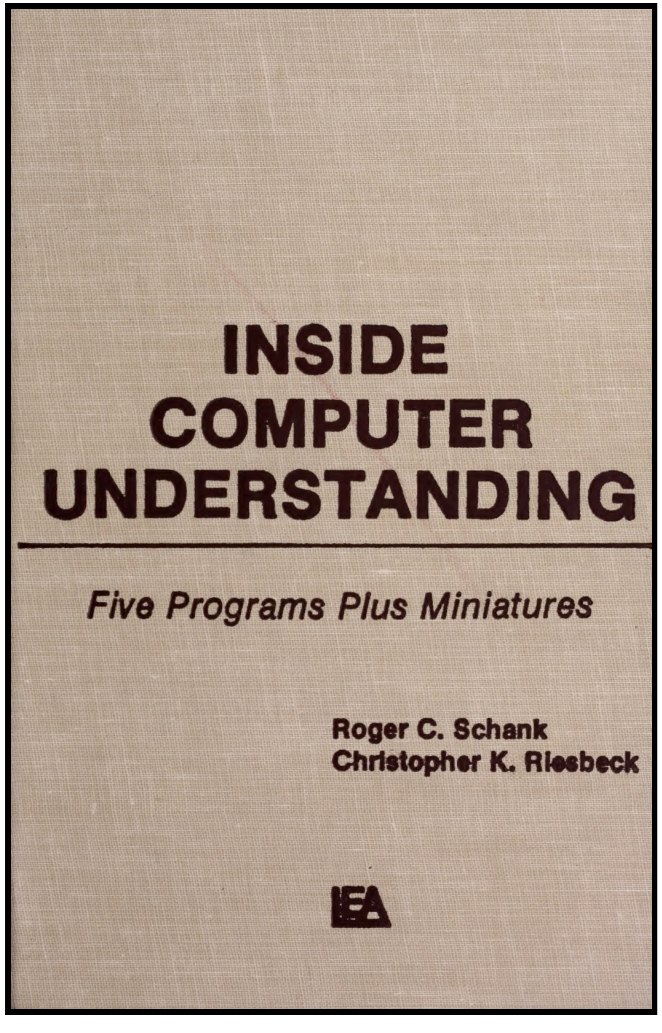
Used in multiple courses worldwide

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

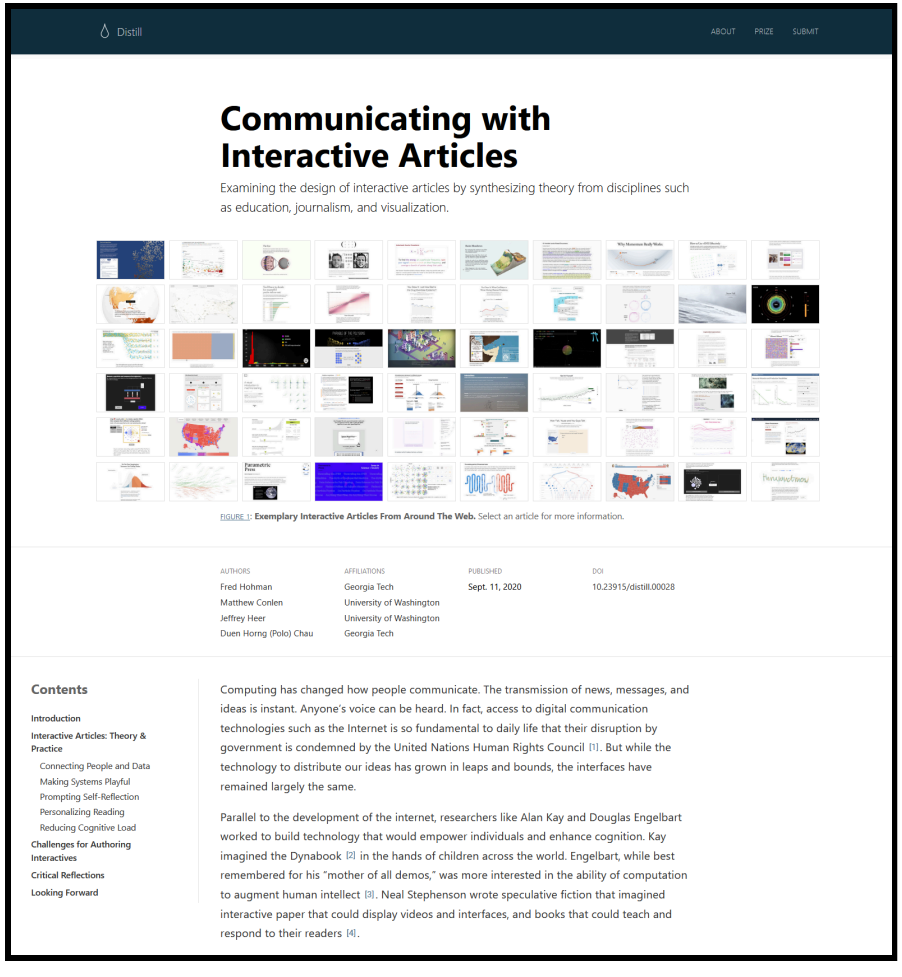
(Schank, Riesbeck, 1981)

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