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?
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
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
Programming Languages

Programming is writing code

Formal semantics, implementation, paradigms, types

We know how to study this!
Programming Systems

Interacting with a stateful system

Feedback, liveness, interactive user interfaces

But how do we study this?
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!
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?
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?
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
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

<table>
<thead>
<tr>
<th>Education</th>
<th>Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writer</td>
<td>Imagine new paradigms</td>
</tr>
<tr>
<td>Understand principles</td>
<td>Variable names</td>
</tr>
<tr>
<td>I hope you'll have fun!</td>
<td>Focus on interaction</td>
</tr>
<tr>
<td>Doing more with less?</td>
<td>Ignore practical details</td>
</tr>
<tr>
<td></td>
<td>New mode of interaction</td>
</tr>
</tbody>
</table>
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
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
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

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

Contents
Introduction
Interactive Articles: Theory & Practice
Connecting People & Data
Making Systems Playful
Prompting Self-Reflection
Personalizing Reading
Reducing Cognitive Load
Challenges for Authoring Interactive Articles
Critical Reflections
Looking Forward

Computing has changed how people communicate. The transmission of news, messages, and ideas is instantaneous. 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 governments is condoned by the United Nations Human Rights Council. However, 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 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. 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.
Programming models
Learning by implementing
Programming models

Language paradigms

→ Functional programming

ató Imperative programming

● Object-oriented programming

● Logic programming
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
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?
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
Operational semantics

Standard approach to programming language theory

Equations vs. Code

Code actually runs! Easier to write?
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
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?
Programming language theory

Ignore implementation and practical features

Prove that the core idea is formally sound
Human-computer interaction (HCI)

Ignore inner working and implementation

Show that users can actually use it and how
Performance evaluation

Ignore usability and design implications

Show that you can do better than a baseline
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
### 110 DRAWING A MAZE

There is a lot of clever hacks that you can do in BASIC with a few lines of code. This ease of getting started contributes to what makes it a fun programming environment. If you found an interesting hack in a computer magazine, you could type it into the console and run it straight away.

The fact that you had to copy code from a paper magazine sounds like a hassle, but it has an educational quality. It keeps the samples that can be distributed in this way reasonably small and it makes you think about the code as you are typing it.

To experience this yourself, you should try typing the following three-line program to the console! It generates a famous maze. This relies on special Commodore character codes: 147 clears the screen, 205 and 206 are backslash and slash crossing the full character size.

```basic
10 PRINT CHR$(147);
20 PRINT CHR$(205.5 + RND(1));
30 GOTO 20
RUN
```

The maze is so famous that it has a whole book written about it [1]. To make it run, you use the command separator `;` which my interpreter does not support. The book inspired not just this example, but some aspects of the style of this article; and it is a great read!

The character set used by Commodore 64 is called PETSCII. The interpreter on this page is using the fantastic C64 TrueType font, which maps Unicode characters to PETSCII, but also includes original key codes in an unused Unicode region, used by the CHR$ function.


---

### 200 CREATING A MOVING BALL

To build our Breakout game, we can proceed gradually. This is yet another nice feature of the programming environment. We want to create a ball that bounces off the wall, but let's start with a ball that just moves to the right.

We will do only a tiny bit of planning. Code that initializes variables with the game state starts at line 1000 and code that handles ball movement will start at 2000. We will also first clear the screen and use DELETE to remove all the previous maze and Hello World code.
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
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!
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

```
initial: 'State
render: 'State -> HTML
update: 'State -> 'Event -> 'State
```
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
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://tomas.net | @tomaspetricek
➡️ https://d3s.mff.cuni.cz/teaching/nprg075
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
  Understanding Five Programs Plus Miniatures
  approach. Addison-Wesley.
- Kamin, S. (1990) PLIBA source code mirror on GitHub
  Kinds of Computer Program, Their Uses and Abuses