Partial Deduction for Assisting Automated Semantic Web Service Composition

Peep Küngas
Norwegian University of Science and Technology
Department of Computer and Information Science
Trondheim, Norway
peep@idi.ntnu.no

Mihhail Matskin
Royal Institute of Technology
IMIT/LECS
Kista, Sweden
misha@imit.kth.se

Introduction

Industry has recently recognised the need for additional descriptions of Web services beyond those provided by WSDL documents and tModels in UDDI. These descriptions should facilitate discovery, integration and composition of Web services in a more efficient way than supported currently. This need sheds light to the progress on Semantic Web services as provided by initiatives like WSMO and OWL-S.

By now many methods have been proposed for composing Web services automatically from existing OWL-S and WSML-like Web service descriptions. The methods range from AI planning (Wu *et al.* 2003; McDermott 2002) to automated theorem proving (McIlraith & Son 2002; Rao, Küngas, & Matskin 2005) and graph search algorithms. However, the usability of these methods is greatly affected by two constraints. Firstly, it is assumed that developers provide consistent declarative descriptions of Web services. Secondly, it is assumed that there exists a universal set of atomic Web services, which would facilitate the composition of all other Web services. In this paper we are going to tackle these two issues. We apply partial deduction for identifying potential inconsistencies in Web service descriptions. Our method also determines atomic Web services, which should be implemented in order to compose a required composite Web service.

We have earlier demonstrated (Rao, Küngas, & Matskin 2005) how linear logic (Girard 1987) (LL) theorem proving can be applied for automated Web service synthesis. At the same time we proposed a formalism (Küngas & Matskin 2005) of partial deduction (PD) for LL. While the former article (Rao, Küngas, & Matskin 2005) provides a process for automated Web service composition, the latter article (Küngas & Matskin 2005) provides a formalism for facilitating interactive composition. In this article we combine these results and propose heuristics for enhancing the Web service composition process. The heuristics determine how the system should perform if theorem proving would not lead to any composition. We apply LL theorem proving and PD for planning as proposed in (Küngas 2003).

The generic Web services composition process is presented in Figure 1. First, a description of existing Web services is translated into extralogical axioms of LL, and the requirements to the composite services are specified in form of a LL sequent to be proven. Then LL theorem proving is applied to determine whether a composition can be found. If no composition is found, PD and gap detection are applied iteratively. While PD generates all possible gaps, gap detection selects most desirable ones according to a particular heuristic.

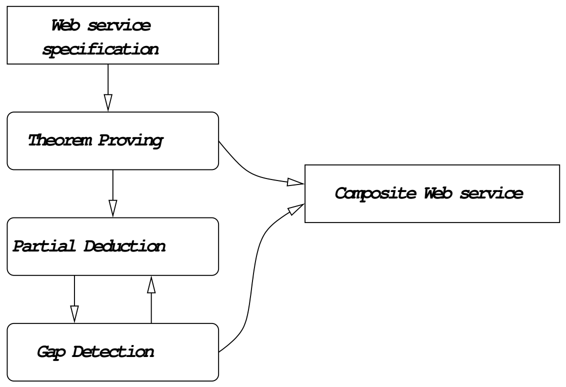

Figure 1: The generic composition process.

Representing Web services in linear logic

LL is a refinement of classical logic introduced by J.-Y. Girard (Girard 1987) to provide a means for keeping track of "resources"—in LL two assumptions of a propositional constant A are distinguished from a single assumption of A. This means that the nonmonotonicity issue is handled implicitly. Although LL is not the first attempt to develop resource-oriented logics (well-known examples are relevance logic and Lambek calculus), it is by now one of the most investigated ones. Since its introduction LL has enjoyed increasing attention both from researchers in proof theory and computer science. Therefore, because of its maturity and well-developed semantic, LL is useful as a declarative language and inference system.

Our fragment of LL consists of multiplicative conjunction $\otimes$ and linear implication $\multimap$. To illustrate Web service presentation in LL let us consider the following example of ski buying service adapted from (Rao, Küngas, & Matskin 2005). The available value-added services are specified as follows:

$$\Gamma_v = \begin{array}{l} \vdash \mathit{PRICE\_LIMIT} \otimes \mathit{SKILL\_LEVEL} \multimap_{selectBrand} \mathit{BRAND} \\ \vdash \mathit{HEIGHT\_CM} \otimes \mathit{WEIGHT\_KG} \multimap_{selectModel} \mathit{LENGTH\_CM} \otimes \mathit{MODEL} \\ \vdash \mathit{LENGTH\_CM} \multimap_{cm2inch} \mathit{LENGTH\_IN} \\ \vdash \mathit{PRICE\_USD} \multimap_{USD2NOK} \mathit{PRICE\_NOK} \end{array}$$

The core service is specified as follows:

$$\Gamma_c = \vdash \mathit{LENGTH\_IN} \otimes \mathit{BRAND} \otimes \mathit{MODEL} \multimap \mathit{PRICE\_USD}$$

The constraints for the composite service are empty, since we would like to keep the example simple:

$$\Delta = \emptyset$$

Finally, the requirements for the composite service are specified as follows:

$$\{\Gamma_v, \Gamma_c\}; \Delta \vdash \mathit{HEIGHT\_CM} \otimes \mathit{WEIGHT\_KG} \otimes \mathit{PRICE\_LIMIT} \otimes \mathit{SKILL\_LEVEL} \multimap \mathit{PRICE\_NOK}$$

The required service can be proven to be correct (and then extracted from the proof) from the specification of available value-added services and the core service. The core service and available value-added services are depicted respectively in Figure 3 and Figure 2. The required service is graphically presented in Figure 4. The structure of a solution for the required service is represented in Figure 5.

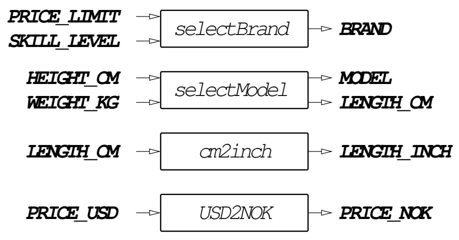

Figure 2: Available value-added services.

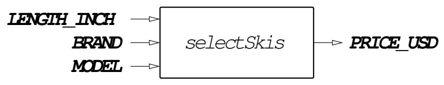

Figure 3: The core service for buying skis.

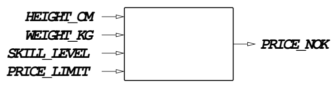

Figure 4: The required service for buying skis.

Partial deduction

Partial deduction (PD) is known as one of optimisation techniques in logic programming. Given a logic program, partial deduction derives a more specific program while preserving the meaning of the original program. Since the program is more specialised, it is usually more efficient than the original program, if executed. For instance, let A, B, C and D be propositional variables and $A \multimap B$, $B \multimap C$ and $C \multimap D$ computability statements in LL. Then possible partial deductions are $A \multimap C$, $B \multimap D$ and $A \multimap D$. It is easy to notice that the first corresponds to forward chaining (from initial states to goals), the second to backward chaining (from goals to initial states) and the third could be either forward or backward chaining.

Although the original motivation behind PD was to deduce specialised logic programs with respect to a given goal, our motivation for PD is a bit different. We are applying PD for determining subproblems, which cannot be solved, but are possibly closer to a solution than an initial task. Similar approach has been applied in (Matskin & Komorowski 1997) for automatic software synthesis. One of the motivations there was debugging of declarative software specification. Here we apply similar technique for debugging and analysing Web services' declarative specifications.

In order to illustrate how to apply PD Web service composition and gap detection, let us consider the above-mentioned example. If we would discard unit conversion services *cm2inch* and *USD2NOK*, there would be no solution available, which would satisfy user's requirements. By discarding these services we assume a situation where these services have not been implemented yet or there are errors in descriptions of these services.

At the same time, by applying PD we would be able to discover these missing Web services. Gaps in a partial solution for a Web service composition task would identify new Web services, which have to be implemented in order to complete the composite Web service. It may be possible also that they represent Semantic Web service description parts, which have to be modified. It could be possible that the developers, who wrote the semantic descriptions of particular Web services, introduced some mistakes into descriptions.

Gap detection heuristics

We propose the following heuristics for gap detection:

1. Construction of all possible combinations of Web services. Any number of instances of a Web service may exist in a solution. Between each two Web services, there may be a gap.
2. Construction of all possible partial solutions through PD. Each solution includes one gap the head and the tail of a partial solution.
3. Select the longest partial solutions from the set of all possible partial solutions constructed through PD. Since a longest partial solutions are most specific ones, they may most precisely describe a desired solution. A developer can always make it shorter or modify further.

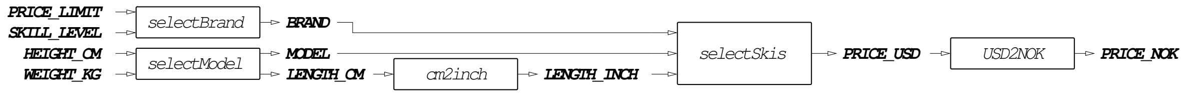

Figure 5: The final service structure for buying skis.

4. Including core Web service(s) to partial solution gaps. A developer can describe Web services, which must be included in solutions. If core services are not included in a partial solution, we try to place them between the head and the tail of the partial solution.

The preceding heuristics can be extended with the following techniques:

- Measure the lexical similarity between input/output names. Since a developer might have made a mistake while describing Web services, just unifying the names might help.
- Use ontologies for deducing subtypes/supertypes and generalise/specialise planning problems on-the-fly.

The preceding techniques would significantly help to debug declarative specifications of Web services and extend the usability of the proposed strategies. Consider for instance a case where a user wants to compose a Web service, which returns *temperature* at a particular location. However, through PD we find a solution, which computes *weather*. Fortunately, *temperature* would be a field in a computed *weather* record. Thus after *weather* has been computed, *temperature* would be extracted from it by applying knowledge in an ontology.

Conclusions

In this paper we applied partial deduction for identifying potential inconsistencies in Web service descriptions. The proposed methodology also determines which atomic Web services should be implemented in order to compose a required composite Web service. These two issues, despite of being important for practical systems, have not yet been considered in the literature of automated Web service composition.

We extended our RAPS planner (Küngas 2003) with PD capabilities to support interactive problem solving and gap detection heuristics. The planner applies LL theorem proving and PD for planning. As a result either complete or partial solutions are constructed automatically. Partial solutions are processed further through the proposed gap detection heuristics.

We identified PD with core Web service inclusion as an effective method for advising developers during automated Web service composition. The method tries to extend partial solutions with core Web service descriptions. Through this process additional gaps are identified, which should be implemented and annotated by developers.

As a future work we would like to perform an analysis of some industrial cases for identifying new methods for gap detection and heuristics for filtering out potentially interesting partial solutions. Additionally we would like to incorporate ontologies and ontological reasoning to our tool. Incorporation of ontologies to our tool would make it more appealing for practical usage.

Acknowledgements

This work was partially supported by the Norwegian Research Foundation in the framework of Information and Communication Technology (IKT-2010) program—the ADIS project.

References

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science* 50:1–102.

Küngas, P., and Matskin, M. 2005. Partial deduction for linear logic—the symbolic negotiation perspective. In *Proceedings of the Second International Workshop on Declarative Agent Languages and Technologies (in conjunction with AAMAS 2004), DALT'2004, New York, USA, July 19, 2004*, volume 3476 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Küngas, P. 2003. Linear logic for domain-independent ai planning (extended abstract). In *Proceedings of Doctoral Consortium at 13th International Conference on Automated Planning and Scheduling, ICAPS 2003, Trento, Italy, June 9-13, 2003*, 68–72.

Matskin, M., and Komorowski, J. 1997. Partial structural synthesis of programs. *Fundamenta Informaticae* 30:23–41.

McDermott, D. 2002. Estimated-regression planning for interaction with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling, Toulouse, France, April 23–27, 2002*. AAAI Press.

McIlraith, S., and Son, T. C. 2002. Adapting Golog for composition of Semantic Web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April 22–25, 2002*, 482–493. Morgan Kaufmann.

Rao, J.; Küngas, P.; and Matskin, M. 2005. Composition of semantic web services using linear logic theorem proving. *Information Systems*. In press.

Wu, D.; Parsia, B.; Sirin, E.; Hendler, J.; and Nau, D. 2003. Automating DAML-S Web Services composition using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference, ISWC 2003, Sanibel Island, Florida, USA, October 20–23, 2003*.