XML TYPECHECKING

Véronique Benzaken
Université Paris 11
http://www.lri.fr/~benzaken

Giuseppe Castagna
C.N.R.S. and Université Paris 7
http://www.pps.jussieu.fr/~gc

Haruo Hosoya
The University of Tokyo
http://arbre.is.s.u-tokyo.ac.jp/~hahosoya

Benjamin C. Pierce
University of Pennsylvania
http://www.cis.upenn.edu/~bcpierce

Stijn Vansummeren
Hasselt University and transnational University of Limburg
http://alpha.uhasselt.be/~lucg5855/

SYNONYMS

None

DEFINITION

In general, *typechecking* refers to the problem where, given a program P, an input type σ, and an output type τ, one must decide whether P is *type-safe*, that is, whether it produces only outputs of type τ when run on inputs of type σ. In the XML context, typechecking problems mainly arise in two forms:

- *XML-to-XML transformations,* where P transforms XML documents conforming to a given type into XML documents conforming to another given type; and
- *XML publishing,* where P transforms relational databases into XML views of these databases and it is necessary to check that all generated views conform to a specified type.

A *type* for XML documents is typically a regular tree language, usually expressed as a schema written in a schema language such as DTD, XML Schema, or a Relax NG (see XML Types). In the XML publishing case, the input type σ is a relational database schema, possibly with integrity constraints.

Typechecking problems may or may not be decidable, depending on (1) the class of programs considered, (2) the class of input types (relational schemas, DTDs, XML Schemas, Relax NG schema, or perhaps other subclasses of the regular tree languages), and (3) the class of output types. In cases where it is decidable, typechecking can be done *exactly*. In cases where it is undecidable, one must revert to *approximate* or *incomplete* typecheckers that may return false negatives—i.e., may reject a program even if it is type-safe. Even when exact typechecking is possible, approximate typechecking may be preferable as this is often computationally cheaper than exact typechecking.

In the programming languages literature, typechecking often not only entails verifying that all outputs are of type

τ, but also requires detecting when the program may abort with a run-time error on inputs of type σ [35]. The above definition encompasses such cases: view run-time errors as a special result value error and then typecheck a program against an output type that does not contain the value error.

HISTORICAL BACKGROUND

Although typechecking is a fundamental and well-studied problem in the theory of programming languages [35], the types necessary for XML typechecking (based on regular tree languages) differ significantly from the conventional data types usually considered (i.e., lists, records, classes, and so on). Indeed, although it is possible to encode XML types into conventional datatypes, this encoding lacks flexibility in the sense that programs tend to need artificial changes when types evolve [22]. For this reason, Hosoya et al. [22] proposed regular tree languages as the "right" notion of types for XML and presented an approximate typechecker in this context. The typechecker was implemented in the XML-to-XML transformation language XDuce [21] whose approach was later extended to general purpose programming by CDuce (functional programming) and Xtatic (object-oriented programming). XDuce's approach also lies at the basis of XQuery's typechecking algorithm [6].

The contemporary study of exact typechecking for XML-to-XML transformations started with an investigation of relatively simple transformation languages [29, 32, 33]. Ironically enough, the fundamentals of exact typechecking for more advanced transformation languages were already investigated a long time before XML appeared [7, 8]. These fundamentals were revived in the XML era by Milo, Suciu, and Vianu in their seminal work on k-pebble tree transducers [30], which was later extended to other transformation languages [39, 26, 40]. The computational complexity of exact typechecking was investigated in [28, 27, 14]. Exact typechecking algorithms for XML publishing scenarios were given by Alon et al. [1].

SCIENTIFIC FUNDAMENTALS

Exact Typechecking

XML-to-XML Transformations. Recall that in this setting, P is a program that should transform XML documents of a type σ into documents of a type τ. When the languages in which the transformation and the types are expressed are sufficiently restricted in power, exact typechecking is possible. There are two major approaches to the construction of an exact typechecking algorithm: *forward inference* and *backward inference*. Forward inference solves the typechecking problem directly by first computing the image O of the input type σ under the transformation P, i.e., $O := \{P(t) \mid t \in \sigma\}$, and then checking $O \subseteq \tau$ [30, 32, 33, 28, 36]. This approach does not work if O goes beyond *context-free tree languages* as checking $O \subseteq \tau$ then becomes undecidable. Sadly, this is already the case when P is written in very simple transformation languages, such as the *top-down tree transducers* (this fact is known as folklore; see, e.g., [14].) Also, computing O itself becomes undecidable for more advanced transformation languages.

Backward inference, on the other hand, first computes the pre-image I of the output type τ under P, i.e., $I := \{t \mid P(t) \in \tau\}$, and then checks $\sigma \subseteq I$. Backward inference often works even when the transformation language is too expressive for forward inference. The technique has successfully been applied to a range of formal models of real-world transformation languages like XSLT, from the top-down and bottom-up tree transducers [7], to macro tree transducers [8, 27, 14], macro forest transducers [34], k-pebble tree transducers [30], tree transducers based on alternating tree automata [39], tree transducers dealing with atomic data values [36], and high-level tree transducers [40].

As mentioned in the definition, static detection of run-time errors can be phrased as a particular form of typechecking by introducing a special output value error. Exact typechecking in this form has been investigated for XQuery programs written in the non-recursive for-let-where return fragment of XQuery without automatic coercions but with the various XPath axes; node constructors; value and node comparisons; and node label and content inspections, in the setting where the input type σ is given by a recursion-free regular tree language. The crux of decidability here is a small-model property: if $P(t) = \mathsf{error}$ for some t of type σ then there exists another input t' of type σ whose size depends only on P and σ such that $P(t') = \mathsf{error}$. It then suffices to enumerate all inputs $t' \in \sigma$ up to the maximum size and check $P(t') = \mathsf{error}$. There are only a finite number of such t' (up to

isomorphism, and P cannot distinguish between isomorphic inputs), from which decidability follows [41]. This small model property continues to hold when we extend the above XQuery fragment with arbitrary primitives satisfying some general niceness properties; see Vansummeren [41].

XML Publishing. In this setting, the input to the program P is a relational database D. Suppose that P computes its output XML tree by posing simple select-project-join queries to D, nesting the results to these queries, and constructing new XML elements. Exact typechecking for programs of this form, when the input type σ is a relational database schema with key and foreign key constraints, and the output type τ is a "star-free" DTD, is decidable [1]. (See [1] for a precise definition and examples of the concept "star-free".) As was the case for detecting runtime errors in XQuery programs, the crux of decidability here is again a small model property. Typechecking remains decidable for output DTDs τ that are not star-free, but then the queries in P must not use projection. Typechecking unfortunately becomes undecidable when the output types τ are given by XML Schemas or Relax NG Schemas [1]. Typechecking also becomes undecidable when P uses queries more expressive than select-project-join queries.

Approximate Typechecking

The expressive power that realistic applications require of practical transformation languages is often too high to allow for exact typechecking. In such cases, one must revert to approximate or incomplete typecheckers that guarantee that all succesfully checked programs are type-safe, but that may also reject some type-safe programs. Existing techniques can be grouped into two categories: type systems and flow analyses.

Type systems Many conventional programming languages (such as C and Java) specify what programs to accept by a *type system* [35]. Typically, such a system consists of a set of *typing rules* that determine the type of each subexpression of a program. Often, in order to help the typechecker, the programmer is required to supply *type annotations* on variable declarations and in other specified places.

The pioneer work applying this approach to the XML setting was the XDuce ("transduce") language [21], whose type system is based on regular tree languages. One significant point in this work is its definition of a natural notion of *subtyping* as the inclusion relation between regular tree languages and its demonstration of the usefulness of allowing a value of one type to be viewed as another type with a syntaticaly completely different structure [22]. In addition, although the decision problem for subtyping is known to be EXPTIME-complete, the "top-down algorithm" used in the XDuce implementation is empirically shown to be efficient in most cases that actually arise in typechecking [22, 37, 13]. Such a type system also needs machinery to reduce the amount of type annotations that otherwise tends to be a burden to the user, in particular when the language supports a non-trivial mechanism to manipulate XML documents such as regular expression patterns [20] or filter expressions [17]. A series of works address this problem by proposing automatic type inference schemes that have certain precision properties in a sense similar to the exact typechecking in the previous section [20, 17, 43]. These ideas have further been extended for XML attributes [19] and parameteric polymorphism [18, 42].

CDuce (pronounced "seduce") extends XDuce in various ways [2]. From a language point of view CDuce embraces XDuce's approach of a functional language based on regular expression patterns [20] and extends it with finer-grained pattern matching, complete two-way compatibility with programs and libraries in the OCaml programming language, Unicode, queries, XML Schema validation, and, above all, higher-order and overloaded functions. XML types are enriched with general purpose data types, intersection and negation types, and functional types. Finally, the CDuce type inference algorithm for patterns is implemented by a new kind of tree automaton and proved to be optimal [10]. Among these extensions he addition of higher-order functions is significant. Theoretically, this extension is not trivial first because functions do not fit well in the framework of finite tree automata, but more deeply because this entails a definitional cycle: the definition of typechecking uses subtyping, whose definition then uses the semantics of types (NB: subtyping is defined as inclusion between the sets denoted by given two types), whose definition in turn uses well-typedness of values; the last part depends on typechecking in the presence of higher-order functions since typechecking of a function abstraction $\lambda x.e$ requires analysis of its internal expression e. Some solutions are known for breaking this circularity [12, 42]. Also, an approach to combine one of these treatments with high-order functions has been proposed [42].

Several research groups explore ways of mixing a XDuce-like type system with an existing popular language.

Xtatic [15] carries out this program for the C# language, developing techniques to blend regular expression types with an object-oriented type system. XJ [16] makes a closely related effort for Java. OCamlDuce [11] mixes with OCaml, proposing a method to intermingle a standard ML type inference algorithm with XDuce-like typechecking. XHaskell [25] is also another instance for Haskell; their approach is, however, to embed XML types into Haskell typing structures (such as tuples and disjoint sums) in the style of data-binding, yet support XDuce-like subtyping in its full flexibility by deploying a coercion technique [38].
The formal semantics of XQuery defined by the W3C contains a type system based on a set of inductive typing rules [6]. Their first draft was heavily based on XDuce's type system [9]. Later, they switched to a different one that reflects the object-oriented hierarchical typing structure adopted by XML Schema.
As byproducts of the above pieces of work, several optimization and compilation techniques that exploit typing information have been proposed [10, 24].

Flow-analysis Flow analysis is a static analysis technique that has long been studied in the programming language community. A series of investigations has been conducted for adapting flow analysis to approximate XML typechecking, concurrently to XDuce-related work, [3, 23, 31]. In this approach, the user needs to write no type annotations for intermediate values like in XDuce, but instead the static anlyzer completely infers them, thus providing a more user-friendly system. One potential drawback is that the specification is rather informal and therefore, when the analyzer raises an error, the reason can sometimes be unclear; empirically, however, such false negatives are rare.
Flow analysis is applied first to Bigwig language system, an extension of Java with an XML-manipulating facility called "templates" [3]. Though this first attempt handles only XHTML types, they naturally generalize it to arbitrary XML types, calling the resulting system XAct [23]. Their techniques are further extended and applied to static analysis of XSLT [31].

KEY APPLICATIONS

XML typechecking is a key component of XQuery, the standard XML query language. As outlined above, XML typechecking in XQuery is based on a set of inductive typing rules that reflects the object-oriented hierarchical typing structure adopted by XML Schema. Different approaches to XML typechecking may be found in research prototypes like CDuce, OcamlDuce, XDuce, XAct, XHaskell, and Xtatic for which references may be found below.

URL TO CODE

CDuce: `http://www.cduce.org`

OCamlDuce: `http://www.cduce.org/ocaml.html`

XAct: `http://www.brics.dk/Xact/`

Xduce: `http://xduce.sourceforge.net`

XHaskell: `http://taichi.ddns.comp.nus.edu.sg/taichiwiki/XhaskellHomePage`

Xtatic: `http://www.cis.upenn.edu/~bcpierce/xtatic`

CROSS REFERENCE

XML Types, Relational Database Schema, Integrity Constraints

RECOMMENDED READING

A gentle introduction to exact typechecking for both XML-to-XML transformations and XML publishing can be found in [36]. A non-technical presentation of Regular Expression Types and Patterns and their use in query languages can be found in the joint DBPL and XSym 2005 invited talk [4]. For a more complete presentation of Regular Expression Types and Patterns and the associated type-checking and subtyping algorithms we

recommend the reader to refer to the seminal JFP article by Hosoya, Pierce, and Vouillon [22]. The joint ICALP and PPDP 2005 keynote talk [5] constitutes a relatively simple survey of the problem of type-checking higher-order functions and an overview on how to derive subtyping algorithms semantically: full technical details can be found in an extended version published in the JACM [13].

[1] Noga Alon, Tova Milo, Frank Neven, Dan Suciu, and Victor Vianu. Typechecking xml views of relational databases. *ACM Trans. Comput. Log.*, 4(3):315–354, 2003.

[2] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: An XML-centric general-purpose language. In *Proceedings of the International Conference on Functional Programming (ICFP)*, pages 51–63, 2003.

[3] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. The `<bigwig>` project. *ACM Transactions on Internet Technology (TOIT)*, 2002.

[4] G. Castagna. Patterns and types for querying XML. In Proceedings of *DBPL 2005, 10th International Symposium on Database Programming Languages* Lecture Notes in Computer Science, n.3774, pages 1-26 Springer (full version) *and XSym 2005, 3rd International XML Database Symposium* Lecture Notes in Computer Science n.3671, pages 1-3, Springer (summary), 2005. Joint DBPL-XSym invited talk.

[5] G. Castagna and A. Frisch. A gentle introduction to semantic subtyping. In Proceedings of *PPDP '05, the 7th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming,* pages 198-208, ACM Press (full version) and *ICALP '05, 32nd International Colloquium on Automata, Languages and Programming,* Lecture Notes in Computer Science n. 3580, pages 30-34, Springer (summary), Lisboa, Portugal, July 2005. Joint ICALP-PPDP keynote talk.

[6] Denise Draper, Peter Fankhauser, Mary Fernández Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics, 2007. `http://www.w3.org/Tr/query-semantics/`.

[7] Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[8] Joost Engelfriet and Heiko Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):710–146, 1985.

[9] Mary F. Fernández, Jérôme Siméon, and Philip Wadler. A semi-monad for semi-structured data. In Jan Van den Bussche and Victor Vianu, editors, *Proceedings of 8th International Conference on Database Theory (ICDT 2001)*, volume 1973 of *lecture Notes in Computer Science*, pages 263–300. Springer, 2001.

[10] Alain Frisch. Regular tree language recognition with static information. In *Proc. IFIP Conference on on Theoretical Computer Science*, pages 661–674. Kluwer, 2004.

[11] Alain Frisch. OCaml+CDuce. In *ICFP '06, International Conference on Functional Programming*, pages 192–200. ACM Press, 2006.

[12] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping. In *Seventeenth Annual IEEE Symposium on logic In Computer Science*, pages 137–146, 2002.

[13] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *The Journal of ACM*, 2008. To appear.

[14] Alain Frisch and Haruo Hosoya. Towards practical typechecking for macro tree transducers. In *11th International Symposium on Database Programming languages (DBPL 2007)*, 2007.

[15] Vladimir Gapeyev, Michael Y. Levin, Benjamin C. Pierce, and Alan Schmitt. The Xtatic experience. In *Workshop on Programming language Technologies for XML (PLAN-X)*, January 2005. university of Pennsylvania Technical report MS-CIS-04-24, Oct 2004.

[16] Matthew Harren, Mukund Raghavachari, Oded Shmueli, Michael G. Burke, Rajesh Bordawekar, Igor Pechtchanski, and Vivek Sarkar. XJ: Facilitating XML processing in Java. In *14th International Conference on World Wide Web (WWW2005)*, pages 278–287. ACM Press, May 2005.

[17] Haruo Hosoya. Regular expression filters for XML. *Journal of Functional Programming*, 16(6):711–750, 2006. Short version appeared in Proceedings of Programming Technologies for XML (PLAN-X), pp.13–27, 2004.

[18] Haruo Hosoya, Alain Frisch, and Giuseppe Castagna. Parametric polymorphism for XML. In *The 32nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages*, pages 50–62, 2005.

[19] Haruo Hosoya and Makoto Murata. Boolean operations and inclusion test for attribute-element constraints. *Theoretical Computer Science*, 360(1-3):327–351, 2006.

[20] Haruo Hosoya and Benjamin C. Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 13(6):961–1004, 2002. Short version appeared in Proceedings of The 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages, pp. 67–80, 2001.

[21] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003. Short version appeared in Proceedings of Third International Workshop on the Web and Databases (WebDB2000), volume 1997 of lecture Notes in Computer Science, pp. 226–244, Springer-Verlag.

[22] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular expression types for XML. *ACM Transactions on*

Programming Languages and Systems, 27(1):46–90, 2004. Short version appeared in Proceedings of the International Conference on Functional Programming (ICFP), pp.11-22, 2000.

[23] Christian Kirkegaard and Anders Møller. Xact - XML transformations in Java. In *Programming language Technologies for XML (PLAN-X)*, page 87, 2006.

[24] Michael Y. Levin and Benjamin C. Pierce. Type-based optimization for regular patterns. In *Database Programming languages (DBPL)*, August 2005.

[25] Kenny Zhuo Ming Lu and Martin Sulzmann. XHaskell: regular expression types for Haskell. Manuscript, 2004.

[26] Sebastian Maneth, Thomas Perst, Alexandru Berlea, and Helmut Seidl. XML type checking with macro tree transducers. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 283–294, 2005.

[27] Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact XML type checking in polynomial time. In *International Conference on Database Theory (ICDT)*, pages 254–268, 2007.

[28] Wim Martens and Frank Neven. Frontiers of tractability for typechecking simple xml transformations. *J. Comput. Syst. Sci.*, 73(3):362–390, 2007.

[29] Tova Milo and Dan Suciu. Type inference for queries on semistructured data. In *Proceedings of Symposium on Principles of Database Systems*, pages 215–226, Philadelphia, May 1999.

[30] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003.

[31] Anders Møller, Mads Østerby Olesen, and Michael I. Schwartzbach. Static validation of XSL Transformations. Technical Report rS-05-32, BrICS, October 2005. Draft, accepted for TOPLAS.

[32] Makoto Murata. Transformation of documents and schemas by patterns and contextual conditions. In *Principles of Document Processing '96*, volume 1293 of *lecture Notes in Computer Science*, pages 153–169. Springer-Verlag, 1997.

[33] Yannis Papakonstantinou and Victor Vianu. DTD Inference for Views of XML Data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 35–46, Dallas, Texas, May 2000.

[34] Thomas Perst and Helmut Seidl. Macro forest transducers. *Information Processing Letters*, 89(3):141–149, 2004.

[35] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[36] Dan Suciu. The XML typechecking problem. *SIGMOD Record*, 31(1):89–96, 2002.

[37] Tadahiro Suda and Haruo Hosoya. Non-backtracking top-down algorithm for checking tree automata containment. In *Proceedings of Conference on Implementation and Applications of Automata (CIAA)*, pages 83–92, 2005.

[38] Martin Sulzmann and Kenny Zhuo Ming lu. A type-safe embedding of XDuce into ML. *Electric Notes in Theoretical Computer Science*, 148(2):239–264, 2006.

[39] Akihiko Tozawa. Towards static type checking for XSLT. In *Proceedings of ACM Symposium on Document Engineering*, 2001.

[40] Akihiko Tozawa. XML type checking using high-level tree transducer. In *Functional and Logic Programming (FLOPS)*, pages 81–96, 2006.

[41] Stijn Vansummeren. On deciding well-definedness for query languages on trees. *J. ACM*, 54(4):19, 2007.

[42] Jerome Vouillon. Polymorphic regular tree types and patterns. In *ACM Symposium on Principles of Programming languages (POPL)*, pages 103–114, 2006.

[43] Jerome Vouillon. Polymorphism and XDuce-style patterns. In *Programming languages Technologies for XML (PLAN-X)*, pages 49–60, 2006.