NPRG075
Mathematics and engineering of types

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
History

Where types come from?
Use types (1900s) to resolve logical paradoxes

\[ p(x) \text{ true if and only if } \neg x(x) \]
But \[ p(p) \text{ if and only if } \neg p(p) \]

Predicate \( p \) can be only applied to entities of lower type hence \( p(p) \) is invalid
"Two types of variable are also permissible: fixed point and floating point."

Called "modes" in more formal description!

Function arguments and results are in one of two modes.
COMTRAN, FLOW-MATIC and COBOL

Languages for business data processing

Built around working with data records
Algol language family

IAL 58 and Algol 60
- Adopts term "type" before publication
- Used just for primitive numeric types
- No explicit reference to Russell & logic

Algol 68, Pascal
- Attempts to make business-friendly language
- Add support for records and more
- Mathematical model of "types as sets"
Abstract data types

Clu and Ada in the 1970s

Type that can be used only through defined operations

Basis for abstraction, information hiding and object-oriented programming
Unifying ideas on types

- Meta-language for a theorem prover
- Abstract data types to represent theorems
- Type checking using methods of logic
- Records and unions for convenience
Types

Viewed by different cultures
History is messy!

Not just adopting logic ideas into programming

Are we even talking about the same thing?

Think cultures of programming!
Cultures and types

- **Hacker**
  - Types used for checking how memory is used
  - Fixed and floating point, but also data structures

- **Mathematical**
  - Types used for proving program properties
  - Simply typed lambda calculus and safety proofs
## Cultures and types

<table>
<thead>
<tr>
<th>Engineering</th>
<th>Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types to support good engineering practices</td>
<td>Types as a mechanism for team structuring</td>
</tr>
<tr>
<td>Information hiding, editor tooling and documentation</td>
<td>Division of labor, control programmer access rights</td>
</tr>
</tbody>
</table>
Cultural analysis

- Abstract data types in Ada and Clu
  Mix of engineering and managerial approaches

- Adding types to JS in TypeScript
  Engineering approach, using mathematical ideas

- Type checking in ML, OCaml
  Mathematical approach, using engineering ideas

- Types and ownership in Rust
  Mix of hacker and mathematical approaches
Type systems
Mathematical look at types
Type systems
Mathematical look at types

☑ Types as a checking mechanism
⚡ Rule out invalid programs
÷ Defined using a formal system
⇄ Use induction to prove properties
Defining a type system

**Simple Language**

\[ e ::= n \mid e_1 + e_2 \mid x \mid \text{if } e \text{ then } e \text{ else } e \]

\[ \gamma ::= \text{num} \mid \gamma \rightarrow \gamma \]

**Typing - Basics**

\[ \Gamma \vdash n : \text{num} \]

\[ \frac{}{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 + e_2 : \text{num}} \] (plus)
Typed lambda calculus

\[ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \text{(var)} \]
\[ \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2} \quad \text{(lam)} \]
\[ \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2} \quad \text{(app)} \]
\[ \frac{\Gamma \vdash e : \tau \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau} \quad \text{(cond)} \]
Type systems
Properties we may want

🛡️ Does it actually prevent bad behaviour?
🔥 Can we check if a program has a given type?
🔍 Can we automatically infer a type?
🏆 Does the system assign just one type?
Properties, more formally

**Determinacy** if \( e \rightarrow e' \) and \( e \rightarrow e'' \) then \( e' = e'' \)

**Safety** if \( e : \gamma \) and \( e \rightarrow^* e' \) then

either \( e' \) is a value or \( \exists e''. e' \rightarrow e'' \)

**Type Inference** given \( e \) one can find \( \gamma \)

such that \( e : \gamma \) or show there is no such \( \gamma \)

**Decidability** given \( \Gamma, e : \gamma \) one can say if \( \Gamma \vdash e : \gamma \)

**Uniqueness** if \( \Gamma, e : \gamma \) and \( \Gamma, e : \gamma' \) then \( \gamma = \gamma' \)
Proofs
Type safety
Type safety

What does it mean

- $5 + (\lambda x.x)$ cannot be reduced!
- Stuck when no evaluation rule applies
- Well-typed programs do not get stuck

Progress + preservation

- Safety = progress + preservation
- Reduction preserves the type
- Well-typed expression is value or can be reduced
Type safety, formally

**Soundness**

**Progress** if $e : \tau \Rightarrow$ then either $e$ is a value or $Fe \cdot e \rightarrow e$.

**Preservation** if $e : \tau$ and $e \rightarrow e'$ then also $\Gamma e : \tau$.

**Safety** if $e : \tau$ and $e \rightarrow e'$ then either $e'$ is a value or $Fe' \cdot e' \rightarrow e''$. 
Proofs about types

What to expect

- Almost always by induction
- Easy with the right property
- Lots of uninspiring cases

Proofs by induction

- Over the (tree) syntax of the expression
- Over the (tree) typing derivation
- Over the (linear) sequence of reductions

By (*) one of the following holds.

\[
\exists e_1', s'. \langle e_1, s \rangle \rightarrow \langle e_1', s' \rangle.
\]

Then by (op1) we have \(\langle e_1 + e_2, s \rangle \rightarrow \langle e_1' + e_2, s' \rangle\), so we are done.

By (**') one of the following holds.

\[
\exists e_2', s'. \langle e_2, s \rangle \rightarrow \langle e_2', s' \rangle.
\]

Then by (op2) \(\langle e_1 + e_2, s \rangle \rightarrow \langle e_1 + e_2', s' \rangle\), so we are done.

case \(e_1\) is a value.

(Now want to use (op+), but need to know that \(e_1\) and \(e_2\) are really integers.)

Lemma 13: For all \(\Gamma, e, T\), if \(\Gamma \vdash e: T\), \(e\) is a value and \(T = \text{int}\) then for some \(n \in \mathbb{Z}\) we have \(e = n\).

Proof: By rule induction. Take \(\Phi'(\Gamma, e, T) = ((\text{value}(e) \land T = \text{int}) \Rightarrow \exists n \in \mathbb{Z} \; e = n)\).

Case (int). ok
If \( e : \tau \) then either \( e \) is a value or \( \exists e' : e \rightarrow e' \)

By induction over the derivation of \( e : \tau \)

- (num) \( e = n \) is a value
- (lol) \( e = \lambda x . e \) is a value
- (plus) \( e = e_1 + e_2 \) by induction \( e_1, e_2 \) values or can reduce:
  - \( e_1, e_2 \) values - by inversion those are \( n_1, n_2 \) - reduce using (plus)
  - \( e_1 \) value and \( e_2 \rightarrow e' \) - reduce using (plus), \( e' = e_1 + e_2 \)
  - \( \exists e_1 : e_1 \rightarrow e' \) - reduce using (plus), \( e' = e_1 + e_2 \)
- (app), (cond) similar
- (var) cannot occur because \( \Pi - \phi \)
Preservation proof sketch

If \( e : \tau \) and \( e \rightarrow e' \) then \( e' : \tau \)

By induction over the derivation \( e \rightarrow e' \)

- (plus) \( \text{ie. } n_1 + n_2 \rightarrow n_3 \) \( , e = n_1 + n_2 \) and so

  \( e : \tau \) used (plus) \( , \gamma = \text{num} \) and so \( \Gamma \vdash n_3 : \text{num} \)

- (plus1) \( \text{ie. } e_1 + e_2 \rightarrow e_1' + e_2' \) \( , e = e_1 + e_2 \) and so

  \( e : \tau \) used (plus) \( , \gamma = \text{num} \) and so \( \Gamma \vdash e_1 : \text{num} \)

By induction \( \Gamma \vdash e_1 : \text{num} \) and so \( \Gamma \vdash e_1 + e_2 : \text{num} \)

- (plus2), (app), (app1), (app2) similar
Fancy types

Interesting type systems
Fancy types

Interesting type systems

- Non-null, ownership & borrowing
- Effects, coeffects and communication
- Specific types for web, data, etc.
- Arbitrary computations in types!
Case study: TypeScript

Literal string types

- Concrete values can be types too!
- Useful paired with union types
- tinyurl.com/nprg075-lt

Design questions

- What was the motivation this?
- Is there another "better" approach?
- What are the benefits and drawbacks?
Billion dollar mistake

Tony Hoare invents null

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. I couldn't resist the temptation to put it in because it was so easy to implement.

Fixing null with types?

- Separate types that can be null
- Allow obj.foo() on non-null types!
- Null checks need special logic
Demo
Flow analysis in TypeScript
Sketch for non-null types

\[ e := \ldots | \text{null} | e ! = c | e . m \]

\[ \gamma := \ldots | t : \gamma, \ldots, m : \gamma^* | \gamma^* \]

\[ \lambda x. \text{if } x ! = \text{null} \text{ then } e_1 \text{ else } e_2 \]

\[
\frac{\Gamma + e : \{ \ldots, m : \gamma^* \}^*}{\Gamma + e : \gamma^*} \quad \text{(mem)}
\]

\[
\frac{\Gamma + x : \gamma + x ! = \text{null} \Rightarrow \Gamma + x : \gamma^*}{\text{(check)}}
\]

\[
\frac{e ! = \text{null}}{\Gamma + e \Rightarrow \Gamma} \quad \text{(otherwise)}
\]

\[
\frac{e \neq x ! = \text{null} + e_1 \Rightarrow \Gamma}{\text{(cond^\star)}}
\]
Conclusions
Mathematics & engineering of types
Designing types

Good language design case study!

Design inspired by logic, engineering concerns, existing real-world code

Mathematicians care for safety, engineering evaluation harder to do
Reading

Type providers

- Design of types for real-world data
- See: tomasp.net/academic/papers/inforich/inforich-ddfp.pdf

Why read this

- Motivations beyond type safety
- Mechanism in F# and The Gamma
- Why don't all typed languages have this!?
Conclusions

Mathematics & engineering of types

- History of types is interesting & messy
- Different cultures think differently
- Type safety is basic formal PLT method!

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz
🌐 https://tomasp.net | @tomaspetricek
🌐 https://d3s.mff.cuni.cz/teaching/nprg075
References (1/2)

Theory and proofs

- Pierce, B. (2002). *Types and Programming Languages*. MIT

Fancy types

- TypeScript (2022). *The TypeScript Handbook*. Online
References (2/2)

History

- Martini, S. (2016). *Several Types of Types in Programming Languages*. HaPoC

Just for fun...

- Tresnormale. *Bertrand Russell: You want to be a philosopher? You do not even smoke!*