Foundations of XML Data Manipulation

Giorgio Ghelli

Type Systems for SSD
Plan of the lesson

- XDuce type system (xduce-toit.2003, google:xduce)
- Tree automata (TATA, google:tree automata)
- DTD’s
- XSD
- Mu-calculus and TQL Logic
- Path inclusion

A Query

```xml
for $b in $doc /bib/book,
    $a in $b /author
where $b /@year > 2000
return booksbyaut [$a,
    for $bb in $doc /bib/book
    where $a isin $bb /author
    return $bb/title
]
```
Its Type

• Type:
  Bookbyaut[ author[String], title[String]* ]*

• If:
  $doc: bib[ book [ title[String],
  author[String]*
  ]*
  ]

Types for SSD

• XDuce type system:
  T ::= B base types
  m[T] tree types
  T, T forest concatenation types
  0 empty tree singleton type
  T | T union type
  X m[ ]-guarded recursion
  T* Kleene star type
Recursion

- Section = para[] | section[Section*]

- Illegal:
  - paraList = para[] | (para[], paraList)

- Equivalent to:
  - paralist = para[], (para[])*

- Illegal:
  - X = a[], X , b[] | 0

Type rules

\[
\begin{align*}
0 & : 0 \\
t & : T \\
t & : T|U \\
m[t] & : m[T] \\
t, u & : T, U \\
t & : T | U \\
0 & : T^* \\
t, u & : T^* \\
X = T & \vdash t : T \\
X = T & \vdash t : X
\end{align*}
\]
Not so different?

- Is $T, T'$ the same as $T \times T'$?
- Is $T^\ast$ the same as $\text{List}(T)$?
- Is $a[T], b[U], c[V]$ the same as $\text{record}[a: T, b: U, c: V]$?

$T, T'$ vs. $T \times T'$

- $(T, T'), T'' = T, (T', T'') = T, T', T''$
- $T, 0 = T$ but $T \times 1 \neq T$
- $\langle t, u \rangle: T \times U$ iff $t: T$ and $u: U$
- It may be that $t, u: T, U$ but not $t: T$ and $u: U$:
  - $(a[], b[]), c[]: (a[], (b[], c[]))$
  - $a[], (b[], c[]): (a[] | a[], b[]), (b[], b[] | c[])$
  - Type checking similar to regular-language testing
- $T \times U < T' \times U'$ iff $T < T'$ and $U < U'$
- It may be that $T, U < T', U'$ but $T \neq T'$ and $U \neq U'$
- Subtyping defined as language inclusion, and checked as automata inclusion
The semantic intuition

- At run time, a value of type $T \times U$ is represented as $\text{pair}(t, u)$
- A value of type $T \times U$ is “bigger” than a value of type $T$
- If $t:T$ then $t$ has NOT type $T \times U$
- A value of type $T, U$ has no $\text{tag}(_, _)\text{structure}$
- If $t:T$ and $0:U$ then $t: (T, U)$

a[T], b[U] vs. tup[a:T] + tup[b:U]

- Record concatenation:
  - $\text{tup}[a:T] + \text{tup}[b:U] = \text{tup}[a:T, b:U]$
- Usually $T+U$ only defined when $T$ is a simple-record type $\text{tup}[a:T]$; $T, U$ accepts any type as $T$, including $(a[]) | 0$
**T* vs. List(T)**

- \( T^* = \mu TS. 0 | (T, TS) \)
- \( \text{List}(T) = \mu LT. 1 | (T \times LT) \)
- Hence:
  - \( T^* = T^{**} \)
  - \( \text{List}(T) \neq \text{List} (\text{List}(T)) \)
  - \( T < T^*: \) no List tag at run-time
  - \( T \neq \text{List}(T) \)

**Guarded recursion**

- Regular grammars:
  - \( X = a_1 | a_2 | a_3.X_1 | a_4.X_n \)
- XDuce “linear” recursive types:
  - \( X = a_1[] | a_2[] | a_3[X_1] | a_4[X_2] \)
  - Correspond to automata
- XDuce tree-like recursive types:
  - \( X = a_1[] | a_2[] | a_3[X_1,X_2] | a_4[X_3,X_4] \)
  - Correspond to tree automata
Horizontal and Vertical RegExps

- **Horizontal**: 
  \[ X = a[](b[])^* \mid (a[])^*b[] \]

- **Vertical**
  - \[ X = a[Y] \mid W \]
  - \[ Y = 0 \mid b[Y] \]
  - \[ W = b[] \mid a[W] \]

- Tree automata because of Vertical
- *Regular* tree automata because of Horizontal

Tree Automata

- \((A,Q,R,F)\) with
  - \(F \subseteq \text{Lists}(Q)\)
  - \(R \subseteq A \times \text{Lists}(Q) \times Q\) (set of \(a[q_1,\ldots,q_n] \rightarrow q\) rules)

- A run:
  - Substitute \(a[]\) with \(q\) \((a^q[])\) if \(a[] \rightarrow q \in R\)
  - Substitute \(a[q_1,\ldots,q_n]\) with \(q\) if \(a[q_1,\ldots,q_n] \rightarrow q \in R\)
  - Accept \(t_1,\ldots,t_n\) if rewritten as \(q_1,\ldots,q_n \in F\)
Unranked Tree Automata

- Ranked Tree Automata:
  - Rules are like $a_2[q_1,q_2] \rightarrow q_3$: for each binary symbol, I need $2^{3|Q|}$ rules at most

- But: $A \times \text{Lists}(Q) \times Q$ is not finite:
  - Rules like $a[q_1,\ldots,q_1] \rightarrow q_2$

- Problem:
  - Representing $R$ and deciding $a[q_1,\ldots,q_n] \rightarrow q \in R$

Regular Tree Automata

- Regular Tree Automata:
  - For each $a$, $q$, the language (in $Q^*$) $\{q_1,\ldots,q_n \mid a[q_1,\ldots,q_n] \rightarrow q \in R\}$ is regular

- $R$ can be represented as a function of type $A \times Q \rightarrow \text{RegExp}(Q)$

- Tree automata correspond to vertical recursion

- Regular in RTA corresponds to horizontal regular recursion
Unranked binary expressions

• Or(F, And(T, T, F), And(F, T))
  where T = And(), F = Or()
• A = {And, Or}; Q = {t, f}
• R:
  – Or(f*) -> f
  – Or( (t|f)*, t, (t|f)* ) -> t
  – And(t*) -> t
  – And ( (t|f)*, f, (t|f)* ) -> f

We like automata

• Recognize trees in linear time
• Closed by union, intersection, complement
• Emptyness is decidable
• A <: A' iff
  A \ A' = A \ \ Co(A') is empty
**DTDs**

- Canonical way to describe the structure of an XML document

- Example:
  ```xml
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, children?)>
  <!ELEMENT children (person+)>  
  <!ELEMENT name (#PCDATA)>  
  <!ELEMENT birthdate (#PCDATA)>
  ```

**An example**

- **DTD:**
  ```xml
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, children?)>
  <!ELEMENT children (person+)>  
  ```

- **Document:**
  ```xml
  <!DOCTYPE people_list SYSTEM "example.dtd">
  <people_list>
    <person>
      <name>Fred Bloggs</name>
      <birthdate>…</birthdate>
      <children>
        <person><name>Jim</name></person>
      </children>
    </person>
    <person><name>Luis Gutierrez</name></person>
  </people_list>
  ```
**XDuce vs. DTD**

- **XDuce**: a set of mutual recursive defs:
  - $A = a[B_1^*, B_2]$
  - $B_1 = b[X]$
  - $B_2 = b[Y]$
  - ...

- **DTD**: the type is identified by the label:
  - $a = a[b^*, c]$
  - $b = b[X]$
  - $c = c[Y]$

**DTD into automata**

- **DTD**:
  
  ```xml
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, children?)>
  <!ELEMENT children (person+)>
  ```

- **Automaton**:
  - $F = PL$
  - $R$:  
    - people_list[P*] -> PL
    - person[N,BD?,C?] -> P
    - children[P+] -> C
    - name[ ] -> N ...

XDuce into automata

- XDuce:
  - Paper where
  - Paper = title[], section[abstract[]], Content
  - Content = ( paragraph[] | section[Content]) *

- Automa:
  - F = T,SA,(P|SC)*
  - R:
    - title[] -> T, abstract[] -> A,
    - section[A] -> SA
    - paragraph[] -> P
    - section[(P|SC)∗] -> SC

XSD

- Local element types are not label-identified, but global element types are:
  - a = a[b[X]]
  - b = b[a[Z]]
  - c = c[a[W]]

- Not every regexp is OK (Unique Particle Attribution):
  - a = a[b[X]^, b[X]]: illegal

- In one element, label identifies type (Element Declarations Consistent):
  - a = a[b[X], b[Y]]: illegal
XSD

• Names are qualified with respect to namespaces
• XSD can specify key and keyref constraints
• Two limited forms of subtyping by name: derivation and substitution groups

XSD Syntax

• Global element declarations:
  – element El of T
  – element El of (type [T] of {…})
  – May be local, in which case El is not a key

• Complex type definitions:
  – type T of
  – Either anonymous, or T is a key (even if local)
XSD Assessment

• Assessment:
  – local validation, schema-validity assessment and infoset augmentation:
    Infoset -> PSVI

• PSVI contains:
  – Normalized and default values for attributes and elements
  – Type definitions for attributes and elements
  – Validation outcome

XSD and Subtyping

• Derivation:
  – Every complex type either extends or restricts another type, starting from xsi:anyType
  – Explicit cast: the derived type can be used for validation only if a corresponding xsi:type attribute is present in the element to validate, or if the element name is in the substitution group of the expected name

• Substitution:
  – An element name may be head of a substitution group, and the other names from the group are valid where the head is required
  – The types of the group elements must be derived from the type of the head
Semantic subtyping

- Rule-based subtyping:
  - Subsumption: \( T <: T' \Rightarrow \forall t. t: T \Rightarrow t: T' \)
- Semantic subtyping:
  - Definition: \( (\forall t. t \in \llbracket T \rrbracket \Rightarrow t \in \llbracket T' \rrbracket) \Rightarrow T <: T' \)
- For example:
  - \((\text{forall } X. X \rightarrow X) <:? \text{ Int } \rightarrow \text{ Int}\)
- XSD: rule-based subtyping
- XDuce, CDuce: semantic subtyping

DTD and \(\mu\)-calculus

- [everywhere] \( A = v_\xi (A \land [\downarrow]_\xi \land [\rightarrow]_\xi) \)
- We extend \( \mu \) with equations:
  - \( A \) where \( \$x_1 = A_1, \ldots, \$x_n = A_n \)
  - \( A(\xi) \) where \( \xi = A_1 \)
    - is the same as
    - \( A(\mu_\xi. A_1) \)
- Still checkable in \( O(2^n) \)
DTD and μ-calculus

• DTD:
  ```
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, children?)>
  <!ELEMENT children (person+)>
  ```

• μ with equations:
  ```
  [everywhere] (people_list ∩ [⊔] $PersonPlus) ∨ (person ∩ [⊔] $NBC) ∨ (children ∩ [⊔] $PersonPlus) ∨ (name ∩ [⊔] False) ∨ ...
  ```

  where $PersonPlus = person ∩ [→] $PersonPlus
  $NBC = name ∩ [→] $BC
  $BC = (birthdate ∩ [→] $C) ∨ children

TQL Logic

• Ordered TQL logic:
  ```
  A ::= B base values
  η[A] trees (η: x or n)
  A , A sequence
  0 empty tree singleton sentence
  T ∨ T disjunction
  ¬A negation
  μζ. A (positive) recursion
  ξ recursion variable
  ∃x.A label quantification
  ∃X.A forest quantification
  X forest variable
  ```
The actual TQL logic

- TQL data model in unordered:
  \[ 0 \mid t = t \; ; \; (t \mid t') \mid t'' = t \mid (t' \mid t'') \; ; \; t \mid t' = t' \mid t \]

- In TQL ordered logic:
  \[ t \vdash (A \mid B) \iff \exists t', t''. t', t'' = t \text{ and } t' \vdash A \text{ and } t'' \vdash B \]

- In TQL logic:
  \[ t \vdash A \mid B \iff \exists t', t''. t' \mid t'' = t \text{ and } t' \vdash A \text{ and } t'' \vdash B \]

TQL Logic

- \( F \vdash True: \text{ always } (True = 0 \lor \neg 0) \)
- \( F \vdash 0 \iff F = 0 \)
- \( F \vdash A \mid B \iff \exists F', F''. F = F' \mid F'', F' \vdash A, F'' \vdash B \)
  \[ F \vdash m[A] \iff F = m[F'], \; F' \vdash A \]

- E.g.:
  - \( a[0] \mid b[0] \vdash b[0] \mid True \?)
  - \( b[0] \vdash b[0] \mid True \?)
  - \( a[0] \mid b[0] \vdash b[0] \?)
  - \( a[b[0]] \vdash a[True] \mid True? \)
Other operators

• $F \models A \wedge B \iff F \models A$ and $F \models B$
• $F \models \neg A \iff \neg (F \models A)$
• Derived operators:
  – $A \lor B =_{\text{def}} \neg (\neg A \land \neg B)$
  – $A \parallel B =_{\text{def}} \neg (\neg A \lor \neg B)$
  – $m[A] =_{\text{def}} \neg m[\neg A]$
• $F \models (a[True] \lor b[True]) \land True$
• $F \models (a[0] \parallel True) \wedge \neg(a[0] \lor a[0] \land True)$
• $F \models author[\Rightarrow Hull] \parallel False$

More than types?

• Complement $\neg A$ dualizes every other operator ($\exists \rightarrow \forall$, $\lor \rightarrow \land$, $\mu \rightarrow \nu$, ...)
• Horizontal recursion is more than $T^*$
• Quantification expresses correlation:
  – $\exists x. x[True] \parallel x[True] \parallel True$
  – $\exists x. x[A] \land x[A]$ \small ($x[A] = x[A] \parallel True$)
  – $\exists x. x[A] \parallel x[A]$ \small ($x[A] = x[A] \parallel True$)
• Logic can express key constraints:
  – $\neg \exists X. .book[t[X]] \parallel .book[t[X]]$
Decidability

- Quantification makes emptiness undecidable
- Quantification makes model-checking (type-checking) PSpace-complete
- Model-checking is often doable in practice

Path containment
Path containment

- As binary relation: $\text{sub}_2$
  - $p \text{sub}_2 q \iff (m \ p \ n \Rightarrow m \ q \ n) \iff [[p]] \subseteq [[q]]$

- Starting from the root: $\text{sub}_1$
  - $p \text{sub}_1 q \iff (\text{root} \ p \ n \Rightarrow \text{root} \ q \ n)$

- Boolean containment, starting from the root:
  - $t \models p$: matching $p$ against the root of $t$ yields non-empty result
  - $p \text{sub}_0 q \iff t \models p \Rightarrow t \models q$

Notions of containment

- If we restrict to child/desc, $\text{sub}_2 \ e \text{sub}_1$ are equivalent

- In the presence of predicates, $\text{sub}_1$ can be mapped to $\text{sub}_0$:
  - $p \text{sub}_1 q \iff p[x] \text{sub}_0 q[x]$, where:
    - $p[x]$ adds a child::$x$ condition to the selection node, and $x$ is fresh (Miklau-Suciu PODS 02)
Complexity for PositiveXPath

• PTime:
  – No disjunction, 2 of // [] * but not all:
    XP(/,//,*), XP(/,[],*), XP(/,//,[]), XP(/,//)+DTD,
  – coNP:
    – XP(/,|); XP(//,|); XP(/,[])+DTD, XP(//,[])+DTD
    – XP(/,//,[],*);
    – XP(/,//,[],*,|) (becomes PSPACE if the alphabet is finite);
• ExpTime:
  – XP(/,//,|) + DTD
  – XP(/,//,[],|,* ) + DTD

Path inclusion and μ-calculus

• Let <<p>> be the translation of a path and <<s>> the translation of a schema:
  – E,L, m \models <<s>> if E,L satisfies s

• p \subseteq_2 q:
  – Valid ( <<p>> \supseteq <<q>> )
  – i.e., for any E,L,m,n:
    E,L,i\rightarrow n, m \models <<p>> \supseteq <<q>>

• p \subseteq_2 q under s:
  – For any E,L,m,n:
    E,L,i\rightarrow n, m \models <<p>> \land <<s>> \supseteq <<q>>

• <<_>> is linear ⇒ inclusion is O(2^n) for NavXPath