The Xtatic Experience

Vladimir Gapeyev, Michael Y. Levin, Benjamin C. Pierce,
and Alan Schmitt

University of Pennsylvania

Regular pattern matching for the masses

Xtatic: Extension of C♯ for statically typed XML processing

- Offspring of the XDuce family
  - Regular types for XML and regular patterns
- Goals:
  - Simplicity: easy to use and understand
  - Flexibility: processing of values of (partially) unknown type
  - Lightweight extension of and tight integration with C♯
- Current status:
  - Xtatic to C♯ source to source compiler
  - Several applications written in Xtatic
    - Online bibtex to HTML / RSS generator
    - Used weekly to generate the Caml Weekly News

Outline

Xtatic: Extension of C♯ for statically typed XML processing

This talk: some language design issues encountered

- What type system for XML values?
- What XML inspection mechanism(s) to use?
- How to realize a tight integration with C♯?

Typing an address book

- XML types: based on regular tree grammars
- Several classes, based on restrictions on the content model
- Content model of an element: sequence of types of its subtrees

A simple address book:

```
<entry> <name>Pat</>, <tel>314-1593</> </entry>
<entry> <name>Jo</>,  <tel>271-8282</> </entry>
```

with type:

```
regtype Name   <name>pcdata</>
regtype Tel    <tel>pcdata</>
regtype AddrBk <entry> Name, Tel </entry>*
```

Local tree grammar: tag specifies content model (DTD)

(Classification of [Murata, Lee, Mani – EML'01])

Typing an address book

Adding categories and new data:

```
<fun>
  <entry> <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</> </entry>
</fun>
<work>
  <entry> <name>Jo</>,  <tel>271-8282</>, <email>Jo@jo.com</> </entry>
</work>
```

with type:

```
regtype Addr      = <addr>pcdata</>
regtype Email     = <email>pcdata</>
regtype FunEntry  = <fun>  <entry> Name, Tel, Addr? </entry> </fun>
regtype WorkEntry = <work> <entry> Name, Tel, Email </entry> </work>
regtype AddrBk    = (FunEntry | WorkEntry)*
```

Single-type tree grammar: downward path from root specifies content model (Schema)

Typing an address book

Putting category information before each entry:

```
<fun />
<entry> <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</> </entry>
<work />
<entry> <name>Jo</>,  <tel>271-8282</>, <email>Jo@jo.com</> </entry>
```

with type:

```
regtype Fun       = <fun />
regtype Work      = <work />
regtype FunEntry  = Fun,  <entry> Name, Tel, Addr? </entry>
regtype WorkEntry = Work, <entry> Name, Tel, Email </entry>
regtype AddrBk    = (FunEntry | WorkEntry)*
```

Restrained-competition tree grammar: downward path and left siblings specifies content

Typing an address book

Putting category information in each entry:

```
<entry>
  <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</>, <fun />
</entry>
<entry>
  <name>Jo</>,  <tel>271-8282</>, <email>Jo@jo.com</>, <work />
</entry>
```

with type:

```
regtype FunEntry  = <entry> Name, Tel, Addr?, Fun  </entry>
regtype WorkEntry = <entry> Name, Tel, Email, Work </entry>
regtype AddrBk    = (FunEntry | WorkEntry)*
```

Regular tree grammar: no restriction for content model (RelaxNG)

Choosing a type system

- Simpler tree grammars (Local, Single-type) have simple and efficient validation and subtyping algorithms
- More powerful grammars have algorithms that remain implementable and practical (the XDuce experience)
- Every grammar is closed under intersection
- Only Regular tree grammars are also closed under union, difference, and concatenation (useful for type inference)
- Reasonable choices:

  Single-type tree grammar: efficiency and Schema compliance

  Regular tree grammar: versatility and closure properties

Outline

- What type system for XML values?
- What XML inspection mechanism(s) to use?
- How to realize a tight integration with C♯?

A taste of patterns

Where does my friend Pat live?

value:

```
<entry>
  <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</>, <fun />
</entry>
<entry>
  <name>Jo</>, <tel>271-8282</>, <email>Jo@jo.com</>, <work />
</entry>
```

A taste of patterns

Where does my friend Pat live?

Pattern: type annotated with variables [Hosoya, Pierce – POPL'01]
Context around and type of the value(s) to be extracted

pattern:

```
any,
<entry>
  <name>Pat</>, any,              <addr>pcdata x</>,        Fun
</entry>,
any
```

value:

```
<entry>
  <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</>, <fun />
</entry>
<entry>
  <name>Jo</>, <tel>271-8282</>, <email>Jo@jo.com</>, <work />
</entry>
```

A taste of patterns

Where does my friend Pat live?

Pattern: type annotated with variables [Hosoya, Pierce – POPL'01]
Context around and type of the value(s) to be extracted

pattern:

```
any,
<entry>
  <name>Pat</>, any,              <addr>pcdata x</>,        Fun
</entry>,
any
```

value:

```
<entry>
  <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</>, <fun />
</entry>
<entry>
  <name>Jo</>, <tel>271-8282</>, <email>Jo@jo.com</>, <work />
</entry>
```

A taste of patterns

Where does my friend Pat live? 42, Wallaby Way

Pattern: type annotated with variables [Hosoya, Pierce – POPL'01]
Context around and type of the value(s) to be extracted

pattern:

```
any,
<entry>
  <name>Pat</>, any,              <addr>pcdata x</>,        Fun
</entry>,
any
```

value:

```
<entry>
  <name>Pat</>, <tel>314-1593</>, <addr>42, Wallaby Way</>, <fun />
</entry>
<entry>
  <name>Jo</>, <tel>271-8282</>, <email>Jo@jo.com</>, <work />
</entry>
```

Pattern matching in Xtatic

```
match (addrbk) {
  case [[ <entry>
            <name>‘Pat‘</>, any, <addr>pcdata x</>, Fun
          </entry>,
          any rest ]]:
    ...
  case [[ (FunEntry | WorkEntry), any rest ]]:
    ...
  case [[ ]]:
    ...
}
```

- Similar to C♯ `switch`, first match policy
- Support from the type checker
  - Matching checked to be exhaustive; every pattern is useful
  - Inference of the type of bound variables
    (`rest` has type `(FunEntry | WorkEntry)*`)

XML manipulation: Patterns vs XPath

- Patterns: types annotated with binders
  - Convenient for splitting XML values horizontally
  - Multiple binders $\Longrightarrow$ extraction of multiple subtrees
- Paths: hierarchical XML navigation
  - Convenient for vertical inspection of XML values
  - Multi match: return all leaves satisfying the path
- In practice, Patterns and Paths are complementary
  - Extension of Xtatic with a subset of XPath in development
  - Common foundation for the two approaches

Schema evolution

- Typical case: extension of a type
- Friends now have an optional Email

```
regtype FunEntry  = <entry> Name, Tel, Addr?, Email?, Fun </entry>
```

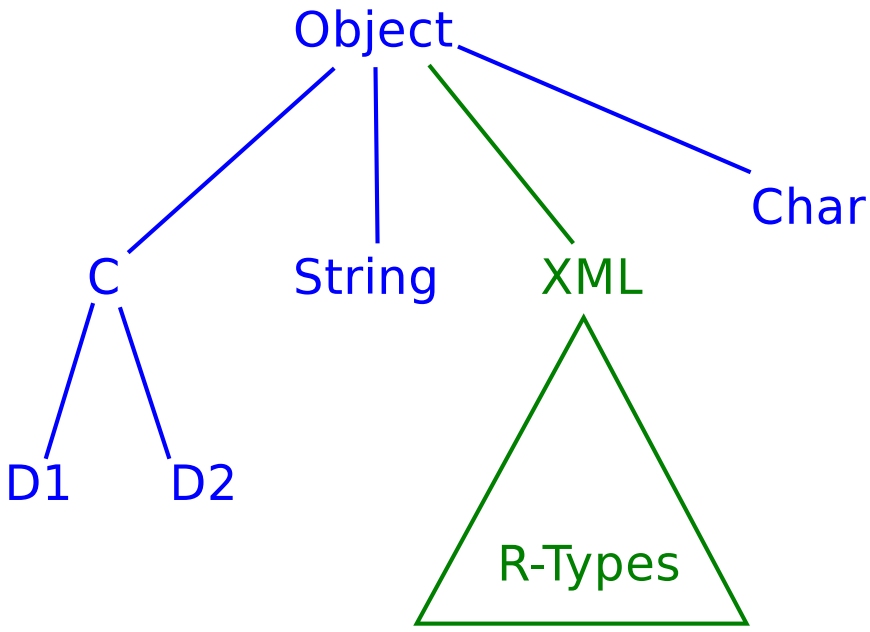
- Paths are too robust confronted to such evolution
  - `//entry[fun][name/text() = "Pat"]/addr/text()`
  - The program still works, the new information is ignored
  - What if the program was printing the data?
- Precise patterns flag an error: match clause not exhaustive
  - The type checker guides the programmer
    - with an example of a value not matched
  - Very useful in practice

Outline

- What type system for XML values?
- What XML inspection mechanism(s) to use?
- How to realize a tight integration with C♯?

XML in the class hierarchy

- Sequences are objects of class XML
  - May be used in collections



- Most languages follow this approach

Objects in XML

- Labels are objects, Label types are classes

$$T = () \mid T_1, T_2 \mid T_1 | T_2 \mid T* \mid <(\mathtt{C})>T</>$$

- XML tags are singleton classes, conceptually subclasses of Tag:
  $<\mathtt{addrbk}>\cdots</> \equiv <(\mathtt{Tag}_{\mathtt{addrbk}})>\cdots</>$
- Characters are singleton classes, conceptually subclasses of $Char$: ‘Pat‘ $\equiv <(\mathtt{Char}_{\mathtt{P}})/><(\mathtt{Char}_{\mathtt{a}})/><(\mathtt{Char}_{\mathtt{t}})/>$
  - Pattern matching used for string regular expressions

```
regtype url_protocols [[ ‘http‘ | ‘ftp‘ | ‘https‘ ]]
regtype url [[ url_protocols , ‘://‘ , (url_char *) ]]
...
case [[ url u, any rest ]] :
  res = [[ res , <a href = u>u</> ]]; p = rest;
```

Imperative idioms: XML modification

- For static type safety reasons, XML values are immutable
  $\Longrightarrow$ no direct assignment as in XJ
- To modify a value, its context must be captured and recreated

```
match (addrbk) {
  case [[ any bef, <entry>
                        <name>‘Pat‘</>, any a, Email?, Fun f
                  </entry>, any after ]]:
    return [[ bef, <entry>
                        <name>‘Pat‘</>, a, <email>‘pat@pat.net‘</>, f
                  </entry>, after ]];
  case [[ any no_pat ]]:
    return no_pat;
}
```

- Simpler in Xact: a primitive creates holes, another fills them

Imperative idioms: repeated concatenation

- Case study: creation of a sequence one element at a time
- Efficient imperative approach: mutation of the end of the list
  - Requires mutable values
- Efficient functional approach: insert all elements at the beginning then reverse the sequence
  - Efficient if good tail recursion compilation
- Xtatic's approach:
  - Naive concatenation of sequences

```
[[ AddrBk ]] p = [[ ]];
while (some_condition) {
  p = [[ p, <entry> ... </> ]];
}
```

  - Compiled to lazy data structures

More in the paper. . .

- Nominal vs Structural type systems
- Simple (as in easy to use) type system for attributes
- Fast downcasting for XML values in collections
- Dealing with legacy representations

Conclusions

- Convenient type grammars for XML values
  - Single type (standard compliance, ease of implementation)
  - Regular (power, closure properties)
  - Efficient implementation of the latter is practical
- Regular pattern matching is a powerful XML processing tool
  - Complements XPath inspection mechanisms
  - Very helpful for dealing with schema evolution
  - Extension of Xtatic with a subset of XPath in development
- Tight integration of XML processing with OO language possible
  - Sequences as objects, objects as labels; Simple and flexible
  - Tension between OO idioms and declarative XML lessened
  - Tighter integration (with objects in sequences) studied in $C\omega$, at the cost of the richness of the type system

The Xtatic experience

- Xtatic's language design difficult but enlightening
  - Goal of keeping things simple requires self-control
  - Many things go "under the hood"
    - Type checker and run-time structures optimizations
    - Transparent interaction with C♯ (separate compilation)
- Building applications is crucial
  - Caml Weekly News rely on Xtatic
  - Takes a lot of time

There is a future for statically typed XML processing in mainstream languages

Questions?

http://www.cis.upenn.edu/~bcpierce/xtatic/