Chapter IX
Analysis of Service Compatibility: Complexity and Computation

Ken Q. Pu
University of Ontario Institute of Technology, Canada

ABSTRACT

In this chapter, the authors apply type-theoretic techniques to the service description and composition verification. A flexible type system is introduced for modeling instances and mappings of semi-structured data, and is demonstrated to be effective in modeling a wide range of data services, ranging from relational database queries to web services for XML. Type-theoretic analysis and verification are then reduced to the problem of type unification. Some (in)tractability results of the unification problem and the expressiveness of their proposed type system are presented in this chapter. Finally, the authors construct a complete unification algorithm which runs in \( \text{EXP-TIME} \) in the worst case, but runs in polynomial time for a large family of unification problems rising from practical type analysis of service compositions.

INTRODUCTION

Service engineering has been of great interest for both researchers as well as practitioners. As part of the Web 2.0 movement of reforming the World Wide Web, service-oriented computing is a fundamental platform to support novel features such as web services (e.g. Curbera et al. (2005)), messaging services (e.g. Maheshwari et al. (2004)), multimedia distribution (e.g. Chawathe (2003),) and dynamic content management and database integration (e.g. Deutsch et al. (2004)). A common approach to the problem of service description and service integration is through functional modeling of the underlying data model and the input/output characteristics of the available services, as discussed by Narayanan & McIlraith (2002); Pu et al. (2006). For the purpose of verification and au-
tomatically discovering relevant and composable services, it becomes necessary to reason about the services both semantically and syntactically.

In this chapter, we present a type-theoretic approach to model the type information of data instances and service functions using a flexible type system. Unlike existing description frameworks such as description logic presented by Baader et al. (2003), and RDF defined by RDF Core Working Group, our proposed type system is used to express structural information of the data instances and the service functions. By extending existing nested record type systems from the programming language community, our type system can be used to for a wide range of data models:

- Semi-structured documents such as XML,
- Structured data such as relational and multidimensional data warehouses.

Furthermore, with the help of a rich set of type variables, we are able to express a wide range of data services:

- Web services
- Query services such as queries in SQL
- Analytical services such as aggregation queries

It is important to incorporate traditional databases and their query languages, such as legacy relational database management systems (RDBMS), in the SOA framework because today’s services still rely heavily on relational back-ends. Since our proposed type system is expressive enough to describe structured data and structured query constructs, it can be used to reason about the compositibility between services and relational data sources.

A type system for service-oriented architectures (SOA) brings exciting possibilities of applying static analysis techniques from programming languages. For instance, type checking is the verification of input/output compatibility between composed services before runtime. Type checking is now a standard feature of all modern compilers. However, in order to apply type checking to compositions of web services, database queries and other services, one must first have a type system capable of expressing the type information of all the service providers. Another well-known type analysis is type inference. Services can often handle data of different structures – this is known as polymorphism. When using polymorphic services, one is required to instantiate the polymorphic input type signature to the actual input data. Type inference is a static procedure which automatically infers the necessary instantiation so that the resulting service invocation is free of type errors. As we will demonstrate, type inference can be applied to the problem of service discovery and automatic service composition. Of course, due to syntactic and semantic ambiguity, service composition cannot be inferred uniquely, however the type inference algorithm, as we will demonstrate in this chapter, offers the following salient features:

- Assist the user with top admissible compositions – a feature similar to method completion in modern integrated development environment (IDE).
- Verify the correctness of existing interoperability of multiple services and backend data sources.
- Complete partially formulated compositions. Services typically have complex input/output structures. User can assist the composition inference process by partially specifying the correspondence between the input and output types, and the type inference algorithm can complete the rest whenever it is possible.

In this chapter, we formally present the proposed type system for SOA, and demonstrate how it can be used to modeling various services. We also present a number of theoretical results on the expressiveness of our type system, and the com-