MAINTAINING XML DATA INTEGRITY IN PROGRAMS

AN ABSTRACT DATATYPE APPROACH

Patrick Michel

Technische Universität Kaiserslautern

SoFTeX Arbeitsgruppe Softwaretechnik

XML Support in Programs

XML Support in Programs

Language Level	Language Support

XML Support in Programs

Language Level	Language Support
XML	Java + DOM / SAX

XML Support in Programs

Language Level	Language Support
XML	Java + DOM / SAX
XML + XML Schema	Java + JAXB / XJ

XML Support in Programs

Language Level	Language Support
XML	Java + DOM / SAX
XML + XML Schema	Java + JAXB / XJ
XML + Relax NG	XDuce

XML Support in Programs

Language Level	Language Support
XML	Java + DOM / SAX
XML + XML Schema	Java + JAXB / XJ
XML + Relax NG	XDuce
XML + Schematron / DSD	???

XML Support in Programs

Language Level	Language Support
XML	Java + DOM / SAX
XML + XML Schema	Java + JAXB / XJ
XML + Relax NG	XDuce
XML + Schematron / DSD	???
Constrained XML (Structure + Integrity)	Java + Abstract Datatype (Atomic Procedures)

Example

Example

Schema:

```
packetheader {
   capacity { INT [ sum(//kind/count) ≤ . ] } &
   kind ∗ {
      count { INT [ . > 0 ] }
   }
}
```

Example

Schema:

```
packetheader {
  capacity { INT [ sum(//kind/count) ≤ . ] } &
  kind ∗ {
    count { INT [ . > 0 ] }
  }
}
```

Procedure:

```
add(ident k, int amount) {
  assume amount > 0;
  if not //kind[k] then
    new //kind[k];
    new //kind[k]/count;
    //kind[k]/count := 0
  fi
  //kind[k]/count := //kind[k]/count + amount
}
```

Example

Schema:

packetheader {
  capacity { *INT* [sum(//kind/count) $\leq$.] } &
  kind $*$ {
    count { *INT* [. $>$ 0] }
  }
}

Procedure:

add(**ident** k, **int** amount) {
  assume amount $>$ 0;
  if not //kind[k] **then**
    new //kind[k];
    new //kind[k]/count;
    //kind[k]/count := 0
  fi
  //kind[k]/count := //kind[k]/count + amount
}

Generated Code:

```
// Preconditions:
// - AssumptionException
//     amount > 0
// - CapacityException
//     sum (//kind/count) + amount <= //capacity
Packetheader add(Ident k, Integer amount) { ... }
```

Example

Schema:

```
packetheader {
  capacity { INT [ sum(//kind/count) ≤ . ] } &
  kind * {
    count { INT [ . > 0 ] }
  }
}
```

Generated Code:

```
// Preconditions:
// - AssumptionException
//     amount > 0
// - CapacityException
//     sum (//kind/count) + amount <= //capacity
Packetheader add(Ident k, Integer amount) { ... }
```

Procedure:

```
add(ident k, int amount) {
  assume amount > 0;
  if not //kind[k] then
    new //kind[k];
    new //kind[k]/count;
    //kind[k]/count := 0
  fi
  //kind[k]/count := //kind[k]/count + amount
}
```

Java Code:

```
void pack(List<Ident> items) {
  Packetheader cur = new Packetheader(42);
  for(Ident item : items) {
    try { cur.add(item, 1); }
    catch(CapacityException e) {
      sendPacket(cur);
      cur = new Packetheader(42).add(item, 1);
    }
  }
  sendPacket(cur);
}
```

XML Data as ADT

XML Data as ADT

- Write access only through interface procedures
  - Encapsulate alien aspects of tree manipulation.
  - Analyze procedures to generate preconditions.

XML Data as ADT

* Write access only through interface procedures
  * Encapsulate alien aspects of tree manipulation.
  * Analyze procedures to generate preconditions.
* Domain experts must be able to write both
  * Schemata and
  * Atomic Procedures

XML Data as ADT

* Write access only through interface procedures
  * Encapsulate alien aspects of tree manipulation.
  * Analyze procedures to generate preconditions.
* Domain experts must be able to write both
  * Schemata and
  * Atomic Procedures
* The rest has to be automated:
  * Code generation (Java library)
  * Weakest precondition generation
  * Simplification to minimal incremental check

Path-based Formalization

Path-based Formalization

$$
\begin{array}{rrl}
\text{values } V & ::= & I \mid S \mid Z \mid \text{clx} \mid D(P) \\
\text{identifier } I & ::= & c_I \mid v_I \mid \text{null} \mid \text{cast}_I(V) \\
\text{strings } S & ::= & c_S \mid v_S \mid \text{cast}_S(V) \\
\text{integer } Z & ::= & 0 \mid 1 \mid v_Z \mid Z + Z \mid Z * Z \mid -Z \mid \text{cast}_Z(V) \\
& & \mid \text{sum}(V^*) \mid \text{count}(V^*) \mid \text{rcount}(V, V^*) \\
\\
\text{labels } L & ::= & c_L \\
\text{paths } P & ::= & \text{root} \mid P/L[I] \mid \text{cast}_P(P^*) \\
\text{documents } D & ::= & \text{blank} \mid D[P \to V] \mid D[P \to] \mid \$ \\
\\
\text{value multisets } V^* & ::= & \{V\} \mid V^* \cup V^* \mid D(P^*) \mid \text{all} \mid v_m \\
\text{path multisets } P^* & ::= & \{P\} \mid P^* \cup P^* \mid P^*/L[V^*] \\
& & \mid . \mid P^*/.. \mid P^*/.L \mid P^*//L[V^*] \\
\\
\text{formulas } G & ::= & \forall v_I . G \mid F \\
F & ::= & \text{false} \mid F \wedge F \mid F \vee F \mid \neg F \\
& & \mid \alpha = \alpha \mid Z < Z \mid P \in D \\
\text{types } T & ::= & \mathit{INT} \mid \mathit{ID} \mid \mathit{STR} \mid \mathit{CLX} \mid \text{typeOf}(V)
\end{array}
$$

Path-based Formalization

$$
\begin{aligned}
\text{values } V &::= I \mid S \mid Z \mid \text{clx} \mid D(P) \\
\text{identifier } I &::= c_I \mid v_I \mid \text{null} \mid \text{cast}_I(V) \\
\text{strings } S &::= c_S \mid v_S \mid \text{cast}_S(V) \\
\text{integer } Z &::= 0 \mid 1 \mid v_Z \mid Z + Z \mid Z * Z \mid -Z \mid \text{cast}_Z(V) \\
&\quad \mid \text{sum}(V^*) \mid \text{count}(V^*) \mid \text{rcount}(V, V^*) \\
\\
\text{labels } L &::= c_L \\
\text{paths } P &::= \text{root} \mid P/L[I] \mid \text{cast}_P(P^*) \\
\text{documents } D &::= \text{blank} \mid D[P \to V] \mid D[P \to] \mid \$ \\
\\
\text{value multisets } V^* &::= \{V\} \mid V^* \cup V^* \mid D(P^*) \mid \text{all} \mid v_m \\
\text{path multisets } P^* &::= \{P\} \mid P^* \cup P^* \mid P^*/L[V^*] \\
&\quad \mid . \mid P^*/.. \mid P^*/.L \mid P^*//L[V^*] \\
\\
\text{formulas } G &::= \forall v_I.G \mid F \\
F &::= \text{false} \mid F \wedge F \mid F \vee F \mid \neg F \\
&\quad \mid \alpha = \alpha \mid Z < Z \mid P \in D \\
\text{types } T &::= \mathit{INT} \mid \mathit{ID} \mid \mathit{STR} \mid \mathit{CLX} \mid \text{typeOf}(V)
\end{aligned}
$$

Example Paths:

- /packetheader
- /packetheader/capacity
- /packetheader/kind[x]
- /packetheader/kind[x]/count

Path-based Formalization

$$
\begin{aligned}
\text{values } V &::= I \mid S \mid Z \mid \text{clx} \mid D(P) \\
\text{identifier } I &::= c_I \mid v_I \mid \text{null} \mid \text{cast}_I(V) \\
\text{strings } S &::= c_S \mid v_S \mid \text{cast}_S(V) \\
\text{integer } Z &::= 0 \mid 1 \mid v_Z \mid Z + Z \mid Z * Z \mid -Z \mid \text{cast}_Z(V) \\
&\quad \mid \text{sum}(V^*) \mid \text{count}(V^*) \mid \text{rcount}(V, V^*) \\
\\
\text{labels } L &::= c_L \\
\text{paths } P &::= \text{root} \mid P/L[I] \mid \text{cast}_P(P^*) \\
\text{documents } D &::= \text{blank} \mid D[P \to V] \mid D[P \to] \mid \$ \\
\\
\text{value multisets } V^* &::= \{V\} \mid V^* \cup V^* \mid D(P^*) \mid \text{all} \mid v_m \\
\text{path multisets } P^* &::= \{P\} \mid P^* \cup P^* \mid P^*/L[V^*] \\
&\quad \mid \; . \mid P^*/.. \mid P^*/.L \mid P^*//L[V^*] \\
\\
\text{formulas } G &::= \forall v_I.G \mid F \\
F &::= \text{false} \mid F \wedge F \mid F \vee F \mid \neg F \\
&\quad \mid \alpha = \alpha \mid Z < Z \mid P \in D \\
\text{types } T &::= \mathit{INT} \mid \mathit{ID} \mid \mathit{STR} \mid \mathit{CLX} \mid \text{typeOf}(V)
\end{aligned}
$$

Example Paths:

`/packetheader`
`/packetheader/capacity`
`/packetheader/kind`$[x]$
`/packetheader/kind`$[x]$`/count`

Derived Constraints, e.g.:

$\forall x.$ `/packetheader/kind`$[x]$`/count` $\in \$ \to$
`/packetheader/kind`$[x]$ $\in \$$

$\forall x.$ `/packetheader/kind`$[x]$`/count` $\in \$ \to$
typeOf($\$$(`/packetheader/kind`$[x]$`/count`))
$= \mathit{INT}$

Example

Schema:

```
inventory {
  time { INT [. >= 0] },
  capacity { INT [. > 0] },

  kind * { size { INT [. > 0] [. <= //capacity] }}
  item * {
    since { INT [. >= 0] [. <= //time] },
    kindref { ID [ //kind[.] ] }
  },

  [ ./capacity >= sum (./kind[./item/kindref]/size) ]
}
```

Procedure:

```
changeKind(ident id, ident kind) {
  set //item[id]/kindref kind;
}
```

Preconditions:

```
1) /inventory/item[id]/kindref
2) /inventory/kind[kind]
3) sum (/inventory/kind[/inventory/item*/kindref]/size)
     + /inventory/kind[kind]/size
     - /inventory/kind[/inventory/item[id]/kindref]/size
     <= /inventory/capacity
```

Summary

* Support for XML + integrity constraints in programs.
* XML data as abstract datatype:
  * With an interface of atomic procedures.
  * Automatically derive minimal preconditions.
  * Automatically generated library.
* Domain experts define schemata and procedures.
  * They are able to read and understand the preconditions.
  * They can react to violations of constraints as they happen.