University of Washington
Department of Computer Science & Engineering

Managing XML and Semistructured Data

Lecture 13: XDuce and Regular Tree Languages

Prof. Dan Suciu

Spring 2001

In this lecture

- Introduction to XDuce
  - types in XDuce
  - subsumption and typechecking in XDuce
- Regular tree languages
  - tree automata
- Connection between regular languages and XDuce types

Resources

XDuce: A typed XML processing language by Hosoya and Pierce

Types in XDuce

- Xduce = a functional programming language (like ML)
- Emphasis: type checking for its functions
- Data model = ordered trees
  - Captures XML elements and attributes
- Types = regular expressions
  - Same expressive power as XML Schema
  - Simpler concept
  - Closer connection to regular tree languages

Values in XDuce

```
<bib>
  <book>
        <title> ML for the Working Programmer </title>
        <author> Paulson </author>
        <year> 1991 </year>
  </book>
  <paper> ... </paper>
  ...
</bib>
```

```
val x = bib[book[title["ML for the Working Programmer"],
                 author["Paulson"],
                 year["1991"]
                ],
            paper[....],
            ...
            ]
```

Types in XDuce

```
<!ELEMENT bib    ((book|paper)*)>
<!ELEMENT book (title, author*, year, publisher?)>
<!ELEMENT title #PCDATA>
...
```

```
type Bib = bib[(Book|Paper)*]
type Book = book[Title, Author*, Year, Publisher?]
type Title = title[String]
...
```

Types in XDuce

- Important idea:
  - Types are first class citizens
  - Element names are second class
- This is consistent with regular expressions and automata:
  - Type = state (we will see later)

Example of Types in XDuce

```
type T1 = b[] | a[T1, T0] | a[T0, T1]
type T0 = a[] | a[T0, T0]
```

Formal Definition of Types in XDuce

T ::= variable
::= base type
::= () /* empty sequence */
::= T,T /* concatenation */
::= T | T /* alternation */

Where are "*" and "?" ?

Types in XDuce

Derived types:

- Given T, the type T* is an abbreviation for:
  - type X = T, X | ()
- Similarly, T+ and T? are abbreviations for:
  - type X = T, T*
  - type Y = T | ()

Types in XDuce

- Danger with recursion:
  - Type X = a[], X, b[] | ()
  - What is is ?
- Need to restrict to *tail recursive* types

Subsumption in Xduce Types

- Definition. T1 <: T2 if the set defined by T1 is a subset of that defined by T2
- Examples
  - Name, Addr <: Name, Addr, Tel?
  - Name, Addr, Tel <: Name, Addr, Tel?
  - T, T, T <: T*

XDuce

- Main goal: given a function, check that it is type correct
  - Come to Benjamin Pierce's talk on Monday
- One note:
  - The type checking algorithm in Xduce incomplete (will see why, in a couple of lectures)
- Important piece of typechecking:
  - Checking if T1 <: T2
- Obviously can't do this for context free languages
- But can do for *regular languages* (next)

Regular Tree Languages

- Given a *ranked alphabet*, $L = L_0 \cup L_1 \cup \ldots \cup L_k$
- Ranked trees are $T ::= a[T_1,...,T_i] \quad a \in L_i$

Definition Bottom-up tree automaton is $A = (L, Q, \delta, Q_F)$ where:

- L = ranked alphabet
- Q = set of states
- δ = transition relation, $\delta: (\cup_{i=0,k} L_i \times Q^i) \rightarrow Q$
- Q_F = terminal states

Bottom Up Tree Authomata

Computation on a tree t

- For each node $t = a[t_1,...,t_i]$, if the roots of $t_1,..., t_i$ are labeled with states $q_1, ..., q_i$ and q in $\delta(a, q_1, ..., q_i)$, then label t with q
- If the root is labeled with a state in Q_F, then accept

The language **accepted** by A consists of all trees t accepted by A

A **regular tree language** is a set of trees accepted by some automaton A

Example of Tree Automaton

- $L_0 = \{b\}, \; L_2 = \{a\}$
- $Q = \{q_1, q_2\}$
- $\delta(b) = q_1, \delta(a,q_1,q_1) = q_2, \delta(a,q_2,q_2) = q_1$
- $Q_{final} = q_1$
- What does this accept ?

  trees such that each leaf is at even height

Properties of Regular Tree Languages

- If T1, T2 are regular, then so are:
  - $T1 \cup T2$
  - $T1 - T2$
  - $T1 \cap T2$
- If A is a nondeterministic bottom up tree automaton, then there exists an equivalent deterministic one
  - Not true for "top-down" automata
- If T1, T2 are regular, then it is decidable whether $T1 \subseteq T2$

Top-down Automata

- Defined similarly, just the computation differs:
  - Start from the root at an initial state, move downwards
  - If all leaves end in an accepting state, then accept
- Here deterministic automata are strictly weaker
  - e.g. cannot recognize the set {a[a,b], a[b,a]}
- Nondeterministic bottom up =
  = deterministic bottom up
  = nondeterministic top down

Example of a Bottom-up Automaton

- $A = (L, Q, , \delta, q_0, Q_F)$ where
  - $L = L_0 \cup L_2, \quad L_0 = \{a, b\}, \quad L_2 = \{a\}$
  - $Q = \{T0, T1\}$
  - $\delta(a) = T0, \delta(b) = T1,$
  - $\delta(a, T1, T0) = T1, \delta(a, T0, T1) = T1$

```
type T1 = b[] | a[T1, T0] | a[T0, T1]
type T0 = a[] | a[T0, T0]
```

Regular Tree Languages and XDuce types

- For ranked alphabets, tail-recursive Xduce types correspond precisely to regular tree languages
- Same is true for unranked alphabets, but there the definition of regular tree lnaugages is more complex

Conclusion for Schemas

A Theoretical View

- XML Schemas = Xduce types = regular tree languages
- DTDs = strictly weaker

A Practical View

- XML Schemas still too complex