

Electronic Notes in Theoretical Computer Science 263 (2010) 31–46

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Composition of Services with Constraints

# Philippe Balbiani[1](#_bookmark0) , Fahima Cheikh Alili [2](#_bookmark0)

*IRIT-CNRS, Toulouse, France*

# P.-C. Héam[3](#_bookmark0)

*LSV-CNRS-INRIA, Cachan, France*

# Olga Kouchnarenko[4](#_bookmark0)

*INRIA-CASSIS-LIFC, Besançon, France*

Abstract

Composition of Web services consists of the interleaving of the sequence of actions executed by the ele- mentary services in accordance with a client specification. We model Web services as automata executing actions and also sending and receiving messages. This paper provides a theoretical study for three service composition problems, and in particular for the problem of computing a Boolean formula which exactly characterises the conditions required for services to answer the client’s request. New complexity results are established for these problems within the framework of service composition with constraints.

*Keywords:* Composition, Décidability.

# Introduction and Related Work

Service oriented computing [[23](#_bookmark35)] is a programming paradigm which considers ser- vices as elementary components. From these components, distributed applications are realised in accordance with a client specification. To realise some distributed applications, elementary components have to be composed. The composition prob- lem has been investigated since the 2000’s with many solutions proposed [[3](#_bookmark17),[2](#_bookmark15),[17](#_bookmark30)]. Often, services are seen as finite automata. In this case, the client specification is given by a finite automaton which represents all computations that a client wants

1 Email: [Philippe.Balbiani@irit.fr](mailto:Philippe.Balbiani@irit.fr)

2 Email: [Cheick@irit.fr](mailto:Cheick@irit.fr)

3 Email: [pcheam@lsv.ens-cachan.fr](mailto:pcheam@lsv.ens-cachan.fr)

4 Email: [Olga.Kouchnarenko@lifc.univ-fcomte.fr](mailto:kouchna@lifc.univ-fcomte.fr)

1571-0661© 2010 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2010.05.003

the services to execute. By executing their transitions, services modify their envi- ronment and that of the client. The problem of combining services becomes that of composing automata, like in [[3](#_bookmark17),[2](#_bookmark15)]. In other cases, services are able to send and to receive messages. Client specification is then given by a logical formula which repre- sents client’s goals he wants services to reach. By communicating together, services modify their knowledge and those of their client (see, e.g., [[17](#_bookmark30)]). This paper follows the line of reasoning suggested in [[5](#_bookmark18),[6](#_bookmark19),[7](#_bookmark20),[8](#_bookmark21)], which consists in giving the semantics of services by means of automata. The paper particularly focuses on the following problem: Given services A1, *.. .*, A*n* and the request A of a client, can A1, *.. .*, A*n* be organised as to answer A? The originality of our approach consists in modeling the services by Boolean automata, i.e. finite automata extended with parametric Boolean conditions. The main motivation for using this model is to manage condi- tional actions or communications of A1, *.. .*, A*n*. For instance, a conditional action may be, for some *i, j* ∈ {1*,... , n*}, that A*i* accepts to communicate with A*j* if and only if A*j* has a security certificate given by some authority. A conditional action may also be, for some *i, j* ∈ {1*,... , n*}, that A*i* accepts to answer A*j* only if the IP address of A*j* is in a selected area. This kind of conditions frequently appear when specifying services. As far as we know, the composition problems studied in the literature do not handle this kind of conditions. This paper provides a theoretical study for three service composition problems, and in particular for the problem of computing a Boolean formula *φ* which exactly characterises the conditions required for A1, *.. .*, A*n* to answer the client’s request. The paper is organised as follows. Section [2](#_bookmark1) introduces the formal background. In Section [3](#_bookmark5), we formally define the valuation decision problem, the Boolean formula decision problem and the Boolean formula synthesis problem for both simulation-based relations and trace-based re- lations. Sections [4](#_bookmark10) and [5](#_bookmark13) contains our complexity results. Finally we conclude in section [6](#_bookmark14).

Related Work In the context of finite automata, many research works on complex- ity results for various finite automata compositions have been done.

In [[10](#_bookmark22)], the authors investigated the following problem: given *n* + *m* finite au- tomata A1*,... ,* A*n*+*m*, what is the complexity of deciding whether A1 × *...* × A*n* is equivalent to A*n*+1 × *...* ×A*n*+*m*. The class of problems considered in [[10](#_bookmark22)] is for non flat systems, i.e. one requires that *n* ≥ 2 and *m* ≥ 2. Another crucial difference w.r.t. our work concerns the product. In [[10](#_bookmark22)], the product is partially synchronised but, contrarily to our work, synchronisation does not produce *ε*-transition but labeled transitions. Their main result shows that the above decision problem is EXPTIME- hard for any relation between the simulation preorder and bisimulation, and that it is EXSPACE-hard for any relation between trace inclusion and the intersection of ready trace equivalence.

A more general problem is considered in [[20](#_bookmark32)] where the product is closer to ours since some actions can be *hidden*, i.e. replaced by an *ε*-transition. The author proved that for non flat systems the equivalence checking is PSPACE-hard for any relation between bisimilarity and trace equivalence, and that the problem is EXPSPACE- complete for trace equivalence, and EXPTIME-complete for bisimilarity. It was

also conjectured in [[20](#_bookmark32)] that the problem is EXPTIME-hard for any relation be- tween bisimilarity and trace equivalence. This conjecture was enhanced and proved in [[22](#_bookmark34)]: the problem is EXPTIME-hard for any relation between bisimilarity and trace preorder.

In [[15](#_bookmark28)], it is shown that deciding whether A is simulated by A1 × *...* × A*n* is still EXPTIME-hard. In this work, the considered product is asynchronous.

Several recent works focus on the use of finite automata based models to address Web services composition problems. In [[18](#_bookmark31),[17](#_bookmark30)], the authors propose a model where Web services compositions expressed in BPEL are formally defined by state transi- tion systems with communicating and internal (unobservable) actions. In [[17](#_bookmark30)], this model was enriched using a knowledge base, and in [[9](#_bookmark23)] using a theory. These ap- proaches were applied to practical applications in [[24](#_bookmark36),[12](#_bookmark24),[11](#_bookmark25)]. In [[1](#_bookmark16),[21](#_bookmark33)], Web services are defined by PWL-S documents and modelled by guarded finite automata. A sim- ilar approach is investigated in [[16](#_bookmark29)] where Web services are defined in BPEL. In [[14](#_bookmark27)], the authors model by Input/Output automata Web services interfaces described in BPEL, OWLS and WSDL.

# Preliminaries

Let *P* be a finite set of Boolean variables (with typical members denoted *p*, *p*', *.. .*), Σ*a* be a countable set of asynchronous actions (with typical members denoted *α*, *β*, *.. .*) and Σ*s* be a countable set of synchronous actions (with typical members denoted *σ*, *τ* , *.. .*). We will assume that *P* , Σ*a* and Σ*s* are disjoint.

Finite automata

A finite automaton is a tuple A = (*Q, E, I, F* ) where

* *Q* is a finite set of states,
* *E* is a function from *Q*×*Q* into the set of all finite subsets of Σ*a* ∪({?*,* !}×Σ*s*)∪{*ϵ*},
* *I* ⊆ *Q* is the set of initial states and *F* ⊆ *Q* is the set of final states. For all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*x, σ*) will be denoted *xσ*.

A is said to be *ϵ*-free iff *E* is a function from *Q* × *Q* into the set of all finite

subsets of Σ*a* ∪ ({?*,* !} × Σ*s*). We shall say that A is weakly asynchronous iff *E* is a function from *Q* × *Q* into the set of all finite subsets of Σ*a* ∪ {*ϵ*}. A is said to be strongly asynchronous iff *E* is a function from *Q* × *Q* into the set of all

finite subsets of Σ*a*. Given a weakly asynchronous automaton A, let *E*∗ be the function from *Q* × *Q* into the set of all finite subsets of Σ*a* such that for all *q, r* ∈ *Q*, for all *α* ∈ Σ*a*, *α* ∈ *E*∗(*q, r*) iff there are sequences (*q*0*, q*1*,... , qm*)*,* (*r*0*, r*1*,... , rn*) ∈ *Q*+ such that

* for all positive integers *i*, if *i* ≤ *m* then *ϵ* ∈ *E*(*qi*−1*, qi*),
* for all positive integers *j*, if *j* ≤ *n* then *ϵ* ∈ *E*(*rj*−1*, rj*),
* *α* ∈ *E*(*qm, r*0),
* *q*0 = *q* and *rn* = *r*.

A1 :

(*p*1 ∨ *p*2*, α*)

(¬*p*2*,* !*σ*1)



*q*1

*q*2

(¬*p*1 ∧ *p*2*,* ?*σ*2)

A2 :

(¬*p*3*, α*)



*r*1

(*p*3*,* ?*σ*1)

*r*2

Figure 1. Boolean automata A1 and A2.

In this case, a run of A on a sequence (*α*1*,... , αk*) ∈ Σ∗ is a sequence (*q*0*, q*1*,... , qk*) ∈ *Q*+ such that for all positive integers *i*, if *i* ≤ *k* then *αi* ∈ *E*∗(*qi*−1*, qi*), *q*0 ∈ *I* and *qk* ∈ *F* . Moreover, the traces of A, denoted *tr*(A), is the set of all sequences (*α*1*,... , αk*) ∈ Σ∗ such that there is a run of A on (*α*1*,... , αk*).

*a*

*a*

Boolean automata

The set B(*P* ) of all Boolean formulas (with typical members denoted *φ*, *ψ*, *.. .*) is defined by: *φ* ::= *p* | ⊥ | ¬*φ* | (*φ* ∨ *φ*), with *p* ∈ *P* . The other constructs are defined as usual. In particular, T = ¬⊥ and (*φ* ∧ *ψ*) = ¬(¬*φ* ∨ ¬*ψ*). A valuation is a function from *P* into {0*,* 1}. Every valuation *V* gives rise to a function *V*^ from B(*P* ) into {0*,* 1} in the usual way. A Boolean automaton is a tuple A = (*Q, E, I, F* ) where

* *Q* is a finite set of states,
* *E* is a function from *Q*×*Q* into the set of all finite subsets of B(*P* )×(Σ*a* ∪({?*,* !}×

Σ*s*) ∪ {*ϵ*}),

* *I* ⊆ *Q* is the set of initial states and *F* ⊆ *Q* is the set of final states.

The notions of *ϵ*-freeness, weak asynchronicity and strong asynchronicity are defined for Boolean automata in the same way as they are defined for finite automata. As example, take the case of the Boolean automata A1 and A2 from Fig. [1](#_bookmark2), with Σ*a* = {*α*} and Σ*s* = {*σ*1*, σ*2}.

Synchronisation

Let A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata. Their synchronisation, denoted A1 ⊗ *...* ⊗ A*n*, is the weakly asynchronous Boolean automaton A = (*Q, E, I, F* ) defined by:

* *Q* = *Q*1 × *...* × *Qn*,
* *E* is the function from *Q* × *Q* into the set of all finite subsets of B(*P* ) × (Σ*a* ∪ {*ϵ*})

such that for all *q, r* ∈ *Q*, for all *φ* ∈ B(*P* ),

* + for all *α* ∈ Σ*a*, (*φ, α*) ∈ *E*(*q, r*) iff there is *i* ∈ {1*,... , n*} such that *q* ≡*i r* and

(*φ, α*) ∈ *Ei*(*qi, ri*),

* + (*φ, ϵ*) ∈ *E*(*q, r*) iff there are *i*1*, i*2 ∈ {1*,... , n*}, there are *φi*1 *, φi*2 ∈ B(*P* ), there is *σ* ∈ Σ*s* such that *i*1 /= *i*2, *q* ≡*i*1 *,i*2 *r*, either (*φi*1 *,* (?*, σ*)) ∈ *Ei*1 (*qi*1 *, ri*1 ) and (*φi*2 *,* (!*, σ*)) ∈ *Ei*2 (*qi*2 *, ri*2 ) or (*φi*1 *,* (!*, σ*)) ∈ *Ei*1 (*qi*1 *, ri*1 ) and (*φi*2 *,* (?*, σ*)) ∈ *Ei*2 (*qi*2 *, ri*2 ) and *φ* = *φi*1 ∧ *φi*2 .

(¬*p*3*, α*) (¬*p*3*, α*)



*q*1*, r*1

(*p*1 ∨ *p*2*, α*)

*q*2*, r*1

*q*1*, r*2

(*p*1 ∨ *p*2*, α*)

*q*2*, r*2

(*p*3 ∧ ¬*p*2*, ϵ*)

Figure 2. Boolean automata A1 ⊗ A2.

A :

(*p*1 ∨ *p*2*, α*)

(¬*p*2*,* !*σ*1)



*q*1

*q*2

(¬*p*1 ∧ *p*2*,* ?*σ*2)

A*V* :

*α*



*q*1

*q*2

?*σ*2

*V* (*p*1)= 0*,V* (*p*2)= 1

Figure 3. Boolean automaton A and the associated automaton A*V* .

* *I* = *I*1 × *...* × *In* and *F* = *F*1 × *...* × *Fn*,

the binary relations ≡*i,* ≡*i*1 *,i*2 ⊆ *Q* × *Q* being such that for all *q, r* ∈ *Q*, *q* ≡*i r* iff for all *j* ∈ {1*,... , n*}, if *i* /= *j* then *qj* = *rj* and *q* ≡*i*1*,i*2 *r* iff for all *j* ∈ {1*,... , n*}, if *i*1 /= *j* and *i*2 /= *j* then *qj* = *rj*. Consider, as example, the Boolean automata *A*1 and A2 from Fig. [1](#_bookmark2) and A1 ⊗ A2 from Fig. [2](#_bookmark3).

From Boolean automata to finite automata

Let A = (*Q, E, I, F* ) be a Boolean automaton and *V* be a valuation. The inter- pretation of A through *V* , denoted A*V* , is the finite automaton A' = (*Q*'*, E*'*,I*'*,F* ') defined by:

* *Q*' = *Q*,
* *E*' is the function from *Q*' × *Q*' into the set of all finite subsets of Σ*a* ∪ ({?*,* !} ×

Σ*s*) ∪ {*ϵ*} such that for all *q, r* ∈ *Q*',

* + for all *α* ∈ Σ*a*, *α* ∈ *E*'(*q, r*) iff there is *φ* ∈ B(*P* ) such that (*φ, α*) ∈ *E*(*q, r*) and

*V*^ (*φ*) = 1,

* + for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, *xσ* ∈ *E*'(*q, r*) iff there is *φ* ∈ B(*P* ) such that

(*φ, xσ*) ∈ *E*(*q, r*) and *V*^ (*φ*) = 1,

* + *ϵ* ∈ *E*'(*q, r*) iff there is *φ* ∈ B(*P* ) such that (*φ, ϵ*) ∈ *E*(*q, r*) and *V*^ (*φ*) = 1,
* *I*' = *I* and *F* ' = *F* .

As example, take the case of A and A*V* from Fig. [3](#_bookmark4).

Trace inclusion and trace equivalence

Let A = (*Q, E, I, F* ), A' = (*Q*'*, E*'*,I*'*,F* ') be weakly asynchronous finite au-

tomata. We shall say that A is trace-included in A', denoted A ± A', iff *tr*(A) ⊆

*tr*(A'). A is said to be trace-equivalent to A', denoted A ≡ A', iff A ± A' and A'

± A.

Simulation and bisimulation

Let A = (*Q, E, I, F* ), A' = (*Q*'*, E*'*,I*'*,F* ') be weakly asynchronous finite au- tomata. We define a binary relation *Z* ⊆ *Q* × *Q*' such that *dom*(*Z*) ∩ *I* /= ∅ and *ran*(*Z*) ∩ *I*' /= ∅ to be a simulation of A by A', denoted *Z*: A ←− A', iff for all *q* ∈ *Q*, for all *q*' ∈ *Q*', if *q Z q*' then

* for all *α* ∈ Σ*a*, for all *r* ∈ *Q*, if *α* ∈ *E*∗(*q, r*) then there is *r*' ∈ *Q*' such that *r Z r*' and *α* ∈ *E*'∗(*q*'*, r*'),
* if *q* ∈ *I* then *q*' ∈ *I*' and if *q* ∈ *F* then *q*' ∈ *F* '.

Note: *dom*(*Z*) and *ran*(*Z*) respectively denote the domain of *Z* and the range of *Z*. If there is a simulation *Z* of A by A' then we write A ←− A'. We define a binary relation *Z* ⊆ *Q* × *Q*' to be a bisimulation between A and A', denoted *Z*: A ←→ A', iff *Z*: A ←− A' and *Z*−1: A' ←− A. If there is a bisimulation between A and A' then we write A ←→ A'.

# Composition of Services

* 1. *Formal definitions*

Let *R* ∈ {±*,* ≡*,* ←−*,* ←→}. The valuation decision (VD) problem for *R* is defined by:

* + - input: a strongly asynchronous finite automaton A = (*Q, E, I, F* ) and *ϵ*-free Boolean automata A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*),
    - output: check if there is a valuation *V* such that A *R* (A1 ⊗ *...* ⊗ A*n*)*V* . The Boolean formula decision (BFD) problem for *R* is defined by:
    - input: a strongly asynchronous finite automaton A = (*Q, E, I, F* ), *ϵ*-free Boolean automata A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) and a Boolean formula *φ*,
    - output: check if for all valuations *V* , A *R* (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1. The Boolean formula synthesis (BFS) problem for *R* is defined by:
    - input: a strongly asynchronous finite automaton A = (*Q, E, I, F* ) and *ϵ*-free Boolean automata A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*),
    - output: find out a Boolean formula *φ* such that for all valuations *V* , A *R* (A1 ⊗

*...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1.

The questions we are interested in are: Is there a valuation of these services such that the desired composition is possible (VD problem)? How to compute a boolean formula *φ* over these predicates that is true iff the desired composition is possible (BFS problem)?

(T*, no*\_*ticket*)

A1 :



*q*1

(T*, ticket*\_*req*)

*q*2

(ч*pblack, c*\_*v*)

*q*3

(T*,* !*ask*\_*bank*1)

*q*4

(T*, send*\_*ticket*)

*q*5

(T*,* ?*resp*\_*bank*1)

Figure 4. Boolean automaton A1

(T*, no*\_*flight*)

A2 :



*q*1

(T*,flight*\_*req*)

*q*2

(*psh* V *pnat, c*\_*f* )

*q*3

(T*,* !*ask*\_*bank*2)

*q*4

(T*, send*\_*flight*)

*q*5

(T*,* ?*resp*\_*bank*2)

Figure 5. Boolean automaton A2

(T*,* !*resp*\_*bank*1)

A3 :



*q*2

(*pcc* V *ppp, pay*\_*ok*)

*q*3

*q*1

(*pcert*1*,* ?*ask*\_*bank*1)

(*pcert*2*,* ?*ask*\_*bank*2)

*q*4

(*pcc* V *ppp, pay*\_*ok*)

*q*5

(T*,* !*resp*\_*bank*2)

Figure 6. Boolean automaton A3

* 1. *Motivating Example*

Consider three services *S*1*, S*2*, S*3.

Service *S*1 provides an on-line booking for European football match places that will take place in Madrid. This service is modeled by the automaton A1 in Fig. [4](#_bookmark6).

When *S*1 receives a request for a football place (*ticket*\_*req*) booking, it checks whether there is available places having the desired price (action *c*\_*v*). This check can be done if and only if the request is provided by a non black listed supporter (predicate ¬*pblack*). If there is no available place, *S*1 informs the requester by the action *no*\_*ticket*. If there is a ticket, *S*1 asks the bank service (*S*3) for the ticket payment (actions !*ask*\_*bank*1). If the bank answers the payment is Ok (message

?*resp*\_*bank*1), the ticket is sent to the requester (action *send*\_*ticket*).

Service *S*2 sells flight tickets. It is modeled by the automaton A2 in Fig. [5](#_bookmark7).

*S*2 works like *S*1 but sells tickets only for people who do not need a Visa for coming to Spain, thus either people whose nationality is in a given list (predicate *pnat*), or if the starting point of the flight is in the Shengen Space (predicate *psh*).

The service *S*3 is the bank service modeled by the automaton A3 in Fig. [6](#_bookmark8).

*no*\_*ticket*

A :



*q*1

*ticket*\_*req*

*q*2

*c*\_*v*

*q*3

*flight*\_*req*

*q*4

*send*\_*flight*

*c*\_*f*

*q*7

*send*\_*ticket*

*no*\_*flight*

*q*6

*pay*\_*ok*

*q*5

*pay*\_*ok*

Figure 7. Automaton A

*S*3 accepts request from services having a security certificate (predicate *pcerti* means that *Si* has such a certificate). Then, if the buyer has either a credit card (predicate *pcc*) or a paypal account (predicate *ppp*), and if the payment is OK (in order to not overload the example, we do not model this communicating point; we just encode it by the action *pay*\_*ok*), the bank validates the payment to the related service.

As the reader can see, services can be composed to provide the intented service *S* (depicted in Fig. [7](#_bookmark9)), with several conditions on the value of predicates *pnat, pcert*1*,*, etc.

# Trace Inclusion and Trace Equivalence

Let us recall that given two finite automata A1 and A2, testing whether A1 ± (A2

can be done in *PSP ACE*.

Upper bound

Let A = (*Q, E, I, F* ) be a strongly asynchronous finite automaton and A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata. We now de- fine a deterministic algorithm which returns the value “accept” iff there is a valuation

*V* such that A ± (A1 ⊗ *...* ⊗ A*n*)*V* :

1. for all valuations *V*
   1. compute (A1 ⊗ *...* ⊗ A*n*)*V* ;
   2. check if A ± (A1 ⊗ *...* ⊗ A*n*)*V* ;
2. if one of these calls returns the value “accept” then return the value “accept” else return the value “reject”;

Obviously, the deterministic algorithm above is exponential-space-bounded. Simi- larly, an exponential-space-bounded deterministic algorithm which returns the value “accept” iff there is a valuation *V* such that A ≡ (A1 ⊗ *...* ⊗ A*n*)*V* can be defined. As a result,

Proposition 4.1 *Let R* ∈ {±*,* ≡}*. The VD problem for R is in EXPSP ACE.*

Similarly,

Proposition 4.2 *Let R* ∈ {±*,* ≡}*. The BFD problem for R is in EXPSP ACE.*

Let A = (*Q, E, I, F* ) be a strongly asynchronous finite automaton and A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata. We now define a deterministic algorithm which returns a Boolean formula *φ* such that for all valuations *V* , A ± (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1:

1. *φ* := ⊥;
2. for all maximal consistent conjunctions *ψ* of *P* -literals
   1. compute the unique valuation *V* such that *V*^ (*ψ*) = 1;
   2. compute (A1 ⊗ *...* ⊗ A*n*)*V* ;
   3. if A ± (A1 ⊗ *...* ⊗ A*n*)*V* then *φ* := *φ* ∨ *ψ*;

Obviously, the deterministic algorithm above is exponential-space-bounded. Simi- larly, an exponential-space-bounded deterministic algorithm which returns a Boolean formula *φ* such that for all valuations *V* , A ≡ (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*)=1 can be defined. As a result,

Proposition 4.3 *Let R* ∈ {±*,* ≡}*. The BFS problem for R is solvable by means of an exponential-space-bounded deterministic algorithm.*

Lower bound

To prove that the VD problem for ± is *EXPSP ACE*-hard, we give a polynomial time reduction of the universality problem for regular expressions with squaring, which is known to be *EXPSP ACE*-hard [[13](#_bookmark26)], to the VD problem for ±. The set of

all regular expressions with squaring (with typical members denoted *exp*, *exp*', *.. .*) is defined by:

*exp* ::= *α* | *ϵ* | (*exp* ◦ *exp*) | (*exp* ∪ *exp*) | *exp*+ | *exp*2*.*

The number of occurrences of operators *ϵ*, ◦, ∪, + and 2 in regular expression *exp* with squaring is denoted *op*(*exp*). Every regular expression *exp* with squaring gives rise to a language denoted *lang*(*exp*) in the usual way. Let *σ*1, *σ*2, *...* be an enumeration of Σ*s*. Given a regular expression *exp* with squaring, let *n* = 2 × *op*(*exp*). Let A = (*Q, E, I, F* ) be the strongly asynchronous finite automaton defined as follows: *Q* =

{*q*}, *E* is the function from *Q* × *Q* into the set of all finite subsets of B(*P* ) ×Σ*a* such that for all *q, r* ∈ *Q*, for all *φ* ∈ B(*P* ), for all *α* ∈ Σ*a*, (*φ, α*) ∈ *E*(*q, r*) iff *φ* = T, *I* =

{*q*} and *F* = {*q*}. For all positive integers *i*, if *i* ≤ *n* then let A*i* = (*Qi, Ei, Ii, Fi*) be the *ϵ*-free Boolean automaton defined as follows: *Qi* = {*q*1*i, q*2*i*}, *Ei* is the function from *Qi* × *Qi* into the set of all finite subsets of B(*P* ) × (Σ*a* ∪ ({?*,* !}×Σ*s*)) such that for all *q, r* ∈ *Qi*, for all *φ* ∈ B(*P* ),

* for all *α* ∈ Σ*a*, (*φ, α*) /∈ *Ei*(*q, r*),
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) ∈ *Ei*(*q, r*) iff *φ* = T, *x* = ?, *σ* = *σi*, *q* =

*q*1*i* and *r* = *q*2*i* or *φ* = T, *x* = !, *σ* = *σi*, *q* = *q*2*i* and *r* = *q*1*i*,

*Ii* = {*q*1*i*} and *Fi* = {*q*1*i*}. Let A0 = (*Q*0*, E*0*, I*0*, F*0) be the *ϵ*-free Boolean automaton defined by induction on *exp* as follows:

Basis: Case *exp* = *α*. In this case, A0 = (*Q*0*, E*0*, I*0*, F*0) is defined as follows: *Q*0

= {*qI, qF* }, *E*0 is the function from *Q*0 × *Q*0 into the set of all finite subsets of

B(*P* ) × (Σ*a* ∪ ({?*,* !}× Σ*s*)) such that for all *q, r* ∈ *Q*0, for all *φ* ∈ B(*P* ),

* for all *β* ∈ Σ*a*, (*φ, β*) ∈ *E*0(*q, r*) iff *q* = *qI*, *r* = *qF* , *φ* = T and *β* = *α*,
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *E*0(*q, r*),

*I*0 = {*qI* } and *F*0 = {*qF* }. The Boolean automaton A0 is represented in Fig. [8](#_bookmark11). The reader may easily verify that for all valuations *V* , *lang*(*exp*) = *tr*(A*V* ). Remark

0

that 0= *n*. Note also that *I*0 and *F*0 are singletons.

*qI* (T*, α*) *qF*



Figure 8. Finite automaton A0 in the case *exp* = *α*.

Hypothesis: *exp*' and *exp*'' are regular expressions with squaring such that there

is an *ϵ*-free Boolean automaton A' = (*Q*' *, E*' *,I*' *,F* ') such that for all valuations

0 0 0 0 0

*V* , *lang*(*exp*') = *tr*((A' ⊗ A1 ⊗ *...* ⊗ A*n*' )*V* ) where *n*' = 2 × *op*(*exp*') and there

0

is an *ϵ*-free Boolean automaton A'' = (*Q*''*, E*''*,I*''*,F* '') such that for all valuations

0 0 0 0 0

*V* , *lang*(*exp*'') = *tr*((A'' ⊗ A1 ⊗ *...* ⊗ A*n*'' )*V* ) where *n*'' = *q* × *op*(*exp*''). We also

0

assume that *I*' , *F* ', *I*'' and *F* '' are singletons.

0 0 0 0

Step: The cases *exp* = *ϵ*, *exp* = *exp*' ◦ *exp*'', *exp* = *exp*' ∪ *exp*'' and *exp* = *exp*'+

are similar.

Case *exp* = *exp*'2. In this case, A0 = (*Q*0*, E*0*, I*0*, F*0) is defined as follows: *Q*0 = *Q*' ∪ {*qI, q, r, qF* }, *E*0 is the function from *Q*0 × *Q*0 into the set of all finite subsets of B(*P* ) × (Σ*a* ∪ ({?*,* !}× Σ*s*)) such that for all *s, t* ∈ *Q*0, for all *φ* ∈ B(*P* ),

0

* for all *β* ∈ Σ*a*, (*φ, β*) ∈ *E*0(*s, t*) iff *s, t* ∈ *Q*' and (*φ, β*) ∈ *E*' (*s, t*),

0

0

* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) ∈ *E*0(*s, t*) iff *s, t* ∈ *Q*'

0

and (*φ, xσ*) ∈

*E*' (*s, t*) or *s* = *qI* , *t* = *q*' , *φ* = T, *x* = ! and *σ* = *σn*'+1 or *s* = *q*' , *t* = *q*, *φ* =

0 *I F*

T, *x* = ! and *σ* = *σn*'+2 or *s* = *q*, *t* = *qI*, *φ* = T, *x* = ? and *σ* = *σn*'+1 or *s* =

*q*' , *t* = *r*, *φ* = T, *x* = ? and *σ* = *σn*'+2 or *s* = *r*, *t* = *qF* , *φ* = T, *x* = ? and *σ*

*F*

= *σn*'+1,

*I*0 = {*qI* } and *F*0 = {*qF* }. The Boolean automaton A0 is represented in Fig. [9](#_bookmark12). The reader may easily verify that for all valuations *V* , *lang*(*exp*)= *tr*((A0 ⊗ A1 ⊗

*...* ⊗ A*n*' ⊗ A*n*'+1 ⊗ A*n*'+2)*V* ). Remark that *n*' +2 = *n*. Note also that *I*0 and *F*0

are singletons.

The reader may easily verify that *lang*(*exp*)= Σ∗ iff there is a valuation *V* such that

*a*

A ± (A0 ⊗ A1 ⊗ *...* ⊗ A*n*)*V* . Similarly, the reader may easily verify that *lang*(*exp*)

= Σ∗ iff there is a valuation *V* such that A ≡ (A0 ⊗ A1 ⊗ *...* ⊗ A*n*)*V* . As a result,

*a*

Proposition 4.4 *Let R* ∈ {±*,* ≡}*. The VD problem for R is EXPSP ACE-hard.*

Similarly,

Proposition 4.5 *Let R* ∈ {±*,* ≡}*. The BFD problem for R is EXPSP ACE-hard.*

(T*,* ?*σn*' +1)



(T*,* ?*σn*' +2)

*r*

*q*

*F*

(T*,* !*σn*' +1)

'

(T*,* !*σn*'+2)

*qI*

*q*' *q*'

A0

*I F*

*q*

(T*,* ?*σn*' +1)

Figure 9. Finite automaton A0 in the case *exp* = *exp*'2.

According to Meyer and Stockmeyer [[13](#_bookmark26)], for all deterministic Turing machines *M* solving the universality problem for regular expressions with squaring, there exists a constant *c >* 1 such that *M* needs at least space *cn* on some input of length *n* for infinitely many *n*. Suppose that there is a deterministic algorithm *f* solving the BFS problem for *R* and such that for all constants *c >* 1, *f* needs at least space *cn* on some input of length *n* for finitely many *n* only. Let *Mf* be the deterministic Turing machine that behaves as follows given a regular expression *exp* with squaring:

1. *Mf* computes *n* = 2 × *op*(*exp*);
2. *Mf* computes the strongly asynchronous finite automaton A = (*Q, E, I, F* )

defined as above;

1. for all positive integers *i*, if *i* ≤ *n* then *Mf* computes the *ϵ*-free Boolean au- tomaton A*i* = (*Qi, Ei, Ii, Fi*) defined as above;
2. *Mf* computes the *ϵ*-free Boolean automaton A0 = (*Q*0*, E*0*, I*0*, F*0) defined by induction on *exp* as above;
3. *Mf* simulates *f* on input A and A0 and A1, *.. .*, A*n* until it is about to return a value *φf* ;
4. if *φf* is a Boolean tautology then return the value “accept” else return the value “reject”;

Obviously, *Mf* solves the universality problem for regular expressions with squaring and for all constants *c >* 1 *Mf* needs at least space *cn* on some input of length *n* for finitely many *n* only: a contradiction. As a result,

Proposition 4.6 *Let R* ∈ {±*,* ≡}*. For all deterministic algorithms f solving the BFS problem for R, there exist a constant c >* 1*, such that f needs at least space cn on some input of length n for infinitely many n.*

# Simulation and Bisimulation

Let us recall that given two finite automata A1 and A2, testing whether A1 simulates

A2 can be done in polynomial time.

Upper bound

Let A = (*Q, E, I, F* ) be a strongly asynchronous finite automaton and A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata. We now de- fine a deterministic algorithm which returns the value “accept” iff there is a valuation

*V* such that A ←− (A1 ⊗ *...* ⊗ A*n*)*V* :

1. for all valuations *V*
   1. compute (A1 ⊗ *...* ⊗ A*n*)*V* ;
   2. check if A ←− (A1 ⊗ *...* ⊗ A*n*)*V* ;
2. if one of these calls returns the value “accept” then return the value “accept” else return the value “reject”;

Obviously, the deterministic algorithm above is exponential-time-bounded. Simi- larly, an exponential-time-bounded deterministic algorithm which returns the value “accept” iff there is a valuation *V* such that A ←→ (A1 ⊗ *...* ⊗ A*n*)*V* can be defined. As a result,

Proposition 5.1 *Let R* ∈ {←−*,* ←→}*. The VD problem for R is in EXPTIME.*

Let A = (*Q, E, I, F* ) be a strongly asynchronous finite automaton, A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata and *φ* be a Boolean formula. We now define a deterministic algorithm which returns the value “accept” iff for all valuations *V* , A ←− (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1:

1. for all valuations *V*
   1. compute (A1 ⊗ *...* ⊗ A*n*)*V* ;
   2. compute *V*^ (*φ*);
   3. check if A ←− (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1;
2. if all these calls returns the value “accept” then return the value “accept” else return the value “reject”.

Obviously, the deterministic algorithm above is exponential-time-bounded. Simi- larly, an exponential-time-bounded deterministic algorithm which returns the value “accept” iff for all valuations *V* , A ←→ (A1 ⊗ *...* ⊗A*n*)*V* iff *V*^ (*φ*)= 1 can be defined.

Proposition 5.2 *Let R* ∈ {←−*,* ←→}*. The BFD problem for R is in EXPTIME.*

Let A = (*Q, E, I, F* ) be a strongly asynchronous finite automaton and A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be *ϵ*-free Boolean automata. We now define a deterministic algorithm which returns a Boolean formula *φ* such that for all valuations *V* , A ←− (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1:

1. *φ* := ⊥;
2. for all maximal consistent conjunctions *ψ* of *P* -literals
   1. compute the unique valuation *V* such that *V*^ (*ψ*) = 1;
   2. compute (A1 ⊗ *...* ⊗ A*n*)*V* ;
   3. if A ←− (A1 ⊗ *...* ⊗ A*n*)*V* then *φ* := *φ* ∨ *ψ*;

Obviously, the deterministic algorithm above is exponential-time-bounded. Simi- larly, an exponential-time-bounded deterministic algorithm which returns a Boolean formula *φ* such that for all valuations *V* , A ←→ (A1 ⊗ *...* ⊗ A*n*)*V* iff *V*^ (*φ*) = 1 can be defined. As a result,

Proposition 5.3 *Let R* ∈ {←−*,* ←→}*. The BFS problem for R is solvable by means of an exponential-time-bounded deterministic algorithm.*

Lower bound

By giving a polynomial time reduction of the simulation problem of a strongly asynchronous finite automaton by means of a product of strongly asynchronous finite automata, which is known to be *EXPTIME*-hard [[15](#_bookmark28)], to the VD problem for ←−, we prove that the VD problem for ←− is *EXPTIME*-hard. Given a strongly asynchronous finite automaton A = (*Q, E, I, F* ) and strongly asynchronous finite

automata A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*), let A' = (*Q*' *, E*' *,I*' *,F* '),

1 1 1 1 1

*.. .*, A' = (*Q*' *, E*' *,I*' *,F* ' ) be the *ϵ*-free Boolean automata defined as follows: *Q*' =

*n n n n n* *i*

*Qi*, *E*' is the function from *Q*' × *Q*' into the set of all finite subsets of B(*P* ) × (Σ*a* ∪

*i i* *i*

({?*,* !}× Σ*s*)) such that for all *q, r* ∈ *Q*', for all *φ* ∈ B(*P* ),

*i*

* for all *α* ∈ Σ*a*, (*φ, α*) ∈ *E*'(*q, r*) iff *φ* = T and *α* ∈ *Ei*(*q, r*),

*i*

* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *E*'(*q, r*),

*i*

*I*' = *Ii* and *F* ' = *Fi*. The reader may easily verify that A ←− A1 ⊗ *...* ⊗ A*n* iff there

*i* *i*

is a valuation *V* such that A ←− (A' ⊗ *...* ⊗ A' )*V* . As a result,

1 *n*

Proposition 5.4 *The VD problem for* ←− *is EXPTIME-hard.*

Similarly,

Proposition 5.5 *The BFD problem for* ←− *is EXPTIME-hard.*

Proof We prove that the Boolean formula decision problem for ←− is *EXPTIME*- hard by giving a polynomial time reduction of the simulation problem of a strongly asynchronous finite automaton by means of a product of strongly asynchronous finite automata, which is known to be *EXPTIME*-hard [[15](#_bookmark28)], to the Boolean formula decision problem for ←−. Given a strongly asyn- chronous finite automaton A = (*Q, E, I, F* ) and strongly asynchronous finite au-

tomata A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*), let A' = (*Q*' *, E*' *,I*' *,*

1 1 1 1

*F* '), *.. .*, A' = (*Q*' *, E*' *,I*' *,F* ' ) be the *ϵ*-free Boolean automaton defined as above

1 *n n n n n*

and *φ* = T. Suppose that A ←− A1 ⊗ *...* ⊗ A*n*. Hence, there is a simulation *Z* of

A by A1 ⊗ *...* ⊗ A*n*. Let *V* be a valuation. Obviously, *Z* is a simulation of A by

(A' ⊗ *...* ⊗A' )*V* . Therefore, for all valuations *V* , A ←− (A' ⊗ *...* ⊗A' )*V* iff *V*^ (*φ*)=

1 *n* 1 *n*

1. Reciprocally, suppose that for all valuations *V* , A ←− (A' ⊗ *...*⊗A' )*V* iff *V*^ (*φ*)=

1 *n*

1. Let *V* be a valuation. Thus, there is a simulation *Z* of A by (A' ⊗*...*⊗A' )*V* . Ob-

1 *n*

viously, *Z* is a simulation of A by A1⊗*.. .*⊗A*n*. Consequently, A ←− A1⊗*.. .*⊗A*n*.

We prove that the VD problem for ←→ is *PSP ACE*-hard by giving a polynomial time reduction of the acceptance problem of a linear-space-bounded deterministic Turing machine, which is known to be *PSP ACE*-hard, to the VD problem for

←→. Let *acc*, *α*1, *α*2, *...* be an enumeration of Σ*a*. Given a linear-space-bounded

deterministic Turing machine *M* = (*QM , q*0 *, q*1 *,* Σ*M , δM* ) and a word *w* ∈ Σ∗

, let *n*

*M M M*

be the length of *w*. Let Σ*f* = {*acc, α*1*,... , αn*}. Let A = (*Q, E, I, F* ) be the strongly

asynchronous finite automaton defined as follows: *Q* = (*QM* × {1*,... , n*}) ∪ {⊥}, *E* is the function from *Q* × *Q* into the set of all finite subsets of B(*P* ) × Σ*f* such that for all (*q, i*)*,* (*r, j*) ∈ *Q*, for all *φ* ∈ B(*P* ),

* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *E*((*q, i*)*,* (*r, j*)) iff *φ* = T, *α* = *αi* and there are *u, v* ∈ Σ*M* , there is *d* ∈ {−1*,* +1} such that *δM* (*q, u, r, v, d*) is defined and *j* = *i* + *d* or *φ* = T,

*α* = *acc*, *q* = *q*1 , *r* = *q*1

and *j* = *i*,

*M M*

* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *E*((*q, i*)*,* ⊥) iff *φ* = T, *α* /= *acc* and *α* /= *αi* or for all *r* ∈

*QM* , for all *u, v* ∈ Σ*M* , for all *d* ∈ {−1*,* +1}, *δM* (*q, u, r, v, d*) is not defined,

* for all *α* ∈ Σ*f* , (*φ, α*) /∈ *E*(⊥*,* (*r, j*)),
* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *E*(⊥*,* ⊥) iff *φ* = T and *α* /= *acc*,

*I* = {(*q*0 *,* 1)} and *F* = ∅. Let A1 = (*Q*1*, E*1*, I*1*, F*1), *.. .*, A*n* = (*Qn, En, In, Fn*) be

*M*

the *ϵ*-free Boolean automata defined as follows: *Qi* = (*QM* × Σ*M* ) ∪ {⊥*i*}, *Ei* is the function from *Qi* × *Qi* into the set of all finite subsets of B(*P* ) × (Σ*f* ∪ ({?*,* !}× Σ*s*)) such that for all (*q, u*)*,* (*r, v*) ∈ *Qi*, for all *φ* ∈ B(*P* ),

* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *Ei*((*q, u*)*,* (*r, v*)) iff *φ* = T, *α* = *αi* and there is *d* ∈ {−1*,* +1}

such that *δM* (*q, u, r, v, d*) is defined,

* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *Ei*((*q, u*)*,* ⊥*i*) iff *φ* = T, *α* /= *acc* and *α* /= *αi* or for all *r*

∈ *QM* , for all *v* ∈ Σ*M* , for all *d* ∈ {−1*,* +1}, *δM* (*q, u, r, v, d*) is not defined,

* for all *α* ∈ Σ*f* , (*φ, α*) /∈ *Ei*(⊥*i,* (*r, v*)),
* for all *α* ∈ Σ*f* , (*φ, α*) ∈ *Ei*(⊥*i,* ⊥*i*) iff *φ* = T and *α* /= *acc*,
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *Ei*((*q, u*)*,* (*r, v*)),
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *Ei*((*q, u*)*,* ⊥*i*),
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *Ei*(⊥*i,* (*r, v*)),
* for all *x* ∈ {?*,* !}, for all *σ* ∈ Σ*s*, (*φ, xσ*) /∈ *Ei*(⊥*i,* ⊥*i*),

*Ii* = {(*q*0 *, wi*)} and *Fi* = ∅. Suppose that *M* does not accept *w*. Let *Z* ⊆ *Q* ×

*M*

(*Q*1 × *...* × *Qn*) be the binary relation such that

* (*q, i*) *Z* ((*q*1*, u*1)*,... ,* (*qn, un*)) iff *q* = *qi* and (*q*0 *,* 1*, w*) =⇒∗

(*q, i, u*1 *... un*),

*M M*

* · *Z* (·*,... ,* ·*,* ⊥*i,* ·*,... ,* ·),
* ⊥ *Z* (·*,... ,* ·).

The reader may easily verify that there is a valuation *V* such that *Z* is a bisimulation between A and (A1 ⊗ *...* ⊗ A*n*)*V* . Hence, there is a valuation *V* such that A ←→ (A1 ⊗ *...* ⊗ A*n*)*V* . Reciprocally, suppose that there is a valuation *V* such that A ←→ (A1 ⊗ *...* ⊗ A*n*)*V* . Therefore, there is a bisimulation *Z* between A and

(A1 ⊗ *...* ⊗ A*n*)*V* . Obviously, *M* does not accept *w*.

Proposition 5.6 *The VD problem for* ←→ *is PSP ACE-hard. The BFD problem for* ←→ *is PSP ACE-hard.*

# Conclusion

This paper presented different new complexity results for the VD problem and the BFD problem within the framework of service composition with constraints. To sum up, the results are given in snapshot of our work below.

|  |  |  |
| --- | --- | --- |
| *R* | valuation decision problem | Boolean formula decision problem |
| ≡ | *EXPSP ACE*-complete | *EXPSP ACE*-complete |
| ⊆ | *EXPSP ACE*-complete | *EXPSP ACE*-complete |
| ← | *EXPTIME*-complete | *EXPTIME*-complete |
| — | *PSP ACE*-hard  in *EXPTIME* | *PSP ACE*-hard  in *EXPTIME* |

As pointed out by the above table, a still open question is to evaluate the exact com- plexity of the valuation decision problem for ←→ and the Boolean formula decision problem for ←→: are they in *PSP ACE* or are they *EXPTIME*-hard? Moreover, we focused on the identification of a relevant abstraction to manage conditional actions in the service composition problem. In the future, we intend to explore practical algorithmic approaches to handle the Boolean formula synthesis problem.

# References

1. Berardi, D., D. Calvanese, G. De Giacomo, R. Hull, M. Mecella: *Automatic composition of transition-* *based semantic Web services with messaging.* In: Very Large Data Bases. ACM (2005) 613–624.
2. Berardi, D., D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: *Automatic services composition* *based on behavioral descriptions.* Int. Journal of Cooperative Information Systems 14 (2005) 333–376.
3. Berardi, D., F. Cheikh, G. De Giacomo, F. Patrizi: *Automatic service composition via simulation.* Int. Journal of Foundations of Computer Science 19 (2008) 429–451.
4. Berardi, D., M. Pistore, P. Traverso: *Automatic Web service composition by on-the-fly belief space* *search.* In: Proceedings of ICAPS’06, (2006) 358–361.
5. Foster, H.: *A Rigorous Approach to Engineering Web Service Compositions.* Thesis of the University of London (2006).
6. Foster, H., S. Uchitel, J. Magee, J. Kramer: *WS-Engineer: a model-based approach to engineering Web service compositions and choreography.* In: Test and Analysis of Web Services. Springer (2007) 87–119.
7. Fu, X., T. Bultan, J. Su: *Analysis of interacting BPEL Web services.* In: Int. World Wide Web Conference. ACM (2004) 621–630.
8. Héam, P.-C., O. Kouchnarenko, J. Voinot: *How to handle QoS aspects in Web services substitutivity veriﬁcation.* In: Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE (2007) 333– 338.
9. Hoffman, J., P. Bertoli, M. Pistore: *Web service composition as planning, revisited.* In: Proceeedings of AAAI’07 AAAI (2007) 1013–1018.
10. Laroussinie, F., Ph. Schnoebelen: *The state explosion problem from trace to bisimulation equivalence.*

In: Foundations of Software Science and Computational Structures. Springer (2007) 192–207.

1. Marconi, A., M. Pistore, P. Poccianti, P. Traverso: *Automated Web service composition at work: the amazon/mps case study* In: Proceedings of ICWS’07, IEEE, (2007) 767–774.
2. Marconi, A., M. Pistore, P. Traverso: *Automated composition of Web services: the astro approach* In: IEEE Data Engineering Bulletin 31 (2008) 23–26.
3. Meyer, A., L. Stockmeyer: *The equivalence problem for regular expressions with squaring requires exponential space.* In: Switching and Automata Theory. IEEE (1972) 125–129.
4. Mitra, S., R.J. Kumar, S. Basu: *Automates CHoregrapher Synthesis for Web service composition using I/O Automata.* In: Proceedings of ICWS’07 IEEE (2007) 364–371.
5. Muscholl, A., I. Walukiewicz: *A lower bound on Web services composition.* In: Foundations of Software Science and Computational Structures. Springer (2007) 274–286.
6. Pathak, J., S. Basu, R. Lutz, V. Honavar: *Parallel web service composition in moscoe: A choregraphy based approach.* In: Proceedings of ECOWS’96, IEEE (2006) 3–12.
7. Pistore, M., A. Marconi, P. Bertoli, P. Traverso: *Automated composition of Web services by planning at the knowledge level.* In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (2005) 1252–1259.
8. Pistore, M., P. Traverso, P. Bertoli: *Automated composition of Web services by planning in asynchronous domains.* In: Proceedings of ICAPS’05, AAAI, (2005) 2–12.
9. Pistore, M., P. Traverso, P. Bertoli, A. Marconi: *Automated synthesis of composite BPEL4WS web services..* In: Proceedings of ICWS’05, IEEE, (2005) 293–301.
10. Rabinovich, A.: *Complexity of equivalence problems for concurrent systems of ﬁnite agents.* In: Information and Computation (1997). volume 139, 111–129.
11. Sardina, S., F. Patrizi, G. De Giacomo: *Behaviour composition in the presence of failure.* In: Proceedings of KR’08, AAAI, (2008). 640–650.
12. Sawa, D.: *Equivalence Checking of Non-flat Systems Is EXPTIME-Hard.* In: Int. Conf. on Concurrency Theory. Springer (2003), 237–250.
13. Singh, M., M. Huhns: *Service-Oriented Computing. Semantics, Process, Agents.* Wiley (2005).
14. Trainotti, M., M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, P. Traverso: *Sipporting composition and execution of Web services.* In: Proceedings of ICSOC’05, Springer, (2005). 495–501.