energies

Article

Formal Asymptotic Analysis of Online Scheduling Algorithms for Plug-In Electric Vehicles' Charging

Asad Ahmed [1,*,†], Osman Hasan [1,†], Falah Awwad [2], Nabil Bastaki [2] and Syed Rafay Hasan [3]

1 School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), H-12 Islamabad, Pakistan; osman.hasan@seecs.nust.edu.pk
2 College of Engineering, United Arab Emirates University, Al-Ain 15551, UAE; f_awwad@uaeu.ac.ae (F.A.); nabil@uaeu.ac.ae (N.B.)
3 Department of Electrical and Computer Engineering, Tennessee Technological University, Cookeville, TN 38505, USA; shasan@tntech.edu
* Correspondence: asad.ahmed@seecs.nust.edu.pk; Tel.: +92-51-9085-2086
† These authors contributed equally to this work.

Received: 27 September 2018; Accepted: 7 November 2018; Published: 21 December 2018

Abstract: A large-scale integration of plug-in electric vehicles (PEVs) into the power grid system has necessitated the design of online scheduling algorithms to accommodate the after-effects of this new type of load, i.e., PEVs, on the overall efficiency of the power system. In online settings, the low computational complexity of the corresponding scheduling algorithms is of paramount importance for the reliable, secure, and efficient operation of the grid system. Generally, the computational complexity of an algorithm is computed using asymptotic analysis. Traditionally, the analysis is performed using the paper-pencil proof method, which is error-prone and thus not suitable for analyzing the mission-critical online scheduling algorithms for PEV charging. To overcome these issues, this paper presents a formal asymptotic analysis approach for online scheduling algorithms for PEV charging using higher-order-logic theorem proving, which is a sound computer-based verification approach. For illustration purposes, we present the complexity analysis of two state-of-the-art online algorithms: the Online cooRdinated CHARging Decision (ORCHARD) algorithm and online Expected Load Flattening (ELF) algorithm.

Keywords: asymptotic analysis; computational complexity; HOL-Light

1. Introduction

Electric road technologies are envisaged as alternatives to traditional fossil-fuel-based transportation due to their low greenhouse gas emissions, energy security, and noise mitigation. These advantages have led to a rapid increase in their volume, e.g., more than 1 million electric vehicles were sold worldwide in the year 2017 alone [1]. This enormous increase has also resulted in a new billion-dollar industry for the development and deployment of electric vehicle (EV) units and their operations [2].

Plug-in electric vehicles (PEVs) are a type of electric vehicle and are characterized by a comparatively larger battery, a charging plug, and an internal combustion engine for powering the vehicle and battery [3]. This technology, in conjunction with the smart grid concept, enables the bidirectional flow of power, i.e., from the grid to the vehicle and from the vehicle to the grid [4]. The use of electric vehicles as a load that can be scheduled in the electricity system, offering storage capability and bidirectional power flow, is anticipated to be effectively used as part of the integration of renewable energy sources, load flattening, peak shaving, and frequency fluctuation mitigation [5]. Therefore, the optimal scheduling of PEV charging/discharging results in an optimized bidirectional flow of power between PEVs and power grid [6], and thus, a reduced capital cost of electricity generation and minimization of the operational cost of the grid.

Mathematically, the PEV scheduling problem for charging/discharging is posed as an optimization problem subject to the constraints imposed by the infrastructure and physical properties of the grid and EVs. The optimization problem is then solved using linear programming [7], nonlinear programming [8], model predictive control (MPC) [9], and queuing theory [10], among many other methods. These scheduling algorithms are further classified as online and offline algorithms, which provide scheduling based on causal and non-causal information, respectively.

In an offline setting, a PEV's charging profile is assumed to be known to the charging station for scheduling purposes, prior to the arrival of the PEV [11,12]. However, in real-world settings, the assumption of prior knowledge is not realistic. Moreover, the PEV charging problem is prone to the uncertainties introduced by the random arrival and departure of PEVs, intermittent power generation from renewable energy sources, and fluctuation in demand and prices of the electricity in the system [5]. In this context, online algorithms cater to the uncertainties associated with the PEV charging problem. In an online setting, the arrival and departure times and charging demand of a PEV are only available to the controller for purposes of scheduling when a PEV is plugged into the charging facility. The scheduler then schedules all the plugged-in PEVs and also allocates the charging rate to each PEV so that the objectives, such as power consumption, are minimized, as shown in Figure 1.

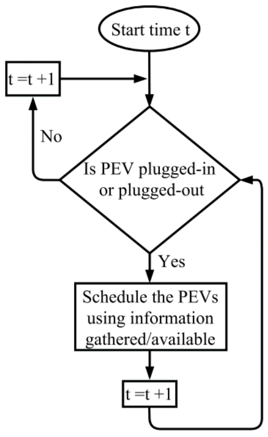

Start time t
t =t +1
No
Is PEV plugged-in or plugged-out
Yes
Schedule the PEVs using information gathered/available
t =t +1

Figure 1. Online plugged-in electric vehicle (PEV) charging scheduling [13].

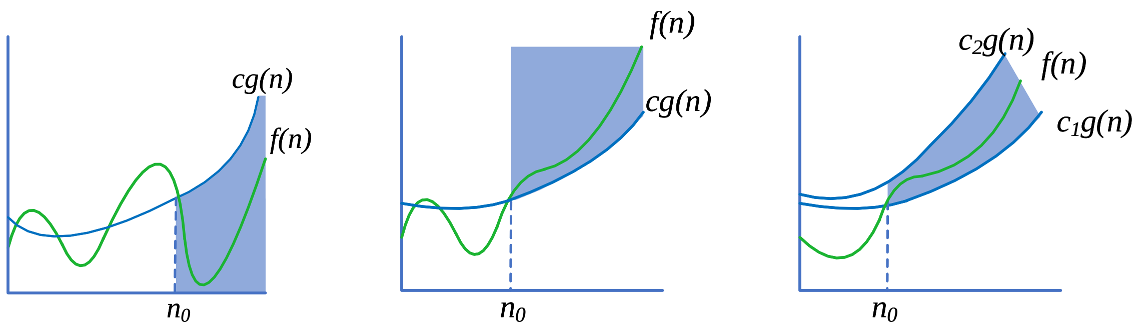
Generally, an online algorithm incorporates information based on the history, knowledge, and statistics available for the future data, in addition to the currently available data, to improve the overall efficiency of the algorithm [13]. Although the use of an online scheduling algorithm for PEV charging facilitates the account of uncertainties, the efficiency of these algorithms is highly sensitive to the size of the PEV population. A large population of PEVs results in exponential growth of the computational complexity of these algorithms which, in turn, can jeopardize the reliability, efficiency, and security of the grid operation. To address this issue, recent online scheduling algorithms have alleviated the unbearable computational cost of the online algorithms by devising various low-complexity routines for the optimal scheduling of PEVs [9,14,15].

The computational complexity of an algorithm refers to the number of primitive operations or steps required to solve a given task using some method or an algorithm [16]. These primitive operations include basic arithmetic operations, such as addition, subtraction, multiplication, and division. In the study of algorithm analysis, a pseudocode is usually used to describe the working of an algorithm using control structures adopted from different conventional programming languages and mathematical notations. The computational complexity for a given algorithm can be judged based on the number of steps from the pseudocode of the given algorithm and also the cost associated with each step. Mathematically, the computational complexity utilizes asymptotic theory, which, by definition, characterizes the limiting behavior of a function [17], with respect to the input size, to describe the notions of the best, worst, and average case computational complexity of an algorithm [16]. Asymptotic notations, such as Big-Oh (Big-O), Big-Omega (Big-Ω), Big-Theta (Big-Θ), little-oh (little-o), and little-omega (little-ω), are used to express the asymptotic bounds for the complexity function of a given algorithm, and thus allows us to characterize the efficiency of an algorithm and also compare relative performance of alternative algorithms [16]. Table 1 presents the asymptotic notions in terms of limits and set theory.

Table 1. Asymptotic Notations.

Notation	Limit Definition	Set Definition
$f = O(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$	$\{f(n) \mid (\exists\, c\, n_o.\, (\forall n_o < n \land 0 < c \Rightarrow \lvert f(n)\rvert \leq cg(n)))\}$
$f = \Omega(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$	$\{f(n) \mid (\exists\, c\, n_o.\, (\forall n_o < n \land 0 < c \Rightarrow cg(n) \leq \lvert f(n)\rvert))\}$
$f = \Theta(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \infty$	$\{f(n) \mid \exists c_1 c_2 n_o.(\forall n_o < n \land 0 < c_1 \land 0 < c_2 \Rightarrow c_1 g(n) \leq \lvert f(n)\rvert \leq c_2 g(n)))\}$
$f = o(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$\{f(n) \mid (\exists\, c\, n_o.\, (\forall n_o < n \land 0 < c \Rightarrow \lvert f(n)\rvert < cg(n)))\}$
$f = \omega(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$\{f(n) \mid (\exists\, c\, n_o.(\forall n_o < n \land 0 < c \Rightarrow cg(n) < \lvert f(n)\rvert))\}$

Big-O is a set that contains all functions, $f(n)$, for which there always exist constants n_o and c, such that the magnitude of $f(n)$ is less than and equal to the constant multiplier of function $g(n)$, i.e., $f(n) \leq cg(n)$, for some $n_o < n$, as shown in Figure 2a. Big-O allows for describing the upper bound of a function and, therefore, it is used to abstract the worst-case running time or space complexity of an algorithm.



(**a**) Big-O Notation (**b**) Big-Omega Notation (**c**) Big-Theta Notation

Figure 2. Asymptotic upper, lower and tight bounds of a function (from (**a**–**c**)).

Big-Ω is a set that contains all functions $f(n)$, for which there always exist constants n_o and c, such that the magnitude of $f(n)$ is greater than and equal to the constant multiplier of the function $g(n)$, i.e., $cg(n) \leq f(n)$, for some $n_o < n$, as shown in Figure 2b. Big-Θ is a set that contains all functions $f(n)$, for which there always exist constants n_o, c_1, and c_2, such that the magnitude of $f(n)$ is bounded by the constant multiplier of function $g(n)$, i.e., $c_1 g(n) \leq f(n) \leq c_2 g(n)$, for some $n_o < n$, as shown in Figure 2c. This notation is used for describing the situation when an algorithm has the same best- or worst-case computational or space complexity. To the contrary, little-o and little-ω are used to describe weak asymptotic lower and upper bounds by not incorporating the equality condition used in the definitions of the Big-O and Big-Ω, respectively.

Traditional analysis techniques, such as simulations and computer algebra systems, cannot be used to conduct a complete asymptotic analysis of algorithms due to their inability to deal with the limiting behavior, as shown in Table 1, in a true manner. Therefore, in practice, the asymptotic analysis is carried out using the paper-pencil method. However, it is also error-prone due to the involvement of humans in the analysis. Moreover, this analysis method is not scalable, as the large-sized models cannot be handled very efficiently on paper. To address these issues, we propose to use a formal asymptotic analysis technique for the online scheduling algorithms for PEV charging schedules.

Formal methods [18] are computer-based mathematical techniques, which are being widely adopted for the specification, analysis, and verification of hardware and software systems. These techniques use logic to formally express and reason with regard to the mathematical models of systems and, therefore, they provide a mechanized platform which is characterized as highly expressive and sound [19]. A suitable logic, such as temporal logic, first-order, predicate, and second- or higher-order logic, with the help of its well-defined syntax, semantics, and proof theory, allows for expressing a discrete or continuous model of the given system and formally reasoning and verifying the correctness of the intended behavior of the system based upon a few axioms of the corresponding logic. A combination of logic and modeling technique, such as finite state machine (FSM) and automaton, leads to a variety of formal method-based solutions [20] for the formal verification of systems, such as sequential or concurrent systems.

Formal methods can be mainly categorized into two mainstream techniques, i.e., *model checking* and *theorem proving* techniques. Model checking [21] employs the finite state machine notion to model the system for the purpose of exhaustive state-space verification of systems in a model checker—a piece of software. Model checking allows automatic verification and thus is easy to use, but it can be subject to state-space explosion for the systems—a situation when the state-space of the corresponding model grows extremely large. Moreover, model checking cannot be used to verify generic mathematical expressions and, therefore, it is not suitable to formally model and verify properties regarding asymptotic notations, which are based upon the limits concept. On the other hand, theorem proving is a formal methods technique that allows for describing any mathematical model by choosing an appropriate logic and then using mathematical reasoning to verify its corresponding properties in a theorem prover—a piece of software. Higher-order logic is quite expressive and allows reasoning about continuous aspects as well. Therefore, we employed theorem proving for the formal asymptotic analysis of online scheduling algorithms for PEV charging.

The primary objective of this paper is to develop a framework for the formal asymptotic analysis using theorem proving for online charging scheduling algorithms for plug-in electric vehicles. A rigorous formal verification of the computational complexity of algorithms results in the explicit specification of assumptions under which results will be valid. These assumptions are about the system parameters, such as number of vehicles, etc., and, therefore, are helpful in providing valuable insights before the implementation phase. Moreover, due to the generic formalization of asymptotic notations, the formalization can be utilized readily for the applications of asymptotic analysis in many other fields, such as applied mathematics and probability theory. The O-notation has been formalized in the Isabelle/HOL (higher-order-logic) theorem prover [22] using the ring theory. However, to the best of our knowledge, this formalization of asymptotic notations has not been used to analyze any practical problem. Moreover, to the best of our knowledge, the other asymptotic notations, i.e., Big-Ω, Big-Θ, little-o, and little-ω, have not been formalized in higher-order logic. In this paper, we propose a real theory-based formalization of all asymptotic notations using the HOL-Light theorem prover and utilize them to analyze the computational complexity of online scheduling algorithms for PEV charging.

Keeping in view the uncertainties coupled to the electric vehicle charging scheduling problem, various online algorithms are available in the literature to alleviate negative effects and harvest potential benefits from the integration of this new type of load into the grid system. For example, an online auction protocol to increase the allocation efficiency of a charging facility in comparison to a fixed price strategy, given the requirements of electric vehicle owners, is described in [15]. Similarly, a decentralized [6,11] approach is used for optimally allocating a charging or discharging schedule to plugged-in electric vehicles for the purpose of using the electricity stored in the mounted batteries with a low computational burden on the central utility. However, the aforementioned approaches require beforehand knowledge of electric vehicle charging demands, which may not be always available.

The nonlinear online maximum sensitivity selection-based charging algorithm (NOL-MSSCA) [23] is used to cater to the current harmonic effects caused by the injection of variable speed drives (VSDs), variable frequency drives (VFDs), energy-efficient lights, switching converters, smart appliances, and plug-in electric vehicles (PEVs) in the grid system. NOL-MSSCA considers the random arrival of PEVs into the system over the span of 24 h and uses a priority-based criterion for allocating resources for charging electric vehicles to minimize the effects of current harmonics caused by nonlinear loads. However, this work does not address the computational complexity issue explicitly. The Online cooRdinated CHARging Decision (ORCHARD) algorithm [9] models an online scenario by considering random arrivals or departures of electric vehicles and their charging demands. Moreover, a low computational complexity routine is devised to avoid computational burden for solving the scheduling problem of PEVs for every arrival or departure. There is another class of load scheduling algorithms which rely on the model predictive control (MPC) approach to utilize future information of the load to enhance the grid operation and achieve various desirable objectives, like energy cost and uncertainty regulation [14,24–26]. In particular, Expected Load Flattening (ELF) [14] is based on the assumption of random arrivals or departures of PEVs along with additional future information, incorporated using a model predictive approach, for an optimal online scheduling of PEV charging.

The algorithms, i.e., ORCHARD and ELF, besides being state-of-the-art, claim to reduce the worst-case computational complexity of the scheduling algorithms for PEV charging. However, the existing computational complexity analysis of these algorithms is based on the traditional paper-and-pencil proof method and is quite informal in its presentation, as well. On the other hand, the computational complexity of online algorithms is usually a foremost consideration for the appropriate and efficient functioning of the real-time systems [27], as a high computational complexity can lead to an unbearable monetary loss in a PEV integrated grid system. Therefore, in this paper, we present the formal verification of the computational complexity of the aforementioned algorithms using the formalization of asymptotic notations in HOL-Light. Besides the formal verification of asymptotic properties in the sound core of HOL-Light, the main challenge involved in the proposed work is the formulation of the mathematical problem—related to the complexity of the given algorithms—for conducting their formal analysis in the HOL-Light theorem prover.

1.1. *Contribution of the Paper*

The main contributions of this paper are:

- A higher-order-logic framework is proposed to conduct formal asymptotic analysis using the HOL-Light theorem prover.
- Formal verification of low computational complexity results are reported for state-of-the-art online scheduling algorithms for PEV charging, i.e., ORCHARD and ELF.
- The formally verified low computational complexity results are then used to compare the effect of the cost of operation of a scheduler and PEV load for ORCHARD and ELF algorithms in MATLAB R2016a [28].

1.2. *Organization of the Paper*

The rest of the paper is organized as follows: Section 2 describes some preliminaries essential for the understanding of the rest of the paper. Section 3 provides the methodology adopted to formally analyze the online scheduling algorithms for PEV charging. Section 4 describes the formalization of asymptotic notations in HOL-Light. Section 5 describes the formal analysis of the chosen algorithms in HOL-Light. Finally, Section 6 concludes the paper.

2. Preliminaries

This section provides a brief introduction to theorem proving and HOL-Light theorem prover to facilitate the understanding of the rest of the paper.

2.1. Theorem Proving

Theorem proving is a widely used formal method [18] for describing and verifying the mathematical model of a system using formal logic, such as propositional, first-order, and second- or higher-order logic. A well-defined proof theory, semantics, and syntax of the formal logic allows the rigorous reasoning of the system's design, properties, and correctness [29]. Various theorem provers, e.g., HOL-Light [30], HOL [31], Coq [32], and PVS [33], have been used for the mechanization of the formal proofs.

Theorem proving is mainly categorized as automated and interactive theorem proving. In automated theorem proving, a decidable logic is employed to model the system, and, therefore, it is possible to develop automatic procedures for the verification of the model. However, the limited expressiveness of the decidable logic does not allow for modeling complex mathematical notions, such as limits, which are commonly employed to analyze the system's behavior. Therefore, a more expressive formal logic, such as higher-order logic, is used to model such behaviors. These logical frameworks, however, require interaction between humans and machines to accomplish the verification task of the system. Generally, this type of theorem proving is referred to as interactive theorem proving. The ability to use a wide range of logic in the verification process makes theorem proving a highly expressive and flexible technique for formal verification.

Theorem proving provides a very flexible platform for the abstraction, formal verification, and specification of mathematical concepts, such as the limiting behavior of functions, and, therefore, we chose this technique for the formal asymptotic analysis of the online scheduling algorithms for PEV charging.

2.2. HOL-Light Theorem Prover

HOL-Light [30] is an interactive higher-order-logic theorem prover that belongs to the HOL family of theorem provers. The higher-order logic is implemented using Objective CAML (OCaml) language [34]—which is a variant of the strongly typed functional programming language—in HOL-Light to express and apply proof strategies and develop logical theories. HOL-Light provides formal reasoning support for many mathematical theories, including sets, natural numbers, real analysis, complex analysis, and vector calculus, and has been particularly successful in verifying many software and hardware systems [18,30,35].

HOL-Light provides a platform for the secure and sound formal verification and specification of systems under verification. Secure theorem proving is ensured because of its small core, which consists of 1500 lines, including a core set of 10 primitive inference rules, such as modus ponens and reflexivity. These primitive inferences are realized as OCaml language, and every new theorem is proved using these rules. These verified theorems can be stored for future use as an HOL-Light theory, which is a computer file. These theories can be loaded by the HOL-Light users in their running sessions, and, therefore, they can be utilized for the verification of new theorems. HOL-Light supports both backward and forward proof methods. The former consists of using inference rules to deduce the proof of the desired theorem, whereas the latter consists of breaking the main goal into subgoals using HOL-Light tactics, which are special ML functions for generating subgoals from a main goal.

We selected the HOL-Light theorem prover for the proposed work as it is equipped with a rich library of mathematical theories of set, natural numbers, and real numbers. Table 2 presents some frequently used HOL-Light functions and symbols to facilitate the understanding of the formalization and verification in this paper.

Table 2. Higher-order-logic (HOL) symbols and functions.

HOL Symbol	Standard Symbol	Meaning
$\wedge$	*and*	Logical *and*
$\vee$	*or*	Logical *or*
$\neg$	*not*	Logical *negation*
λ `x.t`	$\lambda\, x.t$	Function that maps x to $t(x)$
`num`	$\{0, 1, 2, \ldots\}$	Positive integers data type
`real`	All real numbers	Real data type
`SUC n`	$n + 1$	Successor of a *num*
`rpow x y`	x^y	Power with real exponent
`max (f(x), g(x))`	$max(g(x), f(x))$	Maximum among $f(x)$ and $g(x)$

3. Proposed Methodology

In this section, we present the main steps of the proposed methodology, depicted in Figure 3, for conducting the formal asymptotic analysis of online scheduling algorithms for PEV charging.

1. As a first step, we formalize asymptotic notations in higher-order logic to be able to formally verify the properties of asymptotic notations in the sound core of the HOL-Light theorem prover. The formalization is based on the formalized mathematical theories of set, real, and natural number theories available in the library of HOL-Light. This formalization allows the formal verification of the properties of asymptotic notations, which play a pivotal role in the formal verification of the results related to the online scheduling algorithms for PEV charging.
2. The second step in the formal asymptotic analysis of an algorithm for PEVs is to represent its behavior while incorporating its implicit concepts as a pseudocode. As described earlier, this pseudocode then forms the basis on which we can describe the complexity function of an algorithm afterward. Due to the very generic and flexible presentations of algorithms, availability in the PEV literature, and varying audiences, it is not a very easy task to obtain the pseudocodes of the available algorithms. For example, the pseudocode for the Online cooRdinated CHARging Decision (ORCHARD) algorithm, considered in this paper for the formal asymptotic analysis, is not available in the original paper [9]. Moreover, this step also allows the explicit investigation and specification of the pseudocodes of ancillary algorithms (such as insertion sort algorithm) which are required to accomplish the formal analysis of online scheduling algorithms for PEV charging.
3. Finally, the computational complexity functions obtained from Step 2 are formally specified as higher-order-logic functions, utilizing the formal modeling of asymptotic notations from Step 1. Then, the properties of interest for an algorithm are formally verified as higher-order-logic theorems using the sound core of the HOL-Light theorem prover based on the formalized notions and properties of asymptotic notations from Step 1, as shown in Figure 3.

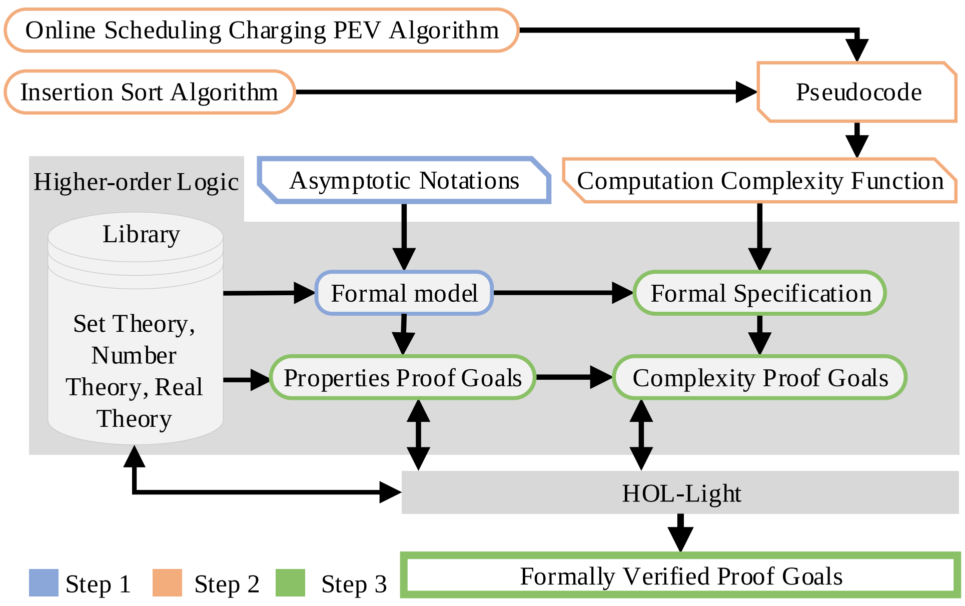


Figure 3. Proposed methodology for PEV charging algorithms in HOL-Light.

4. Formalization of Asymptotic Notations in HOL-Light

In this section, we present a higher-order-logic formalization of asymptotic notations, described in Table 1, in the HOL-Light theorem prover.

Definition 1. $\vdash \forall$ g. $\texttt{BigO}\ (g : \texttt{num} \to \texttt{real}) =$
$\{(f : \texttt{num} \to \texttt{real}) \mid (\exists\ c\ n\_0.\ (\forall\ n.\ n\_0 \leq n$
$\Rightarrow 0 < c\ \wedge\ 0 \leq f(n)\ \wedge\ f(n) \leq c * g(n)))\}$

where $g : \texttt{num} \to \texttt{real}$ and $f : \texttt{num} \to \texttt{real}$ are functions that accept a natural number (num) as an argument and return a real number (real). The constants c and n_0 are of type real and num, respectively, and n is a variable of type (num). BigO is a higher-order-logic function that accepts a function $g : \texttt{num} \to \texttt{real}$ as an argument and returns a set of functions such that every f has a growth rate proportional to the product of function g and a constant multiplier c, i.e., $0 \leq f(n) \leq c * g(n)$. On the other hand, $0 \leq f(n)$ ensures that the member function is a positive function. In algorithm analysis, Big-O notation is used to specify the asymptotic upper bound for the complexity of an algorithm.

Definition 2. $\vdash \forall$ g. $\texttt{BigOmega}\ (g : \texttt{num} \to \texttt{real}) =$
$\{(f : \texttt{num} \to \texttt{real}) \mid (\exists\ c\ n\_0.\ (\forall\ n.\ n\_0 \leq n \Rightarrow 0 < c\ \wedge 0\ \leq c * g(n)\ \leq f(n)))\}$

In the above definition, BigOmega is a higher-order-logic function that accepts a function $g : \texttt{num} \to \texttt{real}$ as an argument and returns a set of functions such that every f has a growth rate proportional to the product of function g and a constant multiplier c, i.e., $0 \leq c * g(n) \leq f(n)$. On the other hand, $0 \leq f(n)$ ensures that the member function is a positive function. In algorithm analysis, Big-Ω notation is used to specify the asymptotic lower bound for the complexity of an algorithm.

Definition 3. $\vdash \forall$ g. $\texttt{BigTheta}\ (g : \texttt{num} \to \texttt{real}) =$
$\{(f : \texttt{num} \to \texttt{real}) \mid (\exists\ c1\ c2\ n\_0.\ (\forall\ n.\ n\_0 \leq n$
$\Rightarrow 0 < c1\ \wedge\ 0 < c2\ \wedge\ 0 \leq c1 * g(n)\ \leq f(n) \leq\ c2 * g(n)))\}$

In the above definition, BigTheta is a higher-order-logic function that accepts a function $g : \texttt{num} \to \texttt{real}$ as an argument and returns a set of functions such that every f growth rate satisfies an inequality, i.e., $\texttt{c1} * \texttt{g(n)} \leq \texttt{f(n)} \leq \texttt{c2} * \texttt{g(n)}$, where c1 and c2 are real constants. In algorithm analysis, Big-Θ notation is used to specify the asymptotic tight bounds for the complexity of an algorithm.

Definition 4. $\vdash \forall$ g. LittleO (g : num $\rightarrow$ real) =
{(f : num $\rightarrow$ real) | ($\exists$ c n_0. ($\forall$ n. n_0 $\leq$ n
$\Rightarrow$ 0 $\leq$ c $\wedge$ 0 < f(n) $\wedge$ f(n) < c $*$ g(n)))}

In the above definition, LittleO is a higher-order-logic function that accepts a function g : num $\rightarrow$ real as an argument and returns a set of functions such that every f growth rate is less than the multiple of a constant c and function g, i.e., f(n) < c $*$ g(n). In algorithm analysis, little-*o* notation is used to specify the strict asymptotic upper bound for the complexity of an algorithm.

Definition 5. $\vdash \forall$ g. LittleOmega (g : num $\rightarrow$ real) =
{(f : num $\rightarrow$ real) | ($\exists$ c n_0. ($\forall$ n. n_0 $\leq$ n $\Rightarrow$ 0 < c $\wedge$ 0 $\leq$ c $*$ g(n) < f(n)))}

In the above definition, LittleOmega notation is a higher-order-logic function that accepts a function g : num $\rightarrow$ real as an argument and returns a set of functions such that the growth rate of every function is greater than the multiple of a constant c and function g, i.e., c $*$ g(n) < f(n).

In algorithm analysis, little-ω notation is used to specify the strict asymptotic lower bound for the complexity of an algorithm.

The above definitions in higher-order logic enable the reasoning of the correctness of their properties in HOL-Light.

Formal Verification of Asymptotic Notations' Properties

In this subsection, we present the formally verified properties of the asymptotic notations in HOL-Light, based on the formal definitions given in the previous section.

We first present the formally verified theorems corresponding to the *transitivity property* of the asymptotic notations:

Theorem 1. $\vdash \forall$ f g h., f $\in$ BigO g $\wedge$ g $\in$ BigO h $\Rightarrow$ f $\in$ BigO h

Theorem 2. $\vdash \forall$ f g h.
f $\in$ BigOmega g $\wedge$ g $\in$ BigOmega h $\Rightarrow$ f $\in$ BigOmega h

Theorem 3. $\vdash \forall$ f g h.
f $\in$ BigTheta g $\wedge$ g $\in$ BigTheta h $\Rightarrow$ f $\in$ BigTheta h

Theorem 4. $\vdash \forall$ f g h.
f $\in$ Littelo g $\wedge$ g $\in$ Littelo h $\Rightarrow$ f $\in$ Littelo h

Theorem 5. $\vdash \forall$ f g h.
f $\in$ Littelomega g $\wedge$ g $\in$ Littelomega h $\Rightarrow$ f $\in$ Littelomega h

In Theorems 1–5, f, g, and h are functions that accept an argument of type num and return a real number. The transitivity property formally verifies the asymptotic relationship of two functions f and h, given that their asymptotic relationship with function g is valid.

Next, we present the formally verified results for the *reflexivity property* of the Big-*O*, Big-Ω, and Big-Θ:

Theorem 6. $\vdash \forall$ f. ($\exists$ n_0. ($\forall$ m. n_0 $\leq$ m $\Rightarrow$ 0 $\leq$ f m) $\Rightarrow$ (f $\in$ BigO f))

Theorem 7. $\vdash \forall \mathtt{f}. (\exists \mathtt{n_0}.\ (\forall\ \mathtt{m}.\ \mathtt{n_0} \leq \mathtt{m} \Rightarrow 0 \leq \mathtt{f\ m}) \Rightarrow (\mathtt{f} \in \mathtt{BigOmega\ f}))$

Theorem 8. $\vdash \forall \mathtt{f}. (\exists \mathtt{n_0}.\ (\forall\ \mathtt{m}.\ \mathtt{n_0} \leq \mathtt{m} \Rightarrow 0 \leq \mathtt{f\ m}) \Rightarrow (\mathtt{f} \in \mathtt{BigTheta\ f}))$

In the above theorems, `f` and `g` are functions that accept a number `num` as an argument and return a real number. Theorems 6–8 are formally verified results which depict that any function `f` is asymptotically related to itself for Big-O, Big-Ω, and Big-Θ notations.

The next property is related to the *summation* of the Big-O and Big-Ω notations and is formally verified as:

Theorem 9. $\vdash \forall\ \mathtt{t1\ t2\ g1\ g2}.$
$(\mathtt{t1} \in \mathtt{BigO\ g1}) \wedge (\mathtt{t2} \in \mathtt{BigO\ g2})$
$\Rightarrow (\lambda \mathtt{n}.\ \mathtt{t1\ n} + \mathtt{t2\ n}) \in (\mathtt{BigO\ (max(g1\ ,\ g2))}$

Theorem 10. $\vdash \forall\ \mathtt{t1\ t2\ g1\ g2}.$
$(\mathtt{t1} \in \mathtt{BigOmega\ g1}) \wedge (\mathtt{t2} \in \mathtt{BigOmega\ g2})$
$\Rightarrow (\lambda \mathtt{n}.\ \mathtt{t1\ n} + \mathtt{t2\ n}) \in (\mathtt{BigOmega\ (min(g1\ ,\ g2))}$

In the above theorems, `t1`, `t2`, `g1`, and `g2` are functions that accept an argument of type `num` and return a real value. Theorem 9 formally verifies the fact that if two functions `t1` and `t2` are in the Big-O of `g1` and `g2`, respectively, then the sum of these two functions will also be in the Big-O of the maximum of the `g1` and `g2` functions. It is due to the fact that the Big-O notation provides an upper asymptotic bound for a function. Similarly, the sum of two functions `t1` and `t2`, which are in the Big-Ω of `g1` and `g2`, will be in the Big-Ω of the minimum of the `g1` and `g2` functions. This is also due to the fact that Big-Ω provides a lower asymptotic bound for a function.

Next, we present the formally verified *symmetry property* of the Big-Ω and Big-Θ notations:

Theorem 11. $\vdash \forall\ \mathtt{f\ g}.\quad \mathtt{f} \in \mathtt{BigOmega\ \ g} \Rightarrow \mathtt{g} \in \mathtt{BigOmega\ f}$

Theorem 12. $\vdash \forall\ \mathtt{f\ g}.\quad \mathtt{f} \in \mathtt{BigTheta\ \ g} \Rightarrow \mathtt{g} \in \mathtt{BigTheta\ f}$

In the above theorems, `f` and `g` are functions that accept an argument of type `num` and return a real number. Theorems 11 and 12 formally verify that if the growth rate of a function `f` is related to a function `g` through the Big-ω or Big-Θ notations, then the growth rate of `g` will also be related to `f` through Big-ω or Big-Θ.

Next, we present the formally verified *transpose symmetry property*:

Theorem 13. $\vdash \forall\ \mathtt{f\ g}.\quad \mathtt{f} \in \mathtt{BigO\ \ g} \Rightarrow \mathtt{g} \in \mathtt{BigOmega\ f}$

Theorem 14. $\vdash \forall\ \mathtt{f\ g}.\quad \mathtt{f} \in \mathtt{Littelo\ \ g} \Rightarrow \mathtt{g} \in \mathtt{LittleOmega\ f}$

In the above theorems, `f` and `g` are functions that accept an argument of type `num` and return a real number. Theorem 13 formally verifies that a function `f` whose growth rate is related to the function `g` through Big-O notations will also satisfy the Big-Ω relationship between function `g` and `f`. Similarly, Theorem 14 is formally verified for the functions related through little-o and little-ω.

Moreover, as the Big-O notation plays a vital role in the formal verification of online scheduling algorithms for PEV charging, two frequently used properties of the Big-O notation are also verified.

Theorem 15. $\vdash \forall\ \mathtt{f\ g}.\quad \mathtt{f} \in \mathtt{BigO\ g} \quad \Rightarrow\ \forall\ \mathtt{k}.\ (\lambda \mathtt{n}.\ \mathtt{k} * \mathtt{f\ n}) \in (\mathtt{BigO\ g})$

Theorem 15, formally verifies that for a function `f` whose growth rate is in the Big-O of a function `g`, then a scalar (`k`) multiplication will not affect the relationship.

Theorem 16. $\vdash \forall$ `t1 t2 g1 g2.`
$$(\mathtt{t1} \in \mathtt{BigO\ g1}) \wedge (\mathtt{t2} \in \mathtt{BigO\ g2}) \Rightarrow (\lambda \mathtt{n.\ t1\ n} * \mathtt{t2\ n}) \in (\mathtt{BigO}\ (\mathtt{g1} * \mathtt{g2}))$$

On the other hand, `t1`, `t2`, `g1`, and `g2` are functions that accept an argument of type `num` and return a real number. Theorem 16 formally verifies that if two functions `t1` and `t2` are in $O(\mathtt{g1})$ and $O(\mathtt{g2})$, respectively, then their multiplication will belong to the set $O(\mathtt{g1} * \mathtt{g2})$.

The above formalization allows for formally reasoning and verifying the computational complexity of an algorithm. For this paper, we particularly use these results for the formal verification of the online scheduling algorithms for PEV charging.

5. Formal Asymptotic Analysis of Scheduling Algorithms for PEVs

In this section, we present the formally verified results for the worst-case computational complexity of the insertion sort algorithm, Online cooRdinated CHARging Decision (ORCHARD) [9] and online Expected Load Flattening (ELF) algorithms [14].

5.1. Formal Analysis of Insertion Sort Algorithm

Sorting is a process of listing the items in some logical order, such as ascending, descending, alphabetical, chronological, or even topological. There are many computer applications where sorting is employed, such as searching, information retrieval, crunching, and data mining. Therefore, there are a number of algorithms designed to accomplish the task, such as insertion sort, mergesort, heapsort, counting sort, quicksort, and radix sort [16]. As the computational complexities of the algorithms, selected as case studies in this paper, are originally computed using the insertion sort algorithm, the formal analysis of these online scheduling algorithms, in this paper, is also based on the insertion sorting algorithm. For a given input, the algorithm picks each entry and sorts the order of the entry for a segment of input up to that entry and places it according to its desired logical order. The algorithm repeats the procedure for all the entries of the input until the desired order is achieved [16]. The computational complexity of the algorithm is the function of the input length n and has an $O(n^2)$ worst-case asymptotic bound [16].

Algorithm 1 is a pseudocode describing the working of the algorithm [16]. It accepts a list of objects—in our case, the information related to PEVs—for sorting its entries in an ascending order. In Line 1, the **FOR** loop is used to perform sorting of the entries of the list, where n is the total length of the input list A. The loop executes n times, and c_1 is the cost associated to each execution of the **FOR** loop statement. In Line 2, the current entry of the list is assigned as a *key*, which is a local variable used for sorting a particular entry. The execution cost for this statement is considered c_2 and it runs for $n-1$ times as the body of a **FOR** or **WHILE** loop runs one time less than the loop statement. Next, in Line 3, a local variable i is assigned the value of $j-1$, and the execution cost of this statement is considered c_3 and it runs for $n-1$ times as well. In Line 4, the **WHILE** loop is used to perform the comparison test on the subarray $A[1,...,i]$. This test expression is executed $\sum_{j=2}^{n} t_j$ times with an execution cost of c_4 for each run, where t_j is the number of times the **WHILE** loop runs for the comparison of a specific *key*. The expressions in the body of **WHILE**, i.e., Lines 5 and 6, are executed $\sum_{j=2}^{n} t_j - 1$ times. Line 5 shifts the entry to one place in the right direction in the list A, given the conditions in the **WHILE** loop are satisfied. Then, Line 6 updates the entry of the subarray for the comparison in the next iteration of the **WHILE** loop, where c_5 and c_6 are the costs for each execution of these expressions. Finally, the *key* is updated for the next iteration of the **FOR** loop in Line 7. The expression is also executed $n-1$ times with a cost of c_7 for each execution. The algorithm outputs a sorted array A in ascending order. The running time complexity function $T(n)$ is obtained by multiplying the cost of each execution, i.e., c_i, with the total time steps each statement is expected to take in the procedure of sorting, and then summing all of these products.

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j$$

$$+c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

In the worst case, the steps of an algorithm are executed the maximum possible number of times by the algorithm for performing the intended task.

$$T(n) = (\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2})n^2 - (c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2}$$

$$+\frac{c_6}{2} + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

```
Algorithm 1 Insertion Sort                          Cost    Time Steps
Input: list $A$
1: for $j = 2$ to $n$                               $c_1$   $n$
2:   $key = A[j]$                                   $c_2$   $n-1$
3:   $i = j-1$                                      $c_3$   $n-1$
4:   while $i > 0$ and $A[i] > key$                 $c_4$   $\sum_{j=2}^{n} t_j$
5:     $A[i+1] = A[i]$                              $c_5$   $\sum_{j=2}^{n}(t_j - 1)$
6:     $i = i-1$                                    $c_6$   $\sum_{j=2}^{n}(t_j - 1)$
7:   $A[i+1] = key$                                 $c_7$   $n-1$
```

The worst-case running time for an insertion sort algorithm is defined in higher-order logic as:

Definition 6. $\vdash \forall$ n c1 c2 c3 c4 c5 c6 c7.
i_sort_wc_t n c1 c2 c3 c4 c5 c6 c7 =
$(\frac{c4}{2} + +\frac{c5}{2} + \frac{c6}{2})$ n pow 2 $-$ $(c_1 + c_2 + c_3$
$+ \frac{c4}{2} + \frac{c5}{2} + \frac{c6}{2} +$ c7)n $-$ (c2 + c3 + c4 + c7)

where c1, c2, c3, c4, c5, c6, and c7 are real constants that represent the cost of performing the corresponding steps in the sorting algorithm, and n is the variable of type num representing the length of the input list A.

Definitions 1–6 and the formally verified properties of the Big-O notation in Section 4 are used to formally verify the computational complexity results for the insertion sort algorithm as follows:

Theorem 17. $\vdash \forall$ n c1 c2 c3 c4 c5 c6 c7.
0 < n $\wedge$ 0 < c1 $\wedge$ 0 < c2 $\wedge$ 0 < c3 $\wedge$
0 < c4 $\wedge$ 0 < c5 $\wedge$ 0 < c6 $\wedge$ 0 < c7
$\Rightarrow$ (λn. i_sort_wc_t n c1 c2 c3 c4 c5 c6 c7 $\in$ BigO n pow 2)

In the above theorem, 0 < n ensures that the list is not empty, whereas 0 < c_i ensures that the execution cost associated with each expression is also not zero. Under these conditions, the above theorem formally verifies that the worst-case computational complexity of the insertion sort algorithm is $O(n^2)$.

Definition 6 and Theorem 17 play a vital role in the formal reasoning and verification of the computational complexity results of the online scheduling algorithms for PEV charging in the next sections.

5.2. *Online cooRdinated CHARging Decision (ORCHARD)*

The "Online cooRdinated CHARging Decision (ORCHARD) algorithm" [9] considers the random arrivals and departures of PEVs with random charging demands. The problem is formulated as a convex optimization problem, which is a special class of mathematical optimization problems and provides efficient and reliable optimal solutions due to its fairly complete theory [36], and it utilizes the speed scaling technique [37] to optimally schedule the charging demands of PEVs. Speed scaling is a power management technique that is widely used in computer and communication systems to reduce energy consumption. In the seminal work by Yao et al. [37], the speed scaling problem is cast as a scheduling problem in which the tasks, along with the release time, deadline, and amount of work, are not only scheduled but also allocated processor speed. Yao et al. also presented two online algorithms, i.e., average rate (AVR) and optimal available (OA), which learn about the tasks when they are available. The AVR runs each task at an average speed, which in itself is a function of the workload and time required to process the job, whereas OA schedules each task while assuming no task in the future. The ORCHARD uses a variant of the OA algorithm, i.e., qOA [38], which improves the results obtained from the OA algorithm. The ORCHARD algorithm solves the convex optimization problem of charging PEVs whenever a PEV arrives or departs using the interior point method. The computational complexity of the interior point method increases exponentially with an increase in input size. To cater to this issue, the authors in [9] presented a low-complexity algorithm to reduce the exponential computational complexity involved in the optimization problem.

The low-complexity routine in [9] exploits the structure of the optimal solution, i.e., an optimization technique such as the interior point method essentially balances the total workload for an optimal solution. Therefore, the proposed low-complexity routine balances the workload by considering the interval density to quantify the amount of the workload in the respective intervals and then shifts the load from high-density intervals to the adjacent interval with a low-density workload to balance the workload, which results in the optimal solution.

Figure 4 depicts a worst-case scenario for an online scheduling problem for PEV charging. In the worst-case scenario, the scheduler records N PEVs' information at any time t, which consists of their arrival and departure times (a_i and f_i, respectively) and charging demands d_i, for scheduling purposes. In the worst-case scenario, when there are N PEVs at time t, with arrival and departure times and charging demands to be scheduled, then there can be at most N possible intervals I_i. These N intervals can, at most, lead to N^2 possible adjacent intervals for load shifting w_i, termed as time windows [9], as shown in Figure 4.

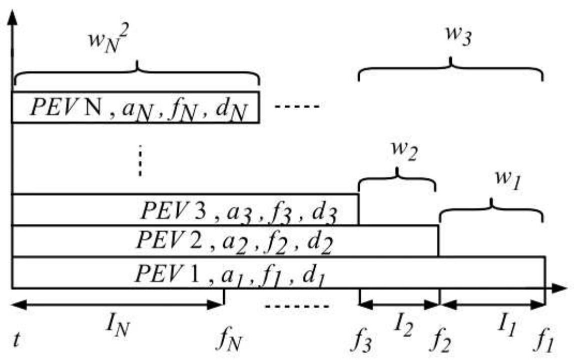


Figure 4. Online PEV charging scheduling.

Algorithm 2 describes the main computational steps of the low-complexity routine. The data required for the PEV low-complexity charging scheduling algorithm is the input, i.e., a set P, a certain time interval τ, workload density ρ_i of each time window, and S_τ. P is the set of PEVs; $\tau \in W := w_1, ..., w_{N^2}$ is a member of the set W, whose members represent all possible time windows; $\rho_i = \frac{1}{\delta_\tau} \sum_{P_i \in S_\tau} d_i$ is the density of the workload of time window τ, which has length δ_τ; $S_\tau \in P$ is the set representing the number of PEVs in time window τ. In Line 1, a **WHILE** loop is used that repeats the procedure until all the PEVs are scheduled. For the worst case, it is assumed that the maximum number of PEVs, i.e., N, are to be scheduled at anytime t, and, therefore, in the worst-case scenario, the loop executes **N** times. In Line 2, a certain time interval τ of highest workload is calculated and selected. This task can be accomplished using any sorting algorithm; however, we considered the commonly used insertion sort algorithm, described above. For N PEVs, there could be, at most, N^2 such time slots which are needed to be sorted and, therefore, for $n = N^2$, it has a $O(N^4)$ worst-case computational complexity, as mentioned in Algorithm 2. The sorting procedure depends on the input, and, therefore, it is imparted to the computational cost of any algorithm, which justifies its inclusion in the complexity analysis of the online scheduling algorithm.

Algorithm 2 Low-complexity Routine — Worst-case

```
Input: $P := P_1, ..., P_N, \tau, \rho_\tau = \frac{1}{\delta_\tau}\sum_{P_i \in S_\tau} d_i, S_\tau \in P$
1: while $P \neq \{\}$                                             $N$
2:     Determine the time interval, $\tau$ of the
       highest intensity, i.e., $\rho_\tau$                         $O(N^4)$
3:     Allocate the charging rate, $\widehat{x}_i$, to all PEVs
       such that $\rho_\tau = \sum_{P_i \in S_\tau} \widehat{x}_i$
4:     Set $P := P \backslash S_I$
5:     Remove $I$ from the time horizon
       and update the departure, arrival
       and residual demands
```

Line 3 assigns to every PEV a charging rate such that the total charging rate is bounded by the maximum charging rate of the interval. In Lines 4 and 5, the algorithm removes the interval and PEVs, which are scheduled, and updates the information for the next iteration. From Lines 3 to 5, the computational cost is not taken into account due to the fact that the complexity associated with these steps is not proportional to the input size. The resulting running time function for the algorithm is then described in HOL-Light as:

Definition 7.
```
⊢ ∀ N c1 c2 c3 c4 c5 c6 c7.
     lcr_wc_t N c1 c2 c3 c4 c5 c6 c7 =
            N * i_sort_wc_t N pow 2 c1 c2 ,c3 c4 c5 c6 c7
```

Definition 7 is a higher-order-logic description of the worst-case computational time of the low-complexity algorithm for execution. `N` represents the number of times a **WHILE** loop is executed for the worst-case scenario. The constants `c1`, `c2`, `c3`, `c4`, `c5`, `c6`, `c7`, and `c8` represent the cost associated with each step in the insertion sort algorithm, whereas $\texttt{N}^2$ denotes the maximum number of intervals that are possible in the worst-case scenario. The above definition is based on the function `i_sort_wc_t` to incorporate the computational complexity due to sorting procedure in Algorithm 2.

Definitions 1, 6, and 7, along with the theorems related to the insertion sort algorithm, i.e., Theorem 17, and the formally verified properties of the Big-O notation, given in Section 4, allow us to formally verify the worst-case complexity result for the ORCHARD algorithm in HOL-Light as follows:

Theorem 18.
```
⊢ ∀ N c1 c2 c3 c4 c5 c6 c7.
     0 < N ∧ 0 < c1 ∧ 0 < c2 ∧ 0 < c3
        ∧ 0 < c4 ∧ 0 < c5 ∧ 0 < c6 ∧ 0 < c7
            ⇒ lcr_wc_t N c1 c2 c3 c4 c5 c6 c7 ∈ BigO N pow 5
```

The application of theorem proving, in this case, requires us to explicitly add all the required specifications of assumptions under which the result holds true in the above theorem. For example, the precondition on the number of intervals, i.e., `0 < N`, in Theorem 16, is added to ensure that the number of intervals is not zero, which is essential for the scheduling task. On the other hand, the preconditions on the constants originate from the formalization of the worst-case scenario of the insertion sort algorithm, presented in the previous section.

5.3. Low-Complexity Online Expected Load Flattening (ELF) Algorithm

The PEV charging scheduling problem has been formulated using the model predictive control (MPC) approach by incorporating first-order statistics of the load on the grid in [14]. This enables accounting for the potential uncertainties in the arrival and departure of PEVs to and from a charging facility and the variation in the electricity load in the grid system. The online algorithm, in [14], considers that the entire system is divided into T equal-length intervals, as shown in Figure 5.

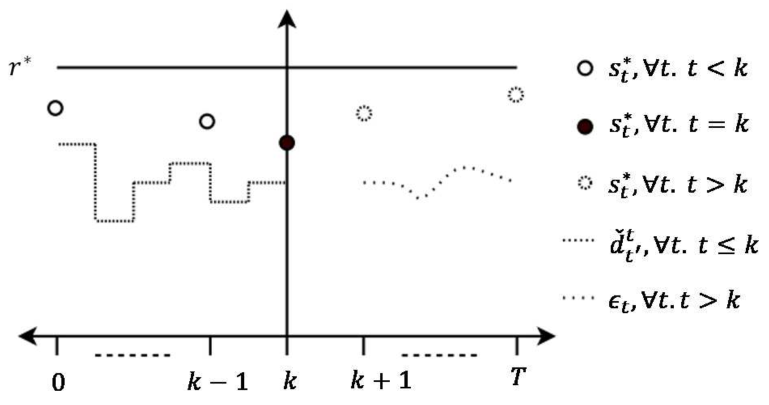

r^*
$s_t^*, \forall t.\ t < k$
$s_t^*, \forall t.\ t = k$
$s_t^*, \forall t.\ t > k$
$\check{d}_{t'}^t, \forall t.\ t \le k$
$\epsilon_t, \forall t. t > k$
0
$k-1$
k
$k+1$
T

Figure 5. Expected Load Flattening (ELF) algorithm.

The remaining charging demand of a PEV i which arrives at time k is defined as $\tilde{d}_i^k = d_i - \sum_{t=a_i}^{k-1} x_{it}$. For the PEVs that have not arrived by time $k-1$, the remaining charging demand is considered to be equal to the charging demand of that PEV, i.e., $\tilde{d}_i^k = d_i$. The total unfinished charging demand at time slot k is defined as the sum of remaining charging demands of all the PEVs, $\tilde{d}_t^k = \sum_{i \in \{i|f_i=t\}} \tilde{d}_i^k, \forall t.k, ..., T$. The total unfinished charging demand is, then, used to define the state of the system at time t as [14]

$$\mathbf{D}_t = [l_t, \tilde{d}_t^t, \tilde{d}_{t+1}^t, ..., \tilde{d}_T^t] \tag{1}$$

where l_t represents the electricity demand excluding the PEV charging at time t. $\tilde{d}_{t'}^t$ is the total unfinished charging demand at time t that must be completed by time t'. Moreover, future load demands at random arrival events, ξ_t, are represented as [14]

$$\xi_t = [\iota_t, \gamma_t^t, \gamma_{t+1}^t, ...,, \gamma_T^t] \tag{2}$$

where ι_t represents the base load at time t and $\gamma_{t'}^t$ represents the total unfinished charging demand that arrives at time t and must be fulfilled by time t'. Finally, the first-order statistics of ξ_t are represented as

$$\epsilon_t = [\alpha_t, \beta_t^t, \beta_{t+1}^t, ...,, \beta_T^t] \tag{3}$$

where $\alpha_t = \mathrm{E}[\iota_t]$ and $\beta_{t'}^t = \mathrm{E}[\gamma_{t'}^t]$ are the expected values for the random variables representing base load and total unfinished charging demands, respectively. The algorithm, using the state of the system (1) and first-order statistical data (3) at every time slot, finds the near-optimal solution, s_k^*, with respect to the offline solution, r^*, to the optimization problem described in [14], as shown in Figure 5. Conventionally, numerical methods, such as the interior point methods, are employed to find the solution at each stage k, which leads to an unbearable computational cost, especially in the case of the large-scale integration of the PEVs with the grid system. Therefore, the authors in [14] used a low-complexity Expected Load Flattening (ELF) algorithm to reduce the computational complexity.

The ELF algorithm relies on the load flattening characteristic of the optimal solution, i.e., the standard numerical methods try to flatten the demand curve. Therefore, the ELF algorithm tries to balance the charging demand among all the time slots $k, ..., T$, where T denotes total time stages in the problem, to flatten the charging demand curve of PEVs. The ELF charging scheduling algorithm incorporates the information of (1) and (3) in $\bar{d}^{t'}_{t''}$

$$\bar{d}^{t}_{t''} = \begin{cases} \tilde{d}^{t'}_{t''}, & \text{for } t'' = k, ..., T, t' = k \\ \beta^{t'}_{t''}, & \text{fro } t'' = t', ..., T, t' = k+1, ..., T \end{cases} \tag{4}$$

Algorithm 3 provides the pseudocode of the ELF algorithm that describes the procedure and computational steps required for the allocation of the charging rate to the PEVs at time slot k.

```
Algorithm 3 Expected Load Flattening (ELF)                              Worst-case
Input:  $\mathbf{D_k}, \epsilon_k, t = k+1, ..., T$
Output: Charging rate, $s_k$, at time slot $k$
1: initialization $i = 0$ and $j = 0$
2: repeat
3:     For all time slots,                                              $O(T^2)$
       $i = k, ..., T, j = i, ..., T$, calculate
       $i^*, j^* = \arg\max_{k \le i \le j \le T} \{\frac{\sum_{t'=i}^{j}(\sum_{t''=t'}^{j} \bar{d}^{t'}_{t''} + \alpha_{t'})}{j-i+1}\}$
4:     set
       $y^* = \{\frac{\sum_{t'=i}^{j}(\sum_{t''=t'}^{j} \bar{d}^{t'}_{t''} + \beta_{t'})}{j-i+1}\}$
5:     delete time slots $i^*, j^*$ and relabel the existing
       time slot $t > j^*$as $t - j + i - 1$
6: until $i \neq k$                                                     $T$
7: Set $s_k = y^* - l_k$
```

The input to Algorithm 3 consists of the data required for the ELF algorithm, i.e., the total unfinished charging demands as the system state $\mathbf{D}_k$ and the expected values of the random events ϵ_t, to schedule the PEVs. The output of the ELF algorithm is the charging rate s_k at time slot k. Line 1 initializes two variables, i.e., i and j. In Line 2, a control structure **REPEAT-UNTIL** is used that is conditioned on the variable i to the index of the time slots, along with j, for the purpose of utilizing information related to the remaining charging demands and first-order statistics of random events $\bar{d}^{t}_{t''}$ and load α_t at time slot k. In the worst-case scenario, the condition of $i \neq k$ may falsify after, at most, T iterations; therefore, the complexity cost of Algorithm 3 is T. In Line 3, the algorithm tries to find the index of the maximum load density based on the total load composed of the remaining and expected charging demands of PEVs and the expected electricity load from sources other than the PEVs. As aforementioned in the case of the ORCHARD analysis, this task is equivalent to sorting the given intervals in descending or ascending order, which is usually performed using sorting algorithms. We again considered insertion sort for this purpose, and, therefore, the worst-case computational cost for the length of T intervals would be $O(T^2)$. Line 4—the maximum value of the charging demand, which is equivalent to the maximum charging rate—is saved in the variable y for future use. In Line 5, the time horizon and the charging demands are readjusted for the next iteration. Finally, in Line 7, the candidate solution is obtained by removing the effect of load other than PEV charging demand. Lines 4, 5, and 6 are not accounted for in the computational complexity analysis, as the operations performed in these lines do not scale with the length of the input, i.e., T. The resulting computational complexity is described in HOL-Light as a theorem as follows:

Definition 8. $\vdash$ `∀ T c1 c2 c3 c4 c5 c6 c7.`
```
    elf_wc_t T c1 c2 c3 c4 c5 c6 c7 =
        T ∗ i_sort_wc_t T pow 2 c1 c2 c3 c4 c5 c6 c7
```

where `T` is the number of times the **WHILE** loop is executed for the worst-case scenario and, therefore, also the maximum number of possible intervals. The above definition is based on the function `i_sort_wc_t` to incorporate the computational complexity due to sorting procedure in Algorithm 3, which accepts the number of intervals, i.e., $\texttt{T}^2$, and real constants `c1`, `c1`, `c2`, `c3`, `c4`, `c5`, `c6`, and `c7` representing the cost of execution of the expressions in Algorithm 1.

Definitions 1, 6, and 8, along with the theorems related to the insertion sort algorithm, i.e., Theorem 17, and the formally verified properties of the Big-O notation, given in Section 4, allow us to formally verify the worst-case complexity result for the ELF algorithm in HOL-Light as follows:

Theorem 19. $\vdash$ `∀ T c1 c2 c3 c4 c5 c6 c7.`
```
 0 < T ∧ 0 < c1  ∧  0 < c2  ∧  0 < c3  ∧  0 < c4  ∧ 0 < c5  ∧  0 < c6  ∧  0 < c7
             ⇒  elf_wc_t T c1 c2 c3 c4 c5 c6 c7 ∈ BigO (T pow 3)
```

The above theorem explicitly specifies the conditions on the variables, such as `T` and $\texttt{c}_\texttt{i}$'s, which are required for the computational complexity of the ELF algorithm, i.e., $O(T^3)$, to hold true. This information can further be utilized in the implementation phase to avoid any subtle errors and thus can enhance the security, reliability, and efficiency of the overall system.

The formalization in this paper provides a foundational framework for the formal verification of the complexity of the online scheduling algorithms for PEV charging. We used the proposed method to formally verify two such algorithms as Theorems 18 and 19 using the sound core of the HOL-light theorem prover. The formal asymptotic analysis resulted in identifying various assumptions which are necessary for the computational complexity results of these algorithms to be valid. It is important to note that the proposed formalization can be used to formally model most of the commonly used control structures of pseudocodes while conducting asymptotic analysis of algorithms. Moreover, the reasoning support presented in this paper allows us to conduct the formal asymptotic analysis of any algorithm almost automatically. These features make the proposed framework a very practical framework for users with very little background in formal methods.

Asymptotic analysis is a fundamental tool for the design and analysis of algorithms [16], in general. Therefore, the proposed formalization can be viewed as a primary resource to conduct the formal asymptotic analysis and verification of algorithms designed using basic design strategies, such as Brute force, dynamic programming, divide-and-conquer etc., for smart grids or elsewhere. For example, the design of the ORCHARD and ELF algorithms can be verified using different sorting algorithms, such as quicksort or mergesort, to compare the performance of these algorithms; this is highly desirable before the implementation phase of these algorithms in the PEV-integrated grid system. Although the proposed logical framework and methodology are aimed for formal asymptotic analysis, in future, the formalization can be easily utilized for the formal verification of detailed and explicit models of the scheduling problem for PEV charging, e.g., incorporating the charging rates, number of PEVs, and computational cost utilized for performing the basic operations. The above-mentioned formally verified results can be used to draw many useful insights about the given scheduling algorithm in the presence of PEV loads. In order to illustrate this process, we used a fixed PEV load and increased the cost of the operations of a scheduler so that the given algorithms approached their upper bound, i.e., the worst-case computational complexity. The formally verified results, i.e., Theorems 18 and 19, facilitate this analysis by providing all the conditions on the parameters readily available, which are usually not known to the users or may need extensive trial and error runs to discover. Moreover, the cost of the operations are associated with the scheduler and, therefore, such an analysis can be useful for estimating the development and deployment cost of the smart grid infrastructure at the early stages of the design. We considered 30 schedulers with a random, but increasing, cost of operations for performing the steps required in the ORCHARD and ELF algorithms. The number of PEVs for ORCHARD and the time horizon for ELF were set at 1000. We ran each of the algorithms using these cost functions to assess the effect on the worst-case computational complexity of the two algorithms. The cost functions, shown in Figures 6a and 7a, i.e., $\texttt{i\_sort\_wc\_t}_i$, are defined using Definition 6, and they represent the cost functions for each scheduler for sorting the array for scheduling purposes. These functions were used to compute the computational complexity of the two functions defined in Definitions 7 and 8. The Big-O for ORCHARD and ELF was mathematically modeled using Definition 1, where $C_0 = max(max(c_i))$, and c_i represents the constants for the $\texttt{i\_sort\_wc\_t}_i$ function and are randomly generated but satisfy the conditions specified in Theorems 18 and 19.

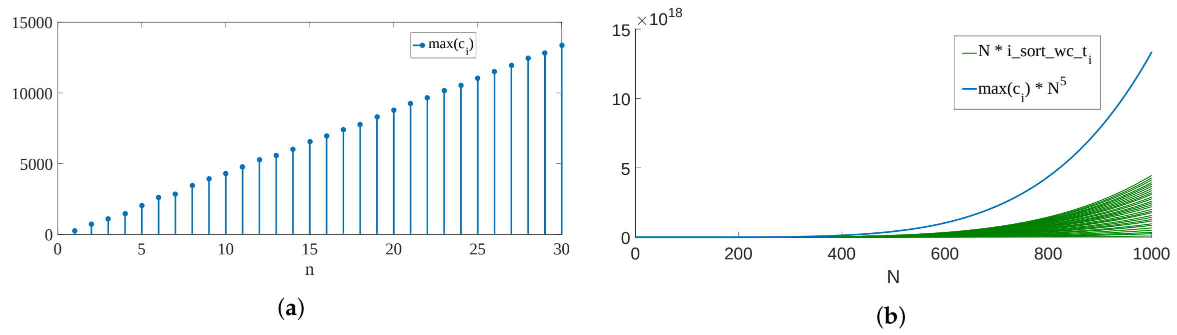


Figure 6. Asymptotic behavior of the ORCHARD algorithm for increasing cost functions. (**a**) Cost function of $n \leq 30$ schedulers; (**b**) Asymptotic growth of $n \leq 30$ schedulers.

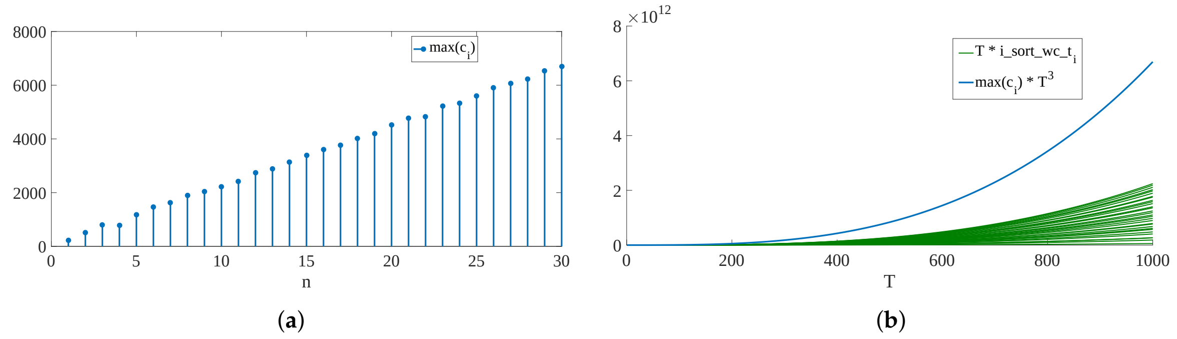


Figure 7. Asymptotic behavior of the ELF algorithm for increasing cost functions. (**a**) Cost function of $n \leq 30$ schedulers; (**b**) Asymptotic growth of $n \leq 30$ schedulers.

5.4. Simulation Results

Figures 6a and 7a show an increasing cost function for 30 schedulers, which were used for running the two algorithms. On the other hand, Figures 6b and 7b show that with an increase in the operation cost, the computational complexities of both algorithms approach their upper bound, i.e., Big-O. The above analysis can be utilized to design schedulers by incorporating their real operations cost into the cost models to meet the desired latency and quality of service in a PEV-integrated grid system at early stages of the design.

6. Conclusions

In this paper, we present an approach for the formal asymptotic analysis of online scheduling algorithms for PEV charging. In this regard, we developed a logical formalization of asymptotic notations and formally verified their properties to assist the formal analysis of such algorithms. To illustrate the usefulness of the proposed approach, we leveraged it to formally analyze the computational complexity of two state-of-the-art PEV scheduling algorithms, namely, the Online cooRdinated CHARging Decision (ORCHARD) and online Expected Load Flattening (ELF) algorithms. We also formally verified the insertion sort algorithm, which is an intermediate step for the complexity analysis of online PEV charging algorithms. The distinguishing features of the proposed analysis work include its accurate results due to the involvement of sound theorem proving and the availability of an exhaustive set of assumptions that are required for the validity of the results. This exhaustive set of assumptions was then utilized in the simulations to asses the effects of the cost of operations on the upper bounds of the two algorithms.

Author Contributions: Conceptualization, F.A., N.B. and S.R.H.; Formal analysis, A.A.; Methodology, A.A. and O.H.; Supervision, O.H.; Writing—original draft, A.A.; Writing—review & editing, O.H., F.A., N.B. and S.R.H.

Funding: This work is supported by ICT Fund UAE, fund number 21N206 at UAE University, Al Ain, United Arab Emirates.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cazzola, P.; Gorner, M. *Global EV Outlook 2018 Towards Cross-Modal Electrification*; International Energy Agency: Paris, France, 2018; doi:10.1787/9789264302365-en.
2. Pasquier, M.; Mintz, J.M.M. *IEA-HEV-TCP Task 24: Economic Impact Assessment of E-Mobility*; International Energy Agency: Paris, France, 2016.
3. Lopes, J.A.P.; Soares, F.J.; Almeida, P.M.R. Integration of electric vehicles in the electric power system. *Proc. IEEE* **2011**, *99*, 168–183, doi:10.1109/JPROC.2010.2066250.
4. Guille, C.; Gross, G. A conceptual framework for the vehicle-to-grid (V2G) implementation. *Energy Policy* **2009**, *37*, 4379–4390.
5. Tang, W.; Jun, Y. *Optimal Charging Control of Electric Vehicles in Smart Grids*; Springer: Berlin, Germany, 2017; doi:10.1007/978-3-319-45862-5.
6. He, Y.; Venkatesh, B.; Guan, L. Optimal scheduling for charging and discharging of electric vehicles. *IEEE Trans. Smart Grid* **2012**, *3*, 1095–1105, doi:10.1109/tsg.2011.2173507.
7. Kurucz, C.; Brandt, D.; Sim, S. A linear programming model for reducing system peak through customer load control programs. *IEEE Trans. Power Syst.* **1996**, *11*, 1817–1824, doi:10.1109/59.544648.
8. Soares, J.; Sousa, T.; Morais, H.; Vale, Z.; Faria, P. An optimal scheduling problem in distribution networks considering V2G. In Proceedings of the 2011 IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG), Paris, France, 11–15 April 2011; pp. 1–8, doi:10.1109/ciasg.2011.5953342.
9. Tang, W.; Bi, S.; Zhang, Y.J.A. Online coordinated charging decision algorithm for electric vehicles without future information. *IEEE Trans. Smart Grid* **2014**, *5*, 2810–2824, doi:10.1109/TSG.2014.2346925.
10. Zhang, T.; Chen, W.; Han, Z.; Cao, Z. Charging scheduling of electric vehicles with local renewable energy under uncertain electric vehicle arrival and grid power price. *IEEE Trans. Veh. Technol.* **2014**, *63*, 2600–2612, doi:10.1109/TVT.2013.2295591.
11. Ma, Z.; Callaway, D.S.; Hiskens, I.A. Decentralized charging control of large populations of plug-in electric vehicles. *IEEE Trans. Control Syst. Technol.* **2013**, *21*, 67–78, doi:10.1109/cdc.2010.5717547.
12. Aghaei, J.; Alizadeh, M.I. Demand response in smart electricity grids equipped with renewable energy sources: A review. *Renew. Sustain. Energy Rev.* **2013**, *18*, 64–72, doi:10.1016/j.rser.2012.09.019.
13. Tang, W.; Bi, S.; Zhang, Y.J. Online charging scheduling algorithms of electric vehicles in smart grid: An overview. *IEEE Commun. Mag.* **2016**, *54*, 76–83, doi:10.1109/mcom.2016.1600346cm.
14. Tang, W.; Zhang, Y.J.A. A model predictive control approach for low-complexity electric vehicle charging scheduling: optimality and scalability. *IEEE Trans. Power Syst.* **2017**, *32*, 1050–1063, doi:10.1109/tpwrs.2016.2585202.

15. Gerding, E.H.; Robu, V.; Stein, S.; Parkes, D.C.; Rogers, A.; Jennings, N.R. Online mechanism design for electric vehicle charging. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems 2011, Taipei, Taiwan, 2–6 May 2011; Volume 2, pp. 811–818.
16. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms Second Edition*; The MIT Press: Cambridge, MA, USA, 2001. Available online: http://web.ist.utl.pt/fabio.ferreira/material/asa/clrs.pdf (accessed on 26 October 2018).
17. Graham, R.L.; Knuth, D.E.; Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed.; Addison-Wesley Longman Publishing Co., Inc.: Chicago, IL, USA, 1994.
18. Hasan, O.; Tahar, S. Formal Verification Methods. In *Encyclopedia of Information Science and Technology*, 3rd ed.; IGI Global: Hershey, PA, USA, 2015; pp. 7162–7170, doi:10.4018/978-1-4666-5888-2.ch705.
19. Qadir, J.; Hasan, O. Applying formal methods to networking: Theory, techniques, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 256–291.
20. Milica, B. The State-of-the-Art in Formal Methods. In *AFOSR Summer Research Technical Report for Rome Research Site*; AFRL/IFGB: Wright, OH, USA, 1998.
21. Clarke, E.M.; Grumberg, O.; Long, D.E. Model checking and abstraction. *ACM Trans. Program. Lang. Syst. (TOPLAS)* **1994**, *16*, 1512–1542.
22. Avigad, J.; Donnelly, K. Formalizing O notation in Isabelle/HOL. In *International Joint Conference on Automated Reasoning*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 357–371.
23. Deilami, S. Online Coordination of Plug-In Electric Vehicles Considering Grid Congestion and Smart Grid Power Quality. *Energies* **2018**, *11*, 2187.
24. Rodrigues, E.; Godina, R.; Pouresmaeil, E.; Ferreira, J.; Catalão, J. Domestic appliances energy optimization with model predictive control. *Energy Convers. Manag.* **2017**, *142*, 402–413.
25. Godina, R.; Rodrigues, E.M.; Pouresmaeil, E.; Catalão, J.P. Optimal residential model predictive control energy management performance with PV microgeneration. *Comput. Oper. Res.* **2018**, *96*, 143–156.
26. Godina, R.; Rodrigues, E.M.; Pouresmaeil, E.; Matias, J.C.; Catalão, J.P. Model predictive control home energy management and optimization strategy with demand response. *Appl. Sci.* **2018**, *8*, 408.
27. Buttazzo, G.C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*; Springer Science & Business Media: Berlin, Germany, 2011; Volume 24.
28. MathWorks. Available online: https://ch.mathworks.com/products/new_products/release2016a.html (accessed on 25 October 2018).
29. Huth, M.; Ryan, M. *Logic in Computer Science: Modelling and Reasoning about Systems*; Cambridge University Press: Cambridge, UK, 2004; doi:10.1017/cbo9780511810275.
30. Harrison, J. The HOL Light Theorem Prover. Available online: http://www.cl.cam.ac.uk/~jrh13/hol-light/ (accessed on 12 September 2018).
31. Harrison, J.; Slind, K.; Arthan, R. HOL. In *The Seventeen Provers of the World*; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2006; Volume 3600, pp. 11–19, doi:10.1007/11542384_3.
32. Bertot, Y.; Castéran, P. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*; Springer Science & Business Media: Berlin, Germany, 2013. Available online: http://www.labri.fr/perso/casteran/CoqArt/ (accessed on 12 September 2018).
33. Owre, S.; Rushby, J.; Shankar, N. PVS: A Prototype Verification System. In *Automated Deduction*; Lecture Notes in Computer Science; Springer: Berlin, Germany, 1992; Volume 607, pp. 748–752, doi:10.1007/3-540-55602-8_217.
34. Harrison, J. HOL Light: An overview. In *International Conference on Theorem Proving in Higher Order Logics*; Springer: Berlin, Germany, 2009; pp. 60–66, doi:10.1007/978-3-642-03359-9_4.
35. Harrison, J. Floating-Point Verification. *J. UCS* **2007**, *13*, 629–638, doi:10.1007/978-1-4471-1591-5_7.
36. Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004; doi:10.1017/cbo9780511804441.

37. Yao, F.; Demers, A.; Shenker, S. A scheduling model for reduced CPU energy. In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, USA, 23–25 October 1995; pp. 374–382, doi:10.1109/SFCS.1995.492493.
38. Bansal, N.; Chan, H.L.; Pruhs, K.; Katz, D. Improved bounds for speed scaling in devices obeying the cube-root rule. In *International Colloquium on Automata, Languages and Programming*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 144–155, doi:10.1007/978-3-642-02927-1_14.