CDuce

Véronique Benzaken, Giuseppe Castagna, Alain Frisch

http://www.cduce.org/

PSD Workshop, 2004-02

Programming with XML

Level 0: textual representation of XML documents

AWK, sed, Perl

Level 1: abstract view provided by a parser

SAX, DOM

Level 2: untyped XML-specific languages

XSLT, XPath

Level 3: XML types taken seriously (aka: related work)

XDuce, Xtatic

Presentation

CDuce in a nutshell

- XML-oriented
- type-centric
- general-purpose features
- efficient (faster than XSLT at least !)

Intended uses

- Small “adapters” between different XML applications
- Larger applications
- Web applications, web services
- XML I/O layer for applications in other languages

Status of the implementation

- Public release available for download
  (+ online web prototype to play with).
- JIT compilation of pattern matching.
- Quite efficient, but many more optimizations are possible (and considered).

Integration with standards

- Unicode, XML, Namespaces: fully supported.
- DTD: external `dtd2cduce` tool.
- XML Schema: implemented at a deeper level.

XML-oriented + data-centric

- XML literals : in the syntax.
- XML fragments : first-class citizens, not embedded in objects.

```
<program>[
  <date day="monday">[
    <invited>[ <title>[ 'Conservation of information' ]
               <author>[ 'Thomas Knight, Jr.' ] ]
    <talk>[ <title>[ 'Scripting the type-inference process' ]
            <author>[ 'Bastiaan Heeren' ]
            <author>[ 'Jurriaan Hage' ]
            <author>[ 'Doaitse Swierstra' ] ] ] ]
```

Types

Types are pervasive in CDuce:

- Static validation
  - E.g.: does the transformation produce valid XHTML ?
- Type-driven semantics
  - Dynamic dispatch
  - Overloaded functions
- Type-driven compilation
  - Optimizations made possible by static types
  - Avoids unnecessary and redundant tests at runtime
  - Allows a more declarative style

Typed XML

$$\boxed{\vdash v : t}$$

```
v  ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
t  ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
```

Typed XML

$$\boxed{\vdash v : t}$$

```
v  ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
t  ==
  <program>[
    <date day=String>[
      <invited>[ <title>[ PCDATA ]
                 <author>[ PCDATA ] ]
      <talk>[ <title>[ PCDATA ]
              <author>[ PCDATA ]
              <author>[ PCDATA ]
              <author>[ PCDATA ] ] ] ]
```

Typed XML

$$\boxed{\vdash v : t}$$

```
v  ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
t  ==
  <program>[
    <date day=String>[
      <invited>[ Title Author ]
      <talk>[ Title Author Author Author  ] ] ]

type Author = <author>[ PCDATA ]
type Title  = <title>[ PCDATA ]
```

Typed XML

$$\boxed{\vdash v : t}$$

```
v  ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
t  ==
  <program>[
    <date day=String>[
      <invited>[ Title Author+ ]
      <talk>[ Title Author+ ] ] ]

type Author = <author>[ PCDATA ]
type Title  = <title>[ PCDATA ]
```

Typed XML

$$\boxed{\vdash v : t}$$

```
v ==
  <program>[
    <date day="monday">[
      <invited>[ <title>[ 'Conservation of information' ]
                 <author>[ 'Thomas Knight, Jr.' ] ]
      <talk>[ <title>[ 'Scripting the type-inference process' ]
              <author>[ 'Bastiaan Heeren' ]
              <author>[ 'Jurriaan Hage' ]
              <author>[ 'Doaitse Swierstra' ] ] ] ]
t ==
  Program

type Program = <program>[ Day* ]
type Day = <date day=String>[ Invited? Talk+ ]
type Invited = <invited>[ Title Author+ ]
type Talk = <talk>[ Title Author+ ]
type Author = <author>[ PCDATA ]
type Title  = <title>[ PCDATA ]
```

Types

- Types describe values.
- A natural notion of subtyping:

$$t \leq s \iff [\![t]\!] \subset [\![s]\!]$$

where

$$[\![t]\!] = \{v \mid \vdash v : t\}$$

Problem: circular definition between subtyping and typing!

Bootstrap method to remain set-theoretic.

Problem: implementation of the complex subtyping relation

- hand-made lightweight solver
  ($\rightsquigarrow$ remove backtracking from XDuce algorithm)
- caching, set-theoretic heuristics

Pattern Matching: ML-like flavor

- ML-like flavor:

```
match e with <date day=d>_ -> d

type E = <add>[Int Int] | <sub>[Int Int]
fun eval (E -> Int)
 | <add>[ x y ] -> x + y
 | <sub>[ x y ] -> x - y
```

- Patterns are “types with capture variables”

Pattern Matching: beyond ML

- Type-based dispatch:

```
match e with
 | x & Int -> ...
 | x & Char -> ...

let doc =
  match (load_xml "doc.xml") with
    | x & DocType -> x
    | _ -> raise "Invalid input !";;
```

Pattern Matching: beyond ML

- Regular expression and capture:

```
fun (Invited|Talk -> [Author+])  <_>[ Title x::Author* ] -> x

fun ([(Invited|Talk|Event)*] -> ([Invited*], [Talk*]))
  [ (i::Invited | t::Talk | _)* ] -> (i,t)

fun parse_email (String -> (String,String))
 | [ local::_* '@' domain::_* ] -> (local,domain)
 | _ -> raise "Invalid email address"
```

Compilation of pattern matching

Implementation of pattern matching

A new kind of push-down tree automata.
$\rightsquigarrow$ Non-backtracking implementation
$\rightsquigarrow$ Uses static type information
$\rightsquigarrow$ Allows a more declarative style.

```
type A = <a>[ Int* ]
type B = <b>[ Char* ]

fun ([A+|B+] -> Int) [A+] -> 0 | [B+] -> 1
≃
fun ([A+|B+] -> Int) [ <a>_ _* ] -> 0 | _ -> 1
```

Functions

- Overloaded, first-class, subtyping, name sharing, code sharing...

```
type Program = <program>[ Day* ]
type Day = <date day=String>[ Invited? Talk+ ]
type Invited = <invited>[ Title Author+ ]
type Talk = <talk>[ Title Author+ ]

let patch_program
  (p :[Program], f :(Invited -> Invited) & (Talk -> Talk)):[Program] =
  xtransform p with (Invited | Talk) & x -> [ (f x) ]

let first_author ([Program] -> [Program];
                  Invited -> Invited;
                  Talk -> Talk)
| [ Program ] & p -> patch_program (p,first_author)
| <invited>[ t a _* ] -> <invited>[ t a ]
| <talk>[ t a _* ] -> <talk>[ t a ]

(* we can replace the last two branches with:
  <(k)>[ t a _* ] -> <(k)>[ t a ]
*)
```

Precise type errors

```
type Title = <title>String
type Author = <author>String
type Talk = <talk>[ Title Author+ ]

let x : Talk = <talk>[ <author>[ 'Alain Frisch' ] <title>[ 'CDuce' ] ]
```

$\rightsquigarrow$

```
let x : Talk = <talk>[ <author>[ 'Alain Frisch' ] <title>[ 'CDuce' ] ]
This expression should have type:
`title
but its inferred type is:
`author
which is not a subtype, as shown by the sample:
`author
```

Precise type errors

```
type Title = <title>String
type Author = <author>String
type Talk = <talk>[ Title Author+ ]

fun mk_talk(s : String) : Talk = <talk>[ <title>s ]
```

$\rightsquigarrow$

```
fun mk_talk(s : String) : Talk = <talk>[ <title>s ]
This expression should have type:
[ Author+ ]
but its inferred type is:
[  ]
which is not a subtype, as shown by the sample:
[  ]
```

Precise type errors

```
type Title = <title>String
type Author = <author>String
type Talk = <talk>[ Title Author+ ]
type Invited = <invited>[ Title Author+ ]
type Day = <date>[ Invited? Talk+ ]

fun (Day -> [Talk+]) <date>[ _ x::_*] -> x
```

$\rightsquigarrow$

```
fun (Day -> [Talk+]) <date>[ _ x::_*] -> x
This expression should have type:
[ Talk+ ]
but its inferred type is:
[ Talk* ]
which is not a subtype, as shown by the sample:
[  ]
```

Precise type errors

```
type Program = <program>[ Day* ]
type Day = <date day=String>[ Invited? Talk+ ]
type Invited = <invited>[ Title Author+ ]
type Talk = <talk>[ Title Author+ ]
type Author = <author>[ PCDATA ]
type Title  = <title>[ PCDATA ]

fun (p :[Program]):[Program] = xtransform p with Invited -> []
fun (p :[Program]):[Program] = xtransform p with <invited>c -> [<talk>c]

fun (p :[Program]):[Program] = xtransform p with Talk -> []
```

⇝

```
fun (p :[Program]):[Program] = xtransform p with Talk -> []
This expression should have type:
[ Program ]
but its inferred type is:
[ <program>[ <date day = String>[ Invited? ]* ] ]
which is not a subtype, as shown by the sample:
[ <program>[ <date day = "">[  ] ] ]
```

Other features

- General-purpose: records, tuples, integers, exceptions, references, . . .
- String + regular expressions (types/patterns)
- Boolean connectives (types/patterns)
- Other iterators

Ongoing work on language design

Already in the CVS

- XML Schema (S. Zacchiroli)
- Interface with OCaml (J. Demouth)

Currently investigated

- XSLT/XPath/XQuery-like primitives
  $\text{XPath} + \text{patterns} + \text{filters} = ?$
- Interface with external languages.
- Module system, incremental programming.
- Parametric polymorphism
  (joint work with H. Hosoya).

Around CDuce

In the CDuce team

- Security & information flow analysis (M. Burelle)
- Query language (C. Miachon)
- Webservices (TBA)

Collaborations

- Interface with a native XML store (joint work with I. Manolescu)
- Verifying CDuce programs with a theorem prover (joint work with the Coq group)
- Graphical programming language (joint work with Brixlogic)

Thank you !

http://www.cduce.org/