NPRG075

Programming language design

Tomáš Petříček, 204 (2nd floor)
petricek@d3s.mff.cuni.cz
https://tomasp.net | @tomaspetricek

Lectures: Tuesday 12:20, S6
https://d3s.mff.cuni.cz/teaching/nprg075

Introduction

What? Why? How?

Making programming
(languages | experience | systems)
better!

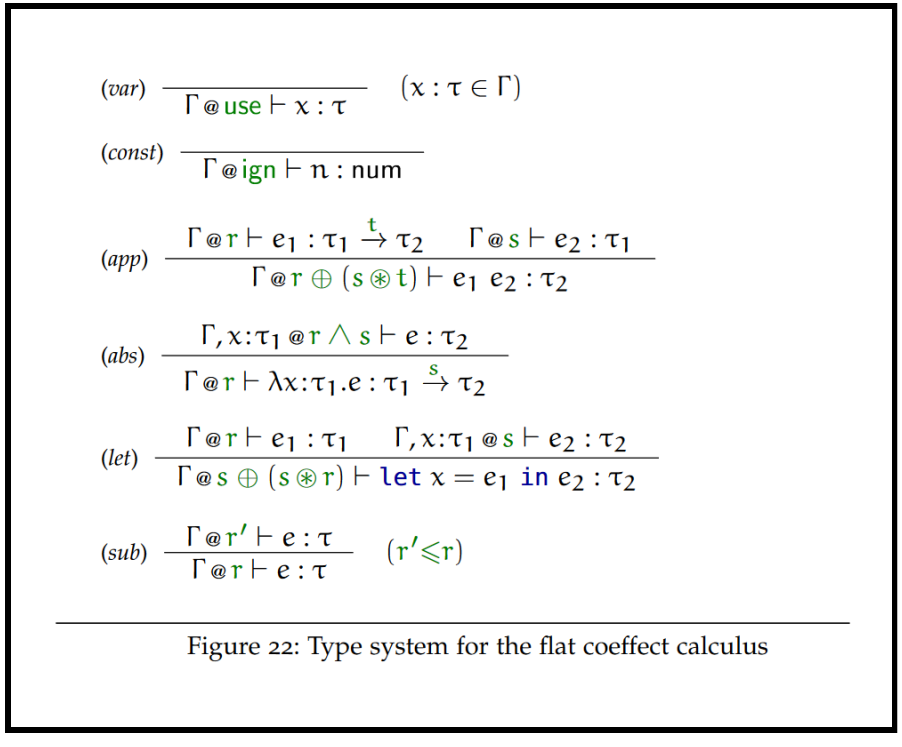
My background

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
Programming systems and learning from the past



$$(var)\ \frac{}{\Gamma @ \mathsf{use} \vdash x : \tau}\ (x : \tau \in \Gamma)$$

$$(const)\ \frac{}{\Gamma @ \mathsf{ign} \vdash n : \mathsf{num}}$$

$$(app)\ \frac{\Gamma @ r \vdash e_1 : \tau_1 \xrightarrow{t} \tau_2 \quad \Gamma @ s \vdash e_2 : \tau_1}{\Gamma @ r \oplus (s \circledast t) \vdash e_1\ e_2 : \tau_2}$$

$$(abs)\ \frac{\Gamma, x{:}\tau_1 @ r \wedge s \vdash e : \tau_2}{\Gamma @ r \vdash \lambda x{:}\tau_1.e : \tau_1 \xrightarrow{s} \tau_2}$$

$$(let)\ \frac{\Gamma @ r \vdash e_1 : \tau_1 \quad \Gamma, x{:}\tau_1 @ s \vdash e_2 : \tau_2}{\Gamma @ s \oplus (s \circledast r) \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \tau_2}$$

$$(sub)\ \frac{\Gamma @ r' \vdash e : \tau}{\Gamma @ r \vdash e : \tau}\ (r' \leqslant r)$$

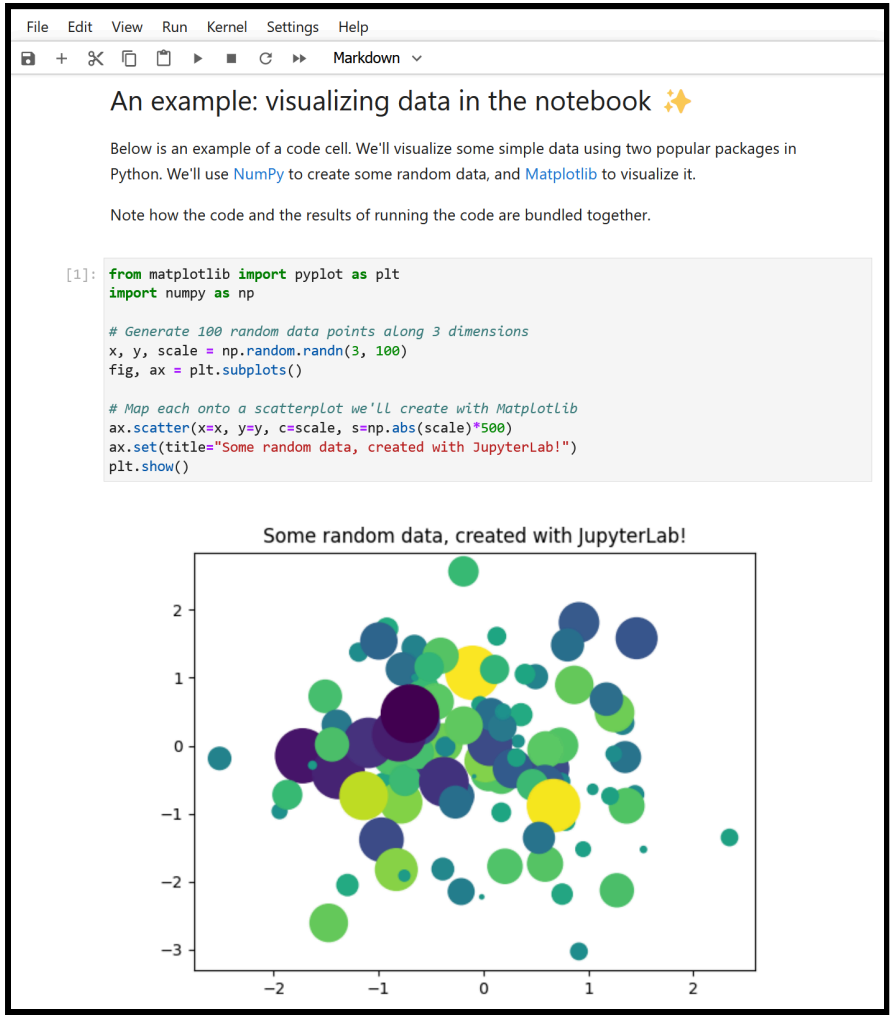
Figure 22: Type system for the flat coeffect calculus

Types for context-aware programming

Program as expression in small formal language

Type system determines what programs are valid

Safety proof shows no unauthorized accesses

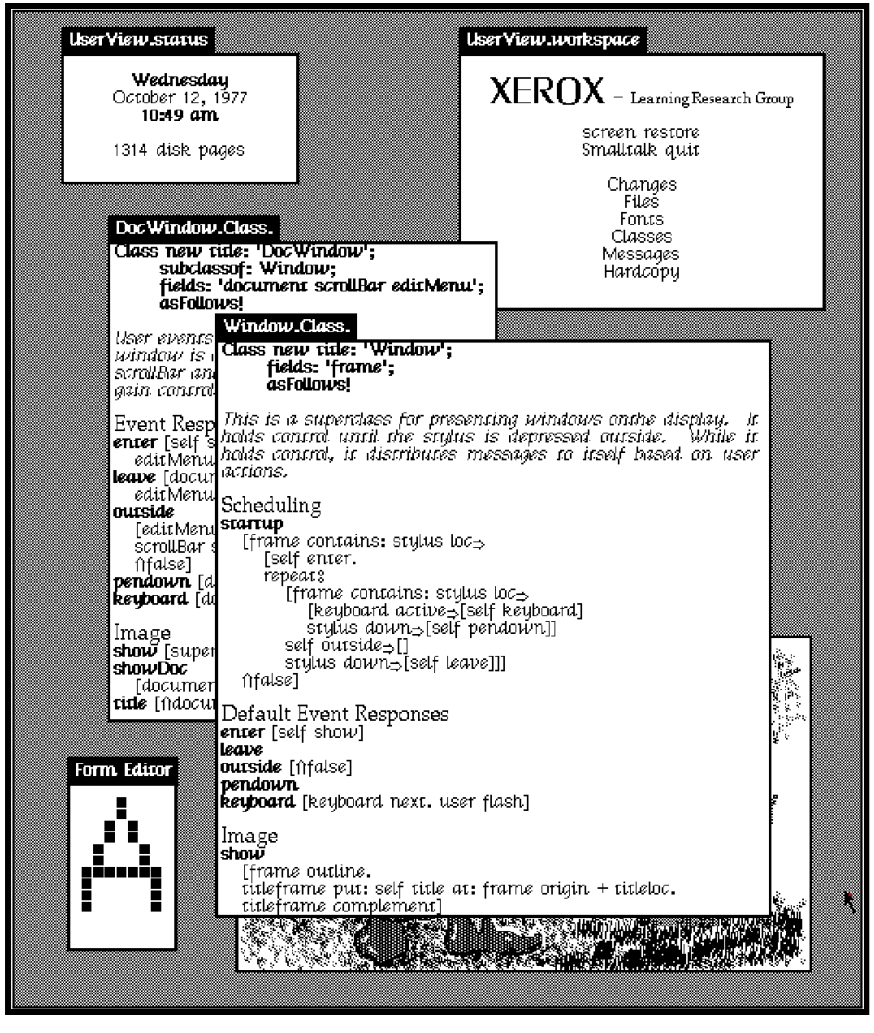


Data science tools and languages

Result is a document not a program

Working with one concrete dataset

Different language and system requirements!



Programming systems & history

Interacting with a stateful environment

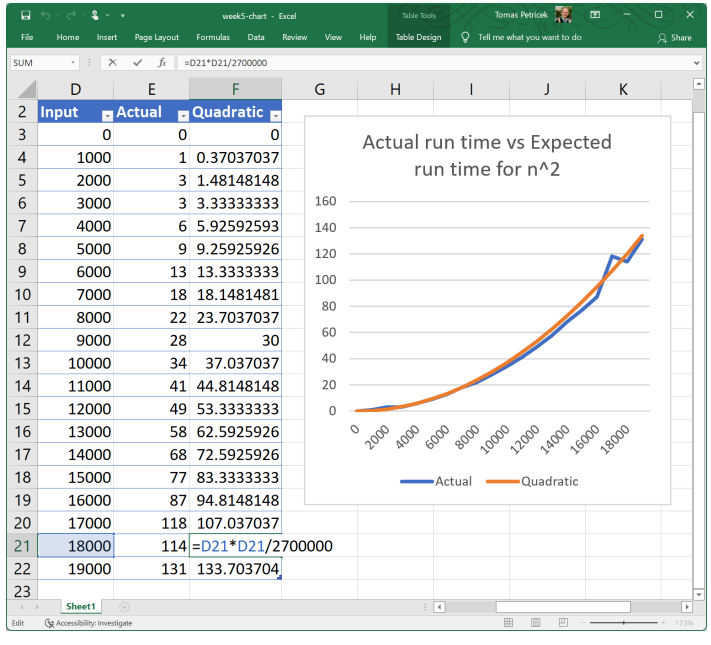
Let programmers do more in new ways...

It's not just a language!

Bringing everything together

Systems ⊃ languages

- Programming process matters
- Tools shape languages
- Harder to formalize & study!



	D	E	F
2	Input	Actual	Quadratic
3	0	0	0
4	1000	1	0.37037037
5	2000	3	1.48148148
6	3000	3	3.33333333
7	4000	6	5.92592593
8	5000	9	9.25925926
9	6000	13	13.3333333
10	7000	18	18.1481481
11	8000	22	23.7037037
12	9000	28	30
13	10000	34	37.037037
14	11000	41	44.8148148
15	12000	49	53.3333333
16	13000	58	62.5925926
17	14000	68	72.5925926
18	15000	77	83.3333333
19	16000	87	94.8148148
20	17000	118	107.037037
21	18000	114	=D21*D21/2700000
22	19000	131	133.703704
23			

Interdisciplinary research

- Formal language models
- Systematic design
- Qualitative and quantitative studies

Case study: LINQ

LINQ queries in Visual Basic .NET and C#

```
Dim db As New northwindDataContext
Dim ukCompanies =
  From cust In db.Customers
  Where cust.Country = "UK"
  Select cust.CompanyName, cust.City
```

Why confuse programmers familiar with SQL?

```
SELECT [CompanyName], [City]
WHERE [Country] = 'UK'
FROM dbo.[Northwind]
```

What to expect?

Content and materials

Many different programming systems

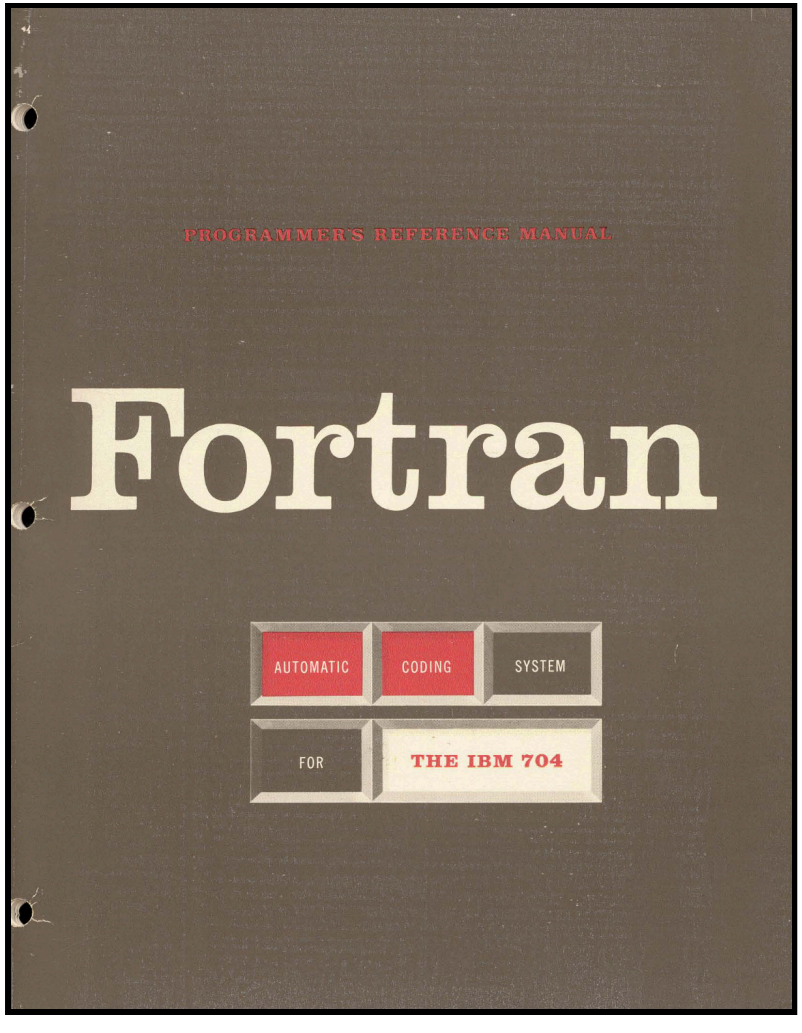
TypeScript, Jupyter, ML/F#, Smalltalk, BASIC

Many different research methods

Design, logic, proofs, user studies

Evolving course from two years ago

Slides on the web, but no textbook



Credit / zápočet

Small independent or group project

Using any of the covered method

Described in a brief report (4 pages)

Deadlines
Topic by December 20
Draft by February 28

Programming Langauges and Systems in Prague

Check out prgprg.org!

Join our reading group (on Wednesday)

Find other cool courses (MFF and firends at FIT)

Sign up for email updates

Programming languages

Conventional topics

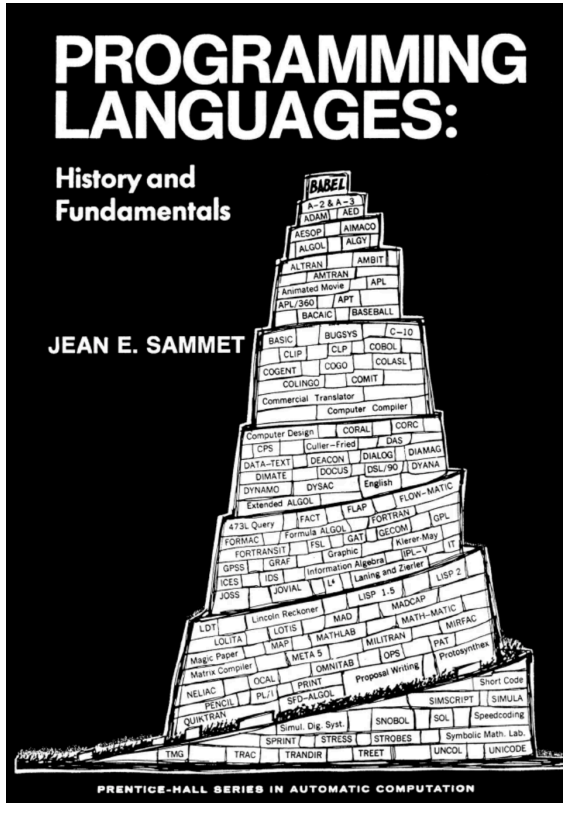
Paradigms and features

Language paradigms

- Functional, OOP, Logic, etc.
- Their fundamental concepts
- Interesting "extreme" designs

Language features

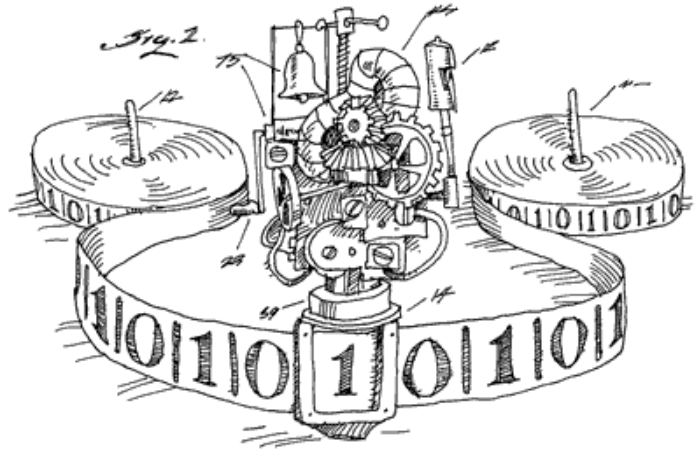
- Variable scoping, pointers
- Lambda abstraction, inheritance
- Design and implementation



Theory and implementation

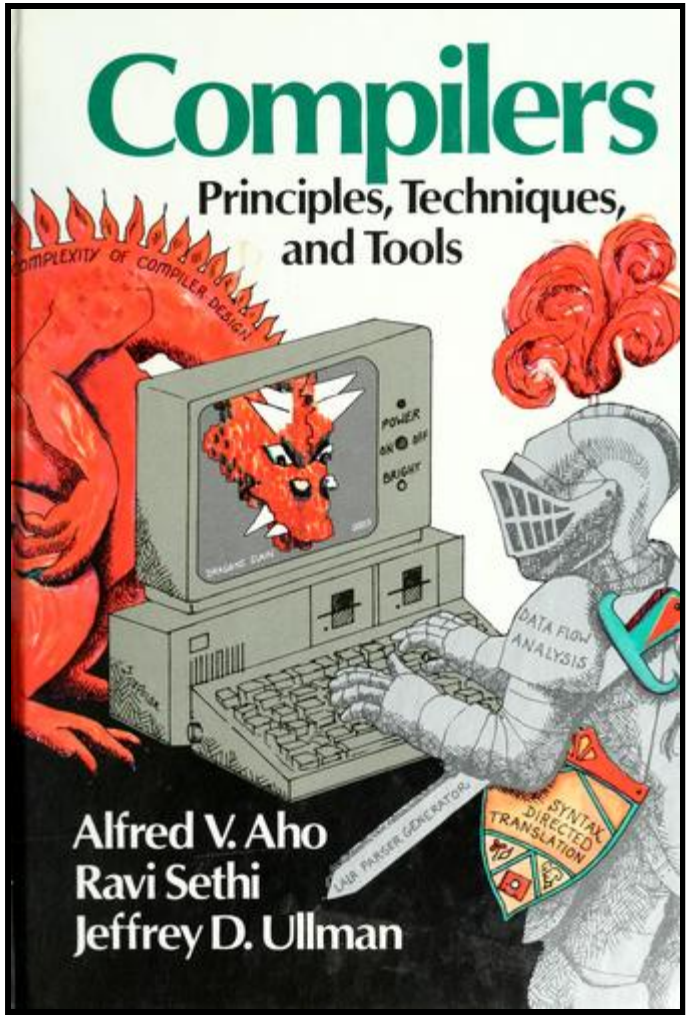
Parsing and automata

- Theory of formal grammars
- Parser implementation
- Computability theory



Compilers and interpreters

- Implementation techniques
- Register allocation
- Meta-circular interpreters



Why is this not enough?

Talks about "how" but not about "what"

Treat design as a research problem!

What can we study about programming systems?

Design

As a research discipline

What is design?

Design is the intentional **solution of a problem**, by the creation of plans for a **new sort of thing**, where the plans would not be immediately seen, by a reasonable person, as an inadequate solution.

Parsons (2015)

Designerly ways

Sciences study natural world

- By experiment, aiming at truth

Humanities study experience

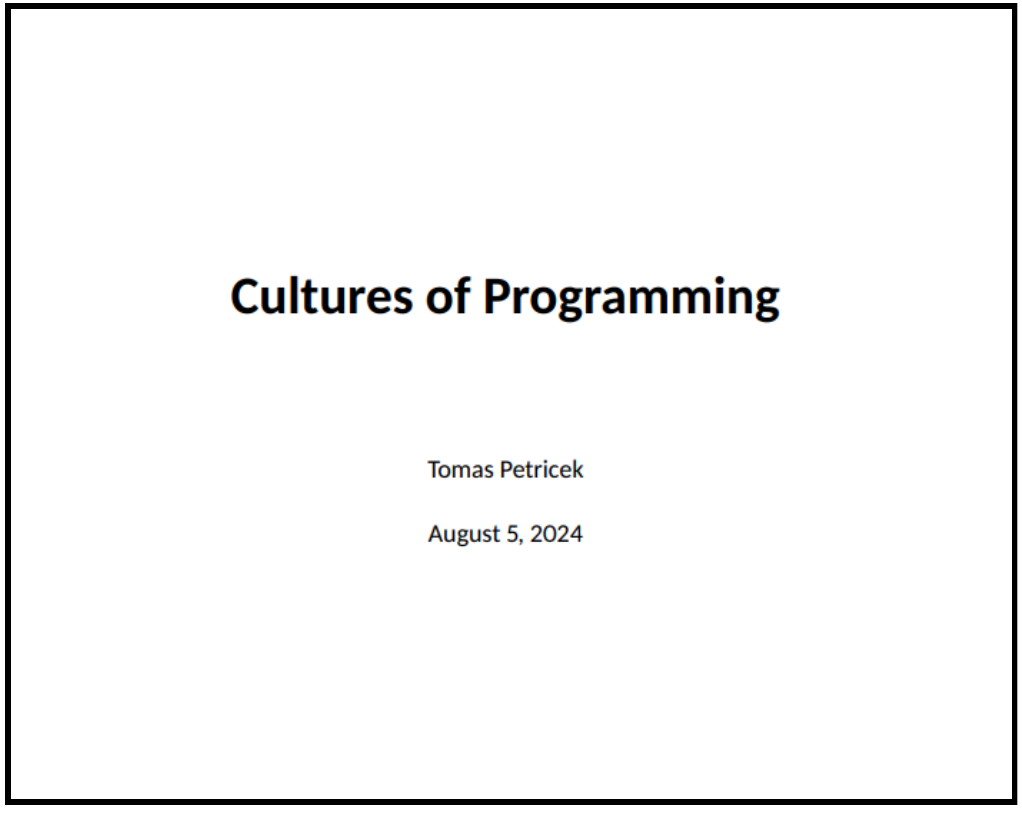
- By analogy, aiming at justice

Design studies the artificial

- By synthesis, aiming at appropriateness

Cultures of programming

Common ways of thinking



Cultures of programming

Different basic ways of thinking

What is a computer program, how to best create one?

Five different cultures of programming!

Case study: TypeScript

Unsoundness by design

- Type checking limitations!
- It's a feature, not a bug?
- tinyurl.com/nprg075-ts

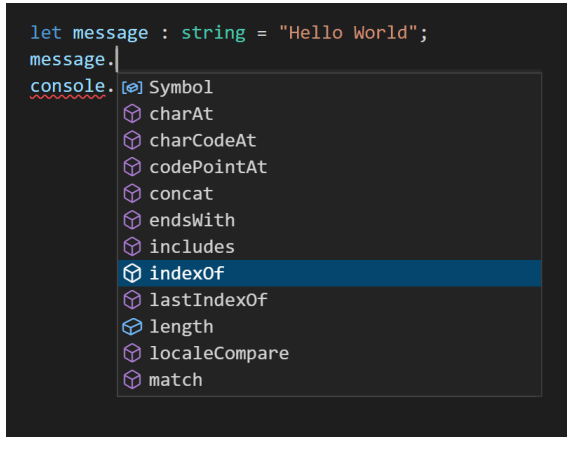
Design questions

- What research methods to use?
- Is partial soundness a thing?
- Is there a better design?
- What does "better" mean?

Cultures of programming

Engineering culture

- Programs are complex systems
- Tools can help us cope
- Careful balance of trade-offs



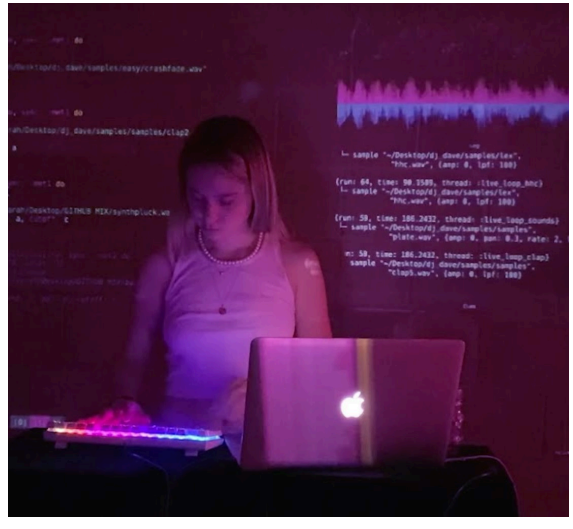
Mathematical culture

- Programs as formal entities
- Like good mathematics...
- Safe, composable, elegant

Cultures of programming

Humanistic culture

- Augmenting human intellect
- Programming helps us think
- Language close to human concepts



Hacker culture

- Programs are fundamentally bits
- Do not restrict the programmer
- Convenience, but full access

Type safety

Different perspectives

 Safety is the very essence of types!

 Useful as long as it makes programming easier

 Sometimes, you need to break the rules

 Does it help programmers think better?

Research methods

Interdisciplinary research

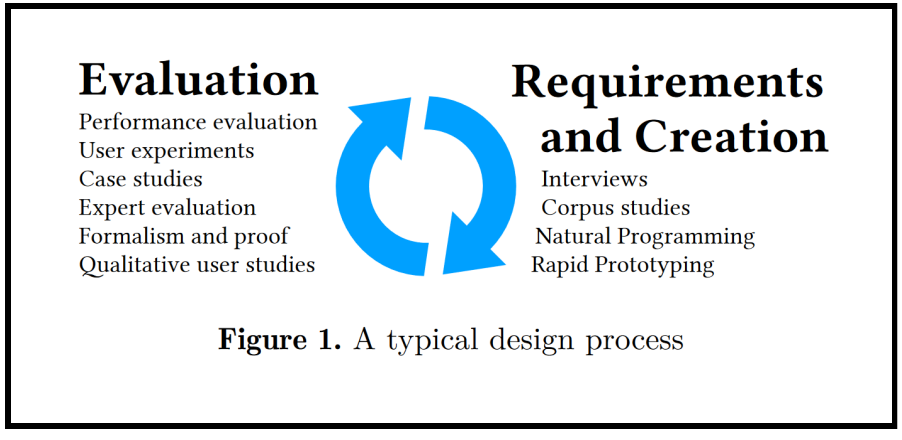


Figure 1. A typical design process

Interdisciplinary programming language research

Creating designs

Interviews, prototyping, formalism, analysis, history

Evaluating designs

Qualitative and quantitative studies, formal proofs

Coblenz et al. (2018)

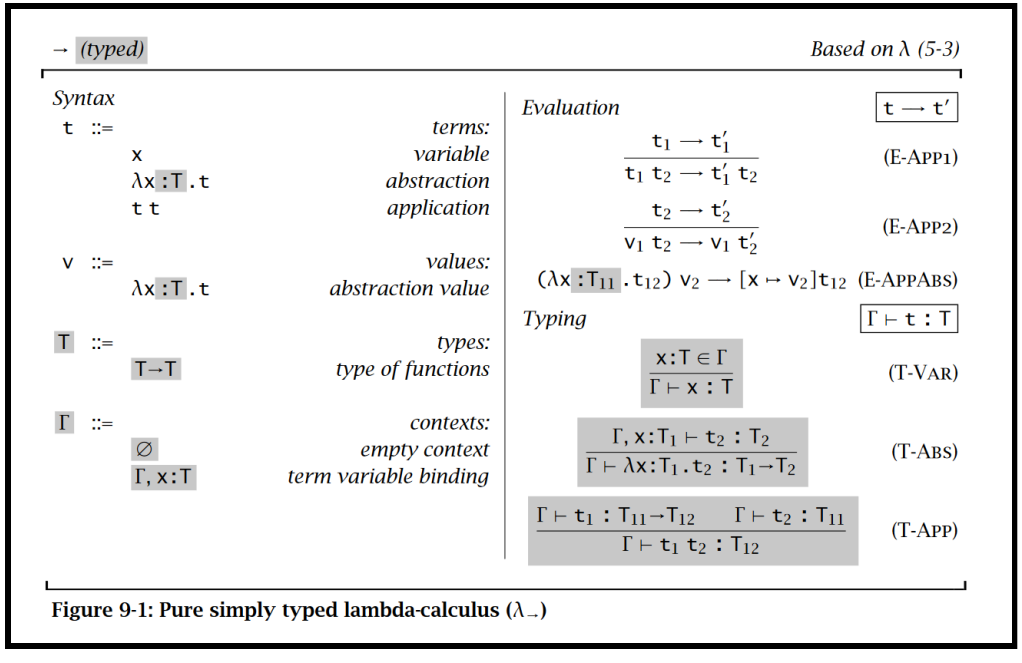
Research methods

≠ Formal and mathematical
Designing a formal mathematical system

Theoretical and conceptual
Derive design from theoretical principles

Empirical methodologies
Measure and evaluate observable data

User-centric methods
Study users and how they actually work



→ *(typed)* — *Based on λ (5-3)*

Syntax

t	::=		*terms:*
		x	*variable*
		λx:T.t	*abstraction*
		t t	*application*
v	::=		*values:*
		λx:T.t	*abstraction value*
T	::=		*types:*
		T→T	*type of functions*
Γ	::=		*contexts:*
		∅	*empty context*
		Γ, x:T	*term variable binding*

Evaluation $t \longrightarrow t'$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

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

Typing $\Gamma \vdash t : T$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

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

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

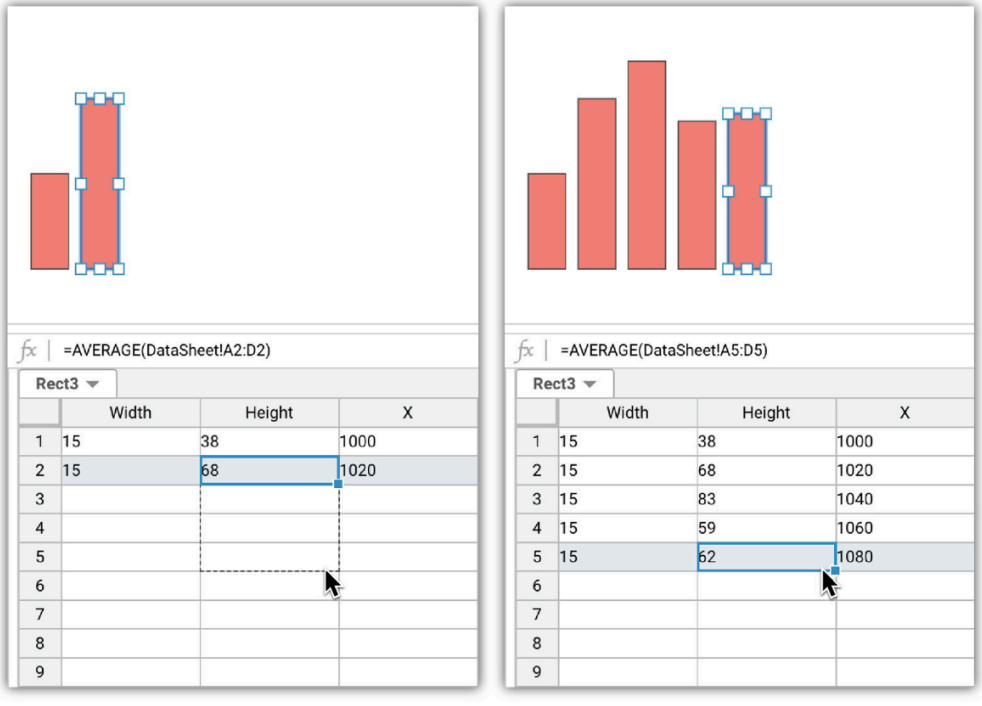
Figure 9-1: Pure simply typed lambda-calculus ($\lambda_{\to}$)

Programming language theory

Prove properties about small formal models

"Well-typed programs do not go wrong"

Discover and avoid subtle mistakes!

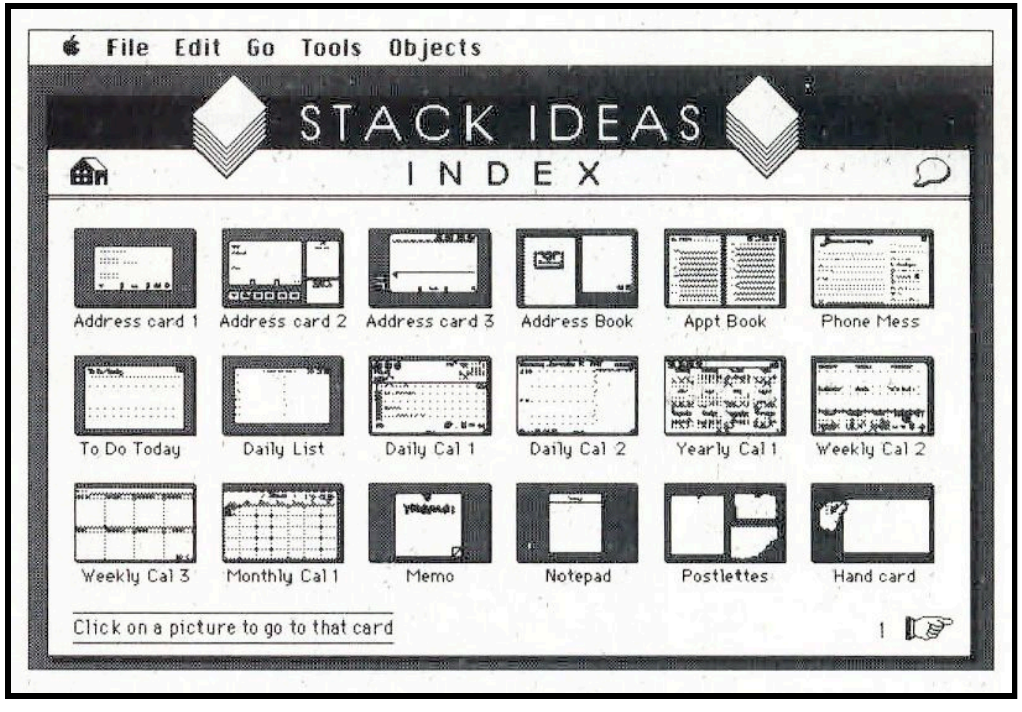


Human-centric system design

User studies, questionnaires, interviews, etc.

Qualitative analysis to design & test ideas

Quantitative analysis to compare designs



History of programming

What interesting past ideas were lost?

And the socio-political reasons for that?

Use history as source for new design ideas!

Conclusions

What to expect

Course outline

Preliminary structure

Design - Design and pattern languages
Usability - Human-centric language design
Semantics - Formal models of programming
Types - Types and type safety proofs
Beyond - Unexpected perspectives on types
Paradigms - History and programming systems
Complementary - Learning from past systems
Cognition - How humans think about programming

Reading

Jeremy Singer on Notebooks

- Notes on Notebooks: Is Jupyter the Bringer of Jollity?
- Available at:

http://www.dcs.gla.ac.uk/~jsinger/notebooks.pdf

Why should you read this?

- You'll get more out of the lecture...
- Perfect for the morning tram ride :-)
- Notebooks are curious programming systems!

Conclusions

How to do research about
programming language design?

- Inherently interdisciplinary topic
- Logic, design, user studies, history & more!

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

References

Methodology

- Coblenz, M., et al. (2018). Interdisciplinary programming language design. ACM Onward!
- Parsons, G. (2015). The philosophy of design. John Wiley & Sons
- Cross, N. (2007). Designerly ways of knowing. BIRD

Assorted examples

- Marasoiu, M. et al. (2019). Cuscus: An end user programming tool for data visualisation. Springer
- Pierce, B. C. (2002). Types and programming languages. MIT Press
- Petricek, T., Jakubovic, J. (2021). Complementary science of interactive programming systems. HaPoC
- Petricek, T. (2017). Context-aware programming languages. University of Cambridge