NPRG075

Programming language design

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

➔ https://tomasp.net | @tomaspetricek

Lectures: Monday 12:20, S7

➔ https://d3s.mff.cuni.cz/teaching/nprg075

Introduction

What? Why? How?

Making programming

(languages | experience | systems)

better!

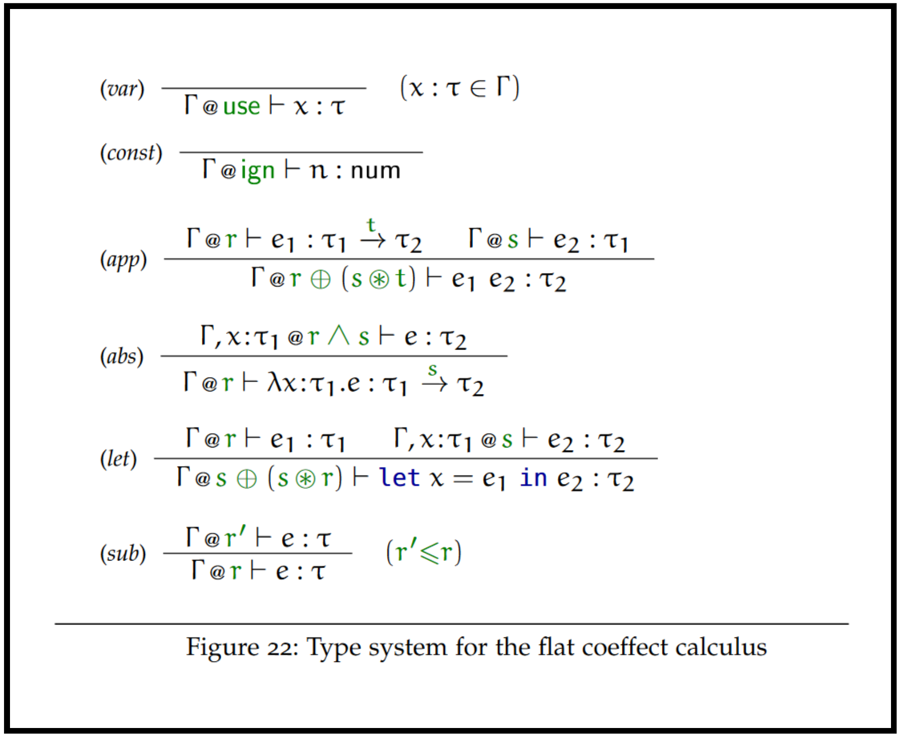
My background

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
History and programming systems



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

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

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

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

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

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

Figure 22: Type system for the flat coeffect calculus

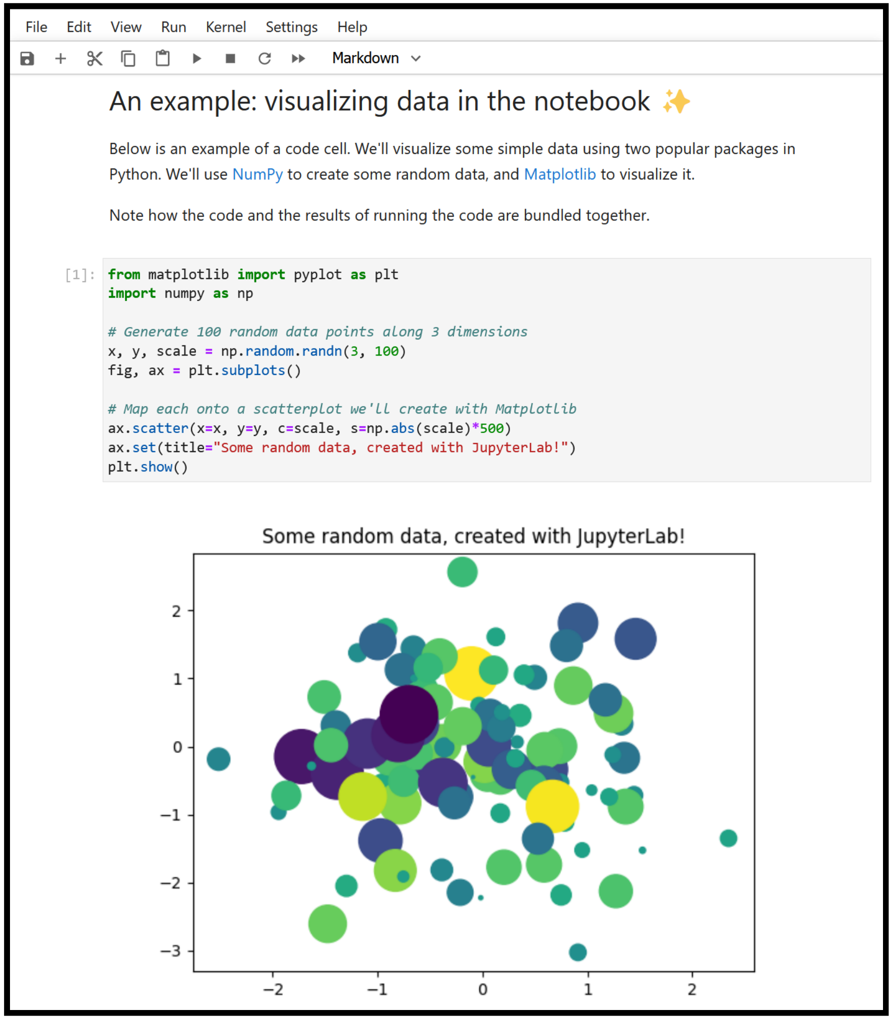
Types for context-aware programming

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



An example: visualizing data in the notebook ✨

Below is an example of a code cell. We'll visualize some simple data using two popular packages in Python. We'll use NumPy to create some random data, and Matplotlib to visualize it.

Note how the code and the results of running the code are bundled together.

```python
from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```

Result is a document not a program

Working with one concrete dataset

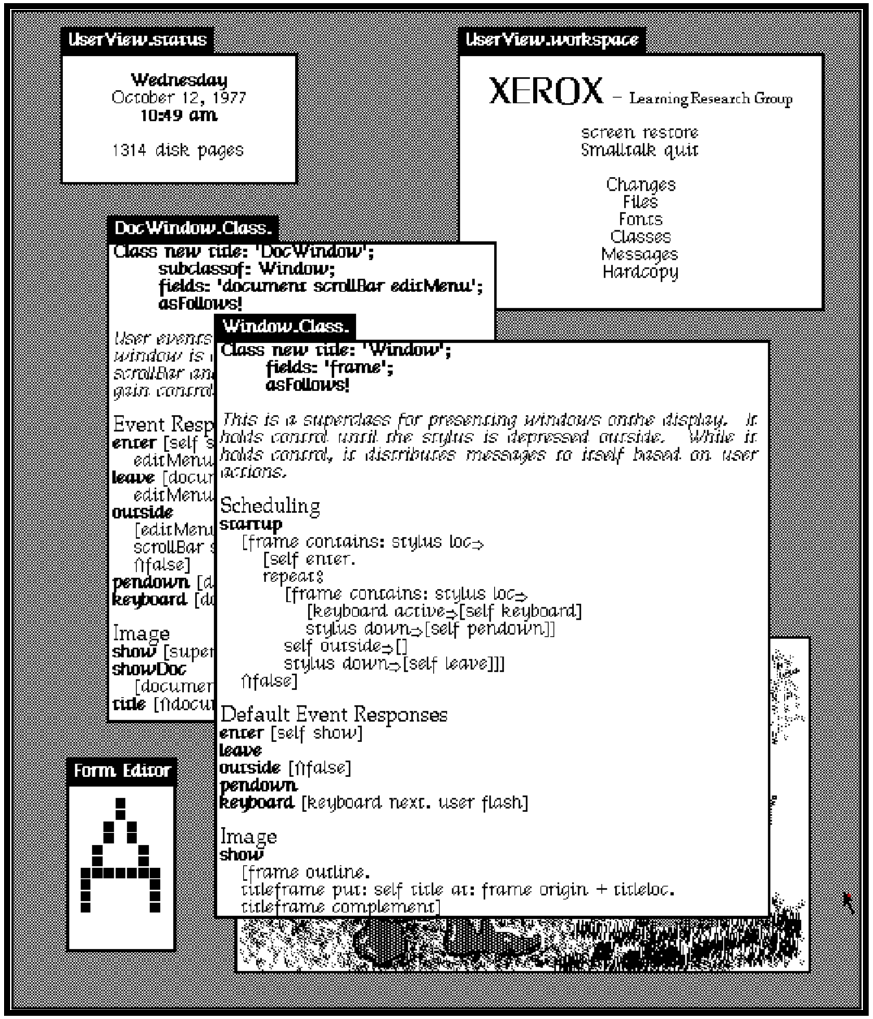
Different language and system requirements!

Programming systems & history

Interacting with a stateful environment

Let programmers do more in new ways...

It's not just a language!



UserView.status

Wednesday
October 12, 1977
10:49 am

1314 disk pages

UserView.workspace

XEROX – Learning Research Group

screen restore
Smalltalk quit

Changes
Files
Fonts
Classes
Messages
Hardcopy

DocWindow.Class.

```
Class new title: 'DocWindow';
        subclassof: Window;
        fields: 'document scrollBar editMenu';
        asFollows!

User events
window is
scrollBar an
gain control

Event Resp
enter [self s
   editMenu
leave [docu
   editMenu
outside
   [editMenu
   scrollBar s
   ⇑false]
pendown [d
keyboard [d

Image
show [super
showDoc
   [documen
title [⇑docu
```

Window.Class.

```
Class new title: 'Window';
        fields: 'frame';
        asFollows!

This is a superclass for presenting windows onthe display.  It
holds control until the stylus is depressed outside.  While it
holds control, it distributes messages to itself based on user
actions.

Scheduling
startup
   [frame contains: stylus loc⇒
      [self enter.
      repeat⁝
         [frame contains: stylus loc⇒
            [keyboard active⇒[self keyboard]
            stylus down⇒[self pendown]]
         self outside⇒[]
         stylus down⇒[self leave]]]
   ⇑false]

Default Event Responses
enter [self show]
leave
outside [⇑false]
pendown
keyboard [keyboard next. user flash]

Image
show
   [frame outline.
   titleframe put: self title at: frame origin + titleloc.
   titleframe complement]
```

Form Editor

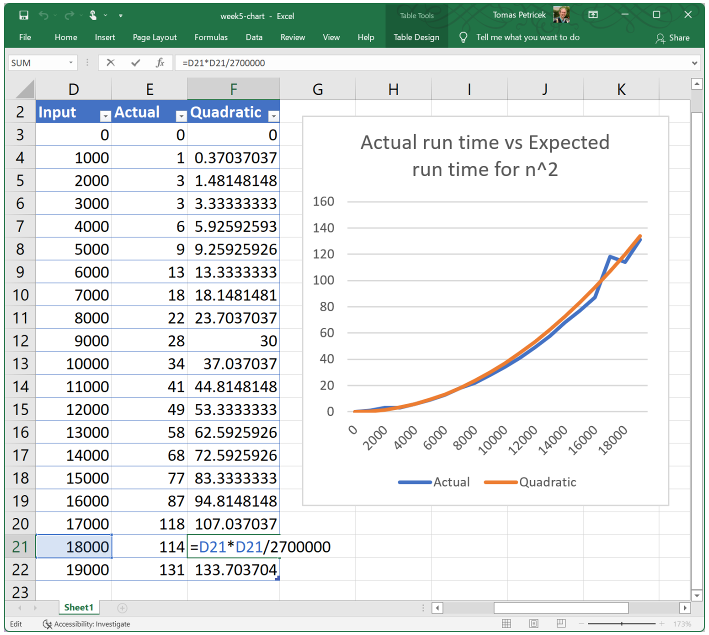
Bringing everything together

Systems ⊃ languages

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

Interdisciplinary research

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



=D21*D21/2700000

	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			

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

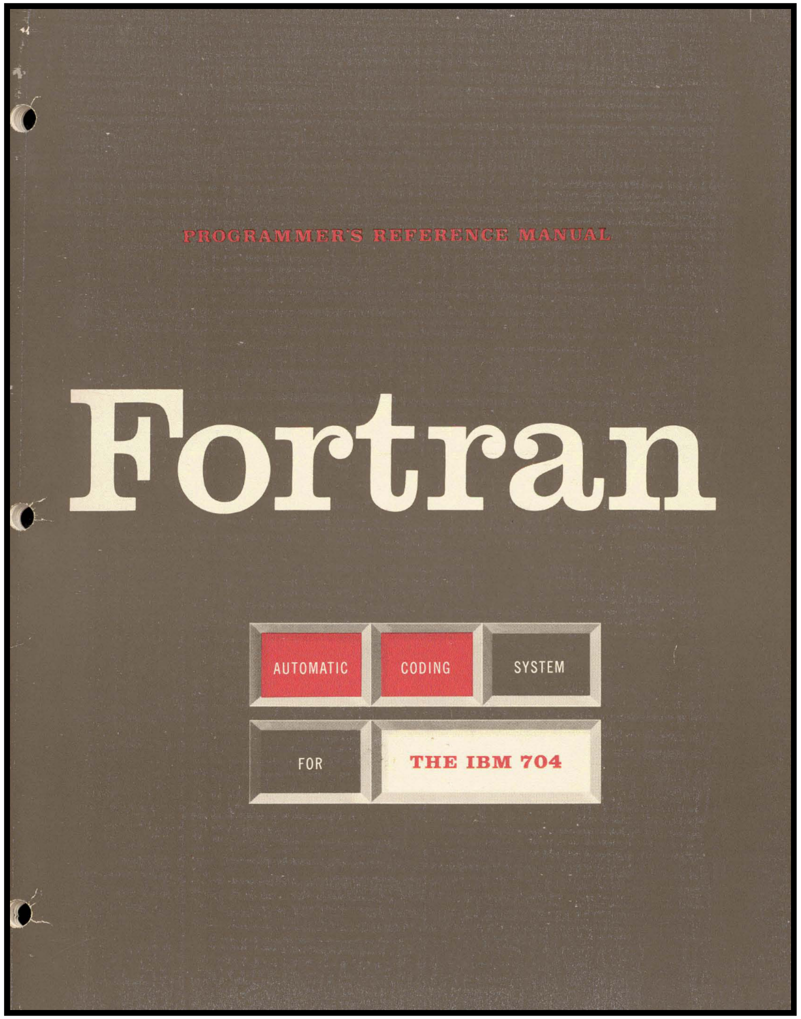
TypeScript, Jupyter, ML/F#, Smalltalk, BASIC

Many different research methods

Design, logic, proofs, user studies

This is a new work-in-progress course

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 (5 pages)

Deadlines
Topic by January 8
Draft by February 28

Programming languages

Conventional topics

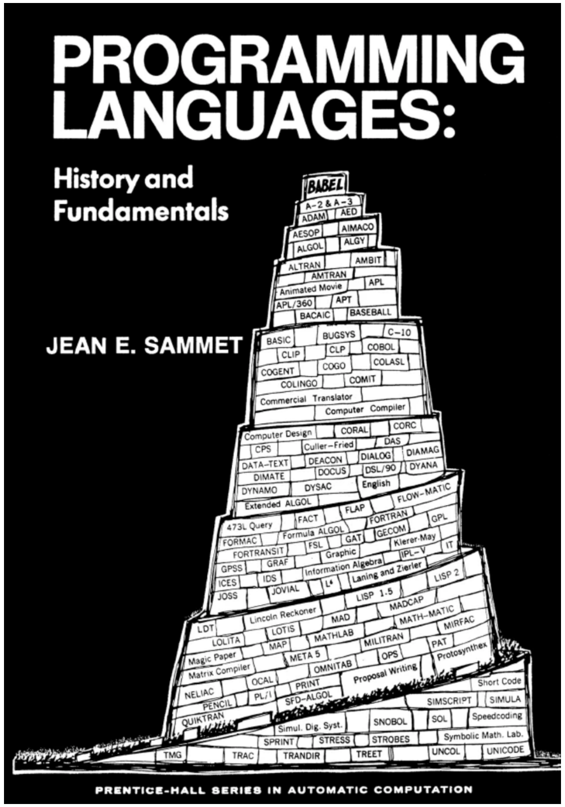
Paradigms and features

Language paradigms

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

Language features

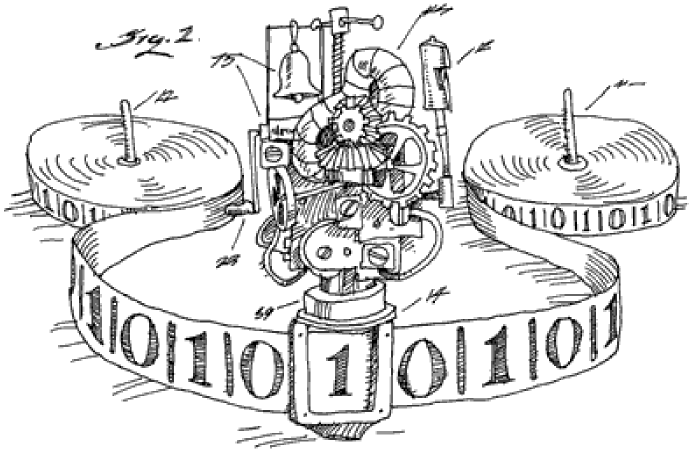
- Variable scoping, pointers
- Lambda abstraction, inheritance
- Design and implementation



Theory and implementation

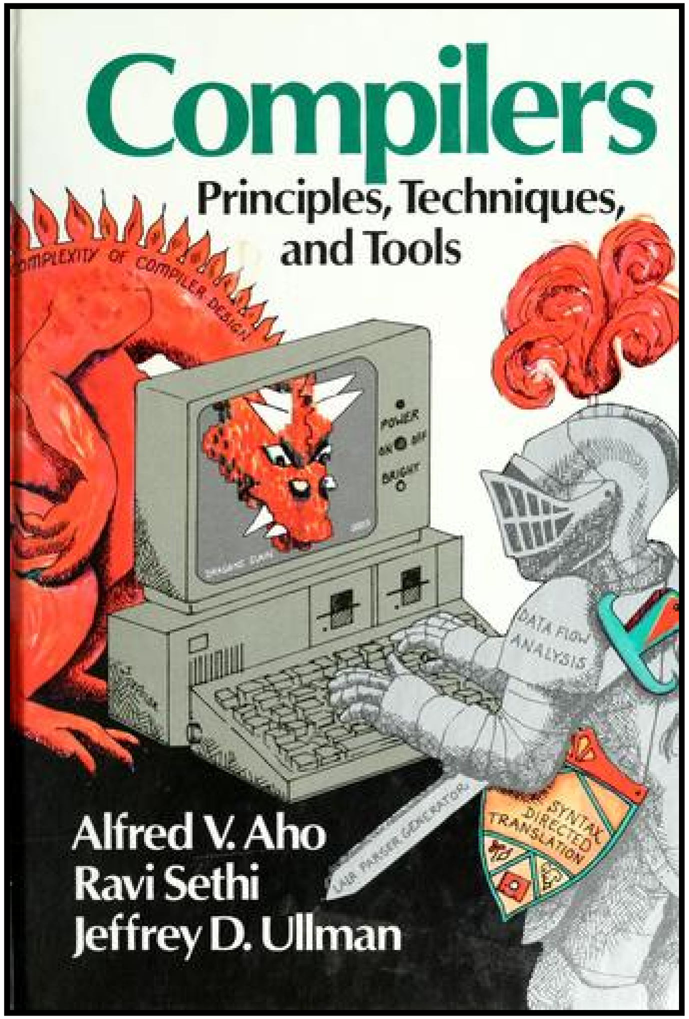
Parsing and automata

- 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 "what" but not about "how"

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

BIRKHÄUSER

Nigel Cross

Designerly Ways
of Knowing

Board of
International
Research in
Design, BIRD

Cultures of programming

Common ways of thinking

Case study: TypeScript

Unsoundness by design

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

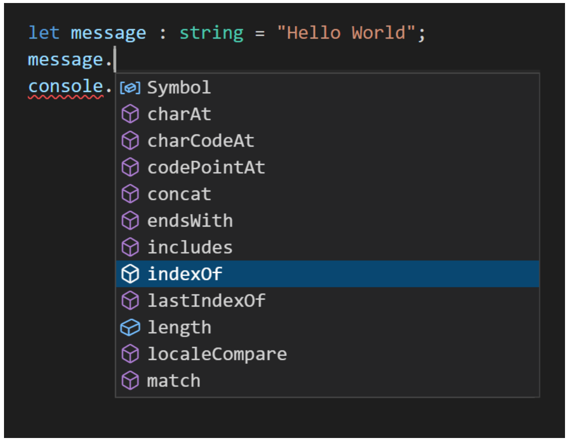
Design questions

- 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



```
let message : string = "Hello World";
message.
console.
```

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

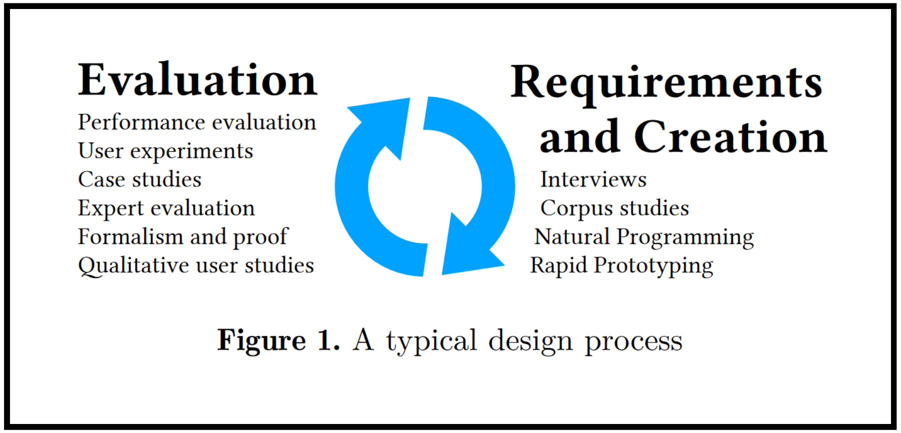
Type safety

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



Evaluation

Performance evaluation
User experiments
Case studies
Expert evaluation
Formalism and proof
Qualitative user studies

Requirements and Creation

Interviews
Corpus studies
Natural Programming
Rapid Prototyping

Figure 1. A typical design process

Interdisciplinary programming language research

Creating designs

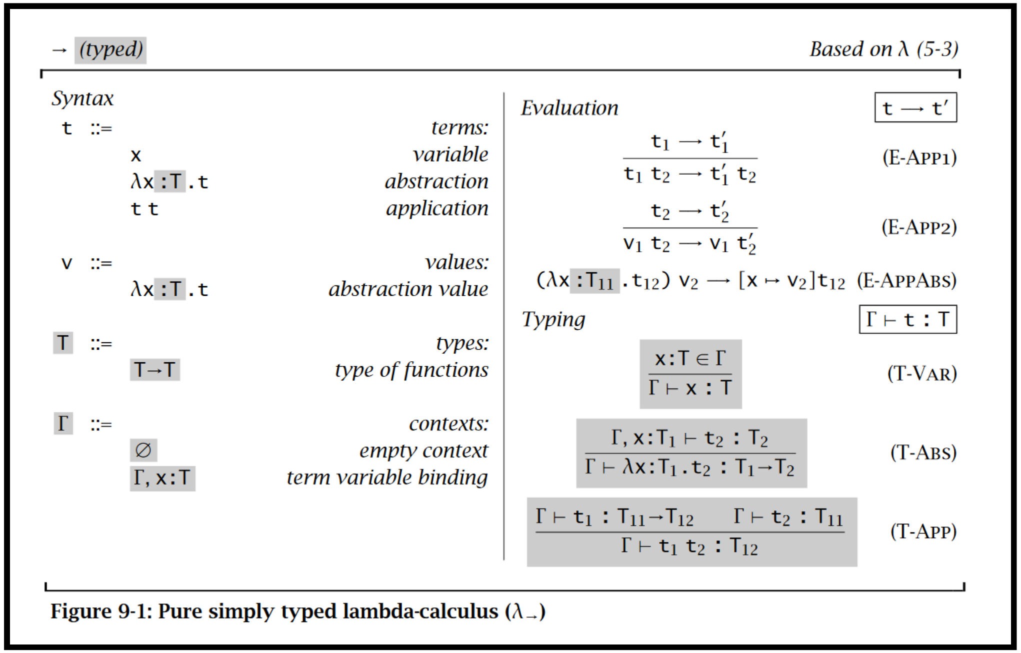
Interviews, prototyping, formalism, analysis, history

Evaluating designs

Qualitative and quantitative studies, formal proofs

(Coblenz et al., 2018)

Programming language theory



$\rightarrow$ *(typed)* *Based on λ (5-3)*

Syntax

$$\begin{array}{llr} \mathtt{t} & ::= & \textit{terms:} \\ & \mathtt{x} & \textit{variable} \\ & \lambda\mathtt{x}\!:\!\mathtt{T}.\mathtt{t} & \textit{abstraction} \\ & \mathtt{t}\ \mathtt{t} & \textit{application} \\ \\ \mathtt{v} & ::= & \textit{values:} \\ & \lambda\mathtt{x}\!:\!\mathtt{T}.\mathtt{t} & \textit{abstraction value} \\ \\ \mathtt{T} & ::= & \textit{types:} \\ & \mathtt{T}\rightarrow\mathtt{T} & \textit{type of functions} \\ \\ \Gamma & ::= & \textit{contexts:} \\ & \varnothing & \textit{empty context} \\ & \Gamma, \mathtt{x}\!:\!\mathtt{T} & \textit{term variable binding} \end{array}$$

Evaluation $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(E-APP2)}$$

$$(\lambda\mathtt{x}\!:\!\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-APPABS)}$$

Typing $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\mathtt{x}\!:\!\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, \mathtt{x}\!:\!\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda\mathtt{x}\!:\!\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1\rightarrow\mathtt{T}_2} \qquad \text{(T-ABS)}$$

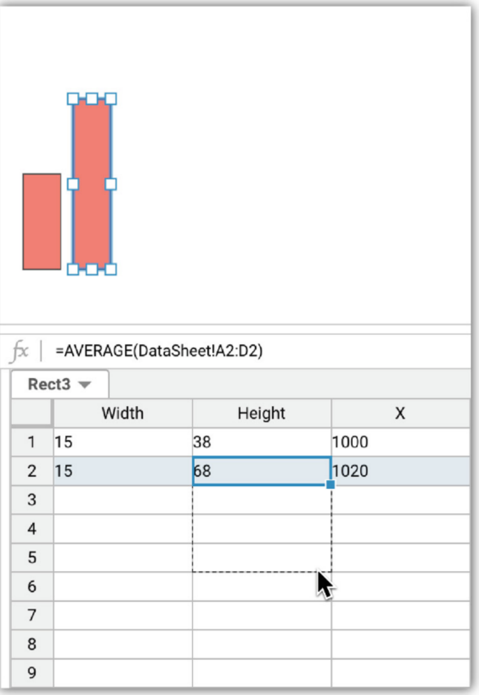
$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}\rightarrow\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

Figure 9-1: Pure simply typed lambda-calculus ($\lambda_\rightarrow$)

Prove properties about small formal models

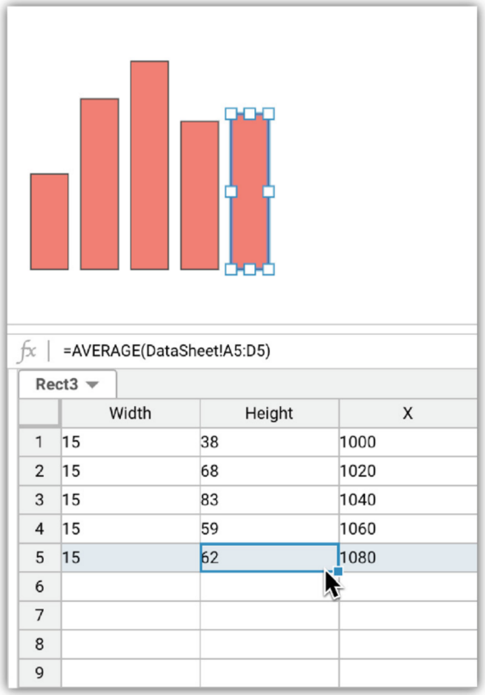
"Well-typed programs do not go wrong"

Discover and avoid subtle mistakes!



=AVERAGE(DataSheet!A2:D2)

	Width	Height	X
1	15	38	1000
2	15	68	1020
3			
4			
5			
6			
7			
8			
9			



=AVERAGE(DataSheet!A5:D5)

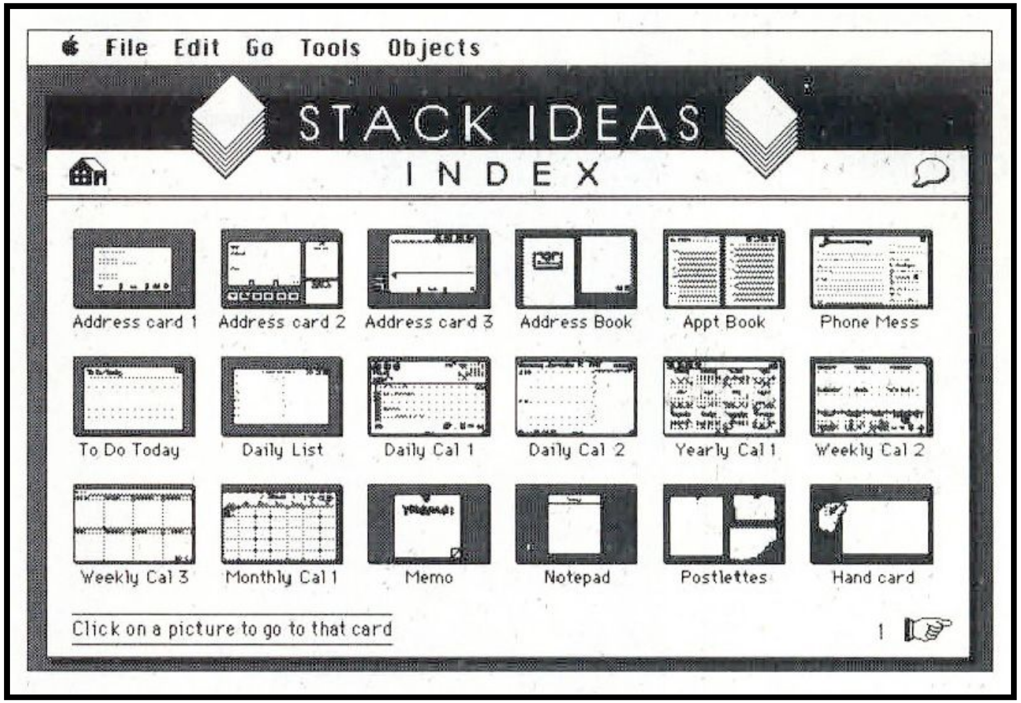
	Width	Height	X
1	15	38	1000
2	15	68	1020
3	15	83	1040
4	15	59	1060
5	15	62	1080
6			
7			
8			
9			

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