

A reformulation of the XDuce type system

(work in progress)

*Jan Van den Bussche
Stijn Vansummeren*

U. Limburg, Belgium

The programming language ML

Functional language working on structured data

Rule-based programming based on
pattern matching

```
fun sumLists =  
  nil => 0  
  nil::YS => sumLists(YS)  
  (x::xs)::YS => x + sumLists(xs::YS)
```

Polymorphic type inference

```
val sumLists = fn : int list list → int
```

The programming language XDuce

Hosoya & Pierce

ML-like language for programming with
semistructured data

Pattern matching

```
match l : (dt[String] | Dd)* with  
dt[t], d as Dd*, rest => ...
```

Longest match!

Type inference of pattern variables

```
match p : person[Name, Email*, Tel?] with  
person[Name, x as (Email|Tel)+]
```

```
⇒ x : (Email+, Tel?) | Tel
```

Weak points of XDuce type system

1. Grammar based
⇒ complicated well-formedness condition
2. Encoding in binary tree automata
⇒ hard to understand and prove correct
3. Type inference only for variables in tail position

Our reformulation:

1. Use standard type system based on regular expressions
2. Algorithm works on same level as type system
3. Sound & complete type inference also for non-tail variables

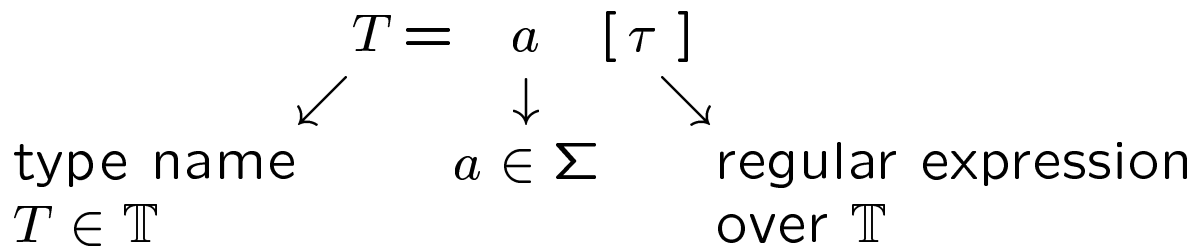
Hedges and types

Hedge: sequence of ordered trees

Node-labeled, finite alphabet Σ

Type environment Δ : set of *type definitions*

Type definition:



Type constraint: $\mathcal{T} = (\Delta, \tau)$

Typing of hedges

Hedge \mathbf{h}

Type assignment on \mathbf{h} : mapping

$$\alpha : \text{Nodes}(\mathbf{h}) \rightarrow \mathbb{T}$$

$\mathbf{h}, \alpha \models (\Delta, \tau)$ if

- α conforms to Δ
- $\alpha(\text{roots}(\mathbf{h})) \in \tau$

$\mathbf{h} \models (\Delta, \tau)$ if $\exists \alpha : \mathbf{h}, \alpha \models (\Delta, \tau)$

Unranked hedge automaton

Patterns

Pattern $\Pi = (\Delta, \tau; r_1, r_2, r_3)$

r_1, r_2, r_3 regular expression types

Result of matching Π to \mathbf{h} : Any subhedge \mathbf{h}' of \mathbf{h} such that

$$\exists \alpha : \mathbf{h}, \alpha \models (\Delta, \tau)$$

and

$$\begin{array}{ccccccc} \underbrace{\hspace{10em}}_{\text{left context}} & \underbrace{\hspace{10em}}_{\mathbf{h}'} & \underbrace{\hspace{10em}}_{\text{right context}} \\ \mathbf{n}_1 & \dots & \mathbf{n}_k & \mathbf{n}_{k+1} & \dots & \mathbf{n}_{k+l} & \mathbf{n}_{k+l+1} & \dots & \mathbf{n}_{k+l+m} \\ \wedge & & \wedge & \wedge & & \wedge & \wedge & & \wedge \end{array}$$

such that

- $\alpha(\mathbf{n}_1) \dots \alpha(\mathbf{n}_k) \in r_1$ as long as possible
- $\alpha(\mathbf{n}_{k+1}) \dots \alpha(\mathbf{n}_{k+l}) \in r_2$ as long as possible
- $\alpha(\mathbf{n}_{k+l+1}) \dots \alpha(\mathbf{n}_{k+l+m}) \in r_3$

Type inference

Input: Pattern Π , type constraint \mathcal{T}_{in}

Output: Type constraint \mathcal{T}_{out} such that for any hedge \mathbf{h}' :

$$\mathbf{h}' \models \mathcal{T}_{\text{out}}$$

iff

\mathbf{h}' is result of matching Π to some $\mathbf{h} \models \mathcal{T}_{\text{in}}$

Aspects of the algorithm

Longest match policy by a 2FA with a pebble

Context by quotient constructions

Accommodate \mathcal{T}_{in} by product construction

Future work

Implementation doable?

Apply to practical pattern languages