

# A Functional Framework for Result Checking<sup>\*</sup>

Gilles Barthe<sup>1</sup>, Pablo Buiras<sup>1,2</sup>, and César Kunz<sup>1</sup>

<sup>1</sup> IMDEA Software, Spain

<sup>2</sup> FCEIA, Universidad Nacional de Rosario, Argentina

**Abstract.** Result checking is a general methodology for ensuring that untrusted computations are valid. Its essence lies in defining efficient checking procedures to verify that a result satisfies some expected property. Result checking often relies on certificates to make the verification process efficient, and thus involves two strongly connected tasks: the generation of certificates and the implementation of a checking procedure. Several ad-hoc solutions exist, but they differ significantly on the kind of properties involved and thus on the validation procedure. The lack of common methodologies has been an obstacle to the applicability of result checking to a more comprehensive set of algorithms. We propose the first framework for building result checking infrastructures for a large class of properties, and illustrate its generality through several examples. The framework has been implemented in Haskell.

## 1 Introduction

Computer programs are error-prone, making it a challenge to assure the validity of computations. Errors arise from many sources: programming mistakes, rounding-off errors in floating-point computations, defects in the underlying hardware, or simply because part of a computation has been delegated to some untrusted party. A general methodology for ascertaining the correctness of the computations performed by a program  $F$  is to rely on an independent result checker  $V$ , which guarantees the correctness of the computation performed by  $F$ . A simple example of result checker is a boolean-valued predicate between inputs and outputs that only returns true on pairs  $(x, y)$  such that  $F(x) = y$ , where  $F$  is a program with a single input  $x$  and single output  $y$ . For example, a result checker for the program  $F$  computing the square root  $y$  of  $x$  is the program  $V$  that returns the boolean expression  $(y^2 \leq x) \wedge (x < y^2 + 2y + 1)$ . However, result checkers may in general rely on additional inputs, called certificates, that guarantee efficient execution. A typical example of result checker which relies on certificates is the checker for greatest common divisor (gcd), which takes as arguments, in addition to  $a$  and  $b$  for which the gcd must be computed, two additional values  $u$  and  $v$  (which constitute the certificate), and the candidate gcd  $d$ , and returns the boolean value  $d = ua + vb \wedge d \mid a \wedge d \mid b$ .

---

<sup>\*</sup> Partially funded by the EU project HATS and Spanish project Desafios-10 and Community of Madrid project Comprometidos.

While result checking offers a general methodology to guarantee the correctness of computations, and thus is potentially applicable to many domains, its applications have been circumscribed to a few and rather specific settings; see Section 2. The main challenge in broadening the scope of result checking is finding a systematic means of building, for a large class of properties, a result checking framework that provides (a) for every property  $P$  in the class, a type  $\text{wit}_P$  of certificates; (b) a means of generating certificates<sup>1</sup>; and (c) a checker  $\text{check}_P : A \rightarrow \text{wit}_P \rightarrow \mathbf{bool}$  (where  $A$  is the carrier of  $P$ ), such that for all  $a : A$  and  $w : \text{wit}_P$  we have

$$(\text{check}_P a w = \mathbf{true}) \Rightarrow P a.$$

The purpose of this article is to provide a framework for building and verifying certified results for a large class of algorithms. The framework is implemented on top of the Haskell [9,10] programming language, and provides:

- *certifying combinators*, which extend the usual combinators of functional programming with facilities for turning certificates of the combinators’ inputs into certificates of the combinators’ outputs. Certifying combinators can be combined to produce certified results;
- a *generic* checker function, that takes a *representation* of a property and behaves like a checker for it.

The combination of certifying combinators and the generic checker allows us to obtain a result checking framework for a large class of properties, including sorting and searching algorithms, or primality testing.

Although our results are developed in the setting of a sequential language, our primary motivation is to provide a certificate-based infrastructure for guaranteeing the correctness of large distributed computations among untrusted hosts [1]. The results of this paper can be embedded in the framework of [16] for this purpose.

*Outline.* In Section 3, we define a generic checking function for a large class of predicates that includes inductively defined ones. In Section 4, we propose two methods for the generation of certificates. Both are defined as an extension of the original producer algorithm. The first one is based on the recursion pattern defining the producer algorithm. The second one is a general approach for the certification of nondeterministic computations.

## 2 Related Work

Blum and Kannan [2] were among the first to recognise the importance of result checking as a general method for discovering bugs in programs, and to advocate its superiority over testing, which can be unreliable, or program verification,

---

<sup>1</sup> Note that no property of the certificate generation mechanism is needed for checked results to be correct, i.e., it is not necessary to trust the certificate generators.