An update on XML types

Alain Frisch

INRIA Rocquencourt (Cristal project)

Links Meeting - Apr. 2005

Plan

Plan

Claim

It is worth studying new programming language features/abstractions to deal with XML natively.

XDuce (Hosoya,Vouillon,Pierce)

Data model

- Functional view of XML trees : no upward pointer, no node identity.
- Values = forests of XML elements.

Types

- Purely structural view of XML types.
- Regular expression types = regular tree languages.
- Subtyping = language inclusion.

Language features

- Functional flavor : recursive functions, few side effects.
- Regular expression patterns = types + capture variables ; very precise type-checking.

XDuce lineage

- XQuery.
- Xtatic (Pierce,Gapeyev,Levin,Schmitt) : $C^{\#}$ + XDuce.
- CDuce (Frisch,Castagna,Benzaken) : extends XDuce into a λ-calculus with overloaded functions.
- Xtatic and CDuce also focus on efficient implementation (runtime data representation and compilation of pattern matching).
- XHaskell (Sulzmann, Zhuo Ming Lu) : Haskell + XDuce (nice encoding of subtyping into an extension of type-classes).

Extensions

Type-checking XDuce is very modular. New XML operations with custom typing rules are easily integrated.

- Filters = "regular-expression based iterators" (Hosoya).
- A query language CQL on type of CDuce (Miachon,Castagna,Benzaken).
- Various disambiguation semantics for pattern matching. With an all-match semantics, patterns can encode XPath navigation (Pierce et al.).

Extensions

Extending the type algebra while retaining its nice properties is more challenging (need to extend subtyping, pattern matching type inference and compilation).

- Attribute-element constraints à la Relax-NG (Hosoya,Murata).
- Extensible records to handle XML attributes.
- Function types (cf CDuce).
- Parametric polymorphism (Hosoya,Frisch,Castagna).

Plan

Parametric polymorphism in XDuce

Motivations

- Theoretical : understand the interaction between set-theoretic types/subtyping and parametric polymorphism.
- Practical :
  - If XDuce-derived languages are used on their own, we need it.
  - Polymorphic manipulation of XML data (e.g. generic processing of envelope formats).

$\rightsquigarrow$ XDuce 0.5.0 release on Tuesday.

Polymorphism in XDuce

First technical challenge

- The natural set-theoretic definition for subtyping gives an extremely complex algorithm.
- Indeed, subtyping can encode constraints on (finite) cardinals. E.g. :

$$\forall\alpha.(t \cap \alpha) \times t \leq (t \cap \alpha) \times \alpha \iff \mathit{Card}(t) \leq 1$$

Proposed solution

- Use another set-theoretic interpretation for types : sets of marked values.
- $\rightsquigarrow$ removes spurious subtyping, simplifies algorithm, gives more parametricity.

Polymorphism in XDuce

Second technical challenge

- How to infer types for quantified variables when calling a polymorphic function ?

Proposed solution

- Pattern-match the type of the actual argument against the input type of the function (its free variables are considered as capture variables).
- If the input type is non-ambiguous, we obtain the best solution.

Polymorphism in XDuce

Limitations

- Type variables only stand for one element in a sequence (or the whole content of an element). No “sub-sequence variable”.
- Inference for type variables does not mix well with function types, even when limiting to toplevel polymorphic functions.
  - State of the art : we know how to do it for a type system without overloaded functions and with strong restrictions on the types of polymorphic functions.
  - Explicit instantiation is easier.

Plan

Combining ML and XDuce

A natural match

- Similarities : functional data, types, patterns, recursive functions.

Extending XDuce to the full power of ML ?

- We know how to extend XDuce either with function types or with parametric polymorphism, but not both.
- We don't know how to do ML-like type inference for XDuce.

Design goals

Language

- Conservatively extend the ML type system with XML types and XML pattern matching (and other fancy operations to come).
- XDuce programs must be easily translated. Less type annotations should be required.

Implementation

- Extend existing ML implementations
  $\rightsquigarrow$ limits the complexity of the underlying theory.
- Keep the structure of XDuce type-checker (propagation + subtyping checks).

Overview of the proposal

XML types as ground types

We simply add XML types to the type algebra (as ground basic types).
We want to have implicit subsumption and to reuse existing type-checking algorithms for XML operations.

Overview of the proposal

A three-pass process

- First ML type-checking pass : collapse all XML types into a single type.
  $\rightsquigarrow$ detect locations in the AST whose type is of kind XML.
- Second ML type-checking pass : introduce two XML variables for each such location, plus a subtyping constraint ; also records XML operations as symbolic constraints between these variables.
  $\rightsquigarrow$ extract data-flow of XML values and a summary of XML operations.
- XDuce-like type-checking : forward propagation of types in the data-flow graph.

Overview of the proposal

- For the third pass to work, we require the data-flow graph to be acyclic.
- The programmer has to provide enough type information :
  - explicit type annotations ;
  - datatype declarations ;
  - module signatures.
- The ML type-checker propagates type-annotation in a way which is simple enough to understand (for an ML programmer).
- Usually, type annotations must only be given for recursive functions.

Example

Example

```
let f b x1 x2 y1 y2 =
if b then (x1@x2,y1) else (x1,y2)
```

Example

Example

```
let f b x1 x2 y1 y2 =
if b then (x1@x2,y1) else (x1,y2)
```

gives :

$$\begin{cases} f : \forall\alpha.\texttt{bool} \to \iota_1 \to \iota_2 \to \alpha \to \alpha \to \iota_3 \times \alpha \\ \texttt{concat}(\iota_1, \iota_2) \leq \iota_3 \ \wedge \ \iota_1 \leq \iota_3 \end{cases}$$

From ML to XML and back

- The extension also provides two constructions to translate an ML value structurally into an XML value, and back.
- The ML type must be fully known, monomorphic, and transparent (i.e. it must have a natural structural representation).
- $\rightsquigarrow$ easy XML parsing/pretty-printing of ML data.
- $\rightsquigarrow$ subsomption from structural subtypes in the ML world.

Conclusion

- Design goals are achieved.
- Type-checking is less compositional (XML type-checking is done for a whole compilation unit at once).
- Implementation on top of OCaml 3.08.2 (a prototype has been made available on Monday).
- Main idea : reusing ML type checker to do slicing + data flow analysis.

Thank you !