Course: Introduction to XQuery and Static Type-Checking

Pierre Genevès
CNRS

(Some examples are inspired from the XQuery tutorial by Peter Fankhauser and Philip Wadler, the slides by Rajshekhar Sunderraman, and the slides by John Rushby)

University of Grenoble, 2016–2017

XQuery - XML Query Language

- XQuery is a SQL-like functional language...
- ... for transforming XML documents into other XML documents
- Allows transforming documents of a given type into another
- A W3C recommendation (www.w3.org/TR/xquery)
- Built around the XPath language
- Turing-complete
- Strongly typed
- Implementations available!

  open-source and commercial (many startups)

Outline

1. XQuery

2. Static Type-Checking

XQuery by Example

- Titles of all books published before 2000:

```
/books/book[@year<2000]/title
```

- Year and title of all books published before 2000:

```
FOR $book in /books/book
WHERE $book/@year < 2000
RETURN <book>{ $book/@year, $book/title }</book>
```

- Books grouped by author:

```
FOR $author in distinct(/books/book/author)
RETURN <author name="{ $author }">
          { /books/book[author = $author]/title }
       </author>
```

XQuery Basics

- General structure (FLWOR expressions):

```
For $var in expr
Let  $var := expr
Where condition
Order By expr
Return expr
```

- All except Return are optional
- Compared to XPath 1.0 node sets, XQuery introduces a notion of order: node sequences
- For (iteration) vs. Let (value is assigned to variable)
- expr can be an XPath expression or another FLWOR expression
- XQuery can be used for a complete document restructuring

More XQuery Examples

Sample XML


```xml
<Transcripts>
  <Transcript>
      <Student StudId="111111111" Name="John Doe"/>
      <CrsTaken CrsCode="CS308" Semester="F1997" Grade="B"/>
      <CrsTaken CrsCode="MAT123" Semester="F1997" Grade="B"/>
      <CrsTaken CrsCode="EE101" Semester="F1997" Grade="A"/>
      <CrsTaken CrsCode="CS305" Semester="F1995" Grade="A"/>
  </Transcript>
  <Transcript>
      <Student StudId="987654321" Name="Bart Simpson"/>
      <CrsTaken CrsCode="CS305" Semester="F1995" Grade="C"/>
      <CrsTaken CrsCode="CS308" Semester="F1994" Grade="B"/>
  </Transcript>
  <Transcript>
      <Student StudId="123454321" Name="Joe Blow"/>
      <CrsTaken CrsCode="CS315" Semester="S1997" Grade="A"/>
      <CrsTaken CrsCode="CS305" Semester="S1996" Grade="A"/>
      <CrsTaken CrsCode="MAT123" Semester="S1996" Grade="C"/>
  </Transcript>
...
</Transcripts>
```

XQuery Example

```
(:  students who took  MAT123  :)
For  $t  In  doc("http://xyz.edu/transcript.xml")//Transcript
Where    $t/CrsTaken/@CrsCode = "MAT123"
Return  $t/Student
```

- Result:

```
<Student  StudId="111111111"  Name="John Doe" />
<Student  StudId="123454321"  Name="Joe Blow" />
```

XQuery and Well-Formedness

- Previous query does not produce a well-formed XML document; the following does:

```
<StudentList>
  {
    For  $t in  doc("transcript.xml")//Transcript
    Where  $t/CrsTaken/@CrsCode = "MAT123"
    Return  $t/Student
  }
</StudentList>
```

- For binds $t to Transcript elements one by one, filters using Where, then places Student-children as children of StudentList using Return

Sample Document Restructuring with XQuery

- Reconstruct lists of students taking each class from transcript.xml
- classes.xml lists course offerings (course code/semester)

```
For  $c  in doc("classes.xml")//Class
Where doc("transcripts.xml")//CrsTaken[@CrsCode = $c/@CrsCode
                                     and @Semester = $c/@Semester]
Return
<ClassRoster CrsCode=$c/@CrsCode Semester=$c/@Semester>
  {
    For  $t  IN  doc("transcript.xml")//Transcript
    Where $t/CrsTaken[@CrsCode = $c/@CrsCode  and
                      @Semester = $c/@Semester]
    Return $t/Student
    Order By $t/Student/@StudId
  }
</ClassRoster>
Order By  $c/@CrsCode
```

Aggregation Example

- Produce a list of students along with the number of courses each student took:

```
For  $t  in  fn:doc("transcripts.xml")//Transcript,
     $s  IN  $t/Student
Let   $c :=  $t/CrsTaken
RETURN
  <StudentSummary
       StudId = $s/@StudId
       Name = $s/@Name
       TotalCourses = fn:count(fn:distinct-values($c)) />
Order By  StudentSummary/@TotalCourses
```

- The grouping effect is achieved because $c is bound to a new set of nodes for each binding of $t

XQuery and Validation

Validating the output

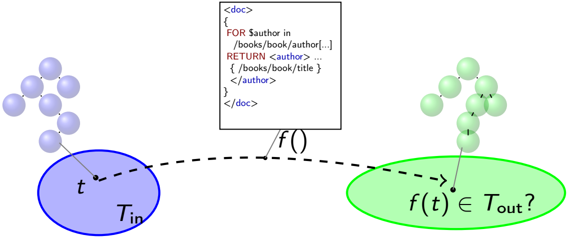
- We can validate the output dynamically for runtime error detection
- Too late for some applications
- Can we detect errors at compile-time?

XQuery is strongly typed

- XQuery is equipped with a sound static type system
- Static type checking is possible to some extent

The Static Type-Checking Problem For XML

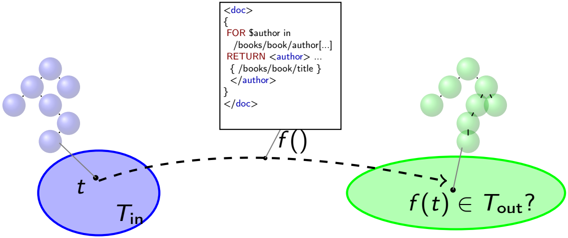
- Given:
  - Source code of some program f (in e.g. XQuery)
  - A type T_{in} for input documents
  - A type T_{out} for expected output documents
- Problem: Does $f(t) \in T_{\text{out}}$ for all $t \in T_{\text{in}}$?



- Crucial problem for avoiding runtime errors, verifying systems of producers/consumers
- Static detection of runtime errors

The Static Type-Checking Problem For XML

- Given:
  - Source code of some program f (in e.g. XQuery)
  - A type T_{in} for input documents
  - A type T_{out} for expected output documents
- Problem: Does $f(t) \in T_{\text{out}}$ for all $t \in T_{\text{in}}$?



- Crucial problem for avoiding runtime errors, verifying systems of producers/consumers
- Static detection of runtime errors $\rightarrow$ impossible in general!

Fundamental Limits: Recall Computability Theory

Rice's Theorem, 1953

Any property which is non-trivial[1] concerning the semantics of a turing-complete programming language is undecidable[2].

(1) non-trivial: neither always true nor always false

(2) Paraphrasing: there is no algorithm that decides a non-trivial property on the program source code, as this would amount to solving Turing's halting problem.

What are the consequences?

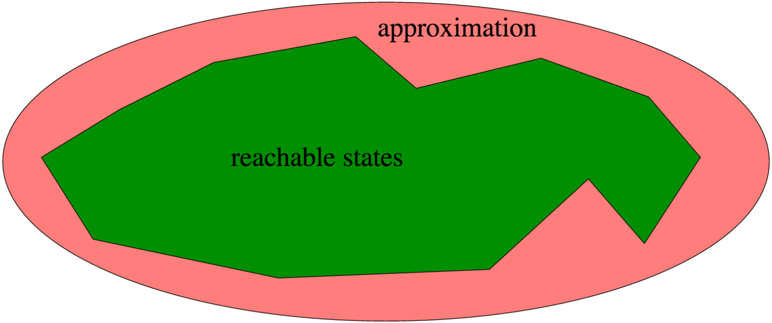
We have to approximate, but there's a price

Rice's Theorem says there are inherent limits on what can be accomplished by automated analysis of programs

- Sound (miss no errors)
- Complete (no false alarms)
- Automatic
- Allow arbitrary (unbounded) memory structures
- Termination (final results)

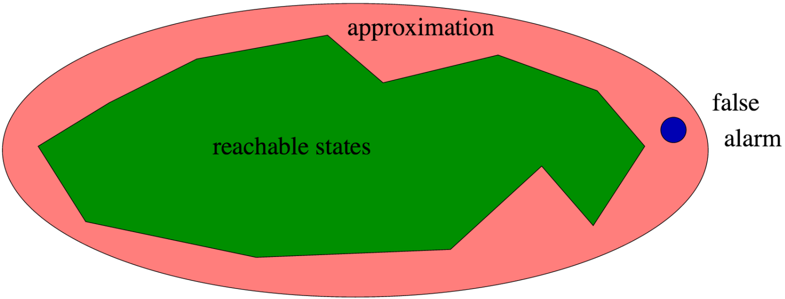
Choose at most 4 of the 5

Approximations



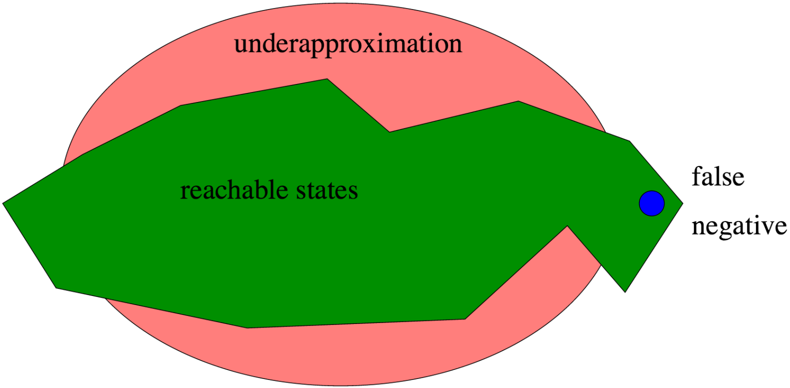
Sound approximations include all the behaviors and reachable states of the real program, but are easier to compute

But Sound Approximations Come with a Price



May flag an error that is unreachable in the real program: a false positive, or false alarm

Unsound Approximations Come with a Price, Too



Can miss real errors: a false negative

Required Choice for XQuery

- We drop one of the following:
  - Sound (miss no errors)
  - Complete (no false alarms)
  - Automatic
  - Allow arbitrary (unbounded) memory structures
  - Termination (final results)

Required Choice for XQuery

- We drop one of the following:
  - Sound (miss no errors)
  - ~~Complete (no false alarms)~~
  - Automatic
  - Allow arbitrary (unbounded) memory structures
  - Termination (final results)
- This means we can theoretically look for method satisfying all other requirements!

Choice Recap

Soundness

- For a given class of errors, a *sound* static analysis method detects errors in a exhaustive manner. (never wrongly considers that a program is safe when actually it is not)
- Soundness provides a "payoff": by soundly checking a web app written in XQuery, one can be **certain** that **certain** errors will **certainly** not happen.
- Short of this, we just have a bug-finder (might be useful, but does not constitute a sufficient basis for establishing guarantees)

Incompleteness

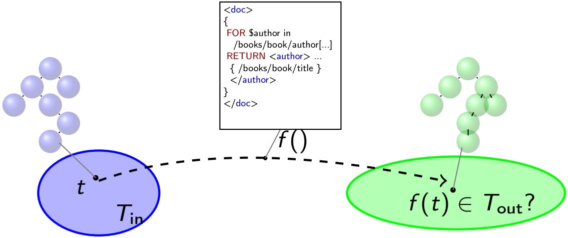
- False alarms

Scientific Challenge

- Reducing the number of false alarms by increasing the precision of the analysis while keeping time and memory costs low

Zoom on Static Type-Checking For XQuery

- Given:
  - Source code of some program f (in e.g. XQuery)
  - A type T_{in} for input documents
  - A type T_{out} for expected output documents
- Problem: Does $f(t) \in T_{out}$ for all $t \in T_{in}$?



- XQuery comes with a sound static type system for a core fragment (optionally implemented)
- There's plenty of promising research technology around

→ So far, we must know more about types for XML...

References

XQuery

- Many tutorials online
- Books
- Standard specifications available from the W3C:
  - XQuery 1.0: An XML Query Language, W3C Recommendation 14 December 2010 (revised 7 September 2015). `http://www.w3.org/TR/xquery/`
  - XQuery 1.0 and XPath 2.0 Formal Semantics, W3C Recommendation 14 December 2010 (revised 7 September 2015). `http://www.w3.org/TR/xquery-semantics/`

XQuery and Static Typing

[1] XQuery and Static Typing: Tackling the Problem of Backward Axes.
Pierre Genevès and Nils Gesbert.
In ICFP'15: Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP), Sept. 2015.
http://tyrex.inria.fr/publications/icfp15.pdf

Analyses: [2,3], Semantics: [4], Transformation and Core XQuery: [5]

[2] Eliminating dead-code from XQuery programs. Pierre Genevès and Nabil Layaïda. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE) - Volume 2, pages 305-306, 2010.

[3] Impact of XML schema evolution. Pierre Genevès, Nabil Layaïda and Vincent Quint. ACM Transactions on Internet Technology (TOIT), volume 11, number 1.

[4] XPath Formal Semantics and Beyond: a Coq based approach. Pierre Genevès and Jean-Yves Vion-Dury. Emerging Trends Proceedings of the 17th International Conference on Theorem Proving in Higher Order Logic: TPHOLs 2004, pages 181-198.

[5] Compiling XPath for streaming access policy. Pierre Genevès and Kristoffer Rose. Proceedings of the 2005 ACM Symposium on Document Engineering, pages 52-54, 2005.