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

<table>
<thead>
<tr>
<th>0: 0</th>
<th>t: T</th>
<th>t: T u: U</th>
</tr>
</thead>
<tbody>
<tr>
<td>m[t] : m[T]</td>
<td>t, u : T, U</td>
<td></td>
</tr>
<tr>
<td>t: T</td>
<td>t: T</td>
<td>U</td>
</tr>
<tr>
<td>0 : T*</td>
<td>t, u : T*</td>
<td></td>
</tr>
</tbody>
</table>

$X=T \vdash t: T$
$X=T \vdash t: X$

Not so different?

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

T,T' vs. TxT'

- $(T, T'), T'' = T, (T', T'') = T, T'$
- $T, 0 = T$ but $T \neq 1 = T$
- $<t,u>: T \times U$ iff $t: T$ and $u: U$
- It may be that $t: T, U$ but not $t:T$ and $u: U$:
  - $(a[],b[]): (a[],b[])$
  - $(a[],b[]): (a[] | a[],b[])$, $(b[],b[] | c[])$
- Type checking similar to regular-language testing
- $T \times U < T', U'$ 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 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 tag(_,_) 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 tup[a:T]; T,U accepts any type as T, including $(a[] | 0)$
### T* vs. List(T)

- \(T^* = \mu T S. 0 | (T, TS)\)
- \(\text{List}(T) = \mu L T. 1 | (T x LT)\)

**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**:
  - \(X = a[](b[])^* | (a[])^*b[]\)
- **Vertical**
  - \(X = a[Y] | W\)
  - \(Y = 0 | b[Y]\)
  - \(W = b[] | a[W]\)
- 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 \(q\) (in \(a[q]\)) 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 \(Q^*\)) \(\{q_1, \ldots, q_n | a[q_1, \ldots, q_n] \rightarrow q \in R\}\) is regular
  - \(R\) can be represented as a function of type \(A x Q \rightarrow \text{RegExp}(Q)\)
  - Tree automata correspond to vertical recursion
  - Regular in RTA corresponds to horizontal regular recursion
Unranked binary expressions

- $\text{Or}(F, \text{And}(T,T,F), \text{And}(F,T))$
  where $T = \text{And}()$, $F = \text{Or}()$
- $A = \{\text{And, Or}\}$; $Q = \{t,f\}$
- $R$:
  - $\text{Or}(t^*) \rightarrow f$
  - $\text{Or}( (t|f)^*, t, (t|f)^* ) \rightarrow t$
  - $\text{And}(t^*) \rightarrow t$
  - $\text{And}( (t|f)^*, f, (t|f)^* ) \rightarrow f$

We like automata

- Recognize trees in linear time
- Closed by union, intersection, complement
- Emptiness is decidable
- $A \triangleleft A'$ iff
  $A \setminus A' = A \cap \co(A')$ is empty

DTDs

- Canonical way to describe the structure of an XML document
- Example:
  $\langle!\text{ELEMENT people_list (person*)}\rangle$
  $\langle!\text{ELEMENT person (name, birthdate?, children?)}\rangle$
  $\langle!\text{ELEMENT children (person+)}\rangle$
  $\langle!\text{ELEMENT name (#PCDATA)}\rangle$
  $\langle!\text{ELEMENT birthdate (#PCDATA)}\rangle$

An example

- DTD:
  $\langle!\text{ELEMENT people_list (person*)}\rangle$
  $\langle!\text{ELEMENT person (name, birthdate?, children?)}\rangle$
  $\langle!\text{ELEMENT children (person+)}\rangle$
- Document:
  $\langle!\text{DOCTYPE people_list SYSTEM "example.dtd"}\rangle$
  $\langle!\text{people_list}\rangle$
    $\langle!\text{person}\rangle$
      $\langle!\text{name}\rangle$Fred Bloggs$\langle!\text{name}\rangle$
      $\langle!\text{birthdate…}\rangle$
      $\langle!\text{children}\rangle$
        $\langle!\text{person}\rangle$
          $\langle!\text{name}\rangle$Jim$\langle!\text{name}\rangle$
          $\langle!\text{children}\rangle$
            $\langle!\text{person}\rangle$
              $\langle!\text{name}\rangle$Luis Gutierrez$\langle!\text{name}\rangle$
  $\langle!\text{person}\rangle$
$\langle!\text{people_list}\rangle$

XDuce vs. DTD

- XDuce: a set of mutual recursive defs:
  - $A = a[B1^*,B2]$
  - $B1 = b[X]$
  - $B2 = b[Y]$
  - ...
- DTD: the type is identified by the label:
  - $a = a[b^*,c]$
  - $b = b[X]$
  - $c = c[Y]$

DTD into automata

- DTD:
  $\langle!\text{ELEMENT people_list (person*)}\rangle$
  $\langle!\text{ELEMENT person (name, birthdate?, children?)}\rangle$
  $\langle!\text{ELEMENT children (person+)}\rangle$
- Automaton:
  - $F = PL$
  - $R$:
    - $\text{people_list}[P^*] \rightarrow PL$
    - $\text{person}[N,BD?,C?] \rightarrow P$
    - $\text{children}[P+] \rightarrow C$
    - $\text{name}[] \rightarrow N$...
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 Syntax

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

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

XSD Assessmnet

• 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: $T <: T'$
  - Semantic subtyping:
    - Definition: $(\forall t. t \in [T]) \Rightarrow t \in [T']$ for $T <: T'$
- For example:
  - $(\forall X. X \rightarrow X) <: ? \text{ Int}$
- XSD: rule-based subtyping
- XDuce, CDuce: semantic subtyping

DTD and $\mu$-calculus

- [everywhere] $A = v_1 \land [\downarrow] A_1 \land [\rightarrow] A_2$
- We extend $\mu$ with equations:
  - $A$ where $X = A_1$, ..., $X = A_n$
  - $A(\xi)$ where $\xi = A_1$
  - is the same as $A(\mu \xi, A_1)$
- Still checkable in $O(2^n)$

DTD and $\mu$-calculus

- DTD:
  ```xml
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, children?)>
  <!ELEMENT children (person+)>  
  ```
- $\mu$ with equations:
  ```
  [\text{everywhere}] (people_list \lor (\downarrow) \text{PersonPlus})
  \lor (person \land (\land) \text{NBC})
  \lor (children \land (\downarrow) \text{PersonPlus})
  \lor (name \land (\land) \text{False}) \lor ... 
  ```
  where $\text{PersonPlus} = \text{person} \land (\rightarrow) \text{PersonPlus}$
  $\text{NBC} = \text{name} \land (\rightarrow) \text{BC}$
  $\text{BC} = (\text{birthdate} \land (\rightarrow) \text{C}) \lor \text{children}$

TQL Logic

- Ordered TQL logic:
  ```
  A ::= B base values
  [A] trees (n: x or n)
  A, A sequence
  0 empty tree singleton sentence
  A \lor B disjunction
  \neg A negation
  \mu A (positive) recursion
  \exists A label quantification
  \exists A forest quantification
  X forest variable
  ```
- $F \models \text{True}$: always ($\text{True} = 0 \lor \neg 0$)
- $F \models 0$ iff $F = 0$
- $F \models A \land B$ iff $F' \models F' \land F' \models A$, $F' \models B$
- $F \models m[A]$ iff $F \models m[F']$, $F' \models A$

TQL Logic

- E.g.:
  ```
  - $a[0] \land b[0] \models b[0] \land \text{True}$ ?
  - $b[0] \models b[0] \land \text{True}$ ?
  - $a[0] \lor b[0] \models a[0] \lor b[0]$ ?
  - $a[b[0]] \models a[\text{True}]$ ?
  ```
Other operators

- $F \models A \land B$ iff $F \models A$ and $F \models B$
- $F \models \neg A$ iff not $(F \models A)$
- Derived operators:
  - $A \lor B \overset{\text{def}}{=} \neg \neg (A \land \neg B)$
  - $A \parallel B \overset{\text{def}}{=} \neg (A \lor \neg B)$
  - $m[A] = \neg m[\neg A]$
- $F \models (a[\text{True}] \lor b[\text{True}]) \mid \text{True}$
- $F \models (a[0] \mid \text{True}) \land \neg(a[0] \mid a[0] \mid \text{True})$
- $F \models \text{author} [\Rightarrow \text{Hull}] \mid \mid \text{False}$

More than types?

- Complement $\neg A$ dualizes every other operator ($\exists \lor \forall, \lor \land, \mu \lor \nu, \ldots$)
- Horizontal recursion is more than $\text{T}^*$
- Quantification expresses correlation:
  - $\exists x \ x[\text{True}] \mid x[\text{True}] \mid \text{True}$
  - $\exists x \ x[A] \land x[A] \ (x[A] = x[A] \mid x[A] \mid \text{True})$
  - $\exists x \ x[A] \mid x[A] \ (x[A] = x[A] \mid \text{True})$
- Logic can express key constraints:
  - $\neg \exists X \ . \ \text{book} [\{X\}] \mid \text{book} [\{X\}]$

Decidability

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

Path containment

- As binary relation: $\text{sub}_2$
  - $p \text{ sub}_2 q \Leftrightarrow (m \ p \ n \Rightarrow m \ q \ n) \Rightarrow [[p]] \subseteq [[q]]$
- Starting from the root: $\text{sub}_1$
  - $p \text{ sub}_1 q \Leftrightarrow (\text{root} \ p \ n \Rightarrow \text{root} \ q \ n)$
- Boolean containment, starting from the root:
  - $t \models p$: matching $p$ against the root of $t$ yields non-empty result
  - $p \text{ sub}_2 q$ 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 $\text{sub}_0$ :
  - $p \text{ sub}_1 q \Leftrightarrow p[x] \text{ sub}_2 q[x]$, 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 \( \mu \)-calculus

- Let \(<p>\) be the translation of a path and \(<s>\) the translation of a schema:
  - E,L; m \( \not\models \) \(<s>\) if E,L satisfies \( s \)

- \( p \) sub \( q \):
  - Valid \( \ocomm{\phi} \Rightarrow \ocomm{\psi} \)
  - i.e., for any E,L,m,n:
    - E,L;i->n, m \( \not\models \) \(<p>\) => \(<q>\)

- \( p \) sub \( q \) under \( s \):
  - For any E,L,m,n:
    - E,L;i->n, m \( \not\models \) \(<p>\land \(<s>\) => \(<q>\)

- \( \com{\_} \) is linear \( \Rightarrow \) inclusion is \( O(2^n) \) for NavXPath