Automatic Quality-of-Service Evaluation in Service-Oriented Computing

Agustín E. Martinez Suñé, Carlos G. Lopez Pombo

Automatic Quality-of-Service evaluation in Service-Oriented Computing *

Agustín E. Martinez Suñé[1] and Carlos G. Lopez Pombo[1,2]

[1] Universidad de Buenos Aires, Facultad de Ciencias Exactas y Naturales, Department of computing.
[2] CONICET–Universidad de Buenos Aires. Instituto de Investigación en Ciencias de la Computación.

Abstract. Formally describing and analysing quantitative requirements of software components might be important in software engineering; in the paradigm of API-based software systems might be vital. Quantitative requeriments can be thought as characterising the *Quality of Service* – QoS provided by a service thus, useful as a way of classifying and ranking them according to specific needs. An efficient and automatic analysis of this type of requirements could provide the means for enabling dynamic establishing of *Service Level Agreements* – SLA, allowing for the automatisation of the *Service Broker*.

In this paper we propose the use of a language for describing QoS contracts based on convex specification, and a two-phase analysis procedure for evaluating contract satisfaction based on the state of the art techniques used for hybrid system verification. The first phase of the procedure responds to the observation that when services are registered in repositories, their contracts are stored for subsequent use in negotiating SLAs. In such a context, a process phase of contract minimisation might lead to great efficiency gain when the second, and recurrent, phase of determining QoS compliance is run.

1 Introduction

Distributed software resulting from paradigms like *Service-oriented Computing* – SOC, and emerging ones like Cloud/Fog computing and the Internet of Things are transforming the world of software systems in order to support applications able to respond and adapt to the changes of their execution environment, giving impulse to what is called the API's economy. The underlying idea of the API's economy is that it is possible to construct software artifacts by composing services provided by third parties and previously registered in repositories.

* Carlos G. Lopez Pombo's research is supported by Universidad de Buenos Aires by grant UBACyT 20020170100544BA, and Consejo Nacional de Investigaciones Científicas y Técnicas by grant PIP 11220130100148CO. The authors want to thank to Marie Skłodowska-Curie Research and Innovation Staff Exchange BehAPI – *Behavioural Application Program Interfaces*.

This envisages a generation of applications running over globally available computational resources and communication infrastructure, which, at run-time, are dynamically and transparently reconfigured by the intervention of a dedicated *Middleware* with the capability to discover and bind a running application with a certain requirement, to a service capable of fulfilling it, subject to the negotiation of a *Service Level Agreement* – SLA, so they can collectively fulfill certain business goals [4].

Requirements have usually been classified in functional and non-functional depending on the nature of the attributes they capture. Functional requirements are usually understood as those describing what the system has to do, while non-functional ones generally express attributes that do not prescribe a particular behaviour but characterise how the system carries out the behaviour described by the functional ones. From non-functional requirements, we propose a further classification by identifying a subset referred to as *quantitative attributes*. We understand quantitative attributes as those that can be interpreted over a particular metric space [1]. This characteristic carries the potential of admitting some form of formal analysis, depending on the expressive power of the language used in specifying requirements over them. The reader shall note that not every metric space has associated formal methods of analysis but, from both practical and theoretical point of view, the real numbers constitute a natural candidate over which many quantitative attributes can be interpreted. As usual, requirements can be used as contracts between software components and satisfaction of such contracts is dealt with by checking whether certain judgement of the form $Pr \vdash Rq$ holds or not, where Pr is the provision contract and Rq is the requirement contract.

From this perspective, those quantitative non-functional requirements formalised over attributes interpreted as real variables might be used to classify functionally equivalent services by the *Quality of Service* – QoS they provide. This means that while services might have the same functional behavior, they could differ on their non-functional one (for example, a service may offer low speed computation at a very low cost while another, functionally equivalent one, might be faster but more onerous), a phenomena that could be useful as a way of classifying and ranking them according to specific needs; a motivation shared with other works like [14].

Identifying and formally describing quantitative properties of a system is not novel, examples of this range from the well known formalisation of the time/space required by an algorithm by means of the asymptotic growth of functions [2, Part. I, Chap. 2], to a very prolific research field dedicated to the verification of hybrid systems. Hybrid systems [6] are dynamic artifacts exhibiting both discrete and continuous behaviour. In general, the continuous aspects of such systems are formalised as constraints over variables taking their values from the real numbers. There is a plethora of analysis techniques that have been proposed for this type of systems, with the vast majority focussing on those aspects laying within the boundary of what is decidable [8]. A relatively new approach to the analysis of hybrid systems' specification, due to Pappas et al. [13], integrates SMT-solving

[10] with convex constraints [5], under the name of *SMC – Satisfiability Modulo Convex Optimization*.

In this work we developed an efficient two phase procedure for evaluating quantitative SLA based on SMC, but adapted to profit from the fact that contracts (both provision Pr and requirement Rq) can be minimised in a preprocessing phase, referred to as *Minimisation through Convex Optimisation – MCO*, when the service is registered in the repository. The expectation is that such preprocessing might produce an efficiency gain when $Pr \vdash Rq$ is checked to evaluate if an SLA is met. Moreover, as finding the minimum size contract requires, at least, enumerating all boolean assignments satisfying a SAT problem, we propose an approach in which contract minimisation can be performed as semantics preserving incremental partial minimizations.

The paper is organised as follows: in Sec. 2 we concentrate both the definition of the formal framework we will use to specify QoS contracts as quantitative requirements, and present our proposal for its analysis, at the same time we state the research questions we want to answer; in Sec. 3 we present the experimental design and results supporting the answers to each of the research questions. Finally in Sec. 4 we draw some conclusions and point out some further lines of research.

2 Formalisation and analysis of QoS contracts

In this section we concentrate on formalising QoS contracts by identifying quantitative non-functional requirements and, in analogy to the continuous elements in hybrid system specification, by interpreting each of the attributes as a real variable. Thereafter, we present state of the art analysis techniques for this kind of formal specifications and our proposal for the optimisation of the procedure based on a preprocess of contract minimisation.

In [13], Pappas et al. adopt *monotone SMC formulae* as specification language; we will refer to this specifications as *convex specification*. Monotone SMC formulae are defined as quantifier-free formulae in conjunctive normal form, with atomic propositions ranging over a subset of the propositional variables and convex constraints.

Definition 1 (Convex specification, Sec. 3.2, [13]). *Let $\mathcal{X}$ be a set of real variables and $\mathcal{P}$ a set of propositional variables, then a* monotone SMC formula *is any formula that can be produced by the following grammar:*

$$
\begin{aligned}
\mathit{formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{literal} \vee\}^* \mathit{literal}) \\
\mathit{literal} &::= p \mid \neg p \mid \top \mid \bot \mid \mathit{conv\_constraint} \text{ , where } p \in \mathcal{P} \\
\mathit{conv\_constraint} &::= \mathit{equation} \mid \mathit{inequality} \\
\mathit{equation} &::= f(x_0, \ldots, x_{arity(f)}) = 0 \\
&\quad \text{, where } f \text{ is an affine function and } x_0, \ldots, x_{arity(f)} \in \mathcal{X}. \\
\mathit{inequality} &::= f(x_0, \ldots, x_{arity(f)}) \ \mathit{relation}\ 0 \\
&\quad \text{, where } f \text{ is a convex function and } x_0, \ldots, x_{arity(f)} \in \mathcal{X}. \\
\mathit{relation} &::= < \mid \leq
\end{aligned}
$$

Then a convex specification *over* $\mathcal{X}$ *and* $\mathcal{P}$ *is a tuple* $\langle\langle \mathcal{X}, \mathcal{P}\rangle, \alpha\rangle$*, where* α *is a monotone SMC formula over* $\mathcal{X}$ *and* $\mathcal{P}$*.*

In the grammar above, *affine_function* and *convex_function* denote, as one could guess, affine and convex functions, respectively. Monotone SMC formulae only admit convex constraints as atoms, in contrast to generic SMC formulae over convex constraints [11], reverse convex constraints are not allowed[3]. As it is mentioned in [13], monotonicity (i.e. the lack of negation to convex constraints) is key to guarantee that whenever a convex specification has a model, it is found by solving one (or more) convex optimization problems.

The following example illustrates a specification written in the language presented in Def. 1.

Example 1 (Service requirement). Let us consider a context of an API-based software application requiring a service accessed and paid for, for the time it is used. In that context we consider as relevant quantitative attributes the cost (formed by: *a. perSec*: the cost (in a given currency) of the service per second elapsed while the communication session is open, and *b. costMb*: the cost (in the same currency) per megabyte of information processed); and execution time (formed by: *a. maxWait*: the maximum waiting time in seconds for the server to effectively attend a request, *b. maxTimeGb*: the maximum amount of time in seconds the service will execute for processing one gigabyte, and *c. netSpeedMb*: an upper bound in seconds for transferring one megabyte). Additionally, costs may change if the system requiring the service has some kind of *promotional code*. Then, such attributes can be formalised by a convex specification $\langle\langle \mathcal{X}, \mathcal{P}\rangle, \alpha\rangle$ where $\mathcal{X} = \{perSec, costMb, maxWait, maxTimexGb, netSpeedMb\}$, $\mathcal{P} = \{promotionMode\}$, and where α is the conjunction of the following formulae[4]:

$$
\begin{array}{l}
0 < maxWait \leq 100 \\
(1000 \leq maxTimexGb \leq 3000) \wedge (0.05 \leq netSpeedMb \leq 0.15) \\
promotionMode \implies 0.0 \leq perSec < 0.1 \\
\neg promotionMode \implies 0.1 \leq perSec \leq 0.3 \\
costMb < 0.5
\end{array}
$$

In [13, Def. 3.4] the authors define the *monotone convex expansion* of a convex specification $\langle\langle \mathcal{X}, \mathcal{P}\rangle, \alpha\rangle$ as a new convex specification $\langle\langle \mathcal{X}, \widehat{\mathcal{P}}\rangle, \widehat{\alpha}\rangle$ where:

- $\widehat{\mathcal{P}} = \mathcal{P} \cup \mathcal{V}$, where $\mathcal{V} = \left\{ v_{f(\vec{x}) \ \mathcal{R} \ 0} \mid f(\vec{x}) \ \mathcal{R} \ 0 \text{ appears in } \alpha \right\}$ and
- $\widehat{\alpha} = \alpha_B \wedge \left[\bigwedge_{v_C \in \mathcal{V}} (\neg v_C \vee C)\right]$, where $\alpha_B = \alpha \,|_C^{v_C}$, the result of substituting every occurrence of an affine/convex constraint C in the monotone SMC formula α by the fresh new propositional variable $v_C \in \mathcal{V}$ associated to C, called the *propositional abstraction* of α.

[3] Note that linear convex constraints could admit negations but, as the negation of a linear convex constraints can be rewritten as a linear convex constraint itself, there is no need for any special treatment.

[4] The reader shall note that there is no impediment in translating the specification below to a formula recognisable by the grammar presented in Def. 1.

Thereafter, in [13, Prop. 3.5], the authors prove that: 1. α and $\widehat{\alpha}$ are equisatisfiable (i.e. for every satisfying assignment for $\widehat{\alpha}$, there exists a satisfying assignment for α) and if $\widehat{\alpha}$ is unsatisfiable, then so is α, 2. any satisfying boolean assignment for α_B reduces the satisfiability problem of $\widehat{\alpha}$ to a conjunction of convex constraints, and 3. the satisfiability problem of $\widehat{\alpha}$ reduces to a finite disjunction (one for every satisfying boolean assignment for α_B) of finite conjunctions of convex constraints.

Roughly speaking, the analysis method proposed by Pappas et al. in [13], sketched in Fig. 1, reduces to enumerating every possible satisfying boolean assignments $\gamma : \widehat{\mathcal{P}} \to \{0, 1\}$ for α_B, and then using a convex constraint solver for testing the feasibility of the set of convex constraints $\{C \mid \gamma(v_C) = 1\}$.

1. Let $\langle \langle \mathcal{X}, \mathcal{P} \rangle, \alpha \rangle$ be a convex specification and α_B the propositional abstraction of α, and a function δ mapping variables in $\mathcal{V}$ to their corresponding convex or affine constraint appearing in α,
2. Feed an SAT-solver with α_B,
3. **if** there exists a satisfying assignment γ, **then**
   1. Feed a convex solver with $\{f(\vec{x}) \ \mathcal{R} \ 0 \mid (\exists v \in \mathcal{V})(\delta(v) = f(\vec{x}) \ \mathcal{R} \ 0 \wedge \gamma(v) = 1)\}$
   2. **if** it is feasible **then**
      1. α **is satisfiable**

      else
      2. Update the SAT-solver information with $\neg minimise(\gamma_\delta)$
      3. **goto** 3,
4. α **is unsatisfiable**

Fig. 1: SMCO analysis procedure

Updating the SAT-solver's clause database by injecting a new clause, as it is done in Line 3(2)2 requires the next assignment to satisfy the clauses in SAT-solver database, plus the new one. Minimising an assignment $\gamma_\mathcal{V}$ (the subassignment of γ considering only variables in $\mathcal{V}$), as stated also in Line 3(2)2, produces a clause $\gamma_\mathcal{V}^{min}$ (subassignment of $\gamma_\mathcal{V}$ such that $\{C \mid \gamma_{min}(v_C) = 1\}$) which is (potentially) smaller that $\gamma_\mathcal{V}$, minimal and still unfeasible. It is reasonably evident that the smaller the amount of variables involved in $\gamma_\mathcal{V}^{min}$, the bigger the pruning of the search space, because no boolean assignment containing such partial assignment will be considered further in the enumeration of satisfying boolean assignments. We will return to this point in Sec. 3 where we discuss some implementation notes. It is worth noting that clause minimisation, used to produce more efficient unsatisfiability certificates during the enumeration of satisfying boolean assignments, is orthogonal to the main contribution of this work on the minimisation of QoS contracts.

This type of automatic analysis is usually understood as a refutation method aiming at finding counterexamples to properties. Assume $\langle \langle \mathcal{X}, \mathcal{P} \rangle, \alpha \rangle$ is a sat-

isfiable[5] convex specification of a system and β a desirable property written in the same language then, if $\alpha \wedge \neg\beta$ (a formula equivalent to $\neg(\alpha \implies \beta)$) is satisfiable, one can conclude that β does not follows from α, and the satisfying assignment constitutes a counterexample.

2.1 From QoS contracts to an efficient determination of SLA

Service-Oriented Computing relies on a notion of software system as a dynamic entity built up from services discovered and bound in run-time. To make this possible, services must be previously registered in public repositories from where they can be procured by a *Service broker* as required by a dedicated *Middleware* who dynamically reconfigures the executing application by binding it to the service. This process of dynamic reconfiguration is triggered automatically by an application reaching a point in its execution where a continuation depends exclusively on the intervention of an external service.

Consider the case of a (satisfiable) QoS requirement contract Rq. A service with a (satisfiable) QoS provision contract Pr will be a good candidate only if $Pr \implies Rq$ holds; or equivalently, the formula $Pr \wedge \neg Rq$ is not satisfiable. Assuming that such a formula can be seen as a convex specification (i.e. a formula that can be produced by the grammar in Def. 1), then it is possible to perform the interoperability check by executing the algorithm of Fig. 1 to determine compliance between the application's QoS requirement contract and the service's QoS provision contract. Although from a theoretical point of view there is no objection to applying this procedure for determining SLA, from a practical perspective, this interoperability check is expected to be performed over many candidates, in order to guarantee that the service chosen by the service broker is the optimum according to the status of the repository.

Such a use context imposes strong efficiency considerations over this type of analysis, and coping with them requires reducing the execution time for performing the analysis as much as possible, even at the expense of investing a bigger amount of time preprocessing the contracts in advance when the services are registered in the repository. Returning to our example, assume that Pr and $\neg Rq$ are satisfiable convex specifications denoting provision QoS contract and the negation of a requirement QoS contract, respectively; both ${Pr}_B$ and ${\neg Rq}_B$ might have plenty of satisfying boolean assignments γ_{Pr} (resp. $\gamma_{\neg Rq}$) leading to non-feasible sets of convex constraints $\{C \mid \gamma_{Pr}(v_C) = 1\}$ (resp. $\{C \mid \gamma_{\neg Rq}(v_C) = 1\}$), suggesting that contracts admit some relative minimisation that can be performed by updating the solver's clause database by injecting a new clause characterising such unfeasibility information. Ideally this process of *Minimisation through Convex Optimisation* – MCO must traverse the whole space of boolean assignments of the propositional abstraction of the contract, determining which of them leads

[5] It is important for the specification to be satisfiable in order to ensure the existence of logical models, which in the case of this particular language, are boolean assignments. The existence of a model can be interpreted as the existence of a concrete implementations of the system satisfying the specification.

to a feasible set of convex constraints. This observation motivates the idea of a two phase analysis approach based on the algorithm of Fig. 1, in which the first phase aims at the minimisation of QoS contracts, performed only once when a service is registered in a repository, and the second phase is the analysis of QoS contract compliance, executed for SLA negotiation.

A two phase analysis algorithm of convex specifications Our proposal for the analysis of QoS contract compliance is motivated by the specific usage scenario of SOC, where specifications are expected to be reused in many analysis. Such a context imposes that time consumption to check whether $Pr \wedge \neg Rq$ is satisfiable or not, has to be as tight as possible due to the fact that such a check has to be performed over every candidate satisfying the functional requirements. To cope with such demand, we devised a preprocess (referred to as *minimisation phase*) aiming at the minimisation of QoS contracts, represented by a convex specification, and a second phase (referred to as *check phase*) that implements the exact same analysis than the algorithm of Fig. 1.

Our first research questions aims at relating the performance of the algorithm in Fig. 1 as distributed by Pappas et al. and our reimplamentation when satisfying boolean assignments are iterated by using Z3 and Minisat as SAT-solvers.

RQ1: Is there any performance gain in a Z3-based implementation of the check phase with respect to that of the algorithm shown in Fig.1? What about between a Minisat-based implementation with respect to the Z3-based implementation of the check phase?

Next we present and evaluate the main contribution of the paper, being the implementation of a QoS contract MCO procedure, aiming at preprocessing QoS contracts in order to prune a significant portion of the space of satisfying boolean assignments of its propositional abstraction. Such minimisation procedure is motivated by two observations about the algorithm of Fig. 1: 1. the analysis procedure relies on enumerating all possible models of the propositional abstraction of the QoS contract, and 2. line 3(2)2 of the algorithm of Fig. 1 alters the enumeration process by using the unfeasibility information obtained from the convex analysis, transformed into a minimal boolean clause, acting as an unfeasibility certificate. The algorithm for QoS contract MCO is based on performing convex analysis to test the feasibility of the set of convex constraints determined by each satisfying boolean assignment. The single difference with SMCO analysis procedure is that, while in Line 3(2)1 of the algorithm of Fig. 1 the problem is reported as satisfiable and the search for more satisfying assignments ends, MCO discards that satisfying boolean assignment to return to the enumeration and continue the search for boolean assignments that lead to non-feasible sets of constraints.

Given a QoS contract, consisting of a monotone SMC formula α, the reader might note that the minimisation process visits all satisfying boolean assignments of the propositional abstraction α_B in order to determine which of them

are declared feasible by the convex solver so they must remain as satisfying boolean assignments, and which of them must be pruned from the space of models of the specification. Then we can design a two phase convex analysis algorithm by considering: 1. a single time application of the process of QoS contract MCO phase of both, a provision contract Pr, and the negation of a requirement contract $\neg Rq$, yielding $\widehat{Pr}$ and $\widehat{\neg Rq}$ respectively, and 2. a second phase of searching for a satisfying boolean assignment for the propositional abstraction of $\widehat{Pr} \wedge \widehat{\neg Rq}$ that leads to feasible set of convex constraints (i.e. the result of applying the algorithm shown in Fig. 1).

An important concern regarding the MCO procedure, is that, as we mentioned before, minimising a QoS contract requires visiting all possible boolean assignments satisfying the propositional abstraction of the formula $\widehat{Pr} \wedge \widehat{\neg Rq}$. It is trivial to see that this procedure is of an exponential nature due to the fact that the size of the space of satisfying boolean assignments of a boolean formula is (potentially) exponential with respect to the amount of boolean variables[6]. Having said that, efficiency improvement resulting from potential optimisations of the process cannot change such a high complexity bound. A direct consequence of this observation is that, even if minimisation is considered as a one time preprocess, full scalability is, by any means, unreachable.

The remaining research questions aims at evaluating the performance of our proposal of a two phase procedure for checking QoS contract compliance. To accomplish that, the comparison of the three different implementations of the SMCO analysis procedure done to answer **RQ1** will serve to set a baseline for the experimental evaluation of the main contribution of this work, being the comparison of the time required for checking QoS contract after performing a *Minimization Through Convex Optimisation* phase.

The second research question aims at identifying whether there is a bound to the size of the QoS contracts that can be fully minimised.

RQ2: What is the size of QoS contracts that can be fully minimised in 3 hs.?

A close look to the proposed minimisation procedure exposes that if it were stopped at any iteration, right before continuing the enumeration of satisfying boolean assignments, the resulting QoS contract shares every feasible model with the original one.

Proposition 1. *Let $\langle \langle \mathcal{X}, \mathcal{P} \rangle, \alpha \rangle$ be a convex specification, α_B the propositional abstraction of α and δ a function mapping variables in $\mathcal{V}$ to their corresponding convex or affine constraint appearing in α. Let $[\alpha_{Bi}]_{0 \leq i \leq n}$ be a sequence of boolean formulae where $\alpha_{B0} = \alpha_B$, α_{Bi+1}, for all $i < n - 1$, is the result of a single iteration of the minimisation algorithm to α_{Bi} and α_{Bn} is the result at the end of the algorithm.*

[6] To be precise, the problem of enumerating all possible satisfying assignment of a SAT-formula is, at least, in the complexity class **#P**.

Then, every convex specification $\langle\langle \mathcal{X}, \mathcal{P}\rangle, \alpha_{B_i}\rangle$*, where* $0 \leq i \leq n$*, share the same feasible models.*

An interesting fact derived from the previous proposition is that the process of QoS contract minimisation can be performed incrementally, one iteration at a time, leading to a succession of partially minimised QoS contracts, leading us to our third research question.

RQ3: How does the nature of the problem change considering successive partial minimisations of a given specification?

3 Implementation and experimental results

In this section we present the answers to the research questions posed in Sec. 2 through experimental evaluation. First we will provide some details about the implementation of the algorithms, the hardware configuration of the experimental setup and the dataset used for the experimental evaluation.

Notes on the implementation In this section we will briefly discuss some aspects of the implementation of the algorithms presented in the paper. The implementation developed by Pappas et al. in [13] of the algorithm of Fig. 1 uses Z3 [12] as SAT-solver. Having said that, we developed two versions of our algorithms, one also using Z3 to check wheter the reimplementation of the algorithm does not introduce any significant difference in performance, a second one resorting to Minisat [3], a well-known SAT-solver whose minimality has made it one of the most efficient ones available.

Checking feasibility of sets of constraints was implemented using IBM ILOG CPLEX Optimization Studio [9] since it is one of the most powerful convex optimization softwares that is available for research and educational purposes.

The algorithm orchestrating the enumeration of satisfying boolean assignments of the propositional abstraction of the contracts with the convex solving of the set of convex constraints determined by corresponding assignment, as well as all the framework needed for the generation of instances according to the experimental design and their systematic execution were developed in Python 2.7 (`https://www.python.org`) resorting to the existent libraries needed to integrate the various tools mentioned above.

Hardware configuration Experiments were run over an x86_64 architecture processor Intel(R) Core(TM) i5 CPU 750 at 2.67GHz (CPU MHz: 2661 – CPU max MHz: 2661 – CPU min MHz: 1197) with 3 level cache (L1d cache: 32K, L1i cache: 32K – L2 cache: 256K – L3 cache: 8192K), 8 Gb of SDRAM at 1333 MHz, and running SMP Debian Linux 4.9.88-1+deb9u1 (2018-05-07). Each individual instance was run for at most 3 hours, unless it is noted, as we believe that what is feasible within that time frame provides enough information to validate our answers. Whenever the analysis of a problem instance, or the construction of the solving infrastructure, exceded the time limit, the corresponding

cell in the tables was marked with "TO" denoting that the process reached the timeout, and whenever the limit was a consequence of the exhaustion of the system memory leading the machine to a sate of trashing was marked with "OoM" denoting the system run out of memory.

Notes on the experimental setting A first note on the experimental designs we need to put forward is that we did not use the experimental setting distributed by the authors of [13] for the following reasons: 1) specifications are constructed hardcoding the solver instance and not providing any interface allowing users to feed a specification in any standard language, making very difficult to test the tool over different data sets, and 2) there is no report on the time needed to construct the solver instance which, after profiling the implementations based on Z3, are proven to be not negligible due to the fact that part of the solving is performed during the addition of the clauses, sometimes consuming more time than the invocation of the solve itself (see column "Initialisation time" of Table 1).

3.1 Experimental evaluation

In this section we evaluate the research questions posed previously and show experimental data supporting our answers.

RQ1: Is there any performance gain in a Z3-based implementation of the check phase with respect to that of the algorithm shown in Fig.1? What about between a Minisat-based implementation with respect to the Z3-based implementation of the check phase?

Experimental design: QoS provision contracts were obtained by first generating random SAT instances of satisfiable provision contracts, Pr_B using the number of boolean and real variables as parameters. The dataset is formed by QoS contracts with boolean variables ranging from 50 to 350, stepping every 50 variables. The number of clauses is 80 times the number of boolean variables. From the total amount of variables we randomly choose 50% to which we associated randomly generated linear convex constraints. Convex constraints were considered to apply over 5 to 30 real variables stepping every 5 variables. Satisfiable contracts $\neg Rq_B$ are obtained from: 1) negating Pr_B, 2) pushing negations to the atoms, and 3) reversing the linear constraints (producing also linear constraints). In this way, $Pr_B \wedge \neg Rq_B$ results satisfiable from the boolean point of view, but having no feasible convex model. This lack of counterexamples for $Pr \implies Rq$ aims at stressing the checking procedure forcing it to traverse the whole space of solutions.

The upper limit in the number of boolean variables respond to the fact that generating a CNF boolean formula $\neg Pr_B$ as a QoS requirement contract, using Tseitin's transformation [15], yields a boolean formula quadratically bigger than Pr_B, both in the number of clauses and variables. If we consider that a propositional abstraction of a provides contract Pr_B of 400 boolean variables and 32000 clauses results in a 35 Mb file, the construction of a QoS contract

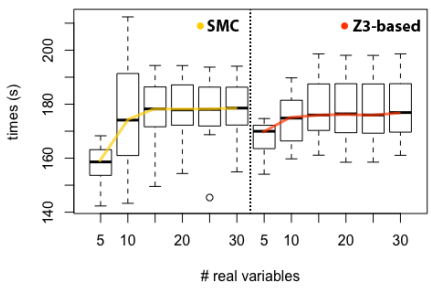
equivalent to $\neg Pr_B$ yields a CNF formula of approximately 30000 boolean variables and 9500000 clauses, consuming approximately 1 Gb. The analysis was performed over 8 instances of each combination of boolean variables and real variables to try palliating the variance among cases. The time was split in two, the time needed to setup the checking infrastructure, and the analysis time itself, as the use of Z3 showed that a significant part of the analysis takes place during the initialisation of the SMT-solver with the clauses.

Experimental results: Table 1 shows the running time comparison between the original implementation of SMCO algorithm presented in [13], reviewed in Fig. 1, and the implementations of the check phase based on Z3 and Minisat. The layout of the table is: 1. the first column shows the amount of boolean variables in the provision contract, 2. the second one shows the solver used to solve the problem, 3. the third column shows the time required for initialising the solver until the exact moment before it is ready to solve the problem, and 4. columns fourth to nine show the average time required to solve the instances of the corresponding number of real variables. Figure 2 shows boxplots graphs containing the run-

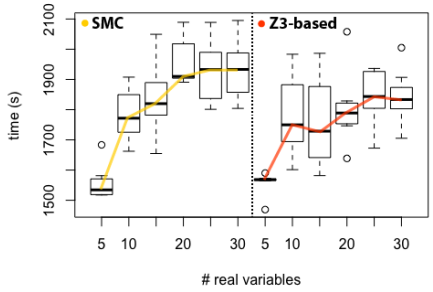
# bool. vars in provides	Solver	Initialisation time (approx.)	# real variables 5	10	15	20	25	30
50	SMC	150	157.48	176.71	176.71	178.28	176.86	178.22
	Z3 check	170	167.82	177.82	177.55	177.01	176.84	177.55
	Minisat check	4	3.21	2.23	2.13	2.12	2.12	2.12
100	SMC	630	1575.70	1802.20	1756.17	1812.62	1854.42	1852.64
	Z3 check	640	1589.20	1819.81	1843.83	1954.84	1925.89	1931.48
	Minisat check	17	16.47	19.89	11.90	10.01	9.11	8.70
150	SMC	1680	4243.10	TO	TO	TO	TO	TO
	Z3 check	1820	4580.25	TO	TO	TO	TO	TO
	Minisat check	45	21.32	174.12	79.92	47.76	26.66	23.60
200	SMC	3100	TO	TO	TO	TO	TO	TO
	Z3 check	3120	TO	TO	TO	TO	TO	TO
	Minisat check	77	35.53	236.32	906.17	488.88	102.78	60.34
250	SMC	4920	TO	TO	TO	TO	TO	TO
	Z3 check	5010	TO	TO	TO	TO	TO	TO
	Minisat check	127	45.93	321.73	1231.22	2929.81	1052.29	405.59
300	SMC	OoM	–	–	–	–	–	–
	Z3 check	7300	TO	TO	TO	TO	TO	TO
	Minisat check	186	60.05	431.87	1714.84	4423.42	8259.39	4720.92
350	SMC	OoM	–	–	–	–	–	–
	Z3 check	OoM	–	–	–	–	–	–
	Minisat check	OoM	–	–	–	–	–	–

Table 1: Comparison of different implementations of algorithm for checking an unsatisfiable formula

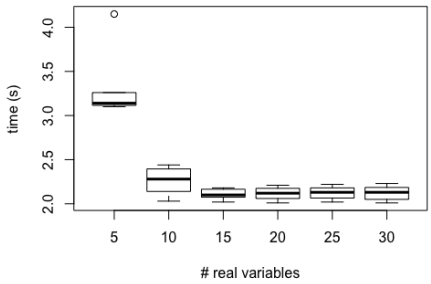
time information for the first two rows of Table 1 exposing the relative stability of the algorithms. Figures 2a and 2b show the running time of the tool using the algorithm in Fig. 1 and the Z3-based implementation of the check phase algorithm, while Figs. 2c and 2d show the running time using the Minisat-based implementation.



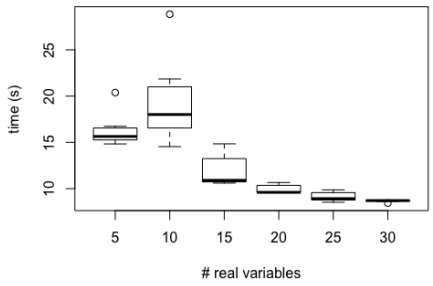
(a) 50 boolean vars. in provides using Z3



(b) 100 boolean vars. in provides using Z3



(c) 50 bool. vars. in provides using Minisat



(d) 100 bool. vars. in provides using Minisat

Fig. 2: Compliance analysis of QoS contracts over 50 and 100 boolean variables in provides (total problem size of +4000 and +8000 bool. vars. respectively)

Conclusions and discussion: Observing the running times obtained from executing the three implementations of SMCP algorithm for checking an unsatisfiable formula shown in Table 1 we derive the following conclusions: 1. running time grows exponentially at a rate of 2.37 on the number of boolean variables, 2. observing the rows it is possible to appreciate that the time required to analyse the satisfiability of the instances grows until it finds a maximum, then decrease until it stabilises in what seems to be a plateau. As for every row the CNF used is the same, also prescribing the amount of convex constraints, this phenomenon seems to expose a relation between the number of constraints and the number of real variables over which those constraints are expressed. It is also observable that decrease, and further stabilisation, of the time is mimicked by the number of iterations of the algorithm. Having said this, we believe that understanding this particular phenomenon has no impact on the experiment conducted to answer this research question, 3. there is no consistent and significant difference between the performance of the SMCP original implementation by Pappas et al. and our implementation based on Z3 as boolean solver. The experimental data shows a running time difference no bigger than 10%. This difference might be

a byproduct of having developed a more abstract implementation of the convex specification encapsulating an iterator for the satisfying boolean assignments of the propositional abstraction, 4. the Minisat-based implementation greatly outperforms both Z3-based implementations taking only approximately 2% of the time required by the latter, considering setting up the analysis infrastructure and analysing compliance altogether.

An important observation about the answer given to **RQ1**, is that only the Minisat-based implementation of the check phase algorithm can produce a significant amount of experimental data useful enough to evaluate our proposal of a two phase QoS contract compliance analysis procedure; thus **RQ2** and **RQ3** will be answered only evaluating the performance of the Minisat-based implementation of both phases of the procedure.

RQ2: What is the size of QoS contracts that can be fully minimised in 3 hs.?

Experimental design: The dataset used to run the experiment was generated using the same criteria used for generating the dataset used to run the experiment performed to answer **RQ1** but considering a finer granularity in the axis of boolean variables (stepping every 5) and starting from 30 variables, as the running time of the minimisation phase of instances with less than 30 was negligible. In those cases in which the minimisation process did not find any unfeasible convex model thus, not producing any minimisation, the running time is reported tagged with **(nm)**, indicating "no minimisation".

Experimental results: Table 2 shows the running time of the minimisation phase. A side by side comparison of the time required by the analysis algorithm, over

	# real variables					
# boolean vars	5	10	15	20	25	30
30	186.68	**(nm)** 200.40	**(nm)** 203.18	**(nm)** 206.43	**(nm)** 209.94	**(nm)** 212.82
35	691.26	**(nm)** 852.93	**(nm)** 921.20	**(nm)** 930.37	**(nm)** 944.27	**(nm)** 953.85
40	5606.34	**(nm)** 9519.08	**(nm)** 9682.34	**(nm)** 9814.39	**(nm)** 10004.27	**(nm)** 7234.67
45	TO	TO	TO	TO	TO	TO

Table 2: Running time of the Minisat-based minimisation algorithm

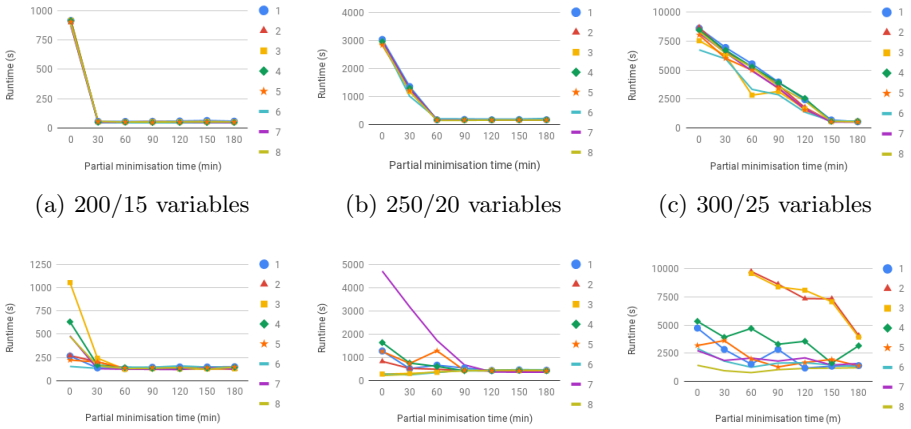
minimised and not minimised instances, is of no interest in this case as the cost associated to the process of full minimisation of QoS contracts is so high that even the smaller instance of Table 1 of 50 boolean variables and 5 real variables could not be minimised within the time bound of 3 hs.

Conclusions and discussion: Results shown in Table 2 expose that a naïve approach to minimisation is not viable as the cost of full minimisation might be too high, even for very small contracts.

RQ3: How does the nature of the problem change considering successive partial minimisations of a given specification?

Experimental design: To provide an answer to this research question we performed partial minimisations (from 0 minutes to 3 hours stepping every 30 minutes) of every pair of QoS contracts of those cells of rows 200, 250 and 300 of Table 1 where the number of real variables ranges between 5% and 10% of the number of boolean variables. Then, we performed the compliance analysis in order to identify how analysis time decreases while more time is invested in the minimisation procedure.

Results: Figure 3 show graphs of the evolution of time required by the check phase over partially minimised QoS contracts, considering the snapshots taken every 30 minutes of minimisation. Table 3 shows the comparison of the solving



(a) 200/15 variables (b) 250/20 variables (c) 300/25 variables

(d) 200/20 variables (e) 250/25 variables (f) 300/30 variables

Fig. 3: Solving times for partial minimisations (from 0 minutes to 3 hours stepping every 30 minutes) of provides and requires QoS contracts

time required by partially minimised QoS contracts up to 3 hours.

Conclusions and discussion: Observing the graphs in Fig. 3 we derive the following conclusions: 1. the running time required by the check phase decreases while the amount of time invested in minimisation grows. For QoS contracts over 200 boolean variables, the time required by the check phase drops dramatically to a plateau after only 30 min. of minimisation. In the case of QoS contracts over 250 boolean variables, the pattern is exactly the same but requiring between 60 to 90 min. of minimisation. Finally, for QoS contracts over 300 boolean variables, this phenomenon can be seen only for instances over 25 real variables after 150

# bool. vars in provides		Initialisation time (approx.)	# real variables 10	15	20	25	30
150	not minimised	43	171.35	94.01	–	–	–
	minimised	44	15.13	38.59	–	–	–
	percentage	102.3	8.8	41	–	–	–
200	not minimised	77	–	896.46	444.53	–	–
	minimised	80	–	50.31	139.34	–	–
	percentage	103.90	–	5.6	31.3	–	–
250	not minimised	127	–	–	2930.41	1310.06	–
	minimised	130	–	–	173.81	429.67	–
	percentage	102.36	–	–	5.9	32.7	–
300	not minimised	186	–	–	–	8083.04	3400.33 / **TO**
	minimised	189	–	–	–	560.94	2265.39
	percentage	101.61	–	–	–	6.9	66.6

Table 3: Comparison of solving time required between not minimised and minimised contracs (3 hs.)

min. but in the case of 30 real variables we can witness cases that cannot be checked within 3 hs. time when QoS contracts are not minimised, that can be checked after 60 min. of minimisation phase, 2. the growth on the number of real variables for instances over the same number of boolean variables shows more dispersion (in the time required for the check phase) over instances minimised for short periods of time, but rapidly collapsing to the plateau,

The results in Table 3 show how 3 hs. of minimisation phase dramatically reduce the cost of the check phase. The reduction naturally depends on the size of the problem under analysis; it is easy to see that bigger problems show smaller reductions over the same amount of time invested in the minimisation phase.

4 Conclusions

We proposed the use of a formal language for QoS contract specification together with an associated two phase compliance checking procedure, based on the algorithm proposed by Pappas et al. in [13] for hybrid system verification, adapted to the concrete scenario of SLA negotiation for the automatic reconfiguration of software systems found in distributed environments such as SOC.

Research questions were posed and experiments were conducted to answer them. The dataset was designed in order to stress the technique by forcing the problem instances to be unsatisfiable thus, requiring the exhaustion of the space of potential solutions; the evaluation of the tool under satisfiable instances remains to be addressed. The experimental results shown in the answering of **RQ2** evidenced that full minimisation of contracts might be unreachable as time and memory consumption, even for small case studies, result too high. To mitigate such demand we proposed the use of incremental minimisation and the experimental results shown in the answering of **RQ3** evidenced that a relatively small amount of time invested in a first phase of QoS contract MCO dramatically reduces the time required by a second phase dedicated to check compliance of minimised versions of the QoS contracts.

References

1. Bryant, V.: Metric spaces: iteration and application. Mathematical systems theory, Cambridge University Press (1985)
2. Cormen, T.H., Clifford, S., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT Press (2001)
3. Eén, N., Sörensson, N.: An extensible sat solver. In: Giunchiglia, E., Tacchella, A. (eds.) Proceedings of 6th. International Conference on Theory and Applications of Satisfiability Testing, SAT 2003. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer-Verlag, Santa Margherita Ligure, Italy (May 2003)
4. Fiadeiro, J.L., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. Formal Aspects of Computing **23**(4), 433–463 (2011)
5. Grünbaum, B.: Convex polytopes, Graduate Texts in Mathematics, vol. 221. Springer-Verlag, Berlin, Germany (1967)
6. Henzinger, T.A.: The theory of hybrid automata. In: Vardi, M.Y., Clarke, E.M. (eds.) Proceedings of Eleventh Annual IEEE Symposium on Logic in Computer Science, 1996. LICS '96. pp. 278–292. IEEE Computer Society (Jul 1996), see also [7]
7. Henzinger, T.A.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems, pp. 265–292. Springer-Verlag, Berlin, Heidelberg (2000), see also [6]
8. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences **57**(1), 94–124 (1998)
9. IBM: IBM ILOG CPLEX Optimization Studio (2004), `https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer`
10. de Moura, L.M., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM **54**(9), 69–77 (2011)
11. Nuzzo, P., Puggelli, A., Seshia, S.A., Sangiovanni-Vincentelli, A.L.: CalCS: SMT solving for non-linear convex constraints. In: Bloem, R., Sharygina, N. (eds.) International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010. pp. 71–79. ieee (October 2010)
12. Microsoft Research: Z3: An efficient smt solver. On-line, available at `http://research.microsoft.com/projects/z3/`
13. Shoukry, Y., Nuzzo, P., Sangiovanni-Vincentelli, A.L., Seshia, S.A., Pappas, G.J., Tabuada, P.: SMC: Satisfiability modulo convex optimization. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control. pp. 19–28. ACM Press, New York, NY, USA (2017)
14. Strunk, A.: QoS-aware service composition: A survey. In: Brogi, A., Pautasso, C., Papadopoulos, G.A. (eds.) Proceedings of 8th IEEE European Conference on Web Services (ECOWS 2010). pp. 67–74. IEEE Computer Society (December 2010)
15. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J.H., Wrightson, G. (eds.) Automation of Reasoning, pp. 466–483. Symbolic Computation, Springer-Verlag (1983)