Computer Laboratory, University of Cambridge

Fork me on GitHub

F# Data: Making structured data first-class

Tomas Petricek
tomas.petricek@cl.cam.ac.uk

supervisor: **Alan Mycroft**
alan.mycroft@cl.cam.ac.uk

Introduction: Structured data in XML, JSON and CSV

- **Structured formats are ubiquitous.** Open-government initiatives release data as CSV, web services communicate using JSON or XML.
- **Schema vs. examples.** Real-world data often do not carry explicit schema. Examples are more common. Documentation for services usually includes examples of "typical" server responses.
- **Type-safe data access is hard.** Data extraction expects known format, but statically-typed languages do not understand it.

Motivation: Printing names and ages from JSON file

```
[ {"name" : null, "age" : 23},
  {"name" : "Alexander", "age" : 1.5},
  {"name" : "Tomas"} ]
```

```
match data with
| Array items →
   for item in items do
     match item with
     | Object o → print (Map.find o "name")
     | _ → failwith "Incorrect format"
| _ → failwith "Incorrect format"
```

Code expects document with a *fixed schema*, but is written using pattern matching designed for handling the *general case*.

Solution: Using the F# Data JSON type provider

```
type People = JsonProvider<"people.json">
let items = People.Parse(data)
for item in items do
  printf "%s" item.Name
  Option.iter (printf "%d") item.Age
item.
```

Age
Name
property People.Age : int option

- **Schema inference.** The `People.Parse` method returns array of entities with `Name` of type string and `Age` of type `int option`. The member names and types are inferred from sample JSON.
- **Ease of use and tooling.** Full type information is available. Used by F# tools to provide auto-complete, type hints and docs (available in Xamarin Studio, Visual Studio, Emacs and more)
- **Safety properties.** Same as in the original implementation. Guaranteed to work if input value is a subtype of sample(s). Otherwise throws a runtime exception that can be handled.

Empty initial context

$\emptyset \vdash e : \tau$

Most type systems assume the initial context is empty

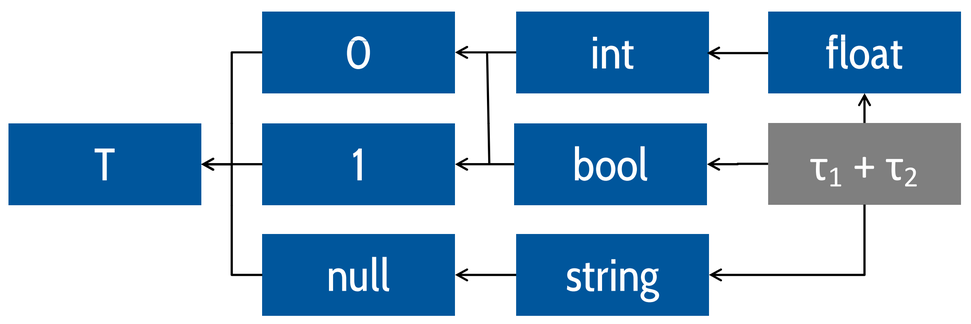
Provided initial context

$\pi(\text{🌍}) \vdash e : \tau$

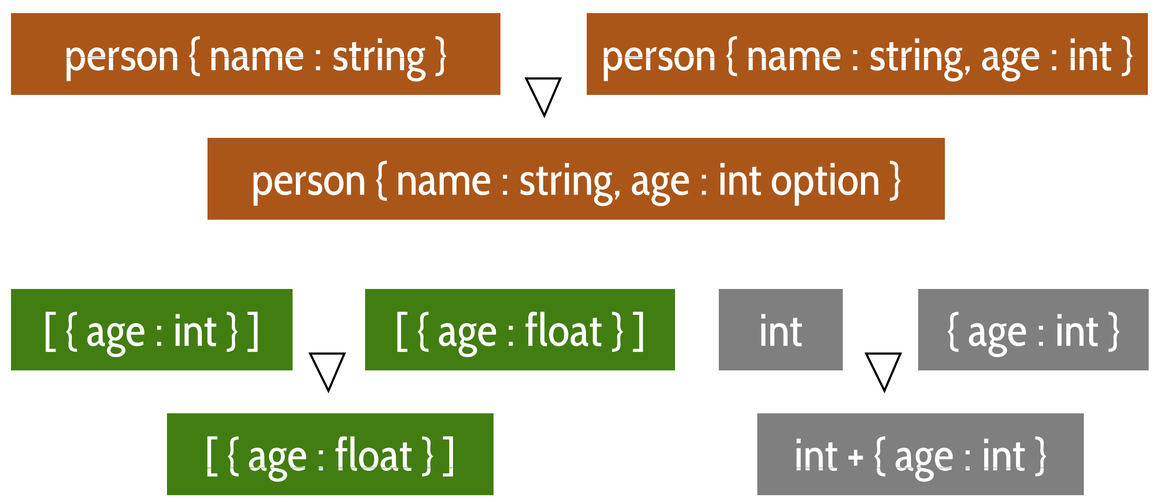
Type provider projects the external world into the context.

Approach: Structural type inference algorithm

Primitive types are inferred from values. The following hierarchy is used to find the most specific common subtype. Note that 0 and 1 are treated as Boolean values and null+τ is an option type.



Records of matching names are unified, introducing optional fields.
Collections are unified by unifying the type of their elements.
Types of different kinds are combined into a flattened sum type.



Summary: What makes F# Data interesting?

Prime example of type provider mechanism Explores relativized type safety property	**F# Type Providers**
Simple yet powerful inference algorithm Unified treatment for XML, CSV and JSON	**Practical Inference**
Used by the industry with 17k downloads Documented, tested with active community	**Industry Adoption**

Thanks to: Don Syme, Gustavo Guerra & other contributors.

Microsoft Research

UNIVERSITY OF CAMBRIDGE