# Architecture-Guided Test Resource Allocation Via Logic *

Clovis Eberhart[1,2][0000−0003−3009−6747], Akihisa Yamada[3][0000−0001−8872−2240],
Stefan Klikovits[1][0000−0003−4212−7029], Shin-ya Katsumata[1][0000−0001−7529−5489],
Tsutomu Kobayashi[4,1][0000−0002−8795−3183], Ichiro Hasuo[1][0000−0002−8300−4650],
and Fuyuki Ishikawa[1][0000−0001−7725−2618]

[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430,
Japan
{eberhart,klikovits,s-katsumata,t-kobayashi,hasuo,f-ishikawa}@nii.ac.jp
[2] Japanese-French Laboratory for Informatics, Tokyo, Japan
[3] National Institute of Advanced Industrial Science and Technology, 2-3-26, Aomi,
Koto-ku, Tokyo 135-0064, Japan
akihisa.yamada@aist.go.jp
[4] Japan Science and Technology Agency, 4-1-8, Honcho, Kawaguchi-shi, Saitama
332-0012, Japan

**Abstract.** We introduce a new logic named Quantitative Confidence Logic (QCL) that quantifies the level of confidence one has in the conclusion of a proof. By translating a fault tree representing a system's architecture to a proof, we show how to use QCL to give a solution to the test resource allocation problem that takes the given architecture into account. We implemented a tool called `Astrahl` and compared our results to other testing resource allocation strategies.

**Keywords:** Reliability · Test Resources Allocation · Logic.

## 1 Introduction

With modern systems growing in size and complexity, asserting their correctness has become a paramount task, and despite advances in the area of formal verification, testing remains a vital part of the system life cycle due to its versatility, practicality, and low entry barrier. Nevertheless, as test resources are limited, it is an important task to most effectively allocate them among system components, a problem commonly known as the test resource allocation problem (TRAP) (see e.g. [12]). In this paper we formulate the TRAP as follows: given a system that consists of multiple components and a certain, limited amount of test resources (e.g. time or money), how much of the budget should we allocate to each component in order to minimise the chance of system failure, i.e. to increase its reliability.

---

We propose an approach to the TRAP, based on a novel logic system called Quantitative Confidence Logic (QCL). QCL differs from classical logic, in that a proof tree does not conclude truth from assumptions, but rather analyses how *confidence* is propagated from assumptions to conclusions. We prove key soundness properties of QCL with respect to a probabilistic interpretation (Section 2).

Learning a new logic is a hard task, especially to practitioners. Thus we do not demand users to learn QCL, but take it as an intermediate language to which already accepted representations of system architectures are translated. As an example, we show how to translate the well-known concept of fault trees (FTs) into QCL proof trees (Section 3).

We then formulate the TRAP as an optimisation problem with respect to a given FT, translated to a QCL proof tree (Section 4). Here we allow users to specify a *confidence function* for each component, which describes how an amount of spent test resources relates to an increase in that component's reliability.

We implement our approach as a tool (`Astrahl`) that takes as input an FT, a confidence function for each component, the current confidence in each component, and the total amount of test resources the user plans to spend. Then `Astrahl` outputs a proposed allocation of the test resources over components. We validate our approach through experiments (Section 5).

An advantage of our method is that it is not tied to a fixed confidence function, and can therefore assign different confidence functions to different components. This will be useful in modelling systems with highly heterogeneous components such as cyber-physical systems (CPSs); for example, hardware components would demand more effort to increase confidence than software, and would depend on the type of components or their vendors.

We expect `Astrahl` to be used continuously in a system's development: the confidence in each component increases as they pass more tests. By rerunning `Astrahl` with updated component confidences, we obtain a test resource allocation strategy. We expect our approach to be promising in product line development, where a number of system configurations are simultaneously developed with common components. In such a situation, updating confidence in a component for one system positively impacts the test strategies for other systems.

## 1.1   Related Work

**Test Resource Allocation** Most approaches to the TRAP use software reliability growth models (SRGMs) such as models based on Poisson Process (e.g. [7]) to capture the relationship between testing efforts and reliability growth, or in our words, confidence functions. A typical TRAP approach formulates the problem using a particular SRGM and provides a solution using exact optimisation [10] or a metaheuristic such as a genetic algorithm [19]. A challenge in this area is to take the structure (inter-module relations) of the target system into account; existing studies consider particular structures such as parallel-series architecture [19] or Markovian architecture [13]. In addition, there is high demand for dynamic allocation methods (e.g. [3]) because in practice SRGMs, system structures, and testing processes often become different from those planned at

first. Our approach has multiple beneficial features over the existing approaches: (1) it is independent of particular SRGMs and optimisation strategies, (2) it can take complex structure into account using FTs, and (3) it can be used for dynamic allocation.

**Fuzzy Logics** Fuzzy logics [8] is a branch of logic interested in deductions where Boolean values are too coarse. In its standard semantics [6], formulas are given a numeric truth value in the interval $[0, 1]$, where 0 represents falsity and 1 truth. These numerical values can be used to represent the confidence one has in an assertion: we give high values to propositions we are confident are true, and low values to those we are confident are false. This is slightly different from our approach, where 1 corresponds to confidence (either in truth or falsity) and 0 to absence of knowledge.

**Dempster-Shafer Theory** Dempster-Shafer theory [4, 15] is a mathematical theory of belief. One of its characteristic features is that if one has a belief $b$ in an assertion, they can have any belief $b' \leq 1 - b$ in its negation, contrary to traditional Bayesian models, where it is necessarily $1 - b$. This feature is crucial to model uncertainty due to absence of knowledge, and Dempster-Shafer theory has been used to model reliability in engineering contexts [14]. Our approach draws inspiration from fuzzy and three-valued logics to model this feature.

**Fault Tree Analysis** Fault trees [17] are tree structures that represent how faults propagate through a system. In qualitative FT analysis, they are used to determine root causes [5]. In quantitative FT analysis, basic events are assigned fault probabilities, and the overall system failure probability is given by propagating the fault probabilities through the fault tree. Our approach uses the same ingredients (assigning numeric values to basic events and propagating them through the fault tree), but repurposed to solve another problem.

## 2    Quantitative Confidence Logic

This section introduces QCL, which we use throughout this paper. A QCL formula is a standard propositional formula $\varphi$ equipped with a pair of reals, written $\varphi \colon (t, f)$, where $t \in [0, 1]$ represents our confidence that $\varphi$ holds and $f \in [0, 1]$ the confidence that $\varphi$ does not hold, so $t + f$ represents how much confidence one has about $\varphi$, and $1 - t - f$ lack of confidence about $\varphi$. Absolute confidence is represented by 1 and total absence of knowledge by 0, so that $\varphi \colon (1, 0)$ means full trust that $\varphi$ holds, $\varphi \colon (0, 0)$ means we have no knowledge about $\varphi$, and $\varphi \colon (1/2, 1/2)$ represents the fact that we know with very high confidence that $\varphi$ holds with 50% chance.

In Section 2.1 we define the syntax of QCL and introduce its proof rules. We also show how to derive standard proof rules from them. In Section 2.2 we give a probabilistic interpretation of QCL formulas and show that QCL proof rules

are sound with respect to it. We also explain how the particular shapes of the rules serve as the basis of our optimisation algorithm (Section 4).

### 2.1   Syntax and proof rules of QCL

This section introduces QCL, starting with *formulas with confidences*, then sequents, and finally proof rules.

**Definition 1.** *Given an arbitrary set of atomic propositions* Prop, *formulas are defined inductively by the following grammar:*

$$\varphi ::= A \mid \top \mid \bot \mid \varphi \Rightarrow \varphi,$$

*for* $A \in$ Prop. *We denote by* Form *the set of all formulas.*

As in classical logic, negation, disjunction, and conjunction can be defined by syntactic sugar $\neg\varphi \equiv \varphi \Rightarrow \bot$, $\varphi \vee \psi \equiv \neg\varphi \Rightarrow \psi$, and $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$.

We now equip such formulas with confidence. We define the *space of confidences* as $\mathbb{C} = \left\{ (t, f) \in [0, 1]^2 \,\middle|\, t + f \leq 1 \right\}$.

**Definition 2.** *A formula with confidence is a pair* $(\varphi, c) \in$ Form $\times \mathbb{C}$, *written* $\varphi : c$. *For* $\varphi : (t, f)$, *we call* $t$ *the* true *confidence in* $\varphi$ *and* $f$ *its* false *confidence.*

Intuitively, a formula with confidence $\varphi : (t, f)$ represents the fact that our confidence that $\varphi$ holds is $t$, and our confidence that $\varphi$ does not hold is $f$. This should mean that the chance that $\varphi$ holds is at least $t$, and the chance that it does not is at least $f$. Another way to look at confidences is *intervals of probability*. Each confidence $(t, f)$ bijectively determines a sub-interval $[t, 1 - f]$ of $[0, 1]$. Then a formula with confidence $\varphi : (t, f)$ represents that the probability of $\varphi$ being true is within the interval $[t, 1 - f]$. We make this intuition more concrete in Section 2.2, where we give an interpretation of QCL in terms of probabilities.

Equipping formulas with numeric values is reminiscent of fuzzy logics[1]. To explain the fundamental difference between our approach and fuzzy logics, let us consider two orders on $\mathbb{C}$. Let $(t, f) \sqsubseteq (t', f')$ if $t \leq t'$ and $f \leq f'$; we call $\sqsubseteq$ the *confidence order*, as $c \sqsubseteq c'$ holds exactly when $c'$ represents more confidence (both true and false) than $c$. Our approach is centred around $\sqsubseteq$, since we are interested in "how confident" we are in an assertion. Similarly, let $(t, f) \leq (t', f')$ if $t \leq t'$ and $f \geq f'$; we call $\leq$ the *truth order*, as $c \leq c'$ intuitively means that $c'$ is "more true" than $c$. Fuzzy logics is centred around the truth order $\leq$ (especially on elements of the form $(t, 1 - t)$), as it is a logic about how true assertions are.

One way to link our approach to fuzzy logics is via three-valued logics [1]. Fuzzy logics can be seen as equipping formulas with a numeric truth value $t \in [0, 1]$ and a falsity value $f \in [0, 1]$ such that $t + f = 1$. This is equivalent to equipping formulas only with a numeric value $t \in [0, 1]$, while $f$ is the implicit difference to 1. With three-valued logics, formulas have three possible outcomes: truth $\top$, falsity $\bot$, and uncertainty $\bot\!\!\!\bot$, and each is given a value $t$, $f$, and $u$, such

---

[1] Here, we mean fuzzy logics interpreted in $[0, 1]$.

that $t + f + u = 1$, which is equivalent to equipping them with $(t, f) \in [0, 1]^2$ such that $t + f \leq 1$, while $u$ is implicit. Dempster-Shafer theory is similar, with $t$ representing our belief in $\varphi$, $f$ our belief in $\neg\varphi$, and $u$ our degree of uncertainty.

We now introduce the sequents on which QCL operates.

**Definition 3.** *A* sequent *in QCL, written $\Gamma \vdash \varphi \colon c$, consists of a finite set $\Gamma \subseteq$ Form $\times \, \mathbb{C}$ of formulas with confidences (written as a list), a formula $\varphi \in$ Form, and a confidence $c \in \mathbb{C}$.*

Such a sequent intuitively means that, if all formulas with corresponding confidences in $\Gamma$ hold, then $\varphi$ holds with confidence $c$ as well.

$$\frac{}{\Gamma, \varphi \colon (t, f) \vdash \varphi \colon (t, f)} \; (ax) \qquad \frac{}{\Gamma \vdash \varphi \colon (0, 0)} \; (unk) \qquad \frac{}{\Gamma \vdash \top \colon (1, 0)} \; (\top_{\mathrm{I}})$$

$$\frac{}{\Gamma \vdash \bot \colon (0, 1)} \; (\bot_{\mathrm{I}}) \qquad \frac{\Gamma \vdash \varphi \colon (t, f) \qquad \Gamma \vdash \psi \colon (t', f')}{\Gamma \vdash \varphi \Rightarrow \psi \colon (f + t' - ft', tf')} \; (\Rightarrow_{\mathrm{I}})$$

$$\frac{\Gamma \vdash \varphi \Rightarrow \psi \colon (t, f) \qquad \Gamma \vdash \varphi \colon (t', f')}{\Gamma \vdash \psi \colon \left(1 - \dfrac{1 - t}{t'}, \dfrac{f}{1 - f'}\right)} \; (\Rightarrow_{\mathrm{E},l}) \text{ if } t' \neq 0 \text{ and } f' \neq 1$$

$$\frac{\Gamma \vdash \varphi \Rightarrow \psi \colon (t, f) \qquad \Gamma \vdash \psi \colon (t', f')}{\Gamma \vdash \varphi \colon \left(\dfrac{f}{1 - t'}, 1 - \dfrac{1 - t}{f'}\right)} \; (\Rightarrow_{\mathrm{E},r}) \text{ if } t' \neq 1 \text{ and } f' \neq 0$$

Fig. 1: Proof rules of Quantitative Confidence Logic

**Definition 4.** *Proof trees in QCL are built from the QCL proofs rules given in Figure 1. There, the notation $\chi \colon (t'', f'')$ in conclusions is a shorthand for*

$$\chi \colon (\min(\max(t'', 0), 1), \min(\max(f'', 0), 1)).$$

Note that $(t'', f'') \in \mathbb{C}$ because $t'' + f'' \leq 1$ in all rules. Note also that rules $(\Rightarrow_{\mathrm{E},l})$ and $(\Rightarrow_{\mathrm{E},r})$ are conditioned so that confidence values do not contain indeterminate forms $0/0$. These side conditions correspond to the facts that, if $\varphi \Rightarrow \psi$ is true, $\varphi$ being false gives no information about $\psi$, and $\psi$ being true gives no information about $\varphi$.

$(ax)$, $(\top_{\mathrm{I}})$, and $(\bot_{\mathrm{I}})$ are self-explanatory, while $(unk)$ states that anything can be proved, but with null confidence. One way to think about $(\Rightarrow_{\mathrm{I}})$ is that, if $\varphi$ and $\psi$ are independent (in a way made precise in Section 2.2), $\varphi$ holds with probability in $[t, 1 - f]$ and $\psi$ with probability in $[t', 1 - f']$, then $\varphi \Rightarrow \psi$ holds with probability in $[f + t' - ft', tf']$. The elimination rules are designed similarly.

From the rules in Figure 1 and the encodings of negation, disjunction, and conjunction, we can derive the introduction rules in Figure 2 (we can also derive

$$\frac{\Gamma \vdash \varphi \colon (t, f)}{\Gamma \vdash \neg\varphi \colon (f, t)} \ (\neg_{\mathrm{I}}) \qquad\qquad \frac{\Gamma \vdash \varphi \colon (t, f) \qquad \Gamma \vdash \psi \colon (t', f')}{\Gamma \vdash \varphi \wedge \psi \colon (tt', f + f' - ff')} \ (\wedge_{\mathrm{I}})$$

$$\frac{\Gamma \vdash \varphi \colon (t, f) \qquad \Gamma \vdash \psi \colon (t', f')}{\Gamma \vdash \varphi \vee \psi \colon (t + t' - tt', ff')} \ (\vee_{\mathrm{I}})$$

Fig. 2: Derivable introduction rules

elimination rules, but do not discuss them here). A point worth attention is that the shape of these rules is quite unorthodox. In particular, one should not need to have confidence in both disjuncts to have confidence in a disjunction. However, this unorthodox shape is exactly the reason why our solution to the TRAP (defined in Section 4) works well, as we show in Example 16. Moreover, we can derive a disjunction from a single disjunct, as:

$$\frac{\Gamma \vdash \varphi \colon (t, f) \qquad \dfrac{}{\Gamma \vdash \psi \colon (0, 0)} \ (unk)}{\Gamma \vdash \varphi \vee \psi \colon (t + 0 - t \cdot 0, f \cdot 0) = (t, 0)} \ (\vee_{\mathrm{I}}),$$

which represents (one of) the usual disjunction introduction rules.

*Remark 5.* QCL rules are different from those of classical logic and do not extend them. Because the rules' design is strongly centred around how confidence flows from hypotheses to conclusions and based on independence of hypotheses, QCL cannot prove sequents such as $\emptyset \vdash A \Rightarrow A \colon (1, 0)$. How to allow reasoning about both confidence and truth in the same logic is left for future work.

*Example 6.* The correctness of a system with software and hardware components is based on the correctness of both components. A classical logical proof that the system is correct assuming both its software and hardware are correct is:

$$\frac{\dfrac{}{\Gamma \vdash \mathrm{software}} \ (ax) \qquad \dfrac{}{\Gamma \vdash \mathrm{hardware}} \ (ax)}{\Gamma \vdash \mathrm{software} \wedge \mathrm{hardware}} \ (\wedge_{\mathrm{I}}),$$

where $\Gamma = \{\mathrm{software}, \mathrm{hardware}\}$. By adding confidence to the formulas, we derive confidence in the assertion that the whole system is correct as a QCL proof:

$$\frac{\dfrac{}{\Gamma \vdash \mathrm{software} \colon (0.5, 0.2)} \ (ax) \qquad \dfrac{}{\Gamma \vdash \mathrm{hardware} \colon (0.3, 0.01)} \ (ax)}{\Gamma \vdash \mathrm{software} \wedge \mathrm{hardware} \colon (0.15, 0.208)} \ (\wedge_{\mathrm{I}}).$$

If we assume the software system is built from components that are difficult to prove reliable (e.g. machine learning algorithms), but much testing effort has been spent on it, then both true and false confidence may be high, as in the example above. On the contrary, if there are no reasons not to trust the hardware (e.g. it is made of very simple, reliable components), but less testing effort has been spent on it, then both true and false confidence may be lower than those of the software system.

It may seem unnecessary to keep track of false confidence, since our ultimate goal is to prove system reliability (and not unreliability). However, there are several cases in which we may want to use it. For instance, the calculation of how often a volatile system fails can be translated to a false confidence problem. Moreover, systems with failsafe mechanisms, e.g. *if A works, use module B, else use module C*, need to take the unreliability of $A$ into account. Otherwise, the whole system's reliability could not be correctly expressed, as it would only depend on $A$ and $B$. Hence, the optimisation would ignore the reliability of $C$.

*Remark 7.* In Figure 2, negation is an involution, and the reader familiar with fuzzy logics will have noticed the product T-norm and its dual probabilistic sum T-conorm in the rules $(\wedge_I)$ and $(\vee_I)$. Negation is an involution of $[0, 1]$ in fuzzy logics, and T-norms and T-conorms are the standard interpretations of conjunction and disjunction in fuzzy logics, which hints at a deep connection between our approach and fuzzy logics. However, implication is not interpreted as a residual, which again differentiates our approach from fuzzy logics.

## 2.2  Interpretation as Random Variables

In this section, we justify the QCL proof rules by giving formulas a probabilistic semantics and showing that these rules are sound.

We start with some measure-theoretic conventions. We write 2 to mean the discrete measurable space over the two-point set $\mathbb{B} = \{\top, \bot\}$. Boolean algebraic operations over 2 are denoted by $\wedge$, $\Rightarrow$, etc. (There should be no possible confusion with formulas.) For a probability space $(\Omega, \mathfrak{F}, P)$ (or $\Omega$ for short), $\mathrm{Meas}(\Omega, 2)$ denotes the set of 2-valued random variables. A *context* is a Prop-indexed family of 2-valued random variables, given as a function $\rho\colon \mathrm{Prop} \to \mathrm{Meas}(\Omega, 2)$.

**Definition 8.** *Given a space $(\Omega, \mathfrak{F}, P)$, we inductively extend a context $\rho$ to a Form-indexed family of 2-valued random variables $\bar{\rho}\colon \mathrm{Form} \to \mathrm{Meas}(\Omega, 2)$:*

$$\bar{\rho}(A) = \rho(A), \ \ \bar{\rho}(\top)(x) = \top, \ \ \bar{\rho}(\bot)(x) = \bot, \ \ \bar{\rho}(\varphi \Rightarrow \psi)(x) = \bar{\rho}(\varphi)(x) \Rightarrow \bar{\rho}(\psi)(x).$$

*The* semantics $[\![\varphi]\!]_{\Omega,\rho}$ *of a formula $\varphi$ in a space $\Omega$ and context $\rho$ is defined to be the probability $P[\bar{\rho}(\varphi) = \top]$ of $\varphi$ being true under $\rho$. We say that $\varphi\colon (t, f)$ holds in $\Omega$ and $\rho$ if $[\![\varphi]\!]_{\Omega,\rho} \in [t, 1 - f]$. We say that a sequent $\Gamma \vdash \varphi\colon c$ holds in $\Omega$ and $\rho$, if $\varphi\colon c$ holds in $\Omega$ and $\rho$ whenever all $\psi\colon c'$ in $\Gamma$ hold in $\Omega$ and $\rho$.*

In other words, the semantics of $\varphi$ is the measure of the space on which $\varphi$ holds, and $\varphi\colon (t, f)$ holds if $\varphi$ is true on at least $t$ of the space and false on at least $f$.

From here on, we only consider *independent* contexts, i.e. $\rho$'s such that the random variables $\rho(A)$ are mutually independent for all atomic propositions $A$.

The following lemma lifts independence of $\rho$ to $\bar{\rho}$.

**Lemma 9.** *Let $\Omega$ be a space and $\rho$ an independent context. If $\varphi$ and $\psi$ share no atomic propositions, then for all $S, T \subseteq \mathbb{B}$, the following holds:*

$$P[\bar{\rho}(\varphi) \in S \wedge \bar{\rho}(\psi) \in T] = P[\bar{\rho}(\varphi) \in S]P[\bar{\rho}(\psi) \in T].$$

*Proof.* By strengthening the proposition to finitely many $\varphi$'s and $\psi$'s, then by induction on: max depth of $\varphi$'s, number of $\varphi$'s of max depth, max depth of $\psi$'s, and number of $\psi$'s of max depth (with lexicographic order).

We can finally prove soundness of the rules.

**Lemma 10.** *For all rules in Figure 1, formulas $\varphi$ and $\psi$ that share no atomic propositions, spaces $\Omega$, and independent contexts $\rho$, if the premise sequents hold in $\Omega$ and $\rho$, then so does the conclusion.*

*Proof.* Simple computations relying on Lemma 9.

**Corollary 11.** *If $\varphi$ is linear (each atomic proposition appears at most once) and a proof $\pi$ of $\Gamma \vdash \varphi \colon c$ only uses base rules and introduction rules, then $\Gamma \vdash \varphi \colon c$ holds in all spaces $\Omega$ and independent contexts $\rho$.*

## 3    Translating System Architectures to Proofs

In this section, we translate FTs [17] to QCL proof trees. This allows us to use a system's architecture—modelled as an FT—in our solution to the TRAP. The way this translation works is close to quantitative fault tree analysis, where FTs are equipped with fault probabilities. In our translation, these fault probabilities are translated to confidences in QCL proofs.

**Definition 12.** *A* fault tree *is a tree whose leaves are called* basic events, *and whose nodes, called* gate events, *are either* AND *or* OR *gates.*

Basic events represent independent components of a system, and the tree structure represents how faults propagate through the system. The system fails if faults propagate through the root node. The usual definition of fault trees is more general than the one we give here, but we use this one for simplicity.
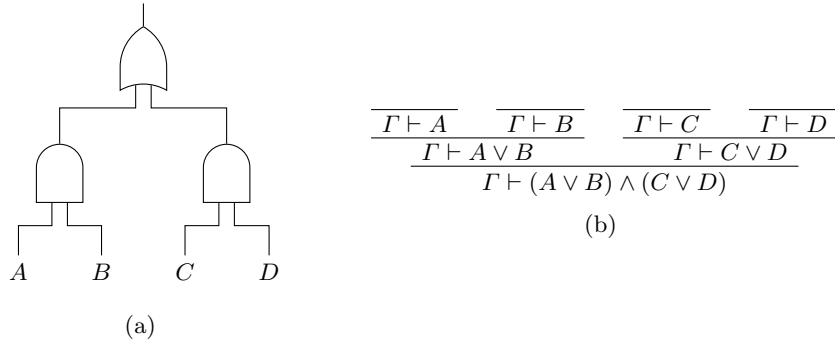


$$\dfrac{\dfrac{\overline{\Gamma \vdash A} \quad \overline{\Gamma \vdash B}}{\Gamma \vdash A \vee B} \quad \dfrac{\overline{\Gamma \vdash C} \quad \overline{\Gamma \vdash D}}{\Gamma \vdash C \vee D}}{\Gamma \vdash (A \vee B) \wedge (C \vee D)}$$

(b)

(a)

Fig. 3: A fault tree and its translation as a proof tree

*Example 13.* The FT in Figure 3a represents a system composed of four basic components $A$, $B$, $C$, and $D$. For a fault to propagate through the system and become a failure, either both $A$ and $B$ have to fail, or both $C$ and $D$ (e. g., $A$ and $B$ could be redundant components, doubled to increase reliability).

In *quantitative fault tree analysis* [5], failure probabilities are assigned to basic events, and they propagate through event gates as if mutually independent. In other words, if the failure probabilities of an AND gate's inputs are $a$ and $b$, then its output failure probability is $ab$, and $a + b - ab$ for an OR gate.

We translate fault trees to QCL proof trees as follows: The set Prop of atomic propositions collects all the names of basic events. $\Gamma$ consists of $A \colon (t_A, f_A)$ for each $A \in$ Prop. AND gates are translated to $(\vee_I)$ rules, and OR gates to $(\wedge_I)$ rules.

The reason for this dualisation is straightforward: while a fault tree represents how faults propagate, proof trees represent confidence in a system's reliability, i. e., how *absence of faults* propagates: the true confidence in each atomic proposition in $\Gamma$ now represents reliability of the component, and the true confidence in the conclusion represents reliability of the whole system.

*Example 14.* The translation of the fault tree of Figure 3a is shown in Figure 3b (with confidences left out for readability). Here, Prop $= \{A, B, C, D\}$ and the true confidence in the conclusion of the proof is $t_A t_B + t_C t_D - t_A t_B t_C t_D$. We can thus link increases in the reliability of components to increases in reliability of the whole system.

Note that the translation of a fault tree only uses base rules and introduction rules (more precisely, only $(ax)$, $(\wedge_I)$, and $(\vee_I)$). This is partly because we only consider AND and OR gates, but more essentially, basic events of fault trees are considered atomic and thus there is no need to eliminate logical connectives. Moreover, an assignment of failure probabilities to basic events translates to a context $\rho$ in QCL terms. Since all basic events are considered independent in an FT, their translation gives an independent context $\rho$. Therefore, the translation of a fault tree always verifies Corollary 11, and our interpretation as a QCL proof tree is sound for any assignment of failure probabilities to basic events. This means that, if the confidence in all basic components corresponds to their reliability, then the confidence of the whole proof cannot overshoot the whole system's reliability.

Since we translate fault probabilities to confidences, and fault probabilities are directly linked to reliability, we may use "reliability" and "confidence" interchangeably in the following, e. g. when we feel that reliability conveys a better intuition than confidence.

## 4   Solving the Test Resource Allocation Problem

In this section, we show how to optimise confidence in the conclusion of a QCL proof. This gives a solution to the TRAP through the translation of FTs to QCL proofs that was described in Section 3.

In order for this approach to be usable in practice, the user has to be able to specify two input parameters: the FT that represents the system's architecture, and functions describing how confidence in each component's reliability grows by spending resource on it. FTs are commonly used in the industry, so modelling a system using them should be no problem in practice. We first describe the latter input parameter in Section 4.1 and then give our solution to the TRAP in Section 4.2.

### 4.1   Confidence Functions

Increasing confidence in a proof's conclusion requires an increase in its premises' confidences. The cost of increasing confidence may vary among premises; when thinking in terms of systems (rather than proofs), for instance, increasing trust in a machine learning algorithm may require more effort than improving hardware reliability. This is a well-known problem, for which many solutions have been designed, especially SRGMs, which are based on mathematical modelling of faults [18]. Here, however, we do not choose a particular fault model and instead introduce the following abstract notion, which makes the approach versatile.

**Definition 15.** *A confidence function is a non-decreasing function* $f : \mathbb{R}_+ \to \mathbb{C}$ *(equipped with $\sqsubseteq$).*

The equality $f(r) = c$ means that after spending $r$ resources on a formula, one will have confidence $c$ in the formula. The monotonicity condition above enforces that, by spending more resources, confidence should not decrease.

Note that $f(0)$ can be different from $(0,0)$, which corresponds to the fact that engineers usually have some confidence in the components they use. This feature also makes it easy to use our approach in continuous development, by using confidence functions $f_s(r) = f(r + s)$ where $s$ is the amount of resource that has been already spent to test a component.

Note also that a confidence function may increase the false confidence, theoretically capturing the fact that faults may found by testing. For an application on the TRAP, however, we assume that faults will be fixed and thus false confidence always stays at 0. In the following, we thus define confidence functions as increasing functions $f \colon \mathbb{R}_+ \to [0,1]$, which represents the true confidence, and assume the false confidence is always 0.

**Designing confidence functions** Of course, expert knowledge on a component can be used to give a good estimate of confidence functions, but other techniques, such as *defect prediction* [11], exist for when knowledge is limited.

When testing is the canonical way to increase confidence, notions of test coverage serve as a good estimate of confidence. If we have a hardware test suite of $n$ tests that achieves 100% coverage (but not enough budget to execute them all), and each test costs $r_0$ resources, then the coverage achieved by spending $r$ resources in testing can be estimated as the confidence $f(r) = \min(r/nr_0, 1)$.

If we do not have such a test suite, then a reasonable way to model confidence is to assume uniform random testing. There we assume that each test covers a

randomly sampled fraction $p$ of the input space, but parts of it might be already covered by previous tests. If running a test costs $r_0$ resources, then a good estimate of confidence function is $f(r) = 1 - (1 - p)^{r/r_0}$.

If more is known about the component, then it is possible to design confidence functions that are better suited for this component. In particular, if we have some *a priori* knowledge about fault distributions, then it is possible to use SRGMs [18] as confidence functions.

### 4.2 The Optimisation Problem

We now formulate the TRAP as an optimisation problem in terms of QCL as follows: given a QCL proof, a confidence function for each premise, and a resource budget to spend, how should we spend the budget on the different premises to maximise confidence in the proof's conclusion?

We only consider the problem of optimising true confidence because the application we are aiming at is about reliability. However, with the same ingredients, we could define similar optimisation problems. For example, we could try to optimise total confidence $t + f$ under limited resources, or try to minimise resources spent to reach a given confidence objective (either in true or total confidence).

We begin with a simple observation: if $\varphi_1 \colon c_1, \ldots, \varphi_n \colon c_n \vdash \varphi \colon c$ is provable, then $c$ is a non-decreasing function of the $c_i$'s (for the confidence order $\sqsubseteq$). Hence, increases in the $c_i$'s confidence lead to increases in $c$.

Because, for the translation of an FT, the true confidence of the conclusion has to be a function $f(t_1, \ldots, t_n)$ of the true confidences of the hypotheses, if the confidence of each hypothesis is given by applying a confidence function $f_i$ to an amount of resources $r_i$ spent on that hypothesis, then the true confidence of the conclusion is itself a function $f(f_1(r_1), \ldots, f_n(r_n))$ of the amount of resources spent on the hypotheses.

The problem is thus the following: given an initial condition $r_1, \ldots, r_n$, confidence functions $f_1, \ldots, f_n$, a proof of $\{\varphi_i \colon (c_i, 0) \mid i \in n\} \vdash \varphi \colon (f(c_1, \ldots, c_n), -)$, and a budget $r$, maximise $f(f_1(r_1 + r_1'), \ldots, f_n(r_n + r_n'))$ under $r_i' \geq 0$ for all $i \in n$ and $\sum_{i \in n} r_1' \leq r$.

We thus reduce the TRAP to a classic constrained optimisation problem, which we can solve using well-known algorithms. In our implementation, we use simulated annealing [16], but any other method (such as CMA-ES [9] or Lagrange multipliers [2]) would work too.

*Example 16.* Take the QCL proof from Example 13. Suppose that the confidence functions of components follow $f(r) = 1 - 1/2^r$, the amount of resources already spent on the components are 0 for $A$, 5 for $B$ and $C$, and 10 for $D$, and we have a test resource budget of 10. Then we want to maximise

$$(1 - 1/2^a)(1 - 1/2^{5+b}) + (1 - 1/2^{5+c})(1 - 1/2^{10+d})$$
$$- (1 - 1/2^a)(1 - 1/2^{5+b})(1 - 1/2^{5+c})(1 - 1/2^{10+d})$$

under the constraints $a, b, c, d \geq 0$, and $a + b + c + d \leq 10$. There are two major points to note here. First, due to the fact that $(\vee_{\mathrm{I}})$ requires both disjuncts to

be proved, the optimisation will try to increase confidence of *both* $A$ and $B$, rather than choose one. Second, since we take system structure into accounr, the algorithm can give $B$ and $C$ different budgets, even though they share the same initial confidence and confidence function.

Our approach has significant advantages over other TRAP solutions. First, it makes use of the system's architecture, which is not the case of most approaches. Even other approaches that take system architecture into account generally only consider simple architectures, such as *parallel-series architecture* [19]. These architectures can be directly translated to FTs, but the converse is not possible without duplicating modules, which puts artificial weight to these duplicated modules. Moreover, we explained how to convert an FT to a QCL proof, but our algorithm is not limited to FTs and would work on other proofs.

Another advantage of our method is that it is not tied to any specific confidence function. The main advantage of this generality is that it allows the user to pick different confidence functions for different components. In particular, this approach should be helpful when allocating test resources for CPSs, where some components are software, while others are hardware, which most likely require to be modelled using different confidence functions.

## 5    Experimental Results

In this section, we describe the results of our experiments, showing that our tool `Astrahl` can increase system reliability more consistently than others. To demonstrate the tool's performance, we designed two experiments. The first one compares `Astrahl`'s confidence gain to other test resource allocation (TRA) strategies. The second, more involved experiment tests whether the increase in confidence provided by `Astrahl` is linked to an increase in system reliability. Given an FT and confidence functions, we simulate existence of component faults, before splitting a fixed testing budget according to different TRA strategies (one of which is our approach). We then mimic component testing according to the allocated budget and fix faults if they are found, thereby increasing system reliability. Our evaluation repeats the probabilistic process to test which method gives the best reliability on average.

We developed `Astrahl`, which implements the TRA algorithm described in Section 4. It takes as input JSON descriptions of the fault tree and the confidence functions (as parse trees), an initial condition (a float for each basic event), and a budget (a float), and returns a splitting of the budget between the different basic events (a float for each basic event).

This section evaluates our claims and analyses `Astrahl`'s system confidence gains to other, more naive approaches. We first ask how much confidence we can gain by using `Astrahl`, rather than simpler TRA approaches. Then, we test whether using `Astrahl` can increase system reliability in practice. Specifically, this section will investigate the following two research questions:

RQ1 Given a certain TRA budget, how much is the calculated confidence gain when using `Astrahl` and how do these figures compare to alternative TRA methods?

RQ2 Does `Astrahl`'s gain in confidence translate to a gain in system reliability in a practical scenario where testing practice is simulated?

**Alternative TRA approaches** There exist numerous solutions to test resource splitting, however some of the most common ones are the uniform and proportional resource allocation strategies (see Figure 4), as they do not require knowledge of the system structure or fault distribution. *Uniform* TRA, for instance, evenly distributes the available resources among the candidate components. This technique is completely agnostic of the current system and component confidences. *Proportional* TRA on the other hand aims to take current component confidence into account and provide proportionally more resources to components in which we have lower confidence. Although it uses current confidences for resource allocation, the system's structure is still not considered.
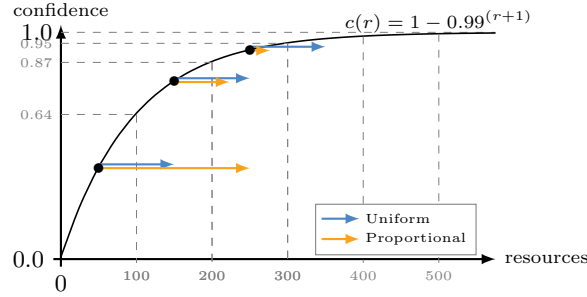


Fig. 4: Allocation of test budget according to common strategies

## 5.1   RQ1: Theoretical Evaluation

Naive approaches might coincidentally be equally good as elaborate techniques, given the right system architecture and initial confidences. We therefore chose to perform our comparison on a set of randomly generated initial confidences (later referred to as starting points (SPs)) and fault trees (FTs). It is natural to expect `Astrahl`'s insight into system structure and component confidences to outperform naive strategies as the systems grow in size and complexity.[2] Therefore, we only verify `Astrahl`'s superiority on relatively small systems and simple confidence functions. We thus fixed an FT size of six devices connected by five binary `AND` and `OR` gates. Furthermore, confidences behave according to

---

[2] Functional optimisation may not be as efficient in larger dimensions, but even a naive estimate should give a better result than completely ignoring system structure.

the function $c(r) = 1 - 0.99^{(r+1)}$ for all devices, where $r$ represents the invested resources and $c$ the confidence, as displayed in Figure 4.

Using these settings, we generated 200 FTs and instantiated each with 100 random SPs in the range of 100 to 300. Using this data set we let `Astrahl` and its competitors distribute total budgets of size 1, 10, 50, 100, 250, 500, and 1000.

## 5.2   RQ2: Empirical Evaluation

To address RQ2 it is necessary to create an evaluation setting that allows the simulated distribution of (hidden) component faults, their (potential) discovery through testing or experimentation effort and subsequent removal, and finally a calculation of the system confidence based on the remaining, undiscovered faults. Our approach is based on the probabilistic creation of fault distributions (FDs), i.e. assignments of faults to components according to their respective confidences. These faults will be probabilistically found and removed by allocating resources to a component, simulating e.g. experimentation or testing. Our hypothesis is that, given initial component confidences that reflect the components' reliabilities, `Astrahl` should be able to outperform its competitor algorithms and on average lead to higher overall system confidence.

The evaluation process is split into three phases. First, faults are assigned to components according to geometric distributions with parameter $p = 1-c$, where $c$ is our initial confidence in the component. Therefore, components in which we have more confidence will on average contain fewer faults. Next, the faults are removed probabilistically during a "testing phase" as follows. We arbitrarily assume that each test costs 10 resources[3]. Each fault has an observability of 0.1, i.e. each test has a 10% chance to detect this particular fault. When a TRA strategy assigns $r$ resources to a component with $n$ faults, $t = \lfloor \frac{r}{10} \rfloor$ full tests are run on it. Each test has a 10% chance to find and remove each of a component's $n$ faults. If $r > 10t$, i.e. there is remaining budget, a "partial" test is run with proportionally reduced chance to find faults. After this phase, we end up with $n'$ faults in each component. Finally, the system fault probability is calculated. As above, during operation each fault's observability is 0.1, so a component's failure probability can be calculated as $1 - 0.9^{n'}$ if it contains $n'$ faults. The entire system's failure probability can then be calculated using all components' failure probabilities and the standard propagation of fault probabilities in FTs.

Due to the probabilistic nature of this evaluation we repeated this process for 50 FTs, 50 SPs (range 10 to 70) for each FT and 50 random FDs for each SP, totalling to 125,000 FDs. We computed test resource allocations using test budgets of 60, 120, 240, 360, 480 and 600, executed the testing and fault removal process 100 times for each FD and test budget, and subsequently calculated the average FT failure probability, for a total of 75,000,000 computations.

---

[3] It would equally be possible to assume a test costs one resource and scale the budget.

### 5.3  Evaluation Results

The experiment results for RQ1 are shown in Figure 5a and Figure 5b. Figure 5a shows the average system reliability according to the total budget and relative difference to `Astrahl`'s score $\frac{(1-r)-(1-r')}{1-r} = \frac{r'-r}{1-r}$, where $r$ is the reliability computed by `Astrahl`, and $r'$ that computed by its competitor (this measure is closer to intuition than $(r-r')/r$ when both confidences are close 1). The error bars in Figure 5b represent mean squared error in system reliability. Note that `Astrahl` outperforms each of the competitors independent of the budget size. It is also noteworthy that although with higher budgets the system becomes very reliable independent of the strategy, the relative performance increase of `Astrahl` when compared to its competitors grows significantly. In other words, spending a large amount of resources increases obviously the system performance, but it is still better to follow `Astrahl`'s suggestions.

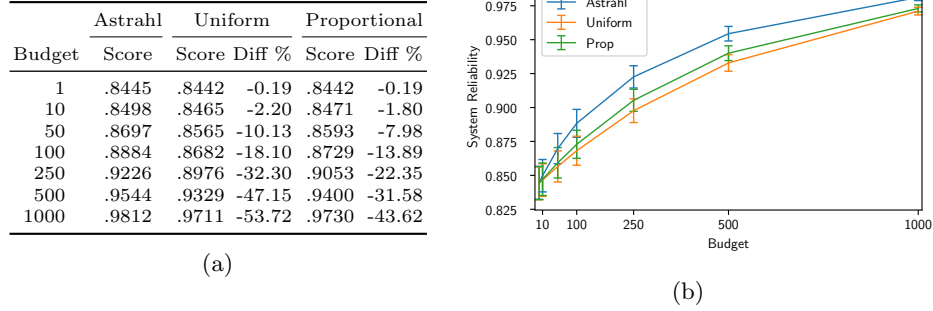| | Astrahl | Uniform | | Proportional | |
|---|---|---|---|---|---|
| Budget | Score | Score | Diff % | Score | Diff % |
| 1 | .8445 | .8442 | -0.19 | .8442 | -0.19 |
| 10 | .8498 | .8465 | -2.20 | .8471 | -1.80 |
| 50 | .8697 | .8565 | -10.13 | .8593 | -7.98 |
| 100 | .8884 | .8682 | -18.10 | .8729 | -13.89 |
| 250 | .9226 | .8976 | -32.30 | .9053 | -22.35 |
| 500 | .9544 | .9329 | -47.15 | .9400 | -31.58 |
| 1000 | .9812 | .9711 | -53.72 | .9730 | -43.62 |

(a)



(b)

Fig. 5: Theoretical evaluation: average system reliability and relative difference

Figure 6a and Figure 6b display the results of the empirical evaluation (RQ2). Here, the error bars correspond to mean squared error of the average over all FDs (for each SP). As can be seen, also here `Astrahl` outperforms other TRA strategies. Interestingly though, `Astrahl`'s relative advantage is not as high. An initial investigation suggests the cause for this observation at the discrete nature of the evaluation setting, where in many cases all faults of a component are removed, which leads to full confidence in this component.

Summarising our evaluations it can be said that `Astrahl` is better-suited for identifying where to place test effort than alternative approaches. The experiments show significant relative gains in both theoretical and practical approaches, even for rather small, straightforward systems as in our setting. For more complex systems, we expect `Astrahl`'s insight into the system's structure and the components' confidence should make its advantage even clearer, although this has yet to be validated by experimental results.

| | Astrahl | Uniform | | Proportional | |
|---|---|---|---|---|---|
| Budget | Score | Score | Diff % | Score | Diff % |
| 60 | .8982 | .8890 | -9.04 | .8887 | -9.33 |
| 120 | .9146 | .9000 | -17.10 | .8995 | -17.68 |
| 240 | .9380 | .9188 | -30.97 | .9179 | -32.42 |
| 360 | .9541 | .9341 | -43.57 | .9329 | -46.19 |
| 480 | .9657 | .9466 | -55.69 | .9451 | -60.06 |
| 600 | .9743 | .9567 | -68.48 | .9550 | -75.10 |

(a)
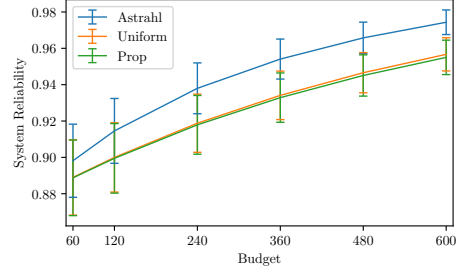


(b)

Fig. 6: Empirical evaluation: system reliability and relative difference

## 6  Conclusion and Future Work

We have defined *Quantitative Confidence Logic*, which represents confidence in assertions and have argued that this logic can help us take system architecture into account when solving the test resource allocation problem (TRAP) and shown the validity of the approach through experimental results. We have also argued that this approach is widely applicable, e. g., because it does not rely on particular assumptions about fault distributions.

The simplicity and versatility of our approach makes it possible to tackle different problems with the same ingredients. An obvious possible future work is to study the TRAP in different settings, for example by implementing multi-objective optimisation, or by studying it in a broader setting, where the confidence gained by running a test depends on the result of the test. We should also experimentally validate our expectation on the scalability of our approach in industry-scale case studies. It would also be interesting to see how solving the TRAP when optimising the total confidence $t + f$ compares to solving it with the current setting, especially on volatile systems. Another possible direction is to study how this approach can be used to solve test prioritisation between different components of a system.

We also want to investigate the logic itself more thouroughly from a purely logical point of view. For example, by changing the interpretation of connectives in three-valued logic, or using different T-norms and T-conorms in the definitions of the rules. Another interesting aspect would be to investigate its links with fuzzy logics and Dempster-Shafer theory deeper, as there seems to be some deep connections. In particular, ties to fuzzy logics would give a bridge between a logic about confidence and a logic about truth, which could help us develop QCL further.

## References

1. Bergmann, M.: An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems. Cambridge University Press (2008).

https://doi.org/10.1017/CBO9780511801129

2. Bertsekas, D.P.: Constrained Optimization and Lagrange Multiplier Methods. Academic press (2014)

3. Carrozza, G., Pietrantuono, R., Russo, S.: Dynamic test planning: a study in an industrial context. International Journal on Software Tools for Technology Transfer **16**(5), 593–607 (2014). https://doi.org/10.1007/s10009-014-0319-0

4. Dempster, A.P.: Upper and lower probabilities induced by a multivalued mapping. The Annals of Mathematical Statistics **36**, 325–339 (1967). https://doi.org/10.1007/978-3-540-44792-4_3

5. Ericson, C.A.: Fault Tree Analysis. In: System Safety Conference, Orlando, Florida. vol. 1, pp. 1–9 (1999)

6. Esteva, F., Godo, L.: Monoidal T-norm based logic: Towards a logic for left-continuous T-norms. Fuzzy sets and systems **124**(3), 271–288 (2001). https://doi.org/10.1016/S0165-0114(01)00098-7

7. Goel, A.L., Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. IEEE transactions on Reliability **28**(3), 206–211 (1979). https://doi.org/10.1109/TR.1979.5220566

8. Hájek, P.: Metamathematics of Fuzzy Logic, vol. 4. Springer Science & Business Media (2013). https://doi.org/10.1007/978-94-011-5300-3

9. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary computation **9**(2), 159–195 (2001). https://doi.org/10.1162/106365601750190398

10. Huang, C.Y., Lo, J.H.: Optimal resource allocation for cost and reliability of modular software systems in the testing phase. Journal of Systems and Software **79**(5), 653–664 (2006). https://doi.org/10.1016/j.jss.2005.06.039

11. Kamei, Y., Shihab, E.: Defect Prediction: Accomplishments and Future Challenges. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER). vol. 5, pp. 33–45. IEEE (2016). https://doi.org/10.1109/SANER.2016.56

12. Pietrantuono, R.: On the Testing Resource Allocation Problem: research trends and perspectives. Journal of Systems and Software **161**, 110462 (2020). https://doi.org/10.1016/j.jss.2019.110462

13. Pietrantuono, R., Russo, S., Trivedi, K.S.: Software Reliability and Testing Time Allocation: An Architecture-Based Approach. IEEE Transactions on Software Engineering **36**(3), 323–337 (2010). https://doi.org/10.1109/TSE.2010.6

14. Sallak, M., Schön, W., Aguirre, F.: Reliability assessment for multi-state systems under uncertainties based on the Dempster–Shafer theory. IIE Transactions **45**(9), 995–1007 (2013). https://doi.org/10.1080/0740817X.2012.706378

15. Shafer, G.: A mathematical theory of evidence, vol. 42. Princeton University Press (1976). https://doi.org/10.2307/j.ctv10vm1qb

16. Van Laarhoven, P.J., Aarts, E.H.: Simulated annealing. In: Simulated annealing: Theory and applications, pp. 7–15. Springer (1987). https://doi.org/10.1007/978-94-015-7744-1_2

17. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. Tech. rep., Nuclear Regulatory Commission Washington DC (1981)

18. Yamada, S., Osaki, S.: Software Reliability Growth Modeling: Models and Applications. IEEE Transactions on Software Engineering **SE-11**(12), 1431–1437 (1985). https://doi.org/10.1109/TSE.1985.232179

19. Zhang, G., Su, Z., Li, M., Yue, F., Jiang, J., Yao, X.: Constraint Handling in NSGA-II for Solving Optimal Testing Resource Allocation Problems. IEEE Transactions on Reliability **66**(4), 1193–1212 (2017). https://doi.org/10.1109/TR.2017.2738660