Idea Paper: Development of a Software Framework for Formalizing Forcefield Atom-Typing for Molecular Simulation

Christopher R. Iacovella, Janos Sallai, Christoph Klein, Tengyu Ma
Vanderbilt University, Nashville, TN
{janos.sallai,christopher.r.iacovella, christoph.klein,tengyu.ma}@vanderbilt.edu

***Abstract*—Forcefields are a crucial ingredient of Molecular Dynamics (MD) simulations, describing the types and parameters of interactions between the simulated particles. These parameter sets, however, are typically specific to the molecule in which the atoms appears, where within the molecule the atom is positioned, the phase or state point of the system, as well as the simulator tool in use. This makes choosing the correct parameter values a tedious and error prone task. Forcefield parameters, furthermore, are often hard to locate: some are published in scientific papers, others come with MD tools, often with no or ambiguous documentation on their applicability. In this paper, we present a framework that aims to solve this data management issue, proposing a common format for forcefields that is self-documenting with machine readable, declarative usage rules. We believe that processes and tools that are commonly used today in software development (e.g, unit testing, verification and validation, continuous integration, and version control) are, with proper infrastructure support, applicable to forcefield development, as well. The paper describes how such an infrastructure can tackle managing and evolving forcefields by the MD community, and proposes a way to encourage and incentivize involvement by the stakeholders.**

I. Introduction

Molecular simulation plays a key role in understanding the atomistic and molecular level interactions that underlie many natural and man-made materials and processes. [1]–[4] Classical molecular simulations rely upon forcefields to describe the various interactions that exist between atoms and/or groups of atoms, including non-bonded interactions (van der Waals and electrostatics) and bonded interactions (bonds, angles, and torsions). These forcefields are typically expressed as a set of analytical function with adjustable fitting parameters for different atomic/molecular species. Considerable efforts have been undertaken by many research groups to develop accurate forcefields, both in determining the mathematical functions and associated fitting parameters, for a large variety of molecular species under different conditions. Numerous forcefields (and associated parameters), have been devised with acronyms such as: AMBER [5], CHARMM [6], OPLS [7], SKS [1], TraPPE [8], COMPASS [9] and GROMOS [10]. The availability of these forcefields can significantly reduce or completely eliminate the difficult and costly task of determining the interactions between species, allowing researchers to instead focus their efforts on the motivating scientific questions. For example, the large number of parameters available for the OPLS forcefield has allowed for the automated screening of drug molecules in order to identify promising candidates for more effective treatments of HIV [11].

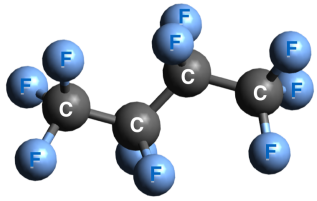

Fig. 1. *Perfluorobutane, CF_3-CF_2-CF_2-CF_3, molecule shown with ball and stick representation.*

However, while researchers do not necessarily need to spend time developing the forcefield parameters, determining *which* parameters to use (*i.e.*, atom-typing) is still often a tedious and error prone task. In many forcefields, the appropriate interaction parameters will depend on the local topological environment of atoms. For example, the non-bonded interactions of carbon atoms in perfluoroalkanes (PFA) are typically different for terminal carbons verses "middle" carbons [12]. Similarly, when identifying the correct parameters for a torsional term, one must typically consider not just the backbone, ($e.g.$, C-C-C-C), but also the bonds of each atom in the backbone ($e.g.$, CH_3-CH_2-CH_2-CH_3, $vs.$ CF_3-CF_2-CF_2-CF_3) [7], [13]. Consequently, a given forcefield may include multiple unique parameters for a given atom, whose usage depends on the *chemical context* of the atom. That is, the appropriate parameters will typically depend on a number of factors, including the specific molecule in which the atom appears ($e.g.$, alkanes $vs.$ perfluoroalkanes), or where in that molecule the atom appears ($e.g.$, terminal $vs.$ middle).As a clear illustration of this, we note that the OPLS all-atom forcefield parameter database (as provided in the TINKER molecular modeling software [14]) contains 427 different "types" of carbon atoms, which are all differentiated based on their chemical context. Furthermore, determining which of the multitude of forcefield parameters is most appropriate, based on the chemical context of an atom, is often accomplished through brief,

unstructured – and sometimes ambiguous – annotations located in the forcefield parameter files. Even journal articles associated with forcefield parameters can be unclear, where parameters are typically listed in table form, often with limited annotations or examples of their usage. Also, given their static nature, parameters provided in journal articles may not be the most up-to-date, whether resulting from typographic errors or modifications in later work. As a result, for many forcefields, the formal logic required to distinguish atom types may be difficult to find, difficult to interpret, out-of-date, or simply ambiguous. This ambiguity is confounded by the fact that many published journal articles that rely on forcefields often provide only vague citations for the source of the parameters.

Most research groups use some combination of hand parameterization and logic based codes to facilitate the process of atom-typing. Identifying atom types by hand, while easy for small/simple molecules and important for validation, becomes impractical for large molecules or systems with significant heterogeneity. For molecules with only a few atomic species and little variability, *ad hoc*, logic-based codes are relatively straightforward to write and test. However, as the number of unique atom types increases, properly defining the appropriate if/else statements required in most logic codes becomes onerous, even for people who are well versed in programming *and* have domain expertise in molecular modeling. Also, nested logic statements are often difficult to adequately test and debug and thus introduce a potential source of error. Since *ad hoc* codes are not typically released to the general community, any errors in the code may go undiscovered and thus data based on flawed atom-typing may appear in published scientific literature. To address this, several community tools and approaches have been developed to aid in atom-typing [15]–[20], some of which are more generally applicable than others. Forcefields developed in the biophysics community tend to have exceptionally well vetted parameterization codes, such as AMBER's antechamber [21], but such tools are typically designed to only work with their associated forcefield and may also produce output specific to a given simulation package, rather than a general form. In general, these tools all rely on a hierarchy where rules that identify more specialized atom types must be called in precise order [21], such that more general atom types are only chosen when more specialized matches do not exist ($i.e.$, they include rule precedence). Maintaining, let alone constructing, these hierarchies is extremely error prone and, just as in *ad hoc* codes, typically results in source code with deeply nested if/else statements. In these hierarchical schemes, in order to add a new atom type or correct an error, a developer must have a complete picture of the hierarchy and know exactly where the relevant rule should be placed, such that it does not inadvertently override other rules. This may impose practical limits on functionality, where, for example, a user is not able to easily extend the rules to include newer parameters, or that such attempts to extend rules result in incorrect atom-typing for other molecular species.

To address these issues, we are developing a new framework for atom-typing, based upon first order logic over graph structures. The novelty of our approach lies in the declarative annotation syntax that allows for 1) decoupling the *definition* of forcefield-specific atom-typing rules from *how* a forcefield-agnostic tool uses these annotations to automatically compute the atom-typing of complex molecular systems, and 2) formally verifying and automatically validating annotated forcefields. Specifically, we proposed to:

- establish a forcefield agnostic ($i.e.$, general) formalism to express the chemical context in which a particular force field entry is applicable ($i.e.$, forcefield usage semantics);
- develop a tool suite that automates atom-typing of molecular structures using the semantics-annotated forcefields;
- and establish a development process to create, incrementally extend, and evolve annotated forcefields, providing tools for automated verification, validation, and continuous integration techniques.

An important goal of the project is to disseminate results and to foster community involvement through the creation of a centralized online forcefield repository containing the formally annotated forcefields, documentation of how to annotate forcefields, along with files for benchmarking and validation of the atom-typing software. Given the importance of accurately applying forcefields to molecular simulation, the new atom-typing framework developed in this work has the potential to significantly impact the community, affording researchers greater confidence in the model parameters used in their studies, eliminate the need to develop *ad hoc* atom-typing codes, make forcefield parameter usage clearer, and significantly reduce incorrect atom-typing as a source of error and inconsistency in published results.

II. Vision

In our vision, forcefield definitions are not just sets of tables with numerical data, but as unambiguous, well-structured documents with rich metadata, including the formal description of the chemical context in which the particular parameter values can be applied. In the future, the tedious and error prone task of manually atom-typing and parameterizing complex molecular models (inputs to simulators) will be eliminated, and replaced by the automated process that relies on the machine-readable forcefield usage semantics.

We envision forcefields as dynamic data entities that evolve over time: parameterizations of additional

chemical species are added, already supported chemical species are specialized, and parameter values are tweaked for better modeling of the chemical interactions. Forcefield definitions will be maintained by online communities and hosted at shared repositories with version control capabilities. These online repositories will also support continuous integration (CI), *i.e.*, automated verification, validation, and testing of the forcefields as they evolve.

We believe that the developers of forcefield definitions deserve credit for their efforts. The online repositories will generate permanent URL links per forcefield versions, as well as document object identifiers (DOI) which allow unambiguously referencing these data artifacts, and properly citing them in scientific publications.

III. Approach

The work to be carried out can be roughly broken down into two main efforts: (1) the development of the atom-typing framework, including the formalism to express forcefield usage semantics, and (2) examination of case studies designed to test, validate, and refine the atom-typing framework. We note that these efforts will be executed concurrently, to provide a continual loop of development, testing, and refinement.

A. Annotation Syntax

One of the main goals of the proposed effort is to design a domain specific language (DSL) for annotating forcefield parameters. The DSL will be used to express the chemical context in which the particular atom type is applicable. This DSL will, effectively, serve as a means to unambiguously document the forcefield. The syntax we propose will be expressive, unambiguous, and both human and machine readable.

Consider the following example of tagging the carbon atoms in the perfluorobutane (shown in Fig. 1) with its usage semantics to create the DSL. Atom type 791 in TINKER's OPLS-aa parameter database [14] corresponds to a terminal carbon of a perfluorobutane chain. This will be annotated with the following statement:

$$\begin{aligned} C_{791} : \mathit{type} = C \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = F)) = 3 \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = C)) = 1. \end{aligned} \quad (1)$$

This means that atom type C_{791} is a carbon (C) atom, which can be used in a chemical context where it has 3 bonded fluorines (F), and one carbon. Similarly, atom type 792, applicable to carbon atoms that are part of a fluorocarbon backbone, would be annotated as

$$\begin{aligned} C_{792} : \mathit{type} = C \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = C)) = 2 \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = F)) = 2. \end{aligned} \quad (2)$$

Notice that the above annotations are *logic statements* consisting of predicates over the topology: $\mathit{type} = C$ evaluates to true if and only if the chemical species of the atom is carbon (here, C is a built-in type); $\mathit{bonded\_atoms}()$ evaluates to the set of all atoms bonded to the one of interest, which can be further filtered by predicates. For instance, $\mathit{bonded\_atoms}(\mathit{type!}{=}C)$ would evaluate to a set of all non-carbon bonded atoms. The proposed DSL will also supports functions on sets, such as $\mathit{count}()$, and common set operations such as union, intersection, difference, etc. Also, the formalism will support the existential and universal quantifiers ($\mathit{exists}()$ and $\mathit{for\_all}()$) to allow evaluating first-order logic statements over sets. Furthermore, the language will include support, through built-in functions, to express common molecular structures that are too verbose to express otherwise, such as, for instance, rings of a particular size.

For convenience, we will allow the annotations to *reference* user-defined types in the language (not just the built-in chemical species, such as C and F). The statement

$$\begin{aligned} C_{791} : \mathit{type} = C \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = F)) = 3 \;\&\; \\ \mathit{count}(\mathit{bonded\_atoms}(\mathit{type} = C_{792})) = 1 \end{aligned} \quad (3)$$

would specify that the carbon atom at the end of the fluorocarbon chain must have a C_{729} type neighbor, which is a carbon in the fluorocarbon backbone. This is clearly a more restrictive atom type usage specification than Eq. 1, allowing it to more specifically express the chemical context.

It is crucial that the annotation syntax we propose be future-proof and **support the evolution of forcefields**. Evolution can mean two things: a.) the forcefield gets extended with support for new chemical species, or b.) already supported species get more specialized. Annotating the atom types for the newly added chemical species can trivially be done incrementally. However, when an existing atom type is specialized, other existing atom-typing rules referencing the specialized one can also be affected. For instance, let us assume that we want to distinguish between C_{791}-type fluorocarbon end groups based on what kind of carbon they are bonded to. Let us assume that to do this, we would remove the C_{791} atom type from the forcefield and replace it with C_{791A} for a chemical context when the carbon neighbor is part of a fluorocarbon backbone (C_{791}), and with C_{791B} if it is not. Since the generic C_{791} type has been removed from the forcefield, all existing annotations that reference it need to be changed to reference C_{791A} and/or C_{791B}.

We want to avoid this, because it would make extending annotated forcefields a laborious and error-prone task, which would hinder the wide-spread acceptance of our proposed formalism, and would jeopardize the success of our efforts. To tackle this issue, the annotation language will allow multiple atom types to be

assigned to a given atom in a molecule, but the atom type annotations will be required to explicitly express the specialization relations (*i.e.*, that atom type C_{792A} *overrides* C_{792}). This way, if multiple atom types are applicable to particular chemical context, it is the most specialized one that will get assigned to the given atom. The following example demonstrates a possible syntax to achieve that, by including the definition of a more generic perfluoroalkane carbon, C_{PFA}.

$$\begin{aligned}
C_{PFA} : & \; type = C \;\&\\
& \; count(bonded_atoms()) = 4 \;\&\\
& \; for_all(bonded_atoms(), type = C \| type = F)\\
C_{791} : & \; type = C_{PFA} \;\&\\
& \; count(bonded_atoms(type = F)) = 3 \;\&\\
& \; count(bonded_atoms(type = C)) = 1\\
& \; @overrides(C_{PFA})\\
C_{791A} : & \; type = C_{791} \;\&\\
& \; count(bonded_atoms(type = C_{792})) = 1\\
& \; @overrides(C_{791})\\
C_{791B} : & \; type = C_{791} \;\&\\
& \; !exists(bonded_atoms(), type = C_{792})\\
& \; @overrides(C_{791})
\end{aligned} \tag{4}$$

Support for overriding annotations is important for two reasons. First, it allows for *incremental* development of forcefields, and second, the overridden, more general atom types can be used as *wildcards* in references. Consider the following annotation of a torsional term defined over four carbon atoms on a fluorocarbon backbone (bonds, and angles are annotated similarly):

$$C13\_C13\_C13\_C13_{PFA} : \quad (C_{PFA}, C_{792}, C_{792}, C_{PFA}) \tag{5}$$

This annotation defines that the torsional term $C13\_C13\_C13\_C13_{PFA}$ is applicable to a series of four carbon atoms such that the first and the last one can be any kind of carbon in a fluorocarbon molecule, but the two middle atoms must be of a more specific type, C_{792}, with exactly two carbon neighbors. This allows us to uniquely identify bonded interactions for atom-types that are of the same "atom class." The DSL can also be extended to provide other parameters that may be required, for example, a molecular length option to unique identify instances when a specific torsion parameter should be used.

Apart from annotating atom types and various bonded interaction types (bonds, angles, torsions) in the forcefield, the proposed language supports global invariants, as well. These invariants can be used to express constraints on the applicability of the forcefield as a whole, and are essential to prevent force field use on unsupported topologies. For example, if the force field cannot handle molecules with ring structures, a global invariant can state that:

$$for\_all(type = C, !in\_ring()). \tag{6}$$

We note that as part of defining the DSL for annotating the forcefield, we will develop various "helper" tools to ensure proper syntax usage.

B. The Atom-Typing Tool

Our approach differs from existing atom-typing tools in a number of ways. First, existing tools are forcefield-specific, while the proposed atom-typer is forcefield-agnostic. Second, the common practice is to hard-code the atom-typing logic into the tool's source code, which makes the code hard to extend as the forcefield evolves. The formalism we propose factors out the atom-typing logic into *declarative* annotations, which, instead of describing *how* atom typing rules are executed, states the *invariants* of the chemical context that must hold true for a given atom type or parameters. The proposed atom-typer will interpret this declarative formalism, and compute the atom type assignments that satisfy the invariants. Third, existing tools often do not cover all chemical contexts supported by the forcefield, but do produce some (potentially bogus) output when run on topologies that include such contexts. Our tool relies on a three-pronged approach to alleviate such problems: 1) formal verification of forcefield annotations that reveal omissions and contradictions, 2) global invariants in the forcefield that map to assertions on the input topologies, warning the user of unsupported molecular features, and 3) a proposed test suite and continuous integration setup that validates the annotated forcefields against known, correctly atom-typed topologies.

An important goal of the proposed effort is to leverage the annotated forcefields to automatically atom-type complex molecular systems. The description of the molecular system must include at least the chemical species of the atoms (element names or atomic numbers), and their connectivity (bonds), both of which could be provided, for instance, from our previously developed mBuild tool [22], [23]. The atom-typer tool we propose to develop will read this topology, along with an annotated forcefield specification, and will produce an output topology with the forcefield specific atom types, as well as bonded interaction types (bonds, angles, torsions, etc.) and associated parameters, which can be used as input to a molecular simulator.

For the atom-typer tool, we will investigate the following implementation approaches:

Naive approach. A naive algorithm first orders the atom type annotations according to the "referenced-by" relationship, that is, if annotation C_{791} references annotation C_{792}, then C_{791} will precede C_{792} in the order. Then, following this ordering, the algorithm evaluates all atom-type annotations for all atoms in the input

topology. This ensures that all referenced atom-typing rules are evaluated before those that reference them. If multiple annotations evaluated to true for a given atom in the input topology ($e.g.$ C_{PFA}, C_{791}, and C_{791A}), the most specialized type (C_{791A}) will be chosen, following the $@overrides$ relations of the annotations. Obviously, this naive approach would fail if no partial ordering of atom type annotations would exist, $i.e.$, if there are annotations that mutually reference each other, or if circular dependencies exist.

Fixpoint-based method. Allowing recursion in annotations would make the language more expressive. But in order to accommodate recursive annotations, we need to tweak the naive approach. First, the fixpoint-based solver evaluates the *primitive* rules, $i.e.$, the ones that do not reference others, for every atom in the topology. If an annotation evaluates to true (on any atom), it *enables* those annotations that reference it. When the enabled annotation is then evaluated, it, in turn, may enable or re-enable others. This continues until no further annotations are enabled (that is, a fixpoint is reached), which is the termination condition of the iteration.

Logic programming. Logic programming languages [24], such as Prolog [25] or Datalog [26], that have been designed for deductive reasoning, are particularly useful in this context. Logic programs consists of *facts*, $i.e.$, things that always hold true ($e.g.$, "Mickey is a mouse", "Pluto is a dog", "Mars is a planet"), and *deduction rules* that can be used to define relations ($e.g.$, "a mouse is an animal", "a dog is an animal"). The program is then run by the user posting queries ($e.g.$, "list all animals"), which the interpreter runtime executes and evaluates (returning Mickey and Pluto in this example). By representing the "bonded-to" relations of the input topology as tuples, and evaluating certain non-logic functions in advance ($e.g.$, enumerating ring structures), we can encode them as *facts* in a logic program ($e.g.$, "atom1 is a carbon", "atom2 is a hydrogen", "atom1 is part of ring1", "atom1 is bonded to atom2", etc.). Similarly, the annotations of the forcefield's atom types will be mapped to *deduction rules*, ($e.g.$, "a carbon atom with 4 bonded hydrogens is a methane carbon"). Then, the logic program can be run by executing queries to list atoms with each atom type ($e.g.$, "list all methane carbons").

An important difference between Prolog and Datalog is that while Prolog is an expressive, general purpuse logic programming language, Datalog is not Turing-complete, and mostly focuses on reasoning about data. Also, Datalog imposes restrictions on the the use of negation and recursion (Prolog does not), however, Datalog queries are always guaranteed to terminate (while in Prolog, there are no such guarantee).

In the proposed effort, we will investigate how forcefield usage specifications and input topologies can be mapped to Datalog, what are the performance implications, including Datalog query execution scales with the topology size and the number of atom types in the forcefield.

Subgraph isomorphism. We expect that many (but not all) of the forcefield annotations will, implicitly, describe the chemical context as a subgraph. While the subgraph isomorphism problem, in the general case, is NP-complete [27], this is not the case with the problem of finding where a particular chemical neighborhood is present in the overall topology [28]. This is due to the fact that chemical topologies belong a special class of graphs (planar graphs) where the maximum number of bonded atoms is well defined ($i.e.$, never more than 4 for covalently bonded systems), and that the chemical neighborhoods in question tend to be relatively small. State of the art graph matching solutions that employ a plethora of optimization techniques and exploit parallelism today can scale up to graphs with 10^9 nodes [29].

We suspect that for certain kinds of annotations, especially those involving ring structures, a subgraph matching based approach would provide better performance than directly mapping such atom type annotations to, for instance, Datalog rules. Therefore, we envision that the final version of the atom-typing tool will borrow from multiple of the above mentioned approaches.

It is important to note that the declarative nature of the proposed annotation language allows us to decouple the forcefield specification (*what* statements must hold true for a correctly atom-typed topology) from execution (*how* it is achieved). That is, the particular execution approach we will eventually choose is orthogonal to the forcefield annotation syntax, so the implementation of the atom-typer tool can evolve independently of the forcefield annotations.

C. Automated Verification, Validation, and Testing

Annotating a forcefield with atom-typing semantics is a major undertaking, and it is inevitable that we make errors on the way. The same is true for a complex piece of software, such as the atom-typer tool. We believe that through proper testing, verification, and validation, we can build quality forcefield annotations and tools that either produce the correct atom-typing results, or fail with adequate warnings or error messages. Our approach is unique in that it provides two very distinct types of testing: 1) verification of the underlying rules to evaluate inconsistencies and 2) validation of the outputted molecular models.

1) Verification

With verification, we want to answer the question: *"Are we annotating the forcefield correctly?"* The fact that our proposed annotation syntax can be mapped to first-order logic statements makes it possible to *"reason about" the annotated forcefield as system of atom-typing rules, even*

without applying them to molecular topologies. That is, in logic programming terms, we can reason about the rules without the facts. We cannot emphasize enough the importance of formal verification here. It may reveal subtle and latent errors in the annotation logic of the forcefield that would be only possible to detect through thorough testing by running the atom-typer on a large swath of different topologies.

Are there rules that are not decidable? Are there any annotations that will never evaluate to true, irrespective of the input chemical topology? Can it ever happen that two rules may hold true at the same time, without one overriding the other? Are there any atom-typing rules that are in contradiction with the global invariants? If the answer to any of the above questions is yes, it indicates an error in the *logical structure* of the forcefield annotations. We propose to verify such properties on the system of forcefield annotations, and will provide a set of development tools that help the forcefield developers pinpoint these inconsistencies early in the forcefield development process. We expect that some of the above violations will be detectable by the Datalog interpreter after the annotations are mapped to Datalog syntax, while others, pertaining to ensuring boolean satisfiability, will require us to integrate a theorem prover such as the widely used Z3 solver from Microsoft Research [30].

Our proposed goal is to prove that the forcefield is *complete* in the logical sense, that is, *if* the forcefield's annotations evaluate to "true" on a correctly atom-typed topology, *then* the atom-typing tool is *guaranteed* to compute the correct atom-typing, given the annotated forcefield, the chemical species of the atoms and their connectivity as inputs. We will strive to reach this goal, even at the cost of reducing the expressiveness of the annotation language.

2) Validation of Forcefield Annotations

With validation, we want to answer the question: **"Did we come up with the correct annotations?"** If we run the atom-typer on a molecular topology and it *does not* produce the expected results, it is vital to know – particularly for an interdisciplinary team of chemical scientists and computer engineers – whether the error is in the forcefield annotations *or* in the atom-typer's source code. With validation, we want to focus on errors in the forcefield annotations, without having to worry about potential software bugs in the atom-typer's implementation.

Notice that if an annotated forcefield passes verification (i.e., it is *complete* in the logical sense), we can validate it in an isolated way, without having to execute the atom-typer tool. It is sufficient to *check* that the forcefield's annotation statements hold true on a set of correctly atom-typed topologies (test cases), which will entail, due to the completeness property, that the atom-typer will be able to compute the correct atom-typing. We propose to develop a validator tool that does just this this: takes an annotated forcefield as an input, and iterates through a large number of correctly atom-typed topologies ensuring that the atom-type annotations and global invariants are never violated. If the validator finds a topology that is in contradiction with the forcefield annotations, it is the chemical scientist who needs to revisit and correct the forcefield, rather than a source code issue.

3) Testing The Atom-Typer Tool

The development and testing of the atom-typer tool is much alike that of any software. Importantly, it can be carried out without chemical domain expertise. This is analogous to developing and testing a database server: the software developers are not concerned with *what* data, in what schemas, will the users store in the database, but rather focus on testing the functionality that implements *how* queries are answered. In the proposed effort, we will use state-of-the-art software testing tools, including unit tests packaged into test suites, as well as coverage tools to quantify the degree to which the source code is tested. The source code of the tools will be stored on GitHub [31], leveraging its collaborative development, version control, and issue tracking facilities.

D. Continuous Integration

Continuous integration (CI) is a software development practice that encourages members of the development team to integrate code into a shared repository several times a day. **On each check-in, the software is automatically built and tested**, allowing teams to detect problems early. Commonly, CI is provided as a cloud service. Developers set up a project with a CI service provider ($e.g.$, Travis CI [32], CodeShip [33], etc.), and it is the CI service that watches the project's source code repository ($e.g.$, GitHub) for changes, attempts to build and test the code in virtual machines in the cloud, and reports build and test results to the developers.

In the proposed effort, we will apply the CI approach to the development process of both the annotated forcefields and the corresponding software tools (atom-typer, verifier, validator, etc.). For the software artifacts, CI will be hosted at Travis CI. For the automated verification and validation (V&V) of annotated forcefields, we propose to develop our own cloud service, based on BuildBot [34], an open-source framework for automating software build, test, and release processes. The proposed **forcefield CI service** (see Fig. 2) will watch the repositories where the annotated forcefield files are hosted. Also, it will integrate with existing online repositories that host correctly atom-typed molecules, which will be automatically downloaded and used as the "ground truth" for forcefield validation. V&V of annotated force field files will be triggered when either a.) changes in the

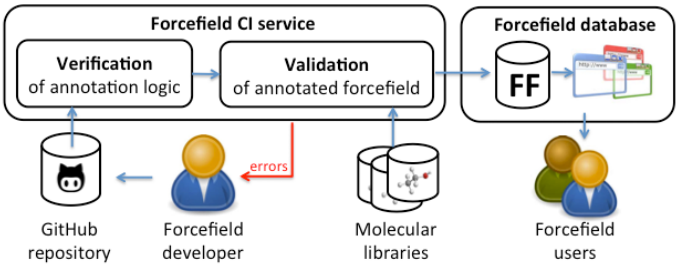

Fig. 2. *Continuous integration workflow of annotated forcefield development.*

repository are detected b.) new correctly-typed molecules are added.

E. Incentivizing Community Involvement

Although there exist some meticulously well maintained and "alive" forcefield repositories, that are, not surprisingly, typically specific to a particular simulator tool or chemical or biomolecular domain, cataloging forcefield development of the past decades into a coherent, unified, searchable online forcefield database would be an enormous undertaking, which is not feasible without community involvement.

The implementation of such an online database would not be technically challenging. Neither would be operating and maintaining such a service. However, convincing the developers of new forcefields to upload their parameter sets, to correctly tag them with how and in what context the values are applicable, to supply test cases, etc., would surely be a futile attempt.

It is well known it is notoriously hard to get outside users to upload their work to a repository. A novel aspect of our **community building approach** is that we will **incentivize** our users to do so by giving them free access to our continuous integration infrastructure: outside users will register their own repositories with the forcefield CI service, which will provide automatic, continuous verification, validation, and testing for their forcefields at no cost.

Whenever a new commit or pull request adds or modifies forcefield related files in the users' registered repositories, the CI service will trigger the verification and validation workflow. Once a forcefield (or a new version of a forcefield) passes all tests, the CI service will publish it (that is, the annotated parameter set, its source URL, and the test results) to our online forcefield repository, assigning to it a permanent URL and a document object identifier (DOI) for referencing and attribution purposes.

IV. Conclusion

Through the development of a new formalism for chemical context and novel atom-typing scheme, our approach unambiguously describes the appropriate usage of forcefield parameters and helps to reduce atom-typing as a source of error during model development. Developing this framework will simplify the rules needed for atom-typing, which is crucial as forcefields continue to grow, specialize, and become more complex. The machine-readable annotations of forcefield usage semantics will enable automation of tedious and error prone tasks, and will enable new application areas, ranging from automated forcefield comparison and cross-validation, to complex simulation workflows integrating multiple forcefields and simulator tools. Through offering our verification, validation, and testing infrastructure as a free-of-charge continuous integration service to the community, we believe that our approach has the potential to foster community involvement through the creation of a online forcefield repository for the annotated forcefields that tagged with DOIs for proper referencing and attribution, the associated software, and documentation of our framework.

Acknowledgments

This work is supported by the National Science Foundation under grant number ACI 1535150.

References

[1] J. I. Siepmann, S. Karaborni, and B. Smit, "Simulating the critical behaviour of complex fluids," *Nature*, vol. 365, pp. 330–332, 1993.

[2] S. Auer and D. Frenkel, "Prediction of absolute crystal-nucleation rate in hard-sphere colloids.," *Nature*, vol. 409, pp. 1020–1023, 2001.

[3] A. Haji-Akbari, M. Engel, A. S. Keys, X. Zheng, R. G. Petschek, P. Palffy-Muhoray, and S. C. Glotzer, "Disordered, quasicrystalline and crystalline phases of densely packed tetrahedra.," *Nature*, vol. 462, pp. 773–777, 2009.

[4] G. Feng and P. T. Cummings, "Supercapacitor capacitance exhibits oscillatory behavior as a function of nanopore size," *Journal of Physical Chemistry Letters*, vol. 2, pp. 2859–2864, 2011.

[5] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weinerl, "A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins," *Journal of the American Chemical Society*, vol. 106, pp. 765–784, 1984.

[6] A. D. MacKerell, N. Banavali, and N. Foloppe, "Development and current status of the CHARMM force field for nucleic acids," *Biopolymers*, vol. 56, pp. 257–265, 2000.

[7] W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives, "Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids," *Journal of the American Chemical Society*, vol. 118, pp. 11225–11236, Jan. 1996.

[8] J. J. Potoff and J. I. Siepmann, "Vapor-liquid equilibria of mixtures containing alkanes, carbon dioxide, and nitrogen," *AIChE Journal*, vol. 47, pp. 1676–1682, 2001.

[9] H. Sun, "COMPASS: An ab Initio Force-Field Optimized for Condensed-Phase Applications s Overview with Details on Alkane and Benzene Compounds," *Journal of Physical Chemistry*, vol. 5647, pp. 7338–7364, 1998.

[10] C. Oostenbrink, A. Villa, A. E. Mark, and W. F. Van Gunsteren, "A biomolecular force field based on the free enthalpy of hydration and solvation: The GROMOS force-field parameter sets 53A5 and 53A6," *Journal of Computational Chemistry*, vol. 25, pp. 1656–1676, 2004.

[11] R. C. Rizzo, J. Tirado-Rives, and W. L. Jorgensen, "Estimation of binding affinities for HEPT and nevirapine analogues with HIV-1 reverse transcriptase via Monte Carlo simulations," *Journal of Medicinal Chemistry*, vol. 44, pp. 145–154, 2001.

[12] M. G. Martin and J. I. Siepmann, "Transferable Potentials for Phase Equilibria. 1. United-Atom Description of n -Alkanes," *The Journal of Physical Chemistry B*, vol. 102, no. 97, pp. 2569–2577, 1998.

[13] E. K. Watkins and W. L. Jorgensen, "Perfluoroalkanes: Conformational Analysis and Liquid-State Properties from ab Initio and Monte Carlo Calculations," *The Journal of Physical Chemistry A*, vol. 105, pp. 4118–4125, 2001.

[14] J. W. Ponder, "TINKER Molecular Modeling Software."

[15] B. L. Bush and R. P. Sheridan, "PATTY: A Programmable Atom Typer and Language for Automatic Classification of Atoms in Molecular Databases," *Journal of Chemical Information and Computer Sciences*, vol. 33, pp. 756–762, 1993.

[16] A. W. Schüttelkopf and D. M. F. Van Aalten, "PRODRG: A tool for high-throughput crystallography of protein-ligand complexes," *Acta Crystallographica Section D: Biological Crystallography*, vol. 60, pp. 1355–1363, 2004.

[17] A. A. S. T. Ribeiro, B. A. C. Horta, and R. B. De Alencastro, "MKTOP: A program for automatic construction of molecular topologies," *Journal of the Brazilian Chemical Society*, vol. 19, no. 7, pp. 1433–1435, 2008.

[18] A. K. Malde, L. Zuo, M. Breeze, M. Stroet, D. Poger, P. C. Nair, C. Oostenbrink, and A. E. Mark, "An Automated force field Topology Builder (ATB) and repository: Version 1.0," *Journal of Chemical Theory and Computation*, vol. 7, pp. 4026–4037, 2011.

[19] K. Vanommeslaeghe and a. D. MacKerell, "Automation of the CHARMM General Force Field (CGenFF) I: bond perception and atom typing.," *Journal of Chemical Information and Modeling*, vol. 52, pp. 3144–54, Dec. 2012.

[20] J. D. Yesselman, D. J. Price, J. L. Knight, and C. L. Brooks, "MATCH: an atom-typing toolset for molecular mechanics force fields.," *Journal of Computational Chemistry*, vol. 33, pp. 189–202, Jan. 2012.

[21] J. Wang, W. Wang, P. A. Kollman, and D. A. Case, "Automatic atom type and bond type perception in molecular mechanical calculations.," *Journal of Molecular Graphics & Modelling*, vol. 25, pp. 247–60, Oct. 2006.

[22] C. Klein, J. Sallai, C. R. Iacovella, C. McCabe, and P. T. Cummings, "Mbuild: A Hierarchical, Component Based Molecule Builder," in *Poster Presentation: Computational Molecular Science and Engineering Forum, American Institute of Chemical Engineers Annual Meeting, Atlanta, GA, November 18*, 2014.

[23] J. Sallai, G. Varga, S. Toth, C. Iacovella, C. Klein, C. McCabe, A. Ledeczi, and P. T. Cummings, "Web- and Cloud-based Software Infrastructure for Materials Design," *Procedia Computer Science*, vol. 29, pp. 2034–2044, 2014.

[24] J. W. Lloyd, "Foundations of logic programming; (2nd extended ed.)," Jan. 1987.

[25] W. Clocksin and C. S. Mellish, *Programming in PROLOG*. Springer Science & Business Media, 2003.

[26] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, pp. 146–166, Mar. 1989.

[27] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), pp. 151–158, ACM, 1971.

[28] D. Eppstein, "Subgraph isomorphism in planar graphs and related problems," in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, (Philadelphia, PA, USA), pp. 632–640, Society for Industrial and Applied Mathematics, 1995.

[29] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proceedings of the VLDB Endowment*, vol. 5, pp. 788–799, May 2012.

[30] L. De Moura and N. Bjø rner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer Berlin Heidelberg, 2008.

[31] Http://github.com, "GitHub: powerful collaboration, code review, and code management for open source and private projects."

[32] Http://travis-ci.org, "Travis CI: A hosted continuous integration service."

[33] Http://codeship.com, "Codeship: A free hosted Continuous Delivery Service."

[34] Http://buildbot.net, "Buildbot: An open-source framework for automating software build, test, and release processes.."