Towards Verifying Exact Conditions for Implementations of Density Functional Approximations

Sameerah Helal, Zhe Tao, Cindy Rubio-González, Francois Gygi, Aditya V. Thakur
Department of Computer Science
University of California, Davis
Davis, U.S.A.
{shelal, zhetao, crubio, fgygi, avthakur}@ucdavis.edu

***Abstract*—Density Functional Theory (DFT) is used extensively in the computation of electronic properties of matter, with applications in solid state physics, computational chemistry, and materials science. Approximating the exchange-correlation (XC) functional is the key to the Kohn-Sham DFT approach, the basis of most DFT calculations. The choice of this density functional approximation (DFA) depends crucially on the particular system under study, which has resulted in the development of hundreds of DFAs. Though the exact density functional is not known, researchers have discovered analytical properties of this exact functional. Furthermore, these *exact conditions* are used when designing DFAs.**

This paper presents XCVERIFIER, the first approach for verifying whether a DFA implementation satisfies the DFT exact conditions. XCVERIFIER was evaluated on five DFAs from the popular LIBXC library and seven exact conditions used in recent work by Pederson and Burke. XCVERIFIER was able to verify or find violations for a majority of the DFA-condition pairs, demonstrating the feasibility of using formal methods to verify DFA implementations. However, it timed out on all conditions of the recent SCAN functional, revealing directions for future work.

***Index Terms*—Density functional theory, Formal methods.**

I. INTRODUCTION

Density functional theory (DFT) [1]–[3] is a widely used approximation in the computation of electronic properties of matter. DFT calculations have been used to predict diverse properties from atomic binding energies and chemical reactivities to electronic conductivities and magnetic phenomena [4]. Consequently, DFT finds applications in a number of scientific and engineering fields, including solid state physics, computational chemistry, and materials science, and has been implemented in widely-used scientific software such as ABINIT [5], cp2k [6], ERKALE [7], Psi4 [8], Octopus [9], Qbox [10], and Quantum ESPRESSO [11].

Originating in the foundational work of Hohenberg and Kohn in 1964 [12], DFT has provided a practical way to reduce the complexity of conventional electronic structure methods by identifying the electron density function $n(r)$—a real function in three-dimensional space—as the fundamental quantity from which all other properties of a physical system can be derived. The Kohn-Sham (KS) [13] approach is currently the basis of most DFT calculations, and states that only the exchange-correlation (XC) energy portion of a functional needs to be approximated. However, the exact expression for this XC energy functional $E_{\mathrm{xc}}[n]$, which describes the complex electron-electron interactions within systems, is not known and is incredibly difficult to approximate [14].

Researchers have developed hundreds of approximations to the XC energy functional, which have the form:

$$\widetilde{E}_{\mathrm{xc}}[n] = \int n(r)\, \tilde{\epsilon}_{\mathrm{xc}}\big(n(r), \nabla n(r), \nabla^2 n(r), \dots\big)\, dr, \quad (1)$$

where the term $\tilde{\epsilon}_{\mathrm{xc}}$ is the *density functional approximation (DFA)*. The choice of DFA in a particular application depends crucially on the known or expected physical properties of the system under study [15]–[17].

This proliferation of DFAs is reflected in the fact that the popular LIBXC software library [18], which provides numerical implementations of DFAs, currently includes over 500 functionals [19]. DFAs are of varying degrees of complexity [20]: (i) local density approximations (LDAs) depend only on the electron density n, (ii) generalized-gradient approximations (GGAs) [21] depend on the electron density n and its gradient ∇n, and (iii) meta-GGAs have a further dependence on the Laplacian $\nabla^2 n$ and the local kinetic energy density τ.

The DFA $\tilde{\epsilon}_{\mathrm{xc}}$ in Equation 1 is a mathematical function with a known, albeit complicated, analytical form. For instance, the following equation shows just the exchange (X) part $\tilde{\epsilon}_{\mathrm{x}}^{\mathrm{PBE}}$ of the Perdew-Burke-Ernzerhof (PBE) functional [22], a commonly used GGA functional [23]:

$$\tilde{\epsilon}_{\mathrm{x}}^{\mathrm{PBE}}(\rho,\sigma) = \frac{2.884\rho^{1/3}\left(-28.944\pi^{4/3}\rho^{8/3} - 0.174\pi^2\sigma\right)}{\pi^{1/3}\cdot\left(77.184\pi^{4/3}\rho^{8/3} + 0.257\pi^2\sigma\right)}$$

However, the correlation part of PBE is significantly more complex with over 300 operations in the LIBXC implementation. The SCAN meta-GGA functional [24] is even more complex with over 1000 operations, including transcendental functions such as $\exp$ and $\log$.

Creating a new DFA is an art mastered by only a few researchers as of today, and the functional forms used to define the DFAs vary considerably. DFA designs fall into two

categories: empirical and non-empirical. Empirically-designed DFAs (e.g., LYP [25]) are tailored for molecular chemistry applications and perform well on molecular benchmarks [26], [27]. Non-empirically designed DFAs are constructed to satisfy some *exact conditions*, which are known analytical properties of the exact functional (Section II). For example, the correlation energy E_C non-positivity condition states that the correlation energy cannot be positive, i.e., $E_{\mathrm{c}}[n] \leq 0$ [28]. Furthermore, so-called norms are imposed on DFAs by requiring that they reproduce correctly some known physical systems, e.g., a hydrogen or a helium atom for which exact results are available. The SCAN functional is built to satisfy as many as 17 constraints and norms [24].

Recently, Pederson and Burke (PB) [28] checked whether the LIBXC implementations of various DFAs satisfy DFT exact conditions. In particular, they checked seven exact conditions by considering their corresponding *local condition* (Section II). These local conditions were assessed for a given DFA by performing a grid search over the inputs to the DFA and checking whether each input-output pair satisfies the local condition. Many of the conditions require gradient calculations, which were numerically approximated. The PB approach is the state of the art in the DFT community, and was the first to perform a large scale study of the role of exact conditions in density functional development. For instance, they found that many empirical DFAs satisfy these exact conditions in certain regions even though they were designed without explicit adherence to these exact conditions.

This paper addresses the problem of automatically verifying whether the implementation of a DFA satisfies the exact conditions of the density functional. It is the first to apply formal-methods techniques to density functional theory. The aim is to provide formal guarantees related to the correctness of existing DFA implementations; viz., to formally verify if the DFA implementation satisfies the exact conditions, and to determine the areas of the input domain where it does not. As a step towards solving this problem, we designed and implemented XCVERIFIER, a tool that verifies whether a LIBXC functional, implemented in Maple [29], satisfies the given local condition (and, hence, the corresponding exact condition) using the `dReal` solver [30] (Section III). XCVERIFIER also computes any required derivatives symbolically, avoiding any potential issues arising from their numerical approximation. We also implemented a domain-splitting technique to improve the performance of XCVERIFIER (Algorithm 1).

We evaluated XCVERIFIER by verifying seven exact conditions (from Pederson and Burke) for five popular DFAs: PBE [22], SCAN [24], LYP [25], AM05 [31], and VWN RPA [32], which cover the different types of DFAs (LDA, GGA, and meta-GGA), as well as different design categories (empirical and non-empirical). Some conditions do not apply to certain DFAs, which left us with 29 DFA-condition pairs. As shown in Section IV-B, XCVERIFIER was successfully able to verify or find counterexamples for 13 pairs, and it is able to partially verify an additional seven pairs (Table I). These results demonstrate the feasibility of using formal-methods techniques to verify DFA implementations. However, XCVERIFIER times out for nine pairs: one property for the PBE DFA, one property for AM05, and *all* of the properties for SCAN. This motivates further research on formal methods for DFT.

To further validate our approach, we compared the results from the PB approach with XCVERIFIER (Section IV-C): both approaches find similar regions where the conditions were violated or satisfied (Table II).

The contributions of the paper are as follows:

- A tool, XCVERIFIER, for automatically verifying exact conditions for density functional approximations (Section III).
- An evaluation of XCVERIFIER using five DFAs and seven exact conditions along with a comparison with the state-of-the-art grid-search approach (Section IV).

Our preliminary results demonstrate the feasibility of using formal methods to prove the correctness DFT implementations, and reveal avenues for future work (Section VI).

II. EXACT CONDITIONS IN DFT

This section lists the exact conditions of the density functional E_{xc} considered in this paper. Our description closely follows that in Pederson and Burke [28]. Each exact condition has a corresponding *local condition* such that if the DFA $\tilde{\epsilon}_{\mathrm{xc}}$ satisfies the local condition, then the functional $\widetilde{E}_{\mathrm{xc}}$ satisfies the (global) exact condition. Note that the converse is not true: violating the local condition does not imply that the exact condition is violated. Furthermore, the region where the local condition is violated for a DFA can depend on the particular implementation of the functional. In this paper, we focus on verifying the LIBXC *implementation of a DFA*.

The local conditions take the exchange (correlation) enhancement factor $\widetilde{F}_{\mathrm{xc}}$, which is a function of $\epsilon_{\mathrm{xc}}\big(n(r)\big)$, the local value of the exchange (correlation) energy for the DFA. Here, $n(r)$ is the electron density at the point r representing the position of an electron. For GGA functionals, the inputs $n(r)$ and $\nabla n(r)$ are usually expressed in terms of the Wigner-Seitz radius $r_s = (4\pi n/3)^{-1/3}$ and $s = |\nabla n|/(2(3\pi^2)^{1/3}n^{4/3})$. Given the expressions for the DFA exchange and correlation energies $\tilde{\epsilon}_{\mathrm{x}}$ and $\tilde{\epsilon}_{\mathrm{c}}$ in terms of s and r_s, we can compute $\widetilde{F}_{\mathrm{x}}$ and $\widetilde{F}_{\mathrm{c}}$ of $\widetilde{F}_{\mathrm{xc}}$:

$$\widetilde{F}_{\mathrm{xc}}[n(r)] = \widetilde{F}_{\mathrm{x}} + \widetilde{F}_{\mathrm{c}} = \frac{\tilde{\epsilon}_{\mathrm{xc}}[n(r)]}{\epsilon_{\mathrm{x}}^{\mathrm{unif}}[n(r)]}. \tag{2}$$

We now list the DFT exact conditions along with their corresponding local condition:

(EC1) The correlation energy (E_{c}) non-positivity condition [28] is defined as $E_{\mathrm{c}}[n] \leq 0$. The corresponding local condition is

$$\epsilon_{\mathrm{c}}(n(r)) \leq 0. \tag{3}$$

It can also be expressed in terms of $\widetilde{F}_{\mathrm{c}}$ as

$$\widetilde{F}_{\mathrm{c}} \geq 0. \tag{4}$$

(EC2) The E_c scaling inequality [28], [33], $(\gamma-1)E_c[n_\gamma] \geq \gamma(\gamma-1)E_c[n]$, has the local condition

$$\frac{\partial \widetilde{F}_c}{\partial r_s} \geq 0. \tag{5}$$

(EC3) The $U_c(\lambda)$ monotonicity condition [28], [34] is $\frac{dU_c(\lambda)}{d\lambda} \leq 0$, where $U_c(\lambda) = d(\lambda^2 E_c[n_{1/\lambda}])/d\lambda$ represents the correlation energy adiabatic connection curves. The corresponding local condition is

$$\frac{\partial^2 \widetilde{F}_c}{\partial r_s^2} \geq \frac{-2}{r_s} \cdot \frac{\partial \widetilde{F}_c}{\partial r_s}. \tag{6}$$

(EC4) Lieb-Oxford bound [28], [35] is $U_{xc} \geq C_{LO} \int d^3r\, n(r)\, \epsilon_x^{\text{unif}}\big(n(r)\big)$, where $U_{xc}[n] = E_{xc}[n] - T_c[n]$ is the potential correlation energy and $C_{LO} = 2.27$ is the Lieb-Oxford constant, following [28]. The corresponding local condition is

$$\widetilde{F}_{xc} + r_s \frac{\partial \widetilde{F}_c}{\partial r_s} \leq C_{LO}. \tag{7}$$

(EC5) The Lieb-Oxford extension to E_{xc} [28], [35] is a generalization of the Lieb-Oxford bound with E_{xc} instead of U_{xc} : $E_{xc} \geq C_{LO} \int d^3r\, n(r)\, \epsilon_x^{\text{unif}}\big(n(r)\big)$. The corresponding local condition is

$$\widetilde{F}_{xc} \leq C_{LO} \tag{8}$$

(EC6) The T_c upper bound condition [28], [34] is $T_c[n_\gamma] \leq -\gamma\left(\frac{\partial E_c[n_\gamma]}{\partial \gamma}\right) + E_c[n_\gamma]$, with corresponding local condition

$$\frac{\partial \widetilde{F}_c}{\partial r_s} \leq \frac{\widetilde{F}_c(\infty) - \widetilde{F}_c}{r_s} \tag{9}$$

where $\widetilde{F}_c(\infty)$ is the limit of $\widetilde{F}_c$ as $r_s \to \infty$.

(EC7) The *conjectured* T_c upper bound [28], [36]–[38] is $T_c[n] \leq -E_c[n]$ with local condition

$$\frac{\partial \widetilde{F}_c}{\partial r_s} \leq \frac{\widetilde{F}_c}{r_s}. \tag{10}$$

III. XCVERIFIER

This section describes the design of the XCVERIFIER tool, which verifies whether a DFA implemented in the LIBXC library [18] satisfies the DFT local conditions in Section II. XCVERIFIER consists of (i) XCENCODER, which encodes the given local condition for a given LIBXC functional into a formula ψ, and (ii) VERIFIER, which verifies whether this encoded formula ψ is always true (valid) in the given input domain. Because DFAs involve non-linear arithmetic and transcendental functions, we chose the `dReal` solver [30] as the core solver in VERIFIER. As a consequence, the formula ψ generated by XCENCODER is a `dReal` formula.

Algorithm 1 VERIFIER(input domain D, formula ψ)

```
1: if D < t then
2:     return
3: result, x ← dReal(φ_D ∧ ¬ψ)
4: if result = UNSAT then
5:     print "Verified condition over domain D"
6:     return
7: if result = SAT then
8:     if valid(x) then
9:         print "Found counterexample x"
10:    else
11:        print "Verification inconclusive over domain D"
12: else
13:    print "Verification timed out over domain D"
14: for all D' in split(D) do
15:    VERIFIER(D', ψ)
16: return
```

A. XCENCODER

Given a LIBXC functional, XCENCODER first translates the Maple code for $\tilde{\epsilon}_{xc}$ to Python using the `CodeGeneration` package from Maple. We implemented a symbolic execution engine for (a subset of) Python that generates the `dReal` expression corresponding to the DFA. Though DFA implementations do not contain loops, arrays, etc., they do contain (non-recursive) function calls and if-then-else statements.

XCENCODER then constructs the `dReal` formula ψ that encodes the given local condition for the particular functional. Encoding the local condition corresponding to the E_c non-positivity condition is straightforward: XCENCODER uses the `dReal` expression for $\tilde{\epsilon}_c$ to directly construct the `dReal` formula $\tilde{\epsilon}_c \leq 0$.

However, the local conditions corresponding to exact conditions such as E_c scaling, Lieb-Oxford, U_c monotonicity, and T_c upper bound require computation of one or more derivatives, In such cases, XCENCODER uses SymPy [39] to symbolically compute the derivatives. Furthermore, the local condition corresponding to the T_c upper bound condition (Equation 9) requires computing $\lim_{r_s \to \infty} \widetilde{F}_c$. Following [28], XCENCODER substitutes an appropriately large value to approximate this limit at infinity, viz. $\widetilde{F}_c|_{r_s=100}$.

B. VERIFIER

Algorithm 1 presents the pseudo-code for VERIFIER. The VERIFIER component of XCVERIFIER takes as input (i) the formula ψ encoding the exact condition for the DFA, and (ii) a domain for the inputs to the DFA. We use the same input bounds as used in Pederson and Burke [28]; for example, for GGA functionals, the domain for r_s is the interval $[0.0001, 5]$, and that for s is $[0, 5]$. Consequently, for a GGA functional, the VERIFIER is trying to prove the validity, or satisfiability, of the following formula:

$$\forall r_s, s\,.\; (r_s \in [0.0001, 5] \wedge s \in [0, 5]) \implies \psi. \tag{11}$$

Proving the validity of Equation 11 is equivalent to proving that the following formula is unsatisfiable:

$$r_s \in [0.0001, 5] \wedge s \in [0, 5] \wedge \neg\psi. \tag{12}$$

VERIFIER uses the `dReal` solver [30] to check the satisfiability of the above formula.

`dReal` implements a delta-complete decision framework: given a formula φ, `dReal` returns `UNSAT`—φ is unsatisfiable—or δ-`SAT`—the δ-weakening φ^δ is satisfiable for the returned model, where a model is a satisfying assignment to the variables in the formula. The δ-weakening φ^δ of a formula φ is numerical relaxation of φ such that (i) a model that satisfies φ will always satisfy φ^δ; however, the reverse is not necessarily true, and (ii) if φ^δ is unsatisfiable, then φ is also unsatisfiable. This relaxation results in `dReal` now being decidable for nonlinear formulas including those with transcendental functions, which are common in DFAs.

Though the δ-satisfiability problem is decidable in principle, in practice, the `dReal` solver could also *timeout*: it was unable to determine whether formula φ is `UNSAT` or δ-`SAT` in the given time limit. Our preliminary results showed that `dReal` would timeout on the formula in Equation 12 for most functional/condition pairs, even when given a time limit of 24 hours. To improve the performance of VERIFIER, we implement a domain-splitting technique that partitions the input domain and uses `dReal` to solve the formula $\neg\psi$ on each subdomain separately. This simple strategy greatly improves the performance of VERIFIER. Furthermore, one of the goals of XCVERIFIER is to determine the input regions where the DFA implementation violates the local condition. Thus, VERIFIER also performs the domain-split when `dReal` returns a valid model: an input that indeed violates the given condition. This allows VERIFIER to isolate the subregions where the DFA implementation violates the local condition.

Algorithm 1 presents the pseudo-code for VERIFIER, and Figure 1f shows a graphical representation of the output of VERIFIER when verifying the conjectured T_c upper bound condition for the PBE functional.

The `dReal` solver is called on Line 3 to find a satisfying assignment to the formula $\varphi_D \wedge \neg\psi$, where φ_D is the formula encoding the domain constraints on the inputs to the DFA and ψ encodes the local condition for the functional. If the result is `UNSAT`, then VERIFIER returns after recording that the condition was verified over domain D (Line 6). This is indicated using ■ in Figure 1f. If the result is δ-`SAT`, `dReal` will also return a model x for the formula. In Line 8, `valid(x)` checks if this model is a valid counterexample by plugging the values back into ψ, which encodes the local condition. If the condition is indeed violated, then VERIFIER records it as a counterexample (Line 9), indicated using ■. Occasionally, `dReal` may return `SAT` with a model that does not actually violate ψ. This happens due to the δ-satisfiability procedure of `dReal`: the model must satisfy the weakened formula $(\neg\psi)^\delta$, but not necessarily the original formula $\neg\psi$. In such cases, VERIFIER records the result as inconclusive (Line 11), indicated using ■ in Figure 1f. We use a two hour time limit for the `dReal` solver; if `dReal` is unable to determine (un)satisfiability in this limit, VERIFIER interrupts it and records that the verification timed out for domain D (Line 13), indicated using ■ in Figure 1f. Increasing the timeout in our experiments did not enable `dReal` to solve more formulas.

Line 14 is executed if the result is `SAT` or when `dReal` times out. VERIFIER calls `split`(D), which partitions each input dimension of D into two equal parts. VERIFIER is recursively called on each subdomain D' in Line 15. We set a lower limit on the size of input domain as the base case for the recursion. On Line 2, VERIFIER returns if the given input domain is too small as determined by the threshold t; we used $t = 0.05$ in our experiments.

IV. EXPERIMENTAL RESULTS

This section evaluates the performance of XCVERIFIER for verifying local conditions for DFA implementations in LIBXC, and compares the results to the prior PB approach [28].

Our experiments were designed to answer the following research questions:

RQ1 Is XCVERIFIER able to verify or find counterexamples for local conditions of DFA implementations (Section IV-B)?

RQ2 How does XCVERIFIER compare to the PB approach (Section IV-C)?

A. Experimental Setup

We use the following five DFAs in our experiments: PBE [22], a popular non-empirical GGA DFA; SCAN [24], a fully constrained non-empirical meta-GGA DFA satisfying all known properties of DFAs; LYP [25], and empirical DFA that is a key component of several commonly-used DFAs; AM05 [31], which shows efficient and superior performance on solids [40]; and VWN RPA [32], an LDA functional.

For each of these DFAs, we consider each of the applicable conditions from Section II. Note that the Lieb-Oxford conditions only apply to functionals with both an exchange and correlation component available (e.g., PBE, SCAN).

The PB approach is the state of the art for assessing condition satisfaction in DFT. For a given DFA and condition, the PB approach draws 10^5 uniform samples each for r_s and s, which are then meshed into a grid. PB then calls the LIBXC implementation of the DFA for each of the points in the grid. This grid is used to numerically compute the limits and gradients necessary for the conditions using the NumPy package in Python [41]. Then the condition is checked at each point in the grid. The condition is assumed to be satisfied for the DFA if all the points in the grid pass the condition.

B. Verifying Local Conditions using XCVERIFIER

Table I summarizes the result of using XCVERIFIER to verify the local condition corresponding to each DFA exact condition in Section II for the five DFAs. The results described below are summarized in Table I. Visualizations for LYP and PBE are shown in Figures 1 and 2, respectively.

TABLE I: Verifying local conditions for DFT exact conditions for DFAs using XCVERIFIER.
✓: XCVERIFIER verified that the DFA satisfies the condition on the entire input domain;
✓*: XCVERIFIER verified that the DFA satisfies the condition on part of the input domain with the rest timing out/inconclusive;
?: XCVERIFIER reported timeout/inconclusive for all of the input domain;
−: the condition does not apply to the DFA;
✗: XCVERIFIER found a counterexample showing that the DFA does not satisfy the local condition.

Local condition	PBE	LYP	AM05	SCAN	VWN RPA
E_c **non-positivity** (Equation 4)	✓*	✗	✓	?	✓
E_c **scaling inequality** (Equation 5)	✓*	✗	✓*	?	✓
U_c **monotonicity** (Equation 6)	?	✗	?	?	✓
T_c **upper bound** (Equation 9)	✓*	✗	✓	?	✓
Conjectured T_c **upper bound** (Equation 10)	✗	✗	✓*	?	✓*
LO bound (Equation 7)	✓*	−	−	?	−
LO extension to E_{xc} (Equation 8)	✓	−	−	?	−

PBE For the E_c non-positivity condition and the Lieb-Oxford bound, XCVERIFIER is able to verify the entire input domain except for some timed-out and inconclusive regions along the s-axis (Figure 1d). The verified region is $r_s > 0.9375$ for the E_c non-positivity condition and $r_s > 0.0781$ for the Lieb-Oxford bound. The Lieb-Oxford *extension* to E_{xc} is verified for the entire input domain (Figure 1e). For the E_c scaling inequality, XCVERIFIER verifies most of the bottom third of the input domain and times out for the rest. There are also a few small inconclusive regions for high s. For the T_c upper bound condition, XCVERIFIER verifies the bottom two-thirds of the input domain and times out for the rest.

For the conjectured T_c upper bound, XCVERIFIER finds a large counterexample region covering the upper left diagonal of the input domain (Figure 1f). There is an inconclusive region along the border of the counterexample region. XCVERIFIER verifies or times out for the rest of the input domain.

Finally, XCVERIFIER times out everywhere for the U_c monotonicity condition.

LYP For the E_c non-positivity condition, XCVERIFIER finds counterexamples at $s > 1.6563$ (Figure 2d) and the remainder of the input domain is verified to satisfy the condition. For the E_c scaling inequality, the counterexamples are at $r_s < 2.5$ and $s > 1.4844$ (Figure 2e). For the T_c upper bound condition (Figure 2f), the counterexamples are in a small region at $r_s > 4.8437$ and $s > 2.4219$, and for the conjectured T_c upper bound condition, the region is $r_s > 0.625$ and $s > 1.3281$. For the U_c monotonicity condition, XCVERIFIER finds counterexamples at $s > 1.4844$ and $r_s < 1.4062$. The rest of the region is partially verified or timed-out. There are some small inconclusive or timed-out regions on the borders of the counterexample regions for each of the conditions.

LYP is the only DFA where XCVERIFIER finds counterexamples to all applicable properties.

AM05 XCVERIFIER verifies that AM05 satisfies the E_c non-positivity and the T_c upper bound conditions in the entire input domain. It also verifies most of the domain for the E_c scaling inequality and the conjectured T_c upper bound conditions, but times out along the s-axis at $r_s < 0.0781$ for the E_c scaling inequality and $r_s < 0.1562$ for conjectured T_c upper bound. For the U_c monotonicity condition, XCVERIFIER times out everywhere.

SCAN For the SCAN functional, XCVERIFIER times out for *all* of the conditions.

VWN RPA XCVERIFIER verifies that VWN RPA satisfies the E_c non-positivity, E_c scaling inequality, and T_c upper bound conditions for the entire input domain. It also verifies the U_c monotonicity condition where it timed out for several of the other functionals. For the conjectured T_c upper bound, it verifies the entire region except along the s-axis at $r_s < 0.0781$, where it returns inconclusive.

Summary for RQ1: In our evaluation of the seven exact conditions for the five DFAs, XCVERIFIER was able to verify or find counterexamples for 13 condition-DFA pairs and partially verify seven, as shown in Table I. This demonstrates the feasibility of using formal-methods techniques like XCVERIFIER in verifying DFA implementations. However, XCVERIFIER also timed out for nine of the 29 applicable pairs, particularly for the SCAN functional, which was designed to satisfy all known properties of DFAs. Thus, there is room for improvement in formal-methods techniques for DFT.

C. *Comparing* XCVERIFIER *to* PB

This section compares the results for XCVERIFIER to the PB approach. Table II shows the consistency between the results of PB and XCVERIFIER.

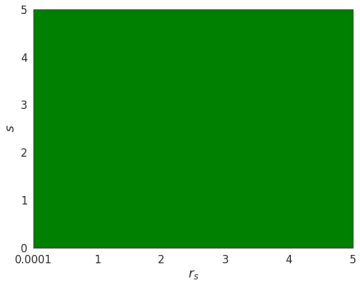
Out of the five functionals verified, XCVERIFIER returns counterexample regions consistent, $\odot$, with PB for the LYP DFA for all applicable properties. It also finds that the counterexample regions are consistent for the conjectured T_c upper bound for PBE.

Several of the results are *not inconsistent*, marked with $\odot^*$, meaning that neither method finds counterexamples for the condition. This is the case for when PB finds no counterexamples, and XCVERIFIER either verifies the entire region or partially verifies and partially times out (e.g., for PBE in Figure 1d).

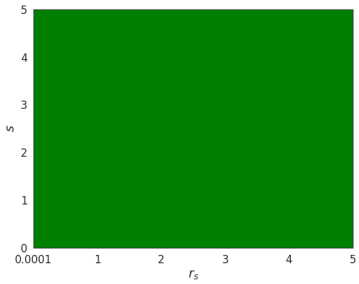
Due to the timeouts of XCVERIFIER for all of the properties for SCAN (Table I), we cannot compare those results for PB and XCVERIFIER. We mark them as ? in Table II.

TABLE II: Comparison between results for XCVERIFIER and PB approach.
⊙: results of PB are consistent with XCVERIFIER; ⊙*: results of PB are not inconsistent with XCVERIFIER;
−: condition does not apply to DFA; ?: XCVERIFIER times out.

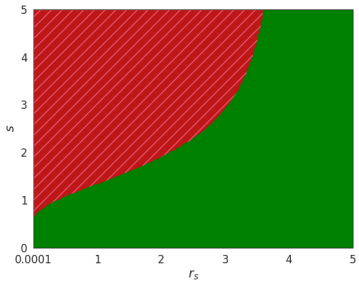
Local condition	PBE	LYP	AM05	SCAN	VWN RPA
E_c **non-positivity** (Equation 4)	⊙*	⊙	⊙*	?	⊙*
E_c **scaling inequality** (Equation 5)	⊙*	⊙	⊙*	?	⊙*
U_c **monotonicity** (Equation 6)	?	⊙	?	?	⊙*
T_c **upper bound** (Equation 9)	⊙*	⊙	⊙*	?	⊙*
Conjectured T_c **upper bound** (Equation 10)	⊙	⊙	⊙*	?	⊙*
LO bound (Equation 7)	⊙*	−	−	?	−
LO extension to E_{xc} (Equation 8)	⊙*	−	−	?	−



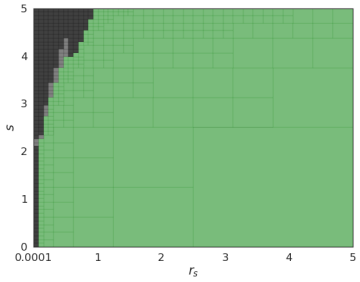
(a) E_c non-positivity w. PB



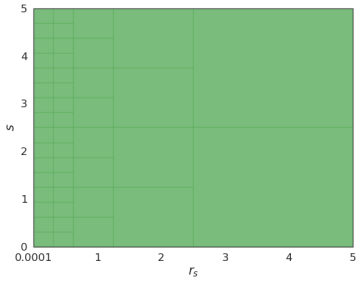
(b) Lieb-Oxford ext. w. PB



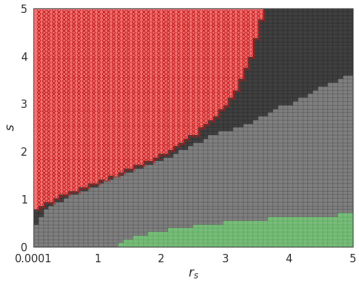
(c) Conj. T_c upper bound w. PB



(d) E_c non-positivity w. XCVERIFIER



(e) Lieb-Oxford ext. w. XCVERIFIER



(f) Conj. T_c upper bound w. XCVERIFIER

Fig. 1: Regions where the PBE functional satisfies or violates conditions according to PB (top) and XCVERIFIER (bottom).
For PB: ■ (region hatched) is a counterexample to the condition, ■ is a point that satisfies the condition.
For XCVERIFIER: ■ is a region that contains a counterexample marked with ×, ■ is a region that is verified to satisfy the condition, ■ indicates a timeout, and ■ indicates an inconclusive result.

Summary for RQ2: The results of XCVERIFIER are consistent with the PB approach (Table II). For DFA-condition pairs for which PB finds counterexamples, XCVERIFIER also finds counterexamples in similar regions. For pairs where PB does not find counterexamples, XCVERIFIER either verifies the input domain, or partially times out and partially verifies the domain.

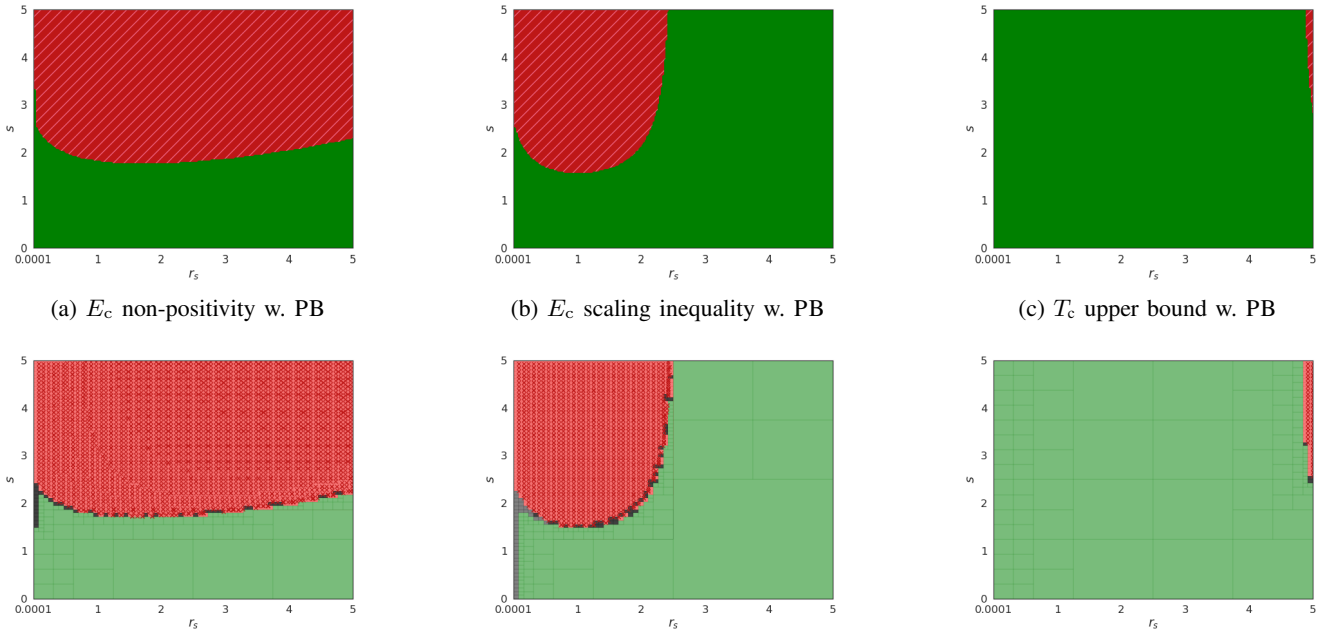
V. RELATED WORK

A. Analysis of Density Functional Theory Approximations

To the best of our knowledge, ours is the first work that uses formal methods to verify correctness in the context of density functional theory, an important scientific computing application. Prior work has used a testing-based approach, which is more scalable but does not provide formal guarantees. As discussed earlier, Pederson and Burke [28] use a grid-search to evaluate whether the DFA satisfies the DFT exact conditions. Lehtola and Marques [42] show that many recent DFAs are numerically ill-behaved by studying their accuracy in computing the total exchange-correlation energy.

B. Correctness in High Performance Computing

Correctness in scientific computing is recognized as a major challenge in HPC [43], [44], which needs formal methods that address the unique challenges in this domain. Progress to date includes the verification of mathematical properties in a conjugate gradient solver, a finite difference stencil, and a mesh quality metric [45], PDE solvers [46], specific properties of CG [47] and LU decomposition [48], and the floating-point equivalence of manually and automatically differentiated code [49]. The above are orthogonal to our goal of verifying



(a) E_c non-positivity w. PB (b) E_c scaling inequality w. PB (c) T_c upper bound w. PB

(d) E_c non-positivity w. XCVERIFIER (e) E_c scaling inequality w. XCVERIFIER (f) T_c upper bound w. XCVERIFIER

Fig. 2: Regions where the LYP functional satisfies or violates conditions according to PB (top) and XCVERIFIER (bottom). For PB: ■ (region hatched) is a counterexample to the condition, ■ is a point that satisfies the condition. For XCVERIFIER: ■ is a region that contains a counterexample marked with ×, ■ is a region that is verified to satisfy the condition, ■ indicates a timeout, and ■ indicates an inconclusive result.

exact conditions of density functional theory approximations. Extending our approach of using `dReal` to verify properties of other scientific computations is an interesting future direction.

C. Analysis of Floating-Point Programs

Many testing and analysis techniques for floating-point programs have been developed in the past decade. The first set of techniques are general approaches that aim to achieve high-coverage of numerical code [50], [51], conduct differential testing of numerical libraries [52], and perform mutation testing of floating-point expressions given a real specification [53]. However, it has been shown that simply achieving high code coverage in numerical programs does not uncover numerical issues in most cases [51]. Additionally, differential testing is not feasible for DFA implementations because most of them are unique.

A large body of work focuses on performing automated error analysis of floating-point programs [54]–[65]. While some of these approaches provide sound error bounds of floating-point programs, they suffer from important limitations with respect to program size and control structures supported. In the absence of techniques that can reason about floating-point error in non-trivial programs, a rich area of research in software testing has focused on how to efficiently generate inputs that *maximize* error in the output of a program [51], [66], [67], which can shed light on the potentially worst error a floating-point program could incur. Similarly, work has proposed techniques to generate inputs that trigger floating-point exceptions [68], [69]. Calculating error bounds of DFAs is orthogonal to our goal of verifying physical and numerical properties of their implementations.

Other existing work on floating-point programs has explored finding function input ranges, also referred to as regimes, with the purpose of improving the accuracy of floating-point expressions [70] or optimizing floating-point efficiency [71]. These approaches are based on either estimating error based on dynamic input sampling, or statically performing an error analysis, which have their own limitations, as described earlier.

VI. DISCUSSION

A. Improving Scalability of the Solver

In our experiments, XCVERIFIER was unable to verify any of the exact conditions of the SCAN functional, which has been designed to satisfy all known exact conditions. SCAN is significantly more complex than the other functionals we considered, and also involves the use of transcendental functions such as exp and log. This causes `dReal` to time out even for the relatively simple E_c non-positivity condition, and even when the input domain is reduced $32\times$. It would be interesting to investigate approaches to improve the performance of the solver so that it can tackle the SCAN functional. Apart from its popularity, the SCAN functional will serve as a fascinating use case: there is a progression of DFAs—rSCAN, r++SCAN, r^{2}SCAN, r^{4}SCAN—proposed

in the literature that were designed with different adherence to exact conditions to improve the numerical stability of the original SCAN functional [72]–[74].

B. Expanding to More DFA-Condition Pairs

Our evaluation demonstrated the robustness of the XCENCODER to handle a wide variety of DFAs and exact conditions. The ultimate goal of our research is to be able to analyze all the 500+ functionals in LIBXC for all known DFT exact conditions. Future work will continue to expand our evaluation, and will aim to integrate our verification tool into LIBXC, e.g., as part of the continuous integration (CI) for LIBXC.

C. Numerical Issues With DFAs

Apart from verifying known exact conditions for DFA implementations, it would be interesting to analyze numerical issues of the implementations with the goal of using formal methods to find and fix numerical issues in DFA implementations. This is a challenging problem involving reasoning about floating points and dealing with transcendental functions like $\sin$, $\log$ and $\exp$. The functional forms of DFAs themselves can also be a source of numerical issues. Some DFAs include different functions that apply to different input domains, and must ensure continuity when switching from one domain to another. Additionally, the parametrization of the DFA may cause issues. Even in the simple case of the Local Density Approximation (LDA), the Perdew-Zunger [75] parametrization of the results of Ceperley and Alder [76] includes potentially inaccurate numerical constants that lead to discontinuities of the exchange-correlation energy at a given matching point.

The functional form of a DFA may also make it sensitive to inaccuracies in its input data. While a given implementation of a DFA may yield correct answers for an exactly known (e.g., exponential) density, it may result in large numerical errors if the input density is noisy, or if the density and its gradient are not numerically consistent. This is particularly problematic in regions of low density, e.g., a point far from a molecule placed in vacuum. Such large errors may lead to inaccurate energies or slow convergence in the solution of the Kohn-Sham equations. For example, the sensitivity of the SCAN functional requires the use of extremely fine grids to represent the electron density in order to avoid large numerical errors. This led some authors to modify the SCAN functional to avoid this numerical issue, resulting in a slightly different DFA [73]. In other cases, an analytical reformulation of a DFA is used to avoid numerical issues [77] without modifying it. However, these fixes are ad hoc, and there is no known general recipe for avoiding the numerical issues of a DFA.

VII. CONCLUSION

This paper presented XCVERIFIER, an approach for verifying whether a DFA implementation satisfies the DFT exact conditions. XCVERIFIER automatically encodes the DFA implementation from LIBXC and a given exact condition into a `dReal` formula, symbolically performing any required derivative calculations. It uses the `dReal` solver to verify whether the condition is true or find a violation to it for a given input domain. XCVERIFIER also implements a domain-splitting technique to improve performance, reducing solver timeouts, and isolating the input regions where the condition is satisfied or violated.

We evaluated XCVERIFIER by verifying seven exact conditions (from Pederson and Burke [28]) for five popular DFAs. XCVERIFIER was successfully able to verify or find a counterexample for 13 out of the 29 (valid) DFA-condition pairs, and it was able to partially verify an additional seven pairs. However, it timed out for nine pairs, which included all the conditions for the SCAN functional. We found that the results of the PB approach, which used grid search to check DFT exact conditions, were largely consistent with those of XCVERIFIER. These results demonstrate promise and future challenges of using formal methods for DFT.

REFERENCES

[1] R. G. Parr and W. Yang, "Density-functional theory of the electronic structure of molecules," *Annual review of physical chemistry*, vol. 46, no. 1, pp. 701–728, 1995.

[2] E. Engel, *Density functional theory*. Springer, 2011.

[3] W. Koch and M. C. Holthausen, *A chemist's guide to density functional theory*. John Wiley & Sons, 2015.

[4] D. S. Sholl and J. A. Steckel, *Density functional theory: a practical introduction*. John Wiley & Sons, 2022.

[5] X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté *et al.*, "ABINIT: First-principles approach to material and nanosystem properties," *Computer Physics Communications*, vol. 180, no. 12, pp. 2582–2615, 2009.

[6] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, "cp2k: atomistic simulations of condensed matter systems," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 4, no. 1, pp. 15–25, 2014.

[7] J. Lehtola, M. Hakala, A. Sakko, and K. Hämäläinen, "ERKALE—A flexible program package for X-ray properties of atoms and molecules," *Journal of computational chemistry*, vol. 33, no. 18, pp. 1572–1585, 2012.

[8] J. M. Turney, A. C. Simmonett, R. M. Parrish, E. G. Hohenstein, F. A. Evangelista, J. T. Fermann, B. J. Mintz, L. A. Burns, J. J. Wilke, M. L. Abrams *et al.*, "Psi4: an open-source ab initio electronic structure program," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 2, no. 4, pp. 556–565, 2012.

[9] A. Castro, H. Appel, M. Oliveira, C. A. Rozzi, X. Andrade, F. Lorenzen, M. A. Marques, E. Gross, and A. Rubio, "Octopus: a tool for the application of time-dependent density functional theory," *physica status solidi (b)*, vol. 243, no. 11, pp. 2465–2488, 2006.

[10] F. Gygi, "Architecture of Qbox: A scalable first-principles molecular dynamics code," *IBM J. Res. Dev.*, vol. 52, no. 1-2, pp. 137–144, 2008.

[11] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[12] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas," *Phys. Rev.*, vol. 136, pp. B864–B871, Nov 1964. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.136.B864

[13] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Phys. Rev.*, vol. 140, pp. A1133–A1138, Nov 1965. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.140.A1133

[14] P. Ziesche, S. Kurth, and J. P. Perdew, "Density functionals from LDA to GGA," *Computational Materials Science*, vol. 11, no. 2, pp. 122–127, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025697002061

[15] A. D. Becke, "Perspective: Fifty years of density-functional theory in chemical physics," *The Journal of chemical physics*, vol. 140, no. 18, 2014.

[16] N. Mardirossian and M. Head-Gordon, "Thirty years of density functional theory in computational chemistry: an overview and extensive assessment of 200 density functionals," *Molecular physics*, vol. 115, no. 19, pp. 2315–2372, 2017.

[17] M. Bursch, J.-M. Mewes, A. Hansen, and S. Grimme, "Best-practice DFT protocols for basic molecular computational chemistry," *Angewandte Chemie International Edition*, vol. 61, no. 42, p. e202205735, 2022.

[18] M. A. Marques, M. J. Oliveira, and T. Burnus, "Libxc: A library of exchange and correlation functionals for density functional theory," *Computer physics communications*, vol. 183, no. 10, pp. 2272–2281, 2012.

[19] S. Lehtola, C. Steigemann, M. J. Oliveira, and M. A. Marques, "Recent developments in libxc—A comprehensive library of functionals for density functional theory," *SoftwareX*, vol. 7, pp. 1–5, 2018.

[20] J. P. Perdew and K. Schmidt, "Jacob's ladder of density functional approximations for the exchange-correlation energy," in *AIP Conference Proceedings*, vol. 577, no. 1. American Institute of Physics, 2001, pp. 1–20.

[21] Y. Wang and J. P. Perdew, "Spin scaling of the electron-gas correlation energy in the high-density limit," *Phys. Rev. B*, vol. 43, pp. 8911–8916, Apr 1991. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.43.8911

[22] J. P. Perdew, K. Burke, and M. Ernzerhof, "Generalized gradient approximation made simple," *Physical review letters*, vol. 77, no. 18, p. 3865, 1996.

[23] D. Rappoport, N. R. M. Crawford, F. Furche, and K. Burke, *Approximate Density Functionals: Which Should I Choose?* John Wiley & Sons, Ltd, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470862106.ia615

[24] J. Sun, A. Ruzsinszky, and J. P. Perdew, "Strongly constrained and appropriately normed semilocal density functional," *Phys. Rev. Lett.*, vol. 115, p. 036402, Jul 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.115.036402

[25] C. Lee, W. Yang, and R. G. Parr, "Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density," *Phys. Rev. B*, vol. 37, pp. 785–789, Jan 1988. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.37.785

[26] L. Goerigk, A. Hansen, C. A. Bauer, S. Ehrlich, A. Najibi, and S. Grimme, "A look at the density functional theory zoo with the advanced GMTKN55 database for general main group thermochemistry, kinetics and noncovalent interactions." *Physical chemistry chemical physics : PCCP*, vol. 19 48, pp. 32 184–32 215, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:25268216

[27] R. Peverati and D. G. Truhlar, "Quest for a universal density functional: the accuracy of density functionals across a broad spectrum of databases in chemistry and physics," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 372, no. 2011, p. 20120476, 2014. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2012.0476

[28] R. Pederson and K. Burke, "The difference between molecules and materials: Reassessing the role of exact conditions in density functional theory," *The Journal of Chemical Physics*, vol. 159, no. 21, Dec 2023. [Online]. Available: http://dx.doi.org/10.1063/5.0172058

[29] A. Heck and W. Koepf, *Introduction to MAPLE*. Springer, 1993, vol. 16.

[30] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Automated Deduction–CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings 24*. Springer, 2013, pp. 208–214.

[31] R. Armiento and A. E. Mattsson, "Functional designed to include surface effects in self-consistent density functional theory," *Phys. Rev. B*, vol. 72, p. 085108, Aug 2005. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.72.085108

[32] S. H. Vosko and L. Wilk, "Influence of an improved local-spin-density correlation-energy functional on the cohesive energy of alkali metals," *Phys. Rev. B*, vol. 22, pp. 3812–3815, Oct 1980. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.22.3812

[33] M. Levy and J. P. Perdew, "Hellmann-feynman, virial, and scaling requisites for the exact universal density functionals. shape of the correlation potential and diamagnetic susceptibility for atoms," *Phys. Rev. A*, vol. 32, pp. 2010–2021, Oct 1985. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.32.2010

[34] ——, "Tight bound and convexity constraint on the exchange-correlation-energy functional in the low-density limit, and other formal tests of generalized-gradient approximations," *Phys. Rev. B*, vol. 48, pp. 11 638–11 645, Oct 1993. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.48.11638

[35] E. H. Lieb and S. Oxford, "Improved lower bound on the indirect Coulomb energy," *International Journal of Quantum Chemistry*, vol. 19, no. 3, pp. 427–439, 1981. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qua.560190306

[36] M. Levy and J. P. Perdew, "Hellmann-Feynman, virial, and scaling requisites for the exact universal density functionals. shape of the correlation potential and diamagnetic susceptibility for atoms," *Phys. Rev. A*, vol. 32, pp. 2010–2021, Oct 1985. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.32.2010

[37] D. Frydel, W. M. Terilla, and K. Burke, "Adiabatic connection from accurate wave-function calculations," *The Journal of Chemical Physics*, vol. 112, no. 12, pp. 5292–5297, 03 2000. [Online]. Available: https://doi.org/10.1063/1.481099

[38] S. Crisostomo, R. Pederson, J. Kozlowski, and et al., "Seven useful questions in density functional theory," *Lett Math Phys*, vol. 113, p. 42, 2023.

[39] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "SymPy: symbolic computing in Python," *PeerJ Computer Science*, vol. 3, p. e103, Jan 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[40] A. E. Mattsson, R. Armiento, J. Paier, G. Kresse, J. M. Wills, and T. R. Mattsson, "The AM05 density functional applied to solids," *The Journal of Chemical Physics*, vol. 128, no. 8, p. 084714, 02 2008. [Online]. Available: https://doi.org/10.1063/1.2835596

[41] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[42] S. Lehtola and M. A. Marques, "Many recent density functionals are numerically ill-behaved," *The Journal of Chemical Physics*, vol. 157, no. 17, 2022.

[43] M. Gokhale, G. Gopalakrishnan, J. Mayo, S. Nagarakatte, C. Rubio-González, and S. F. Siegel, "Report of the DOE/NSF workshop on correctness in scientific computing, june 2023, orlando, FL," *CoRR*, vol. abs/2312.15640, 2023.

[44] G. Gopalakrishnan, P. D. Hovland, C. Iancu, S. Krishnamoorthy, I. Laguna, R. A. Lethin, K. Sen, S. F. Siegel, and A. Solar-Lezama, "Report of the HPC correctness summit, jan 25-26, 2017, washington, DC," *CoRR*, vol. abs/1705.07478, 2017.

[45] J. Hückelheim, Z. Luo, S. H. K. Narayanan, S. F. Siegel, and P. D. Hovland, "Verifying properties of differentiable programs," in *SAS*, ser. Lecture Notes in Computer Science, vol. 11002. Springer, 2018, pp. 205–222.

[46] P. Bientinesi, J. A. Gunnels, M. E. Myers, E. S. Quintana-Ortí, and R. A. van de Geijn, "The science of deriving dense linear algebra algorithms," *ACM Trans. Math. Softw.*, vol. 31, no. 1, pp. 1–26, 2005.

[47] T. B. Marcilon and F. H. de Carvalho Junior, "Derivation and verification of parallel components for the needs of an HPC cloud," in *SBMF*, ser. Lecture Notes in Computer Science, vol. 8195. Springer, 2013, pp. 51–66.

[48] A. Bath and D. Kozen, "Equational verification of cache blocking in LU decompo- sition using Kleene algebra with tests," Cornell University, Tech. Rep., 10 2002.

[49] M. Schordan, J. Hückelheim, P. Lin, and H. Menon, "Verifying the floating-point computation equivalence of manually and automatically differentiated code," in *CORRECTNESS@SC*. ACM, 2017, pp. 34–41.

[50] Z. Fu and Z. Su, "Achieving high coverage for floating-point code via unconstrained programming," in *PLDI*. ACM, 2017, pp. 306–319.

[51] H. Guo and C. Rubio-González, "Efficient generation of error-inducing floating-point inputs via symbolic execution," in *ICSE*. ACM, 2020, pp. 1261–1272.

[52] J. Vanover, X. Deng, and C. Rubio-González, "Discovering discrepancies in numerical libraries," in *ISSTA*. ACM, 2020, pp. 488–501.

[53] C. Jeangoudoux, E. Darulova, and C. Q. Lauter, "Interval constraint-based mutation testing of numerical specifications," in *ISSTA*. ACM, 2021, pp. 388–399.

[54] Z. Fu, Z. Bai, and Z. Su, "Automated backward error analysis for numerical code," in *OOPSLA*. ACM, 2015, pp. 639–654.

[55] A. Solovyev, M. S. Baranowski, I. Briggs, C. Jacobsen, Z. Rakamaric, and G. Gopalakrishnan, "Rigorous estimation of floating-point round-off errors with symbolic taylor expansions," *ACM Trans. Program. Lang. Syst.*, vol. 41, no. 1, pp. 2:1–2:39, 2019.

[56] A. Solovyev, C. Jacobsen, Z. Rakamaric, and G. Gopalakrishnan, "Rigorous estimation of floating-point round-off errors with symbolic taylor expansions," in *FM*, ser. Lecture Notes in Computer Science, vol. 9109. Springer, 2015, pp. 532–550.

[57] A. Das, I. Briggs, G. Gopalakrishnan, S. Krishnamoorthy, and P. Panchekha, "Scalable yet rigorous floating-point error analysis," in *SC*. IEEE/ACM, 2020, p. 51.

[58] D. Zou, M. Zeng, Y. Xiong, Z. Fu, L. Zhang, and Z. Su, "Detecting floating-point errors via atomic conditions," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, pp. 60:1–60:27, 2020.

[59] A. Das, T. Tirpankar, G. Gopalakrishnan, and S. Krishnamoorthy, "Robustness analysis of loop-free floating-point programs via symbolic automatic differentiation," in *CLUSTER*. IEEE, 2021, pp. 481–491.

[60] G. Singh, B. Kundu, H. Menon, A. Penev, D. J. Lange, and V. Vassilev, "Fast and automatic floating point error analysis with CHEF-FP," in *IPDPS*. IEEE, 2023, pp. 1018–1028.

[61] R. Abbasi and E. Darulova, "Modular optimization-based roundoff error analysis of floating-point programs," in *SAS*, ser. Lecture Notes in Computer Science, vol. 14284. Springer, 2023, pp. 41–64.

[62] D. Lohar, M. Prokop, and E. Darulova, "Sound probabilistic numerical error analysis," in *IFM*, ser. Lecture Notes in Computer Science, vol. 11918. Springer, 2019, pp. 322–340.

[63] E. Darulova, A. Izycheva, F. Nasir, F. Ritter, H. Becker, and R. Bastian, "Daisy - framework for analysis and optimization of numerical programs (tool paper)," in *TACAS (1)*, ser. Lecture Notes in Computer Science, vol. 10805. Springer, 2018, pp. 270–287.

[64] A. Izycheva and E. Darulova, "On sound relative error bounds for floating-point arithmetic," in *FMCAD*. IEEE, 2017, pp. 15–22.

[65] N. Damouche and M. Martel, "Salsa: An automatic tool to improve the numerical accuracy of programs," in *AFM@NFM*, ser. Kalpa Publications in Computing, vol. 5. EasyChair, 2017, pp. 63–76.

[66] W. Chiang, G. Gopalakrishnan, Z. Rakamaric, and A. Solovyev, "Efficient search for inputs causing high floating-point errors," in *PPoPP*. ACM, 2014, pp. 43–52.

[67] D. Zou, R. Wang, Y. Xiong, L. Zhang, Z. Su, and H. Mei, "A genetic algorithm for detecting significant floating-point inaccuracies," in *ICSE (1)*. IEEE Computer Society, 2015, pp. 529–539.

[68] E. T. Barr, T. Vo, V. Le, and Z. Su, "Automatic detection of floating-point exceptions," in *POPL*. ACM, 2013, pp. 549–560.

[69] I. Laguna and G. Gopalakrishnan, "Finding inputs that trigger floating-point exceptions in gpus via Bayesian optimization," in *SC*. IEEE, 2022, pp. 33:1–33:14.

[70] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock, "Automatically improving accuracy for floating point expressions," in *PLDI*. ACM, 2015, pp. 1–11.

[71] R. Rabe, A. Izycheva, and E. Darulova, "Regime inference for sound floating-point optimizations," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, pp. 81:1–81:23, 2021.

[72] A. P. Bartók and J. R. Yates, "Regularized SCAN functional," *The Journal of chemical physics*, vol. 150, no. 16, 2019.

[73] J. W. Furness, A. D. Kaplan, J. Ning, J. P. Perdew, and J. Sun, "Accurate and numerically efficient r2SCAN meta-generalized gradient approximation," *The journal of physical chemistry letters*, vol. 11, no. 19, pp. 8208–8215, 2020.

[74] ——, "Construction of meta-GGA functionals through restoration of exact constraint adherence to regularized SCAN functionals," *The Journal of Chemical Physics*, vol. 156, no. 3, 2022.

[75] J. P. Perdew and A. Zunger, "Self-interaction correction to density-functional approximations for many-electron systems," *Physical Review B*, vol. 23, no. 10, p. 5048, 1981.

[76] D. M. Ceperley and B. Alder, "Ground state of the electron gas by a stochastic method," *Physical Review Letters*, vol. 45, no. 7, p. 566, 1980.

[77] J. Wu and F. Gygi, "A simplified implementation of van der Waals density functionals for first-principles molecular dynamics applications," *The Journal of Chemical Physics*, vol. 136, no. 22, 2012.