Internal behavior reduction for partner synthesis

Niels Lohmann

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Communication is a unique feature of services and allows for reusing services in different compositions. To make a statement about the correctness of a service in isolation, *partner synthesis* is a proven technique. It overapproximates the service's behavior in any possible composition. Unfortunately, the complexity of partner synthesis is an order of magnitude higher than that of classical model checking techniques. This paper approaches this problem by tackling one source of complexity, namely *internal behavior* (also called silent or τ-transitions). By applying rules known from compositional verification, we reduce the internal behavior of a service while preserving its external behavior, viz. its communication protocol.

1 Introduction

Correctness plays an important role in service-oriented systems, as they increasingly realize business processes or other important infrastructures. Because failures of a single service may affect all other participants of the composition, thorough testing or verification is of paramount importance. In previous work [16], we argued that *partner synthesis* is not only an effective means to check the correctness of single services, but can also be used to synthesize communication skeletons, construct operating guidelines, generate test suites, realize interaction models, correct choreographies, configure business processes, or synthesize adapters. Further references can be found in a survey [10]. Thereby, a partner of a service is another service such that their composition is correct (e. g., deadlock-free, sound, or weakly terminating).

Conceptually, a partner is synthesized by first overapproximating the service's behavior in any possible composition and then removing undesired states yielding deadlocks or livelocks. Unfortunately, the complexity of the partner synthesis is exponential in the size of the service; that is, *both* in the number of states and the size of the interface. Even worse, the service's behavior itself can already suffer from the state space explosion problem [13], which makes the overall complexity of partner synthesis devastating.

As partner synthesis focuses on the external behavior of a service (i. e., its communication protocol), the internal behavior is only important when internal decisions are modeled. To this end, *this paper aims at reducing internal behavior of a service while preserving its external behavior.* As a result, we can construct partner services with reduced effort. We therefore sketch partner synthesis in the

next section. As one contribution, we also survey in Sect. 3 different approaches to leverage the complexity to classify our reduction. Section 4 provides the main contribution of this paper: We adjusted several state space reduction rules to reduce internal service behavior. To assess our approach, we implemented it and discuss in Sect. 5 some first experimental results and the impact of the reduction to the partner synthesis, before Sect. 6 concludes the paper.

2 Partner synthesis in a nutshell

We shall briefly sketch the partner synthesis approach of Wolf [16]. Given a service model, a partner cannot observe the service's state at runtime. Hence, the only information a partner can rely on are (1) the service model and its behavior (i. e., its state space), (2) its own actions from the past, and (3) the asynchronous messages it receives from the service or synchronizations with the service.

Consequently, a partner can only make a vague statement on the concrete state of a service, and can only guess a *set* of states (called *knowledge*) the service might be in. For instance, the initial knowledge of a partner consists of all states of the service it can reach without influence of the partner; that is, all states the service can reach by performing internal transitions (also called silent or τ-transitions) or by sending asynchronous messages to the partner. Each action of the partner results in changed knowledge. For instance, sending a message to the service may result in additionally enabled receiving transitions. After building all possible knowledges and removing "bad" knowledges; that is, knowledges which imply unwanted behavior of the composition (such as deadlocks or livelocks), the remaining graph (unless empty) can be used as a partner. The partner synthesis algorithm is implemented in two tools, Fiona [11] and Wendy [9], we shall discuss later.

3 Reduction techniques for partner synthesis

There are several aspects that yield in the high complexity of the partner synthesis approach. For some of theses aspects already exist approaches to leverage the associated complexity.

State space. On source of complexity is the size of the state space of the service. Each knowledge is a subset of the service's states. Since services often employ concurrency, already this state space may be exponentially in the size of the service model. This state explosion can be fought in different fashions, and these state space reduction approaches can be classified as follows.

One idea is to reduce the original model (e. g., the Petri net or WS-BPEL process) before the calculation of the state space. The most prominent example for such an *a priori reduction* are Petri net reduction rules [12]. Applied to service models, these rules already allow to remove some internal behavior. However, experiments with business process models [3] show that their effect does not hardly justifies the required calculation time.

Another idea is not to generate the complete state space, but only a smaller fragment of it. An example for such an *on-the-fly reduction* are partial order techniques, for instance CTL* preserving partial order reduction [4]. First experiments with such a technique implemented in the tool Fiona are promising.

Finally, state space reduction techniques can also be applied *a posteriori*; that is, after the full state space is built, but before the partner synthesis. Such reduction rules [5] were already employed to reduce a characterization of all livelock-freely interaction partners [17], but not for the partner synthesis itself. This shall be the contribution of this paper.

Knowledge. Once the state space is built, the number of knowledges is another source of complexity. Here, we face two problems. First, a lot of "bad" knowledges are generated, but later removed because they contain unwanted behavior. For instance, a service model may contain a deadlock which is only reached after following a certain communication protocol. To avoid such unnecessary calculation, static analysis techniques can be used to preprocess the state space and to avoiding the calculation of "bad" knowledges as early as possible [9].

Second, not all knowledges need to be calculated in case only the *existence* of a partner is relevant. Such a (possibly less permissive) partner is usually much smaller. Weinberg [14] presents several partner reduction rules which turn out to be very effective during the partner synthesis of industrial service models [9].

Representation. Finally, symbolic data structures such as binary decision diagrams [2] may help to represent the state space and the knowledges in a compact manner. Early experiments [6] show that this technique known from model checking is also very effective when applied during partner synthesis.

The presented reduction techniques are modular, although they cannot be arbitrarily mixed. For instance, the tool Fiona [11] generates the state space on the fly and implements partial order reduction techniques, partner reduction rules, and symbolic representation. A reimplementation of Fiona, Wendy [9], generates the complete state space a priori to perform static analysis to avoid the calculation of "bad" knowledges. It also implements partner reduction rules. A case study [9] demonstrates that Wendy clearly outperforms Fiona. As Wendy does not implement any state space reductions, we shall focus on this aspect in the remainder of the paper.

4 Reduction of internal behavior

As mentioned earlier, Petri net reduction techniques are not effective enough to fight the state space reduction. Furthermore, on-the-fly reduction techniques cannot be combined with the powerful preprocessing techniques that avoid the generation of "bad" knowledge. Consequently, we shall investigate how a posteriori reduction techniques can be combined with partner synthesis.

To reduce the complexity of partner synthesis, we follow one idea: *States of the service, that would always appear in the same knowledge, should be merged before*

(a) redundant parallel transitions (b) fusion of internal loops

(c) states linked by internal transition (1) (d) states linked by internal transition (2)

Fig. 1. Reduction rules

the actual partner synthesis. As sketched in Sect. 2, this reduction particularly affects internal transitions, because — by definition — knowledge consists of those states that can be reached without interaction with the partner. Our approach takes the state space of a service model as input and constructs a reduced state space such that both the original and the reduced state space yield the same generated partner.

To achieve this goal, we employed state space reduction rules from Juan et al. [5]. These rules were defined to preserve IOT failure equivalence [5], which is very closely related to the preservation of external behavior we are interested in. Figure 1 depicts four of these rules. Thereby, a dashed arc stands for an arbitrary number of transitions with an arbitrary label.

Redundant parallel transitions. As the environment cannot distinguish which transition was taken, and both transitions reach the same state s_2, one transition can be safely removed, see Fig. 1(a). Note that this rule is not restricted to τ labels, but can also be applied for arbitrary communication transitions.

Fusion of internal loops. The internal transitions between the two states s_1 and s_2 are not observable. When in either state, the other state remains reachable without influence of a partner. Consequently, both states can be merged to a new state s_{12}. The internal loop is replaced by a self-loop, see Fig. 1(b).

States linked by internal transition. s_1 and s_2 are linked by an internal transition. For each outgoing transition of s_1 exists an outgoing transition of s_2 with the same label that reaches the same state. As a result, s_2 does not restrict any behavior compared to s_1 and the states can be safely merged to a new state s_{12}, see Fig. 1(c).

Table 1. Experimental results: effect of reduction to internal behavior

service model	sizes before reduction states	trans.	τ trans.	sizes after reduction states	trans.	τ trans.	reduction time
Deliver goods	4,148	13,832	9,288	150	397	12	3 s
Car analysis	11,381	39,865	27,231	420	1,211	164	64 s
Identity card	14,569	71,332	66,500	25	37	0	108 s
Product order	14,990	50,193	34,159	504	1,458	135	104 s
SMTP protocol	26,667	110,065	80,137	23,381	99,304	70,646	2,101 s
Philosophers	92,206	427,312	113,023	19,683	98,415	0	7,236 s

The rules are defined on the behavior of a service model and may appear very technical when considered in isolation. Nevertheless, the first two rules may allow the application of other rules. As special case of the third rule is when s_1 has no outgoing transitions other than the internal transition, see Fig. 1(d). Only this restricted setting is covered by a Petri net reduction rule [12]. In general, state space reduction rules allow for more reduction, because they can be applied on a simpler model and do not need to take concurrency into account.

Juan et al. [5] present more rules, but we refrain from a discussion of all of them. For instance, several rules deal with initial states. Furthermore, no original rule was aware of final states which are important in the area of services to distinguish desired final states from deadlocks or to detect livelocks.

5 Experimental results: effect to partner synthesis

We implemented the reduction rules described in the prior section as a component of Wendy [9]. It takes a Petri net service model as input, calls LoLA [15] to generate a state space and then iteratively applies the reduction rules until a fixed point is reached; that is, no more rules can be applied. The output is a reduced state space which is then used during the partner synthesis. The integration of the reduction component is still in an early stage of development.

As a proof of concept, we analyzed several WS-BPEL services from a consulting company. Each process consists of around 40 WS-BPEL activities and models communication protocols and business processes of different industrial sectors. We translated the WS-BPEL processes into Petri nets using the compiler BPEL2oWFN implementing a feature-complete Petri net semantics [7]. Furthermore, the "Philosophers" service is an academic example.

Table 1 summarizes the results regarding the reduction: For most industrial models, nearly all internal transitions could be removed and the state space could be reduced dramatically. This is particularly important, because knowledges consist of subsets of these state spaces, so even a small reduction may have an exponential effect. The SMTP protocol shows, however, that the reduction is

Table 2. Experimental results: effect of reduction to partner synthesis

	synthesis without reduction			synthesis with reduction		
service model	knowledges	time	memory	knowledges	time	memory
Deliver goods	1,376	3 s	18 MB	1,376	0 s	3 MB
Car analysis	1,448	75 s	368 MB	1,176	2 s	13 MB
Identity card	1,536	88 s	427 MB	1,536	0 s	2 MB
Product order	57,996	299 s	1,467 MB	53,324	12 s	75 MB
SMTP protocol	13,456	210 s	249 MB	—[1]	—[1]	—[1]
Philosophers	481,646	4,098 s	6,078 MB	19,682	35 s	98 MB

not always effective. One reason might be that we have not implemented all applicable rules of Juan et al. [5] yet.

The partly devastating runtime can be explained by the prototypic status of the implementation. Nevertheless, the runtime of the reduction can be seen as a worthwhile investment, as shown by Table 2. We see that the synthesis times are usually much faster when the reduction is applied. Of course, we also need to take the reduction time into account. Nevertheless, only when analyzing the "Philosopher" model, the additional time does not pay off. Experiences from the implementation of Petri net reduction rules (i. e., parallel execution or index structures) may help to decrease the runtime by an order of magnitude.

More importantly, we can observe a dramatical reduction of around 95 % in the consumed memory. This allows us to synthesize partners for service models using a few megabytes rather than gigabytes.

6 Conclusion

Summary. In this paper, we discussed several sources of complexity of partner synthesis. We identified a large state space and in particular internal transitions as one reason partner synthesis might be intractable for larger service models. To tackle this problem, we presented a reduction technique that aims at reducing the internal behavior of service models. This technique is modular; that is, can be integrated in existing partner synthesis approaches. A prototypic integration into the partner synthesis tool Wendy [9] demonstrated principal effectiveness of the reduction. We observed a dramatic decrease in memory consumption which allowed us to apply partner synthesis to models we could not analyze before. This reduced memory reduction, however, is currently traded by a suboptimal runtime of the reduction.

The approach has another advantage: it is *compositional*. Suppose the state space of the net is too large to be calculated. As this calculation is a prerequisite for the synthesis algorithm, no partner could be computed. The rules, however, allow for a compositional approach. That is, we can (1) divide the net into parts, (2) apply the reduction rules to the state space of each part, and (3) compose

[1] We currently face a software bug when analyzing the reduced SMTP protocol model.

reduced state spaces. The interested reader is referred to [5] for a detailed discussion.

Lessons learnt. In retrospective, the results of this paper seem obvious and the approach straightforward. However, two questions were open in the run-up of this paper: First, little experimental results were published on the practical applicability of the reduction rules from Juan et al. [5] and their effectiveness to real-life service models. Second, the exact effect of the reduction to partner synthesis was unclear. In particular, we did not foresee that reduced internal behavior could have such a positive effect on the memory consumption. Also the fact that already four reduction rules have such an effect was unclear. The "Philosophers" model further showed that much fewer knowledges need to be calculated when synthesizing partners.

We would like to point out that only a prototypic implementation and access to realistic service models allowed us to perform experiments and to answer these questions. Thereby, the modular architecture of the partner synthesis tool Wendy facilitated the integration of the reduction rules to the partner synthesis. These experiences follow the observations we described in a recent survey [10]. Both the tool Wendy [9] and the experimental results are available via the Web site `http://service-technology.org/live` [8].

Future work. In future work, several open issues need to be approached. As already pointed out, we need to improve the efficiency of the rule application. From a conceptual point of view, a combination of the presented approach and partial order reduction techniques would be promising. Partial order reduction aims at avoiding the state space explosion by not enumerating all possible orders of transitions and intermediate states. This usually results in very small and also simpler structured state spaces. This in turn should boost the applicability of the reduction rules. Finally, a look at related rules [1] may allow for further reduction.

Acknowledgments. The author thanks Christian Stahl for his feedback on an earlier version of this paper and for pointing out the compositionality aspect.

References

1. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From public views to private views — correctness-by-design for services. In: WS-FM 2007. pp. 139–153. LNCS 4937, Springer (2008)
2. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Computers C-35(8), 677–691 (1986)
3. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: BPM 2009. pp. 278–293. LNCS 5701, Springer (2009)
4. Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. Inf. Comput. 150(2), 132–152 (1999)

5. Juan, E.Y.T., Tsai, J.J.P., Murata, T.: Compositional verification of concurrent systems using Petri-net-based condensation rules. ACM Trans. Program. Lang. Syst. 20(5), 917–979 (1998)
6. Kaschner, K., Massuthe, P., Wolf, K.: Symbolic representation of operating guidelines for services. Petri Net Newsletter 72, 21–28 (2007)
7. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. pp. 77–91. LNCS 4937, Springer (2008)
8. Lohmann, N.: service-technology.org/live – replaying tool experiments in a Web browser. In: BPM Demos 2010. pp. 64–68. CEUR Workshop Proceedings 615, CEUR-WS.org (2010)
9. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010), tool available at `http://service-technology.org/wendy`.
10. Lohmann, N., Wolf, K.: How to implement a theory of correctness in the area of business processes and services. In: BPM 2010. pp. 61–77. LNCS 6336, Springer (2010)
11. Massuthe, P., Weinberg, D.: Fiona: A tool to analyze interacting open nets. In: AWPN 2008. pp. 99–104. CEUR Workshop Proceedings Vol. 380, CEUR-WS.org (2008), tool available at `http://service-technology.org/fiona`.
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
13. Valmari, A.: The state explosion problem. In: Advanced Course on Petri Nets. pp. 429–528. LNCS 1491, Springer (1996)
14. Weinberg, D.: Efficient controllability analysis of open nets. In: WS-FM 2008. pp. 224–239. LNCS 5387, Springer (2009)
15. Wolf, K.: Generating Petri net state spaces. In: PETRI NETS 2007. pp. 29–42. LNCS 4546, Springer (2007), tool available at `http://service-technology.org/lola`.
16. Wolf, K.: Does my service have partners? LNCS ToPNoC 5460(II), 152–171 (2009)
17. Wolf, K., Stahl, C., Ott, J., Danitz, R.: Verifying livelock freedom in an SOA scenario. In: ACSD 2009. pp. 168–177. IEEE Computer Society (2009)