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

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

Polymorphic type inference

```ml
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 $\Sigma$

*Type environment $\Delta$: set of type definitions*

Type definition:

$$ T = a \left[ \tau \right] $$

- type name
- $a \in \Sigma$
- regular expression over $\Sigma$

Type constraint: $T = (\Delta, \tau)$
Typing of hedges

Hedge $h$

*Type assignment* on $h$: mapping

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

$h, \alpha \models (\Delta, \tau)$ if

- $\alpha$ conforms to $\Delta$

- $\alpha(\text{roots}(h)) \in \tau$

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

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 $\Pi$ to $h$: Any subhedge $h'$ of $h$ such that

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

and

\[
\begin{array}{cccccc}
\text{left context} & & \text{h'} & & \text{right context} \\
\hline
n_1 & \land & \ldots & \land & n_k & n_{k+1} \\
\hline
\land & \ldots & \land & \ldots & \land & \ldots & \land & n_{k+\ell} & n_{k+\ell+1} & \land & \ldots & \land & n_{k+\ell+m}
\end{array}
\]

such that

- $\alpha(n_1) \ldots \alpha(n_k) \in r_1$ as long as possible
- $\alpha(n_{k+1}) \ldots \alpha(n_{k+\ell}) \in r_2$ as long as possible
- $\alpha(n_{k+\ell+1}) \ldots \alpha(n_{k+\ell+m}) \in r_3$
Type inference

**Input:** Pattern $\Pi$, type constraint $\mathcal{T}_{in}$

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

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

iff

$h'$ is result of matching $\Pi$ to some $h \models \mathcal{T}_{in}$
Aspects of the algorithm

Longest match policy by a 2FA with a pebble

Context by quotient constructions

Accommodate $\mathcal{T}_\text{in}$ by product construction
Future work

Implementation doable?

Apply to practical pattern languages