

Electronic Notes in Theoretical Computer Science 260 (2010) 109–123

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

Component Simulation-based Substitutivity Managing QoS Aspects [1](#_bookmark0)

# Pierre-Cyrille Héam[2](#_bookmark0) Olga Kouchnarenko[2](#_bookmark0) Jérôme Voinot [2](#_bookmark0)

*INRIA-CASSIS* *LIFC*

*University of Franche-Comté*

*16 route de Gray, 25030 Besançon Cedex, France*

Abstract

Several scientific bottlenecks have been identified in existing component-based approaches. Among them, we focus on the identification of a relevant abstraction for the component expression and verification of properties like substitutivity: When is it possible to formally accept or reject the substitution of a component in a composition? This paper suggests max/plus automata to tackle this problem when considering a new factor – Quality of Service (QoS). Four notions of simulation-based substitutivity managing QoS aspects are proposed, and related complexity issues on max/plus automata are investigated.

*Keywords:* Substitutivity, Component, Simulation, max/plus automata, Quality of Service

# Introduction

This paper is dedicated to the verification of substitutivity of components modelled by max/plus automata while considering a new factor – Quality of Service (QoS). In this context modelling and verifying both functional and non-functional properties is possible. For these verification problems, we provide new theoretical decidability results.

Component-based development provides significant advantages – portability, adaptability, re-usability, etc. – when developing, e.g., Java Card smart card ap- plications or when composing Web services within Service Component Architecture (SCA) – a relatively new initiative advocated by users of Java technology. Several scientific bottlenecks have been identified in existing component-based approaches. Among them, we focus on the identification of a relevant abstraction for the compo- nent expression and verification. When is it possible to accept or reject the substi-

1 This work is partially funded by the French project ARA COPS.

2 {heampc,kouchna,voinot}@lifc.univ-fcomte.fr

1571-0661© 2009 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.2009.12.034

tution of a component in a composition? Moreover, with the increasing importance of QoS in the design of component-oriented applications, like Web services, it is of great interest for users or developers to be able to determine, possibly dynamically, that a Web service performs the same tasks as another possibly failing service, with comparable/higher quality.

There is a lot of well-known component models and frameworks for developing service-oriented applications. Practical aspects of dynamic service selection and its implementation are presented e.g. in [[4](#_bookmark14)] reporting on the ANSO project bringing together major European industrial actors. Formal aspects of Web service composi- tions have been studied when using guarded automata in [[11](#_bookmark24)], or finite state machines in [[2](#_bookmark15)], or a finite state process algebra [[12](#_bookmark25)]. However, to our knowledge, for verify- ing Web services properties like substitutivity, none of these models allows taking service costs into account. To make up for this lack, this paper proposes and uses a formal model, called max/plus automata. This paper gives formal definitions of four – (partial) substitutivity and (partial) strong substitutivity – problems based on a simulation of automata taking path costs into account. New decision/complexity results for different classes of max/plus automata are then presented for these sub- stitutivity problems.

*Related works.* Weighted automata – an extension of max/plus automata – is a formalism widely used in computer science for applications in images compres- sion [[19](#_bookmark32),[22](#_bookmark35)], speech-to-text processing [[26](#_bookmark38),[3](#_bookmark16)] or discrete event systems [[13](#_bookmark26)]. These large application areas make them intensively studied from the theoretical point of view [[23](#_bookmark36),[33](#_bookmark46),[14](#_bookmark27),[21](#_bookmark34)]. See [[6](#_bookmark19)] for more detail on max/plus automata.

There are numerous works dealing with component substitutivity or interop- erability [[28](#_bookmark39),[9](#_bookmark22),[8](#_bookmark21)]. Our work is close to that in [[9](#_bookmark22)], where the authors addressed component substitutability using equivalences between component-interaction au- tomata, which are defined with respect to a given set of observable labels. In the present work, in addition to a set of labels, path costs are taken into account when comparing max/plus automata.

Different solutions have been proposed to allow taking QoS into account while specifying Web services and their compositions [[24](#_bookmark37),[31](#_bookmark44),[10](#_bookmark23),[7](#_bookmark20)]. These approaches are promising, but they are currently W3C submissions, not W3C recommendations [3](#_bookmark1) . That is why following [[15](#_bookmark28)], we proposed in [[16](#_bookmark29)] to extend both BPEL and WSDL specifications with a notion of service costs for being closer to the Web services real- ity. In [[15](#_bookmark28)] the substitutivity problem has been investigated for the trace equivalence and an automatic approach to convert Web services description files into max/plus automata has been provided.

To compare processes or components, trace equivalences are in general not ex- pressive enough and there are stronger equivalence relations permitting to consider deadlocks, livelocks, branching behaviours, causality, etc. Among them, the strong bisimulation equivalence by Milner [[25](#_bookmark40)] and Park [[27](#_bookmark41)] is widely used in computer science because of its numerous advantages: It preserves branching behaviours and,

3 A good introductory reference to the Web services standard is [[17](#_bookmark30)].

consequently, most of dynamic properties; there is a link between the strong bisim- ulation and modal logics [[18](#_bookmark31)]; this is a congruence for a number of composition operators, e.g. parallel composition, prefixing by an action, etc.

Bisimulation relations over max/plus automata were investigated in [[5](#_bookmark18)]. In that paper authors consider that a max/plus automaton simulates another one if it can perform at the same moment the same action with the same weight. Our main purpose is to handle QoS aspects which are global notions over components. This is why in our paper, unlike [[5](#_bookmark18)], weights are related to successful paths of max/plus automata.

*Layout of the paper.* The remainder of the paper is organised as follows. A moti- vating example is given in Sect. [2](#_bookmark2). Section [3](#_bookmark4) recalls max/plus automata and defines four simulation-oriented substitutivity notions based on them. The verification is- sues on components substitutivity are presented in Sect. [4](#_bookmark6) and [5](#_bookmark9) before concluding in Sect. [6](#_bookmark13).

# Motivating Example

We illustrate the substitutivity problem by a characteristic toy example. Let’s con- sider the two following components, represented by finite automata where, to be intuitive [4](#_bookmark3) , communicating actions are labelled with ? or !.

*b*

?*a*

!*c*

*b*

!*c*

C1

?*a*

*b*

!*c*

*b*

C2

* Component C1 works as follows. Action ?*a* encodes that C1 receives a request; at this stage, C1 performs action *b* once or twice (depending on the abstracted value passed through the ?*a* request). Then C1 sends a message !*c* to acknowledge that its task is successfully performed.
* Component C2 works similarly but can perform action *b* as many times as it is required.

Obviously, the C1 component can be functionally substituted by C2. Further, when considering e.g. energy costs over components represented by finite automata C3 and C4 below, the cost of each action is put on each transition, as presented below.

4 In the rest of the paper, the actions are not partitioned into input/output/internal actions.

*b*|2

?*a*|1

!*c*|3

*b*|2

!*c*|3

C3

?*a*|1

*b*|1

!*c*|3

*b*|4

C4

For both C3 and C4, receiving a request ?*a* costs 1 energy unit and sending the confirmation !*c* costs 3 energy units. However, for C3 each action *b* costs 2 energy units. For C4, performing the first *b* action costs only 1 energy unit but other *b* actions cost 4 energy units. The intuition behind this modelling is as follows. C3 has a low-cache memory allowing it to locally compute action *b* twice. C4 has a high performance low-cache memory that allows it to locally compute action *b* with a cost of 1 energy unit but only once. C4 also has a local hard drive that makes more *b* computations possible. However, reading and writing on hard drives has a high energy cost of 4 energy units. In this situation, we do not want to say that C4 can substitute C3 since performing ?*abb*!*c* on C3 has the cost of 8 energy units whereas the same sequence of actions costs 9 energy units on C4.

# Theoretical Background

In this paper, Σ denotes a finite set of actions. We first introduce the notion of max/plus automata.

Definition 3.1 A *finite max/plus automaton* A over Σ is a quintuplet

A = (*Q,* Σ*, E,I,F* )

where *Q* is the finite set of states, *E* ⊆ *Q* ×Σ ×Z × *Q* is the set of transitions, *I* ⊆ *Q*

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

Notice that there is a restriction on *E*: for every action *a*, every pair of states *p, q*,

*a,c*

there exists in *E* at most one transition of the form (*p, a, c, q*), also written *p* −→*A q*.

Now we formally define an execution of a max/plus automaton and related notions.

A *partial execution* or a *path* of a finite max/plus automaton A is a sequence *π* = (*p*0*, a*0*, c*0*, q*0)*,* (*p*1*, a*1*, c*1*, q*1)*,... ,* (*pn, an, cn, qn*) of transitions of A such that for every 0 ≤ *i < n*, *qi* = *pi*+1. If we add the conditions: *p*0 is an initial state, *qn* is a final state, then we call *π* an *execution* or *a successful path*. The *trace tr* of the

(partial) execution *π* is the word *a*0*a*1 *... an*, and the *cost* of the (partial) execution

*π* is the sum of the *ci*’s: cost*A*(*π*)= Σ*n ci*. A state *p* of a max/plus automaton is

*i*=0

*accessible/reachable* (resp. *co-accessible/co-reachable*) if there exists a path from an initial state to *p* (resp. from *p* to a final state). Basically, given A, *L*(A) denotes its set of execution traces.

An automaton is *trim* if its states are all both accessible and co-accessible. It is well known that for every automaton A, there exists a trim automaton with the same set of successful executions. Moreover, computing this trim automaton can be done in polynomial time. An automaton A is *finitely ambiguous* if there exists a positive integer *k* such that for every word *w* there exists at most *k* successful paths in A labelled by *w*.

Let A1 = (*Q*1*, A, E*1*, I*1*, F*1) and A2 = (*Q*2*, A, E*2*, I*2*, F*2) be two automata. A binary relation ≤*A*1*,A*2 ⊆ *Q*1 × *Q*2 is a simulation if (*p*1*, p*2) ∈≤*A*1 *,A*2 implies, for all *a* in *A* and all *c*1 in Q,

* for every *q*1 ∈ *Q*1, if (*p*1*, a, c*1*, q*1) ∈ *E*1 then there exist *q*2 ∈ *Q*2 and *c*2 ∈ Q such that (*p*2*, a, c*2*, q*2) ∈ *E*2 and (*q*1*, q*2) ∈≤*A*1 *,A*2 , and
* if *p*1 is final, then *p*2 is final too.

If there is no ambiguity on A1 and A2, we just say that *p*2 ≤- simulates *p*1, written *p*1 ≤ *p*2, when there is a simulation containing (*p*1*, p*2). It is easy to see that the largest simulation on *Q*1 × *Q*2 exists. To simplify the notations, the largest simulation on *Q*1 × *Q*2 is also denoted by ≤*A*1 *,A*2 .

The above relation is extended to paths of A1 and A2 in the following way: an execution *π*2 of A2 ≤- simulates an execution *π*1 of A1 if and only if they have the same label (and consequently the same length) and for every *i*, *π*1[*i*] ≤ *π*2[*i*]. Finally, we write A1 ≤ A2 if for every co-accessible initial state *i*1 of A1 there exists an initial state *i*2 of A2 such that *i*1 ≤ *i*2. For our example in Sect. [2](#_bookmark2), it is easy to see that C3 ≤ C4.

In the rest of the paper, the actions are not partitioned into in- put/output/internal actions (see [[15](#_bookmark28),[16](#_bookmark29)] for a conversion of Web services description files into max/plus automata). Consequently, the communications are not covered by *τ* -transitions. The only *τ* -transitions appear when translating the assign and empty BPEL activities into max/plus automata. Those transitions can be abstracted without loosing behaviours since they do not take part in services exchanges.

In this setting, i.e. without silent *τ* -transitions, the ≤-simulation relation is com- patible with a sequential composition operator modeling e.g. the sequence BPEL structured activities, and with an asynchronous parallel composition operator im- plementing e.g. the flow BPEL structured activities. Both operators are important

in practice since they allow building complex services by a composition of services.

Definition 3.2 Let A1 = (*Q*1*, A*1*, E*1*, I*1*, F*1) and A2 = (*Q*2*, A*2*, E*2*, I*2*, F*2) be two automata. The sequential composition of A1 and A2, denoted A1*.*A2, is an automa- ton A12 = (*Q*12*, A*12*, E*12*, I*12*, F*12) where

* *Q*12 = {*p*1; *p*2 | ∀*p*1 ∈ *Q*1*, p*2 ∈ *Q*2}∪ *Q*2,
* *A*12 = *A*1 ∪ *A*2,
* *I*12 = {*p*1; *p*2 | ∀*p*1 ∈ *I*1*, p*2 ∈ *I*2},
* *F*12 = *F*2,

and where the transition relation *E*12 obeys the following rules:

*a*1*,c*1

[SEQ1] *p*1 −→*A*1 *q*1

*a*1*,c*1

*p*1; *p*2 −→*A*1 *.A*2 *q*1; *p*2

*a*2*,c*2

[SEQ2] *p*2 −→*A*2 *q*2

*a*2*,c*2

*p*1; *p*2 −→*A*1 *.A*2 *q*2

*p*1 ∈ *F*1

This definition means that all moves of sequential composition are moves of either A1 or of A2 if A1 is in a final state. It is easy to establish the following compatibility result whose proof is rather basic for strong simulation relations handling deadlocks.

Proposition 3.3 *Let* A1*,* A2*,* A3 *be three automata, let* A1 ≤ A3*. One has:*

1. A1*.*A2 ≤ A3*.*A2
2. A2*.*A1 ≤ A2*.*A3

Definition 3.4 Let A1 = (*Q*1*, A*1*, E*1*, I*1*, F*1) and A2 = (*Q*2*, A*2*, E*2*, I*2*, F*2) be two automata. The parallel composition of A1 and A2, denoted A1  A2, is an automaton A12 = (*Q*12*, A*12*, E*12*, I*12*, F*12) where

| *p*

| *p*

∈*/*

* *Q*12 = {*p*1 *p*2 | ∀*p*1 ∈ *Q*1*, p*2 ∈ *Q*2} ∪ {*pi*

12

12

12

12

*i* ∈*/* (*Q*1 ∪ *Q*2)} ∪ {*pf f*

(*Q*1 ∪ *Q*2)},

* *A*12 = *A*1 ∪ *A*2 ∪ {*ϵ*},
* *I*12 = {*pi*

12

* *F*12 = {*pf*

12

| ∃*pi*

| ∃*pf*

12

12

∈ *Q*12 \ (*Q*1 ∪ *Q*2)},

∈ *Q*12 \ (*Q*1 ∪ *Q*2)},

and where the transition relation *E*12 obeys the following rules:

[START]

*i є,*0

*a*1*,c*1

*pk* ∈ *Ik* [PAR1] *p*1 −→*A*1 *q*1



*p*1 *p*2 −→*A*1 *A*2 *q*1 *p*2

*a*1*,c*1

*p*12 −→*A*1 *A*2 *p*1 *p*2

*a*2*,c*2

[PAR2] *p*2 −→*A*2 *q*2



*p*1 *p*2 −→*A*1 *A*2 *p*1 *q*2

*a*2*,c*2

[END]

*є,*0

*f qk* ∈ *Fk*

*q*1 *q*2 −→*A*1  *A*2 *p*12

This definition means that all moves of parallel composition are moves of either A1 or of A2, or an initial/ending move of both A1 and A2. The following proposition claims that the ≤-simulation is compatible with the parallel composition above.

Proposition 3.5 *Let* A1*,* A2*,* A3*,* A4 *be four automata, such that* A1 ≤ A3 *and*

A2 ≤ A4*. We have:*

1. A1  A2 ≤ A3  A4
2. A2 A1 ≤ A4 A3
   1. *Modelling Substitutivity*

A problem occurring while managing Web services is to determine that a Web service performs the same tasks as another possibly failing service, with comparable or higher quality. More formally, for two Web services modelled by their max/plus

automata A1 and A2, the problem is to decide whether A2 can have the same behaviour as A1 with a similar or higher quality. To address this problem, four notions of simulation-based substitutivity managing QoS aspects are proposed in this section.

The notion of substitutivity means that a service *S*1 can be substituted by a service *S*2 if *S*2 has a way to act as *S*1 and the *cost* of this way is comparable or better that the cost in *S*1. The notion of strong substitutivity means that a service *S*1 can be substituted by a service *S*2 if *S*2 has a way to act as *S*1, and whatever the way chosen by *S*2 to act as *S*1 is, its quality is similar or higher.

Substitutivity Problem

Input: Two automata A1 and A2.

Output: True if for every successful path *π*1 of A1 there exists a successful path *π*2

of A2 such that *π*1 ≤ *π*2 and cost*A*2 (*π*2) ≤ cost*A*1 (*π*1), false otherwise.

Strong Substitutivity Problem

Input: Two automata A1 and A2.

Output: True if for every successful path *π*1 of A1 there exists a successful path *π*2

of A such that *π* ≤ *π* and for every *π* of A such that *π* ≤ *π* , cost (*π* ) ≤

2

1 2

*'*

2

2

1

*'*

2

*A*

2 2

*'*

cost*A*1 (*π*1), false otherwise.

It is sometime fruitful to compare successful executions costs only on subtraces. This leads to the following *partial* substitutivity problems that are similar to the ones above. For these problems, we want to compare parts of executions, not paths that cannot be related to a successful path. Consequently, automata are required to be trim, and comparisons are done for all paths, not only for successful paths.

Partial Substitutivity Problem Input: Two trim automata A1 and A2.

Output: True if for every path *π*1 of A1 there exists a path *π*2 of A2 such that

*π*1 ≤ *π*2 and cost*A*2 (*π*2) ≤ cost*A*1 (*π*1), false otherwise.

Partial Strong Substitutivity Problem

Input: Two trim automata A1 and A2.

Output: True if for every path *π*1 of A1 there exists a path *π*2 of A2 such that

1 2

otherwise.

*π* ≤ *π* and for every *π* of A such that *π* ≤ *π* , cost (*π* ) ≤ cost (*π* ), false

*'*

2

2

1

*'*

*'*

2

*A*

2

2

*A* 1

1

Notice that in the above definitions we choose that cost(*π*2) ≤ cost(*π*1) modeling that the lower is the cost the better is the service, what is intuitive for connection time or financial cost. One can give a dual definition if the lower is the cost the worse is the service by changing cost(*π*2) ≤ cost(*π*1) into cost(*π*2) ≥ cost(*π*1). All notions, algorithms, etc. described in this paper may be trivially adapted to this dual definition. In order to not overload the reader, we do not consider that case.

We end this section by recalling some results on decision procedures for finite max/plus automata.

Theorem 3.6 *Given two max/plus automata* A1 *and* A2*, it is*

* *undecidable to test whether for every u* ∈ *L*(A1)*,* cost*A*1 (*u*) ≤ cost*A*2 (*u*) *[*[*23*](#_bookmark36)*]; the same problem is decidable if* A1 *and* A2 *are both finitely ambiguous [*[*14*](#_bookmark27)*,*[*33*](#_bookmark46)*],*
* *undecidable to test whether for every u* ∈ *L*(A1)*, there exists an execution π of label u in* A1 *such that* cost*A*1 (*π*) ≥ 0 *(resp.* cost*A*1 (*π*) ≤ 0*) [*[*23*](#_bookmark36)*],*
* *decidable in polynomial time to test whether for every u* ∈ *L*(A1)*,* cost*A*1 (*u*) ≤

cost*A*2 (*u*) *if* A1 *and* A2 *are both finitely ambiguous [*[*14*](#_bookmark27)*,*[*33*](#_bookmark46)*],*

* *decidable in polynomial time to test whether* A1 *is finitely ambiguous [*[*34*](#_bookmark47)*].*
* *PSPACE-complete to decide whether L*(A1) ⊆ *L*(A2) *[*[*1*](#_bookmark17)*].*

# Strong Substitutivity Problems

This section provides decidability results for the (partial) strong substitutivity prob- lems.

Lemma 4.1 *One has* A1 ≤ A2 *if and only if for every successful path π*1 *of* A1

*there exists a successful path π*2 *of* A2 *such that π*1 ≤ *π*2*.*

Proof Assume first that for every successful path *π*1 of A1 there exists a successful path *π*2 of A2 such that *π*1 ≤ *π*2. Let *i*1 be a co-accessible state of A1. By definition of co-accessibility, there exists a successful path *π*1 in A1 starting from *i*1. By hypothesis, there exists a successful path *π*2 of A2 such that *π*1 ≤ *π*2. Therefore, *π*1[1] ≤ *π*2[1]. But *π*1[1] = *i*1 and since *π*2 is a successful path, *π*2[1] is an initial state of A2. Consequently, A1 ≤ A2.

Assume now that A1 ≤ A2. Let *π*1 be a successful path of A1. Since *π*1[1] is an initial state and since A1 ≤ A2, there exists an initial state *q*1 in A2 such that *π*1[1] ≤ *q*1. Therefore, if we denote by (*π*1[1]*, a*1*, c*1*, π*1[2]) the first transition of *π*1, there exists a state *q*2 in A2 and *d*1 ∈ Z, such that (*q*1*, a*1*, d*1*, q*2) is a transition of A2 and *π*1[2] ≤ *q*2. Iterating this construction, one can, by a direct induction, build a successful path *π*2 of A2 such that *π*1 ≤ *π*2, which concludes the proof.

Theorem 4.2 *The strong substitutivity problem is P-complete.*

Proof Let A1 = (*Q*1*, A, E*1*, I*1*, F*1) and A2 = (*Q*2*, A, E*2*, I*2*, F*2) be two automata. We denote by B the automaton (*Q, A, E, I, F* ) where

- *Q* = {(*q*1*, q*2) ∈ *Q*1 × *Q*2 | *q*1 ≤ *q*2},

- *E* = {((*p*1*, p*2)*, a, c,* (*q*1*, q*2)) | (*p*1*, a, c*1*, q*1) ∈ *E*1*,* (*p*2*, a, c*2*, q*2) ∈ *E*2*, c* = *c*1 −

*c*2*, a* ∈ *A*},

- *I* = (*I*1 × *I*2) ∩ *Q* and *F* = (*F*1 × *F*2) ∩ *Q*.

We claim that A1 and A2 satisfy the strong substitutivity problem if and only if A1 ≤ *A*2 and for every successful path *π* of B, cost*B*(*π*) ≥ 0.

* Assume that A1 and A2 satisfy the strong substitutivity problem. By Lemma [4.1](#_bookmark7), for every successful path of A1 there exists an ≤-related path in A2. Thus A1 ≤ A2. Consider now a successful path *π* in B,

*π* = (*p*0*, a*1*, α*1*, p*1)*,* (*p*1*, a*2*, α*2*, p*2) *...* (*pn−*1*, an, αn, pn*)*.*

By definition of B, there exist *p*0*, p*1*,... , pn* states of A1, *q*0*, q*1*,... , qn* states of

A2, integers *c*1*, c*2*,... , cn, d*1*, d*2*,... , dn* such that

* *π*1 = (*p*0*, a*1*, c*1*, p*1)*,* (*p*1*, a*2*, c*2*, p*2)*,... ,* (*pn−*1*, an, cn, pn*) is a successful path in

A1,

* *π*2 = (*q*0*, a*1*, d*1*, q*1)*,* (*q*1*, a*2*, d*2*, q*2)*,... ,* (*qn−*1*, an, dn, qn*) is a successful path in

A2,

* for every 1 ≤ *i* ≤ *n*, *αi* = *ci* − *di*,
* for every 0 ≤ *i* ≤ *n*, *pi* = (*pi, qi*) and *pi* ≤ *qi*.

Thus, one has *π*1 ≤ *π*2. Therefore, since A1 and A2 satisfy the strong substitu- tivity problem, the following inequality holds:

*n n*

Σ *di* ≤ Σ *ci.*

*i*=1 *i*=1

Consequently,

Σ *αi* ≥ 0*.*

*i*=1

*n*

* Assume now that A1 and A2 satisfy A1 ≤ A2 and for every successful path *π* of

B, cost*B*(*π*) ≥ 0.

Since A1 ≤ A2, by Lemma [4.1](#_bookmark7), for every successful path in A1 there exists a

≤-related successful path in A2.

Finally, consider two successful paths

*π*1 = (*p*0*, a*1*, c*1*, p*1)*,* (*p*1*, a*2*, c*2*, p*2)*,... ,* (*pn−*1*, an, cn, pn*)

in A1 and

*π*2 = (*q*0*, a*1*, d*1*, q*1)*,* (*q*1*, a*2*, d*2*, q*2)*,... ,* (*qn−*1*, an, dn, qn*)

in A2 such that *π*1 ≤ *π*2.

By definition there exists an successful path *π* in B,

*π* = (*p*0*, a*1*, α*1*, p*1)*,* (*p*1*, a*2*, α*2*, p*2) *...* (*pn−*1*, an, αn, pn*)*.*

such that

* for every 1 ≤ *i* ≤ *n*, *αi* = *ci* − *di*,
* for every 0 ≤ *i* ≤ *n*, *pi* = (*pi, qi*) and *pi* ≤ *qi*. Moreover, by hypotheses, one has cost(*π*) ≥ 0:

cost(*π*)= Σ *αi* ≥ 0*.*

*n*

*i*=1

*n n*

Consequently,

Σ *di* ≤ Σ *ci.*

*i*=1 *i*=1

It follows that cost*A*2 (*π*2) ≤ cost*A*1 (*π*1), proving the claim.

Deciding whether A1 ≤ A2 is known to be P-complete [[29](#_bookmark42),[30](#_bookmark43)]. Now deciding whether for every successful path *π* of B, cost*B*(*π*) ≥ 0 is a basic polynomial problem on weighted graphs which can be solved for instance by Bellman-Ford’s algorithm.

The P-completeness is trivially obtained using the claim on automata with nil weights and the P-completeness of testing whether A1 ≤ A2.

Theorem 4.3 *The partial strong substitutivity problem is P-complete.*

Proof Let A1 and A2 be two trim automata. Let B be the automaton constructed as in the proof of Theorem [4.2](#_bookmark8). We claim that A1 and A2 satisfy the partial strong substitutivity problem if and only if A1 ≤ *A*2 and if every transition of B has a positive weight.

The proof is quite similar to the one of Theorem [4.2](#_bookmark8): if A1 and A2 satisfy the partial strong substitutivity problem, then using the property on paths of length 1, each transition of B has to be positively weighted. Conversely, if every transition of B has a positive weight, it is clear by a direct induction on paths lengths, that A1 and A2 satisfy the partial strong substitutivity problem.

The P-completeness is also trivially obtained using the claim on automata with

nil weights and the P-completeness of testing whether A1 ≤ A2.

# Substitutivity Problems

This section provides decidability results for the (partial) substitutivity problem.

Theorem 5.1 *The substitutivity problem is polynomial time decidable if* A2 *is finitely ambiguous.*

Proof Let A1 = (*Q*1*,* Σ*, E*1*, I*1*, F*1) be a max/plus automaton and A2 = (*Q*2*,* Σ*, E*2*, I*2*, F*2) a finitely ambiguous max/plus automaton. Set A3 = (*Q*1*,* Σ × *Q*1 × *Q*1*, E*3*, I*1*, F*1) and A4 = (*Q*2*,* Σ × *Q*1 × *Q*1*, E*4*, I*2*, F*2) where:

* *E*3 = {(*p,* [*a, p, q*]*, c, q*) | (*p, a, c, q*) ∈ *E*1},
* *E*4 = {(*p,* [*a, r, s*]*, c, q*) | (*p, a, c, q*) ∈ *E*2*,* ∃*x* ∈ Z*,* (*r, a, x, s*) ∈ *E*1 *, r,s* ∈

*Q*1and *r* ≤ *p* and *s* ≤ *q*}.

Notice that A3 is unambiguous and that A4 is finitely ambiguous. Indeed, if *u* = [*a*1*, q*1*, q*2][*a*2*, q*2*, q*3] *...* [*an, qn, qn*+1] is accepted by A3, then there is a unique execution (*q*1*, a*1*, c*1*, q*2) *...* (*qn, an, cn, qn*+1) labelled by *u* because of restric- tion on *E* in Sect. [3](#_bookmark4). Now assume that A2 is *l*-ambiguous and that the word *u* = [*a*1*, q*1*, q*2][*a*2*, q*2*, q*3] *...* [*an, qn, qn*+1] is accepted by A4. Since there are at most *l* executions in A2 accepting *a*1*a*2 *... an*, there is at most *l* executions in A4 accepting

*u*. Thus A4 is finitely ambiguous.

Let B = A3 × (−A4), where −A4 is obtained from A4 by multiplying the weight of each transition by −1.

We claim that A1 and A2 satisfy the substitutivity problem if and only if A1 ≤ A2

and for every *u* ∈ *L*(B), there exists an execution *π* in B such that cost*B*(*π*) ≥ 0.

(⇒) Assume first that A1 and A2 satisfy the substitutivity problem. Then A1 ≤ A2.

Now let *u* ∈ *L*(B).

By definition of the product, one also has *u* ∈ *L*(A3). Consequently, there exists an execution *π*3 in A3 of label *u* of the form

*π*3 = (*q*1*,* [*a*1*, q*1*, q*2]*, c*1*, q*2)*,* (*q*2*,* [*a*2*, q*2*, q*3]*, c*2*, q*3) *...* (*qn,* [*an, qn, qn*+1]*, cn, qn*+1)*.*

Consequently, by construction of A3,

*π*1 = (*q*1*, a*1*, c*1*, q*2)*,* (*q*2*, a*2*, c*2*, q*3) *...* (*qn, an, cn, qn*+1)

is an execution in A1.

Since A1 and A2 satisfy the substitutivity problem, there exists an execution

*π*2 in A2 of label *a*1*a*2 *... an* such that

cost*A*2 (*π*2) ≤ cost*A*1 (*π*1) and *π*1 ≤ *π*2*.* (1)

Set

*π*2 = (*p*1*, a*1*, d*1*, p*2)*,* (*p*2*, a*2*, d*2*, p*3) *...* (*pn, an, dn, pn*+1)*.*

Now, by construction of A4,

*π*4 = (*p*1*,* [*a*1*, q*1*, q*2]*, d*1*, p*2)*,* (*p*2*,* [*a*2*, q*2*, q*3]*, d*2*, p*3) *...* (*pn,* [*an, qn, qn*+1]*, dn, pn*+1)

is an execution of A4. Since cost*A*2 (*π*2)= cost*A*4 (*π*4) and cost*A*1 (*π*1)= cost*A*3 (*π*3) and by ([1](#_bookmark10)), the execution *π* in B corresponding to *π*3 and *π*4 has label *u* and a positive cost.

(⇐) Let assume now that A1 and A2 satisfy A1 ≤ A2 and for every *u* ∈ *L*(B), there exists an execution *π* in B such that cost*B*(*π*) ≥ 0.

Let

*π*1 = (*q*1*, a*1*, c*1*, q*2)*,* (*q*2*, a*2*, c*2*, q*3) *...* (*qn, an, cn, qn*+1)

be an execution of A1. By construction of A3, one has in A3 the following execu- tion

*π*3 = (*q*1*,* [*a*1*, q*1*, q*2]*, c*1*, q*2)*,* (*q*2*,* [*a*2*, q*2*, q*3]*, c*2*, q*3) *...* (*qn,* [*an, qn, qn*+1]*, cn, qn*+1)*.*

Consequently, since A1 ≤ *A*2, there exists a successful path *π*4 in A4 such that *π*3 ≤ *π*4. It follows that *u* = [*a*1*, q*1*, q*2][*a*2*, q*2*, q*3] *...* [*an, qn, qn*+1] is in *L*(B). By hypothesis, there exists an execution *π* in B of label *u* such that

Let *π'*

3

and *π'*

4

cost*B*(*π*) ≥ 0*.* (2)

be the corresponding executions of respectively A3 and A4 cor-

responding to *π*. Using ([2](#_bookmark11)), one has:

cost*A*4 (*π'* ) ≤ cost*A*3 (*π'* )*.*

4 3

Therefore, since A3 is unambiguous, *π*3 = *π'* and one has:

3

Set

4

cost*A*4 (*π'* ) ≤ cost*A*3 (*π*3)*.* (3)

*π*4 = (*p*1*,* [*a*1*, q*1*, q*2]*, d*1*, p*2)*,* (*p*2*,* [*a*2*, q*2*, q*3]*, d*2*, p*3) *...* (*pn,* [*an, qn, qn*+1]*, dn, pn*+1)*.*

By construction of A4, there exists an execution *π*2 of A2 of the form:

*π*2 = (*p*1*, a*1*, d*1*, p*2)*,* (*p*2*, a*2*, d*2*, p*3) *...* (*pn, an, dn, pn*+1)*.*

Since cost*A*4 (*π*4)= cost*A*2 (*π*2) and by ([3](#_bookmark12)) one has:

cost*A*2 (*π*2) ≤ cost*A*3 (*π*3)*.*

Since by construction *π*2 ≤ *π*1, the proof of the claim is completed.

This finishes the proof of the theorem, the polynomial time decidability resulting from Theorem [3.6](#_bookmark5).

Theorem 5.2 *The partial substitutivity problem is decidable in polynomial time.*

Proof Let A1 and A2 be two trim automata. We claim that automata A1 and A2 satisfy the partial substitutivity problem if for every transition (*p*1*, a, c*1*, q*1) of A1 there exists a transition (*p*2*, a, c*2*, q*2) of A2 such that *c*2 ≤ *c*1, *p*1 ≤ *p*2 and *q*1 ≤ *q*2. Indeed, if A1 and A2 satisfy the partial substitutivity problem then, using the property on paths of length 1, one has the desired result. Conversely, if for every

transition (*p*1*, a, c*1*, q*1) of A1 there exists a transition (*p*2*, a, c*2*, q*2) of A2 such that *c*2 ≤ *c*1, *p*1 ≤ *p*2 and *q*1 ≤ *q*2, a direct induction on paths lengths shows that A1 and A2 satisfy the partial substitutivity problem.

Computing relation ≤ can be done in polynomial time. Next, it suffices to check the above property by a simple walk of the transitions list.

# Conclusion

In this paper we proposed to manage both functional and non functional aspects of components. To sum up, this paper exposes how max/plus automata can be used to address substitutivity issues in the context of component-based applications. We defined four kinds of substitutivity managing QoS aspects. Several complexity results were provided.

On the implementation side, following our work in [[15](#_bookmark28),[16](#_bookmark29)], an automatic transla- tion of Web services description files into max/plus automata has been implemented, as well as an algorithm for the trace-based substitutivity problem. The prototype has been tested on a Dell Latitude D600 with an Intel Pentium M 1.4Ghz and 512Mo of RAM. The tests have been performed on different versions of a movie store example, a book store example provided by Oracle [[20](#_bookmark33)], and the classical loan approval example. For the book store example specified in BPEL/WSDL, the au- tomaton is built in less than 3 seconds, and it has 67 states and 101 transitions due to the flow activities putting in parallel three activities. We intend to continue the implementation and extend that work to simulation-based substitutivity problems presented in this paper. It would necessitate considering, e.g., correlation sets which are currently not supported in the translator.

To go further, more expressive formalisms like Mealy machines, process algebra or Petri nets would provide more precise component abstractions. In this context, ex- tending substitutivity definitions to these formalisms is easy, but algorithmic studies have to be performed again. In other respects, the problem of whether the substi- tutivity problem is decidable in the general case, remains open. In the context of the trace-based substitutivity, this problem is undecidable. We conjecture the same result holds for the simulation-based substitutivity.

Polynomial time decidability shows the substitution notion presented in the pa- per is reasonable and practical. For example, it would be possible to include into consideration the fact that performance/reliability metrics of a component service are not only a function on the service or the service trace, but also on parameters such as the execution environment, the performance/reliability of externally called services and the usage profile. In fact, the decidability being polynomial time, it could be possible to apply the algorithms for each of that parameters.

In a more general context, modelling quantitative aspects is of great interest for modelling and verifying component-based applications. Work continues on mod- elling and verifying properties simpler than substitutivity, and on considering other applications, e.g. business protocols. In addition, the proposed framework seems to be well-adapted to handle energy dispersion associated with actions, that is partic- ularly relevant for embedded systems or sensor networks (see for instance [[32](#_bookmark45)]).

# Acknowledgement

We would like to thank the anonymous referees for their interesting and helpful comments and suggestions to improve this work.

# References

1. A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms*, pages 395–400. Addison-Wesley, 1974.
2. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *VLDB’05, Trondheim, Norway*, 2005.
3. A.L. Buchsbaum, R. Giancarlo, and J. Westbrook. An approximate determinization algorithm for weighted finite-state automata. *Algorithmica*, 30(4):503–526, 2001.
4. A. Bottaro and R. S. Hall. Dynamic contextual service ranking. In *6th International Symposium on* *Software Composition (SC 2007), Braga, Portugal*, pages 129–143, 2007.
5. P. Buchholz and P. Kemper. Weak bisimulation for (max/+) automata and related models. *Journal* *of Automata, Languages and Combinatorics*, 8(2):187–218, 2003.
6. J. Berstel and Ch. Reutenauer. *Rational Series and Their Languages*. Springer-Verlag, 1988.
7. F. Baligand, N. Rivierre, and Th. Ledoux. A declarative approach for qos-aware web service compositions. In *ICSOC*, pages 422–428, 2007.
8. S. Chouali, M. Heisel, and J. Souquières. Proving component interoperability with B refinement. *Electr. Notes Theor. Comput. Sci.*, 160:157–172, 2006.
9. I. Cerná, P. Vareková, and B. Zimmerova. Component substitutability via equivalencies of component- interaction automata. *Electr. Notes Theor. Comput. Sci.*, 182:39–55, 2007.
10. A. D’Ambrogrio. A Model-driven WSDL Extension for Describing the QoS of Web Services. In

*ICWS’06, Chicago, Illinois, USA*, 2006.

1. X. Fu, T. Bultan, and J. Su. Analys of Interacting BPEL Web Services. In *WWW’04, New York, NY, USA*, 2004.
2. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Ws-Engineer: A model-based approach to engineering web service compositions and choreography. In *Test and Analysis of Web Services*, pages 87–119. 2007.
3. S. Gaubert. Performance Evaluation of (max,+) Automata. *IEEE Trans. on Automatic Control*, 40(12), 1995.
4. K. Hashiguchi, K. Ishiguro, and S. Jimbo. Decidability of the Equivalence Problem for Finitely Ambiguous Finance Automata. *IJAC*, 12(3), 2002.
5. P.-C. Héam, O. Kouchnarenko, and J. Voinot. How to Handle QoS Aspects in Web Services Substitutivity Verification. In *WETICE’07, Paris, France*, 2007.
6. P.-C. Héam, O. Kouchnarenko, and J. Voinot. Towards formalizing QoS of web services with weighted automata. Technical Report 6218, INRIA, 2007.
7. T. Hauser and U. Löwer. *Web Services. Die Standards*. Galileo Computing, November 2003.
8. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.
9. K. Culik II and P.C. von Rosenberg. Generalized weighted finite automata based image compression.

*J. UCS*, 5(4):227–242, 1999.

1. M. B. Juric. *A Hands-on Introduction to BPEL, Part 2: Advanced BPEL*, 2005.

<http://www.oracle.com/technology/pub/articles/matjaz_bpel2.html>.

1. I. Klimann, S. Lombardy, J. Mairesse, and Ch. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.
2. F. Katritzke, W. Merzenich, and M. Thomas. Enhancements of partitioning techniques for image compression using weighted finite automata. *Theoretical Computer Science*, 313(1):133–144, 2004.
3. D. Krob. The Equality Problem for Rational Series with Multiplicities in the Tropical Semiring is Undecidable. *IJAC*, 4(3), 1994.
4. H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. *Web Service Level Agreement (WSLA) Language Speciﬁcation, Version 1.0*. IBM Corporation, January 2003.
5. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.
6. M. Mohri, F. Pereira, and M. Riley. Weighted automata in text and speech processing. March 28 2005.
7. D. Park. Concurrency and automata on infinite sequences. In *Lecture Notes in Computer Science*, volume 104, pages 167–183. Springer Verlag, 1981.
8. H. W. Schmidt, I. Crnkovic, G. T. Heineman, and J. A. Stafford, editors. *Component-Based Software Engineering, 10th International Symposium, CBSE 2007, Medford, MA, USA, July 9-11, 2007, Proceedings*, volume 4608 of *Lecture Notes in Computer Science*. Springer, 2007.
9. Z. Sawa and P. Jancar. P-hardness of equivalence testing on finite-state processes. In *SOFSEM*, pages 326–335, 2001.
10. Z. Sawa and P. Jancar. Behavioural equivalences on finite-state systems are ptime-hard. *Computers and Artiﬁcial Intelligence*, 24(5), 2005.
11. Min Tian. *QoS integration in Web services with the WS-QoS framework*. PhD thesis, Freie Universitat Berlin, 2005.
12. Xiaoling Wu, Jinsung Cho, Brian J. d’Auriol, and Sungyoung Lee. Energy-aware routing for wireless sensor networks by ahp. In *Software Technologies for Embedded and Ubiquitous Systems*, volume 4761 of *LNCS*, pages 446–455, 2007.
13. A. Weber. Finite-valued Distance Automata. *Theoretical Computer Science*, 134, 1994.
14. A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.