Formalizing Atom-typing and the Dissemination of Force Fields with Foyer

Christoph Klein[a,b], Andrew Z. Summers[a,b], Matthew W. Thompson[a,b], Justin Gilmer[c,b], Clare McCabe[a,d,b], Peter T. Cummings[a,b], Janos Sallai[e], Christopher R. Iacovella[a,b]

[a] *Department of Chemical and Biomolecular Engineering, Vanderbilt University, Nashville, Tennessee 37235, United States*
[b] *Vanderbilt Multiscale Modeling and Simulation (MuMS) Center, Vanderbilt University, Nashville, Tennessee 37235, USA*
[c] *Interdisciplinary Materials Science Program, Vanderbilt University, Nashville, Tennessee 37235, United States*
[d] *Department of Chemistry, Vanderbilt University, Nashville, Tennessee 37235, United States*
[e] *Institute for Software Integrated Systems, Vanderbilt University, Nashville, Tennessee 37235, United States*

Abstract

A key component to enhancing reproducibility in the molecular simulation community is reducing ambiguity in the parameterization of molecular models used to perform a study. Ambiguity in molecular models often stems from inadequate usage documentation of molecular force fields and the fact that force fields are not typically disseminated in a format that is directly usable by software. Specifically, the lack of a generally applicable scheme for the annotation of the rules of a particular force field and a general purpose tool for performing automated parameterization (i.e., atom-typing) based on these rules, may lead to errors in model parameterization that are not easily identified. Here, we present `Foyer`, an open-source Python tool that enables users to define and apply force field atom-typing rules in a format that is both human- and machine-readable and provides a framework for force field dissemination, thus eliminating ambiguity in atom-typing and improving reproducibility. Foyer defines force fields in an XML format, where SMARTS strings are used to define the chemical context of a particular atom type and "overrides" are used to set rule precedence, rather than a rigid hierarchical scheme. Herein we describe the underlying methodology and force field annotation scheme of the Foyer software, demonstrate its application in several use-cases, and discuss specific aspects of the Foyer approach that are designed to improve reproducibility.

Keywords: Molecular simulation; Force fields; Reproducibility; Open-source software

1. Introduction

Considerable efforts have been undertaken by many research groups to develop accurate classical force fields for a wide range of systems.[1, 2, 3, 4, 5, 6, 7] Force fields are often expressed as a set of analytical functions with adjustable fitting parameters that describe the interactions between constituents of a system (often discrete atoms but, more generally, interaction sites). Classical force fields are typically able to achieve high accuracy by creating sets of highly specific fitting parameters (i.e., atom types), in which each atom type describes an interaction site within a different chemical context. The chemical context is typically defined by the bonded environment of an interaction site (e.g., the number of bonds and the identity of the bonded neighbors) and may also consider, among other factors, the bonded environment of the neighbors, and/or the specific molecule/structure within which the interaction site is included. Consequently, a force field may include tens or even hundreds of different atom types for a given element. For example, there are 347 atom types that apply to carbon in the OPLS force field parameter set distributed with GROMACS[8] where each atom type corresponds to a carbon atom within a different chemical context. Thus, while force field development efforts have reduced – or in some cases completely eliminated – the need for researchers to generate their own fitting parameters, determining *which* parameters (i.e., atom types) to use can still be a tedious and error prone task. Failure to properly identify the chemical context and atom type of an interaction site will inevitably lead to the unfaithful implementation of the force field and thus inconsistent results.

Part of the difficulty in performing atom-typing (i.e., determining which atom type applies to an interaction site) stems from the fact that there is not yet a standardized way of unambiguously expressing chemical context and parameter usage. As such, journal articles that report novel force field parameters may vary significantly in terms of their clarity. In many cases, parameters are reported in a tabular format with minimal annotations and few (if any) examples of how to appropriately assign the atom types. Since this approach does not allow for automated evaluation, different users of the force field may apply the atom types differently based on their own interpretation of the information provided. Journal articles that utilize existing force fields often do not report the specific fitting parameters and typically do not specify which atom types were chosen for the interaction sites, instead providing citation(s) to the source of the force field parameters. Even if the source of the parameters is clearly and fully specified, usage may again depend on the clarity of the original source(s) and the interpretation by the end user, hampering reproducibility. Force field parameter files that aggregate a large number of atom types (often thousands) into a single source suffer from some of the same issues. Often, they include only brief, unstructured – and sometimes ambiguous – annotations as to parameter usage, and may, or may not, provide clear citations of the original source of the parameters.

To apply force fields, users can perform atom-typing manually (e.g., creation of an atom-typed template of a molecule or unit cell), although manual

assignment of parameters becomes tedious and error prone for large molecules and/or complex systems, and manual manipulation of files is not considered a good practice in terms of reproducibility[9]. Furthermore, manual assignment of parameters does not lend itself well to workflows such as screening[10], where thousands of unique systems with different chemical constituents and structures may need to be atom-typed in an automated fashion. To avoid manual assignment, end-users often develop in-house software to apply force fields in an automated fashion; however, such software is not typically made freely available to the community and may be very limited in scope and applicability. Without access to the same software, the exact atom-typing cannot be reproduced by others and if the source code is not made freely available, the logic used to interpret and apply the force field is unknown and if there are errors in the software/logic, these cannot be identified. There exist a number of freely available atom-typing tools that read in a force field parameter file and execute a set of rules to apply the force field to a chemical topology [11, 12, 13, 14, 15, 16, 17], enabling the exact atom-typing process to be reproduced. However, many of these atom-typing tools are either closed-source[12, 15], simulation engine-specific[14, 17], and/or force field-specific[13, 15, 16], which limits their utility. Furthermore, these tools almost universally rely on a rigid hierarchy of rules[11], where rules must be called in a precise order such that more general atom types are only chosen when more specialized matches do not exist (*i.e.*, the order of rules defines the precedence). Maintaining, let alone constructing, these hierarchies is challenging, especially for a large number of atom types. In order to add a new atom type or correct an error in hierarchical schemes, 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 new atom types, or that such attempts to extend the rules result in incorrect atom-typing for other systems. For many tools, this approach is further complicated by the encoding of the hierarchy as a set of heavily nested if/else statements within the source code of the software. These heavily nested if/else hierarchies may be difficult to validate and debug, and any changes or extensions to the rules, no matter how trivial, require modification of the source code itself. Reproducibility issues may therefore arise if users make modifications or extensions to a piece of software and these changes are not made freely available to the larger community and/or incorporated into the main software distribution. This also creates a situation where there are effectively two sets of rules since there is no guarantee that the logic statements in the source code (i.e., the machine readable rules) agree with the textual annotations in the force field parameter file (i.e., the human readable rules).

Several atom-typing tools have been developed that remove the need to encode atom type usage rules within the source code itself. A unifying feature of these tools is the use of the simplified molecular-input line-entry system (SMILES) [18] language, or variants thereof, for describing chemical structures associated with an atom type. For example, Yesselman, et al. [17] developed an atom-typing toolset for the CHARMM simulation engine, termed MATCH, that

relies on assigning parameters by representing a molecule of interest as a graph and performing subgraph matching against a library of fragments with known parameters. These fragments are represented as "super smiles", an extension of the SMILES language. By using super smiles and storing these fragments in text files separate from the software, chemical context is expressed without the need to define a rigid if/else hierarchy within the software and thus new atom types and rules (i.e., fragments encoded as super smiles) can be added without modifying the code used to evaluate them. In recent work, Mobley and coworkers[19] have developed an approach to defining force fields, termed SMIRNOFF, that effectively eliminates explicit atom types altogether, instead using SMIRKS (another language related SMILES [20]) to identify chemical fragments that are associated with a set of force field parameters. Similar to Yesselman, et al. [17], application of the force field relies on representing the system as a graph and rules as subgraphs. In other work, the Enhanced Monte Carlo (EMC) software developed by in't Veld [21] encodes chemical context of an atom type using SMILES. In all cases, the use of the SMILES-based approaches not only removes the need to encode usage within the source code, but associates parameters with a human and machine readable definition of their chemical context, although, these approaches all still require rules be specified in a particular order to enable correct atom-typing.

In this work, we present `Foyer`, a Python library for performing atom-typing based upon first-order logic over graph structures, designed to address many of the aforementioned issues, with a particular emphasis on reproducibility and the dissemination of force fields to the community. `Foyer` relies upon a force-field-agnostic formalism to express atom-typing and parameterization rules in a way that is expressive enough for human consumption while simultaneously being machine readable, allowing a single, unambiguous format to be constructed for both dissemination and use by software. This logic is implemented via SMARTS[20] to encode chemical context and "`overrides`" statements to define rule precedence. SMARTS extends the SMILES language to support substructure definitions and allows expression of greater chemical detail and logic operations within the chemical patterns. In `Foyer`, SMARTS has been extended such that it allows user-defined "elements" (not in the periodic table) to be leveraged within the chemical context definitions, thus enabling both atomistic and non-atomistic force fields to be used. By using SMARTS to define chemical context, atom type definitions do not appear in the source code, and thus force fields can be created and evolved without modification to the code used to evaluate them. Rule precedence is explicitly defined by the aforementioned `overrides` statements, thus atom-typing rules can appear in any order in the file and include recursive definitions to other atom types, eliminating physical placement in the file as a source of error and providing increased flexibility. Since this iterative approach used by `Foyer` evaluates all rules, automated evaluation can be used to help ensure that `Foyer` force field definitions (1) encompass all atom types in the force field and (2) are sufficiently descriptive without conflicting rules, both necessary conditions for publishing force fields in a way that is unambiguous and reproducible. The `Foyer` software provides routines that create

syntactically correct input files for a variety of common simulation engines and is designed to take, as input, chemical topologies from several of other community developed tools (e.g., `ParmEd`[22], `OpenMM`[23, 24], and `mBuild`[25, 26, 27]). The manuscript is organized as follows. Sec. 2 introduces the use of SMARTS and `overrides` for encoding force field usage. This section also presents the XML file format used by `Foyer`, which builds upon the OpenMM force field file format[23, 24] extended to support the definition of the associated SMARTS, and `overrides` statements, textual descriptions of the parameters, and digital objective identifiers (DOIs) for the source of each atom type. Sec. 3 provides an overview of the `Foyer` software used to evaluate the SMARTS and `overrides` statements encoded in the XML file format, including the iterative process – and its optimizations – used for determining atom types. This section also discusses force field validation and verification within `Foyer`. Sec. 4 provides examples of the use of the `Foyer` software to perform atom-typing of several different chemical systems. Sec. 5 discusses best practices for the use of `Foyer` and force field annotation scheme in terms of reproducibilit, focusing on the use of version control and related open-source software development tools to enable the creation and evolution of force fields in a transparent, testable manner. Sec. 6 provides concluding remarks.

2. Defining chemical context and rule precedence

2.1. XML File Format

`Foyer` utilizes the OpenMM force field XML format[24] to encode parameters, where this format is extended to allow for the definitions of chemical context and rule precedence (discussed below). To briefly summarize the OpenMM file format, atom types and forces are encoded as XML tags with various attributes defining the types of elements that they apply to (by name only), as well as the associated parameters for that interaction (e.g., the equilibrium bond length and spring constant for a harmonic bond). Listing 1 provides an example of encoding the OPLS force field parameters for linear alkanes in the OpenMM XML format (note, this Listing does not include our extensions). As shown in Listing 1, the XML format provides clear descriptions of each of the parameters/properties defined in the file (e.g., `element="C"` indicates the entry is defining a carbon atom), along with additional tags that provide unambiguous descriptions of the types of interactions being used (e.g., the `<HarmonicBondForce>` tag is used to define the use of a harmonic force to define bonds). As such, this file format includes a wealth of metadata that is both human and machine readable. For more detailed information, we refer the reader to the OpenMM manual where this force field file format is extensively documented [24].

The flexible nature of XML allows it to be readily extended via the addition of new tags/attributes without fundamentally changing the original format, as new tags/attributes can simply be ignored by software that does not require them. As shown in Table 1, and discussed in detail later, four new attributes have been added to the atom type entries in the existing OpenMM XML file

Table 1: Extensions to the atom type definitions in the OpenMM XML format.

Attribute	Description	Example
`def`	Chemical context of an atom type via SMARTS	`[C;X4](H)(H)(H)C`
`desc`	Textual description of the atom type	`Alkane CH3`
`doi`	Digital object identifier to the atom type source	`10.1021/ja9621760`
`overrides`	Atom type(s) the current rule is given precedence over	`opls_136`

format to enable the functionality needed to encode usage rules in `Foyer`: `def`, `desc`, `doi`, and `overrides`. The use of XML additionally allows sanity checks to be performed by using XML schemas to ensure the expected attributes have been provided in the file.

Listing 1: OpenMM formatted XML file for linear alkanes using the OPLS force field.

```
<ForceField>
  <AtomTypes>
    <Type name="opls_135" class="CT" element="C" mass="12.01100"/>
    <Type name="opls_136" class="CT" element="C" mass="12.01100"/>
    <Type name="opls_140" class="HC" element="H" mass="1.00800"/>
  </AtomTypes>
  <HarmonicBondForce>
    <Bond class1="CT" class2="CT" length="0.1529" k="224262.4"/>
    <Bond class1="CT" class2="HC" length="0.1090" k="284512.0"/>
  </HarmonicBondForce>
  <HarmonicAngleForce>
    <Angle class1="CT" class2="CT" class3="CT" angle="1.966986067"\\
        k="488.273"/>
    <Angle class1="CT" class2="CT" class3="HC" angle="1.932079482"\\
        k="313.800"/>
    <Angle class1="HC" class2="CT" class3="HC" angle="1.881464934"\\
        k="276.144"/>
  </HarmonicAngleForce>
  <RBTorsionForce>
    <Proper class1="CT" class2="CT" class3="CT" class4="CT" c0="2.9288"\\
        c1="-1.4644" c2="0.2092" c3="-1.6736" c4="0.0" c5="0.0"/>
    <Proper class1="CT" class2="CT" class3="CT" class4="HC" c0="0.6276"\\
        c1="1.8828" c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
    <Proper class1="HC" class2="CT" class3="CT" class4="HC" c0="0.6276"\\
        c1="1.8828" c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
  </RBTorsionForce>
  <NonbondedForce coulomb14scale="0.5" lj14scale="0.5">
    <Atom type="opls_135" charge="-0.18" sigma="0.35"\\
        epsilon="0.276144"/>
    <Atom type="opls_136" charge="-0.12" sigma="0.35"\\
        epsilon="0.276144"/>
    <Atom type="opls_140" charge="0.06" sigma="0.25"\\
        epsilon="0.12552"/>
  </NonbondedForce>
</ForceField>
```

2.2. Using SMARTS to define chemical context

The chemical context of an interaction site is typically defined by its bonded environment, notably the number of bonds and the identities of bonded neigh-

Table 2: 2D depictions of molecular fragments referred to in the text

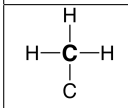
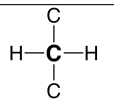
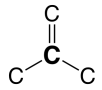
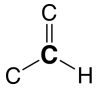
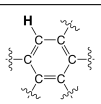
Alkane			Alkene			Benzene	
C, CH_3	**C, CH_2**	**H**	**C, (R2-C=)**	**C, (RH-C=)**	**H**	**C**	**H**
							
opls_135	opls_136	opls_140	opls_141	opls_142	opls_144	opls_145	opls_146

Table 3: Currently implemented SMARTS atomic primitives[a]

Symbol	Symbol name	Atomic property requirements	Default
*	wildcard	any atom	(no default)
A	aliphatic	aliphatic	(no default)
r<n>	ring size	in smallest SSSR[b] ring of size <n>	any ring atom
X<n>	connectivity	<n>total connections	exactly one
#n	atomic number	atomic number <n>	(no default)

[a]This table has been adapted from the Daylight SMARTS website.
[b]Smallest set of smallest rings.

Table 4: Extensions to SMARTS atomic primitives

Symbol	Symbol name	Atomic property requirements	Default
_A	non-element	non-atomistic element	(no default)
%<type>	atomtype	of atomtype <type>	(no default)

Table 5: SMARTS Logical Operators[a]

Symbol	Expression	Meaning
exclamation	!e1	not e1
ampersand	e1&e2	e1 and e2 (high precedence)
comma	e1,e2	e1 or e2
semicolon	e1;e2	e1 and e2 (low precedence)

[a]This table has been adapted from the Daylight SMARTS website.

boring interaction sites, but may also include longer range information, such as the bonded environment of neighbors. To encode this information, `Foyer` utilizes SMARTS[20], a language for defining chemical patterns. SMARTS is an extension of the more commonly used SMILES[18] notation, providing additional tokens that enable users to express greater chemical detail and logic operations. SMARTS notation is expressed as strings that simultaneously include arbitrary chemical complexity but are concise and clear enough for human consumption, in addition to being machine readable. As an example, consider defining the chemical context of OPLS-AA atom types for carbon and hydrogen atoms in a linear alkane, as shown in Listing 2 (note, only the `<AtomTypes>` section of the file is shown, as this is the only section that differs from Listing 1). The reader is referred to Table 2 for a visual depiction of these atom types. To encode the chemical context, the `def` attribute is added to the OpenMM XML format to encode the corresponding SMARTS string. Here, the atom type that specifies the terminal methyl group, `opls_135` ("-CH3") can be expressed as `[C;X4](C)(H)(H)H` in the SMARTS notation. In this SMARTS notation, `[C;X4]` indicates that the element of interest – always the first token in the SMARTS string – is a carbon atom (i.e., `C`) and this carbon atom has 4 total bonds (i.e., `;X4`, where `;` indicates the logical operator `AND`). The identities of the 4 bonded neighbors are 1 carbon atom and 3 hydrogen atoms, expressed as `(C)(H)(H)H`. Similarly, the `opls_136` atom type, which describes a methylene group in an alkane, is expressed in SMARTS notation as `[C;X4](C)(C)(H)H`. Here, the only change from the `opls_135` definition lies in the identity of the 4 bonded neighbors (2 carbon atoms and 2 hydrogen atoms). Increased chemical complexity can be described by adding details about each of neighboring interaction sites within SMARTS. For example, the `opls_140` atom type, which describes a generic alkane hydrogen, is defined as `H[C;X4]` - a hydrogen atom bonded to a carbon atom with 4 bonds. Multiple valid SMARTS can be defined for each atom type, where, e.g., `opls_140` could be defined simply as `def="H"` since there is only a single hydrogen atom type defined in Listing 2. However, such a definition would not necessarily provide a user of the force field with a clear understanding of the chemical context for which this atom type applies, and may limit future evolution of the force field. Our extension of the XML file format also includes the `desc` attribute (e.g., shown in Listing 2) that allows for unstructured comments to be provided for each entry if desired.

We note that the parser in the `Foyer` libraries does not currently support the full SMARTS language, instead providing support for the subset that was found to be relevant to the definition of chemical context for atom types. Table 3 lists the currently supported primitives, Table 4 shows our extensions to the language, and Table 5 outlines the logical operators supported.

Listing 2: Atom type definitions for carbon and hydrogen atoms in a linear alkane using the OPLS force field. Note, only the section that applies to atom types is shown for clarity.

```
<ForceField>
  <AtomTypes>
    <Type name="opls_135" class="CT" element="C" mass="12.01100"\\
          def="[C;X4](C)(H)(H)H" desc="alkane CH3"/>
    <Type name="opls_136" class="CT" element="C" mass="12.01100"\\
          def="[C;X4](C)(C)(H)H" desc="alkane CH2"/>
    <Type name="opls_140" class="HC" element="H" mass="1.00800"\\
          def="H[C;X4]" desc="alkane H"/>
  </AtomTypes>
</ForceField>
```

2.3. Establishing rule precedence

Rule precedence must be established when multiple atom type definitions can apply to a given interaction site. In typical hierarchical schemes, this is determined implicitly by the order in which rules are evaluated; in general, more specific rules are evaluated first and when a match is found, the code stops evaluating rules altogether. While this approach works, it becomes more challenging to maintain the correct ordering of rules as the number of atom types grows and as chemistries become more complex and specific. Users may find it difficult, if not impossible, to make even small additions to a larger force field without breaking existing behavior. `Foyer` allows rule precedence to be explicitly stated via the use of the `overrides` attribute added to the XML file format. This allows atom type usage rules to be encoded in any order within the file, eliminating incorrectly placed rule order as a source of error. `Foyer` iteratively evaluates all rules on all interaction sites in the system, maintaining for each interaction site a "whitelist" consisting of rules that evaluate to `True` and a "blacklist" consisting of rules that have been superseded by another rule (i.e., those that appear in the `overrides` attribute). The set difference between the white- and blacklists of an interaction site yields the correct atom type if the force field is implemented correctly (incorrect/incomplete definition of force fields is discussed later). As an example of a system where `overrides` need to be defined, consider describing alkenes and benzene in a single force field file, as shown in Listing 3 (note, only the `<AtomTypes>` section of the force field file is shown). The reader is again referred to Table 2 for visual depictions of the relevant atom types.

Listing 3: Atom type definitions for alkenes and benzene using the OPLS force field highlighting the overrides syntax and mechanism for referencing other atom types. Note, only the section that applies to atom types is shown for clarity.

```
<ForceField>
  <AtomTypes>
    <Type name="opls_141" class="CM" element="C" mass="12.01100"\\
        def="[C;X3](C)(C)C" desc="alkene C (R2-C=)"/>
    <Type name="opls_142" class="CM" element="C" mass="12.01100"\\
        def="[C;X3](C)(C)H" desc="alkene C (RH-C=)"/>
    <Type name="opls_144" class="HC" element="H" mass="1.00800"\\
        def="[H][C;X3]" desc="alkene H"/>
    <Type name="opls_145" class="CA" element="C" mass="12.01100"\\
        def="[C;X3;r6]1[C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6]1"\\
        overrides="opls_142"/>
    <Type name="opls_146" class="HA" element="H" mass="1.00800"\\
        def="[H][C;%opls_145]" overrides="opls_144" desc="benzene H"/>
  </AtomTypes>
</ForceField>
```

When atom-typing a benzene molecule, the carbon atoms in the ring will match the SMARTS patterns for both `opls_142` (an alkene carbon) and `opls_145` (a benzene carbon). Without the `overrides` attribute, `Foyer` will find that multiple atom types apply to each carbon atom. Providing the `overrides` indicates that if the `opls_145` pattern matches, it will supersede `opls_142`. Thus, the difference between the whitelist (containing `opls_142` and `opls_145`) and blacklist (containing only `opls_142`) would be `opls_145`.

Note that multiple atom types can be listed in a single `overrides` attribute. The approach taken here also allows atom types to inherit `overrides` from the atom types they override. For example, consider a case in which atom types 1, 2 and 3 each evaluate to `True` for an interaction site. If atom type 3 overrides atom type 2 (i.e., adds atom type 2 to the blacklist) and atom type 2 overrides atom type 1 (i.e., adds atom type 1 to the blacklist), then atom type 3 will implicitly override atom type 1. Additionally, in `Foyer`, the SMARTS grammar has been modified such that specific atom type names can also be included within the definition (see Table 4). For example, `opls_146`, the hydrogen atom attached to carbon atoms in a benzene ring, has the SMARTS definition `[H][C;%opls_145]`, as shown in Listing 3. This states that the interaction site of interest is a hydrogen atom (`H`) and is bonded to a carbon atom that has atom type `opls_145` (`C;%opls_145`). Because `Foyer` evaluates rules iteratively for each interaction site, such recursive definitions can be utilized without the need to explicitly define atom types in a chemical topology input file. For example, in this case, when `Foyer` identifies the interaction site of a carbon atom to be `opls_145`, the next iteration to evaluate the hydrogen atom will find that `opls_146` now evaluates to `True`. Similar to how an `overrides` statement clearly defines precedence, this recursive definition provides a clear way to identify chemical context and the relationship between different atom types for highly specific parameters. We note, that one could also replace the recursive reference to `opls_145` with its SMARTS string, although, in this case, it would result in a more complex, less human readable definition.

Because the logic used to define chemical context is separated from the source code used to evaluate it, one can construct a force field file that contains only the relevant subset of atom types need for a given application area. Using the above example of benzene and alkenes, if a system only contained benzene molecules, one could avoid specifying the `overrides` attributes altogether by simply creating a force field file containing only atom types relevant to benzene and eliminating those associated with alkenes. In many cases, considering smaller subsets is beneficial as the amount of effort required to differentiate and set rule precedence between atom types is reduced. Additionally, using smaller files will reduce the likelihood of errors related to defining chemical context and rule precedence, reduce the number of test molecules with known atom types required to fully validate the rules, and increase the readability of the force field files by limiting the number of entries.

2.4. Extension of SMARTS for non-atomistic systems

`Foyer` is able to atom-type systems in which an interaction site does not represent a single atom with a standard element, but instead may represent a group of atoms (relevant to united-atom and coarse-grained force fields) or a generic site (relevant to simplified models). Standard SMARTS notation does not support non-atomic species due to its reliance on the presence of an element specification for each interaction site. To circumvent this limitation, the `Foyer` SMARTS parser allows users to define custom "elements" by prefixing their string representation with an underscore (see Table 4). For example, `_CCC` could represent a coarse-grained interaction site intended to model three carbon atoms. In its current implementation, `Foyer` makes a first pass through force field files to detect any custom element definitions. These are injected into the grammar that parses SMARTS strings and are given priority over standard elements. This allows non-atomistic and atomistic atom types to be used either separately or together.

In practice, united-atom and coarse-grained force fields can be defined in an almost identical fashion to all-atom force fields, where the only difference is that "elements" are user-defined strings prepended with an underscore. As an example, consider an alkane modeled with the united-atom TraPPE force field[28, 29]. An interaction site in this force field represents both carbon and the hydrogen atoms bonded to it. Thus, this force field contains two distinct atom types, one that represents CH_3 (`_CH3`) and one that represents CH_2 (`_CH2`). These can be encoded as shown in Listing 4. Focusing on atom type `CH3_sp3`, usage is encoded with the definition `[_CH3;X1][_CH3,_CH2]` which states that the base "element" is `_CH3` with one bond (i.e., `;X1`) to either a `_CH3` or a `_CH2` group. In SMARTS, a comma indicates an "OR" logic statement and a semicolon is used to denote an "AND" logical statement (see Table 5 for a complete list of SMARTS logical operators). In this example, `[_CH3;X1]` states the element must be `_CH3` "AND" have only a single bond. Atom-type `_CH2_sp3`, which represents a "middle" alkane carbon and its 2 associated hydrogen atoms, is defined similarly as `[_CH2;X2]([_CH3,_CH2])[_CH3,_CH2]`. Here, the base "element" is a `_CH2` with two bonds (i.e., `;X2`), each of which may be either "element"

`_CH3` or `_CH2`. Note, the interaction sites defined in the input chemical topology would need to follow the same naming convention as the force field file, namely they would need to be labeled as `_CH3` and `_CH2`.

Listing 4: Atom:type definitions for united atom alkanes using TraPPE.

```
<ForceField>
 <AtomTypes>
    <Type name="CH3_sp3" class="CH3" element="_CH3" mass="15.03500"\\
        def="[_CH3;X1][_CH3,_CH2]" desc="Alkane CH3, united atom"/>
  <Type name="CH2_sp3" class="CH2" element="_CH2" mass="14.02700"\\
        def="[_CH2;X2]([_CH3,_CH2])[_CH3,_CH2]"\\
        desc="Alkane CH2, united atom"/>
 </AtomTypes>
</ForceField
```

2.5. Determining bonded parameters

Once a chemical topology is atom-typed, bonded interactions can be determined by simply searching for the matching pairs, triplets, and quartets (bonds, angles, and torsions, respectively). In many force fields, the bonded parameters are not as specific as the non-bonded interactions, and thus are not defined directly based on atom types. Thus, rather than atom types, a more general `class` identifier (sometimes referred to as the "bond family") is used to identify these interactions. In Listing 5, both `opls_136` and `opls_962` are part of the same `class` "CT". Thus a bond between `opls_136-opls_962` would have the same parameters (defined as `class1="CT" class2="CT"` in Listing 5) as a bond between `opls_136-opls_136` (also defined as `class1="CT" class2="CT"`). However, this general approach breaks down for certain chemical topologies. For example, while the atom types for carbon atoms in alkanes[3] and perfluoralkanes[30] are both of `class` "CT" and share the same bond and angle parameters for carbon atoms, they differ in terms of torsional parameters. In order to handle this conflict, many codes require users to comment out the more general set of parameters or include statements within the code that accomplish the same task. However, in this approach, one would not be able to atom-type a system composed of a mixture of alkane and perfluoroalkane molecules, since only one set of parameters can be included simultaneously. Note that while one could define a new `class` to differentiate between alkanes and perfluoroalkanes, this wouldresult in a force field file with many duplicate parameters sets that simply have different labels.

The OpenMM format allows bonded parameters to be defined using the `type` attribute in place of the `class` attribute, where `type` refers directly to the `name` attribute that stores the atom type, allowing for bonded interactions to be defined with increased specificity. Additionally, mixed use of `type` and `class` in the definition of these bonded interactions is supported. Referring to Listing 5, to provide the necessary distinction between torsional parameters for perfluoroalkanes and alkanes, one could define perfluoroalkane torsions using type attributes (i.e., `type1="opls_962" type2="opls_962" type3="opls_962"`

type4="opls_962", where opls_962 is defined in the <AtomTypes> XML section), and alkane torsions with the more general quartet for alkanes of class1="CT" class2="CT" class3="CT" class4="CT" that uses class attributes. However, when iterating through bonded parameter definitions, OpenMM assigns parameters based on the first match found. In the example described above, perfluoroalkane torsional parameters would therefore need to be defined before alkane parameters in the torsional section, and thus the ordering shown in Listing 5 would result in the incorrect assignment of torsional parameters for perfluoroalkanes. Several approaches can be taken to address this. overrides statements could be used to set rule precedence for bonded topologies and thus eliminate the need to specify order in the file, however additional modification to the force field file format would be required because bonded parameters do not have a "name" attribute like atom types. In the approach taken by SMIRNOFF [19], bonded parameters are defined directly using their chemical context (i.e., via SMIRKS), eliminating this issue altogether; however, taking a similar approach would result in the duplication of many parameters in the same was as defining a new class attribute. A more simple approach taken by Foyer is to perform a preprocessing step on the bonded parameters. This step orders bonded parameters such that the most specific cases are sorted to the top of the list to set precedence. This accomplished by assigning a weight to each entry proportional to the number of type attributes included (as these are the most specific). For example, a torsion that explicitly defines the atom types for which it applies (i.e., has 4 type attributes) would be given the highest weight, and sorted to the top of the list, whereas an entry that specifies only class attributes would be given the lowest. Thus, for the force field XML shown in Listing 5 Foyer would reverse the order of the two defined dihedrals during preprocessing.

Listing 5: Force field XML snippet showing atom types defined for carbon in CH_2 and CF_2 substructures, a bonded definition between carbons, and C-C-C-C dihedral definitions for hydrogenated and perfluorinated alkanes.

```
<ForceField>
  <AtomTypes>
   ...
    <Type name="opls_136" class="CT" element="C" mass="12.01100"\\
        def="[C;X4](C)(C)(H)H" desc="alkane CH2" />
    <Type name="opls_962" class="CT" element="C" mass="12.01100"\\
        def="[C;X4](C)(C)(F)F" desc="perfluoroalkane CF2" />
   ...
  </AtomTypes>
  <HarmonicBondForce>
   ...
    <Bond class1="CT" class2="CT" length="0.1529" k="224262.4"/>
   ...
  </HarmonicBondForce>
   ...
  <RBTorsionForce>
   ...
    <Proper class1="CT" class2="CT" class3="CT" class4="CT" c0="2.9288"\\
        c1="-1.4644" c2="0.2092" c3="-1.6736" c4="0.0" c5="0.0"/>
    <Proper type1="opls_962" type2="opls_962" type3="opls_962"\\
        type4="opls_962" c0="14.91596" c1="-22.564312" c2="-39.41328"\\
        c3="11.614784" c4="35.446848" c5="0.0"/>
   ...
  </RBTorsionForce>
   ...
</ForceField>
```

3. Foyer software

In order to read the force field usage specification discussed above and perform atom-typing, the `Foyer` software has been developed as an open-source Python library. Python allows for portability between platforms and provides a wealth of freely available modules (e.g., NumPy[31], SciPy[32], NetworkX[33]) to facilitate many of the underlying operations. The source, documentation, tutorials, and examples of `Foyer` are freely available and can be found on the GitHub project repository[34], tutorial repository[35], and website[36]. Figure 1 provides an overview of the general software workflow, which we will discuss here.

3.1. Inputs and Preprocessing

`Foyer` accepts, as input, the XML force field file and an input chemical topology for which to apply the force field. In addition to sorting bonded parameters by specificity as described in the previous section, the XML force field file undergoes a preprocessing and validation step via application of an XML schema definition. Here `Foyer` enforces which elements (e.g. `HarmonicBondForce`) are valid and how their attributes should be formatted. While this does not test the accuracy of the parameters, it does ensure that all of the expected parameters are defined. Additionally, the schema ensures that atom types are

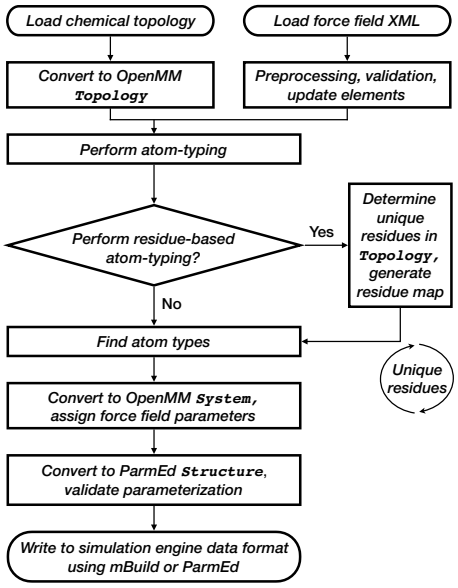


Figure 1: Flowchart of the Foyer software from chemical topology and force field XML inputs to a simulation data file output.

not defined more than once and that atom types referenced in other sections (e.g., `<HarmonicBondForce>`) are actually defined in the `<AtomTypes>` section. Next, the SMARTS strings defined by the `def` attribute for all atom types are parsed and checked for validity. This does not validate whether a SMARTS string is correctly defined for a given interaction site but simply ensures that the SMARTS string can be interpreted by `Foyer` and does not contain any erroneous characters. Parsing errors are captured and re-raised with error messages that allow a user to pin point the location of the problem in the XML file and within the SMARTS string. Wherever possible, `Foyer` attempts to provide helpful suggestions for fixing detected errors.

Input chemical topologies can be passed to `Foyer` through various data structures; the current version supports the `OpenMM Topology` object[23, 24], the `ParmEd Structure` object[22, 37], and the `mBuild Compound` object[26, 25, 27]. Each of `OpenMM`, `ParmEd`, and `mBuild` topologies support inputs from a variety of common molecular file formats, such as PDB and MOL2, and thus it is typically straightforward to convert a given system into a data structure that `Foyer` can accept. Regardless of the input format, once read into `Foyer` the chemical topology is converted to an `OpenMM Topology` object. The `OpenMM Topology` object provides a standardized data container to store the necessary system information and allows for leveraging of routines already defined within `OpenMM`'s library.

3.2. Atom-typing

A flowchart of `Foyer`'s atom-typing procedure is shown in Fig. 3. To perform atom-typing, `Foyer` constructs a graph of the complete system defined by the chemical topology (or alternatively a graph of each unique residue, see the Residue-based Atom-typing section below) and iteratively searches for SMARTS matches via subgraph isomorphism (where subgraphs are generated for each SMARTS definition). Graph construction and matching are performed using the `NetworkX` package[33], an open-source Python project that provides an intuitive interface for a multitude of graph-based algorithms and is the de facto standard network analysis library in Python. During this step, the iterative process of determining the atom type is undertaken, adding rules to the white and back lists for each interaction site in the system.

The implementation of the SMARTS based atom-typing scheme is comprised of several steps and internally relies on a subgraph isomorphism to detect matches as highlighted in Figure 2. First, a SMARTS string is parsed into an abstract syntax tree (AST) from which we populate a `SMARTSGraph` object. This class inherits from the `Graph` class in the `NetworkX` package. Elements in this `SMARTSGraph` are represented as nodes and chemical bonds as edges. Inheriting from `NetworkX` is convenient in that it allows us to leverage most of the algorithms and visualization methods already implemented there. The primary distinguishing feature of the `SMARTSGraph` is the set of methods that encode the logic for matching the more complex SMARTS tokens. These methods can be directly used by `NetworkX`'s implementation of the VF2 subgraph isomorphism algorithm[38]. A thin wrapper provided by the `find_matches` method allows a `SMARTSGraph` instance to search for all subgraph isomorphisms within a bare chemical topology (an non-atom-typed graph of just elements and bonds). This method returns the indices of all elements that match the first token in the SMARTS string, which defines the atom type that we are looking for. Successfully matching elements have the atom type definition added to their whitelist and any overridden types added to their blacklist. The appropriate atom type for an interaction site is determined by examining the difference between white- and blacklists, where a sufficiently descriptive force field should yield only a single atom type as the difference between the two lists.

The use of white- and blacklists provides users with a means to validate the completeness of the chemical contexts defined by the set of SMARTS strings and `overrides`. For example, when considering a test molecule, if multiple valid atom types are found as the difference between white- and blacklists, this indicates the rules are not sufficiently unique and likely have incomplete information provided to the `override` attributes. `Foyer` provides the list of conflicting types to aid in resolving such issues. If no atom types exist as the difference, the interaction site of interest cannot be described by the force field rules as implemented. Typically, this will require adding a new atom type or amending an existing atom type's definition. This may also indicate, that there is an error in how rule precedence has been defined, such as, all the rules on the whitelist "overriding" each other. The efficacy of this type of validation in `Foyer` will depend on providing a sufficient range of systems to fully explore the

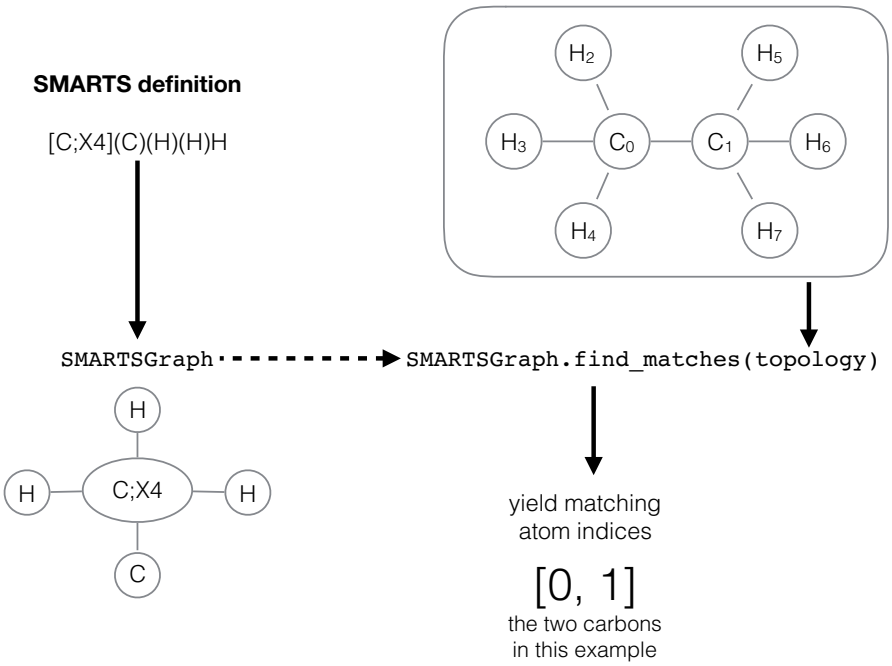


Figure 2: Schematic of the workflow to apply SMARTS patterns to chemical topologies. The SMARTS strings used to define atomtypes are read into a `SMARTSGraph` class which inherits from `NetworkX`'s core data structure. Using the `find_matches` method, a `SMARTSGraph` instance can search for subgraph isomorphisms of itself within a provided chemical topology and will yield all atoms that match the first token in the original SMARTS string - the atom type that we are looking for.

combinations of atom types that can be applied, where, as a general rule, the set of systems chosen to perform validation tests should collectively utilize all atom types defined in the force field. Note, these validation tests can be done to identify conflicts and under-defined systems, but do not necessarily indicate that the force field has been implemented corrected; separate verification tests are needed to ensure proper implementation, whereby the atom types identified by `Foyer` are compared to that of molecules with known, validated atom types, as discussed later.

3.2.1. Residue-based Atom-typing

Many systems of interest to molecular simulation contain topologies that consist of duplicates of smaller molecules or repeat units, each with identical topologies. A brute-force implementation of the atom-typing process wastes time by repeating subgraph isomorphism computation on each repeat unit and thus would not scale well with system size. To eliminate unnecessary calculations, a map of atom-typed residues is saved after each unique residue is atom-typed the first time. Then, when an identical residue is found, it copies

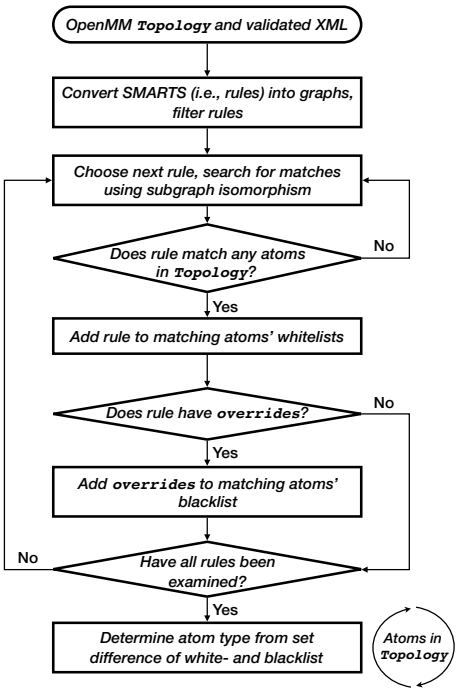


Figure 3: Flowchart of Foyer's atom-typing process.

the atom-typed information from the residue map instead of repeating the subgraph isomorphism. This feature is enabled by default but can optionally be turned off.

As an example, consider a box of N hexane molecules. After the subgraph isomorphism is called on the first molecule, the result is copied and saved into a map. Then, when molecules 2 to N are encountered, those results are copied into the running topology. The time it takes for the `apply` function to finish is timed for each case and plotted in Figure 4 relative to the brute force approach, where significant speed improvements are observed for common system sizes.

3.3. Force field assignment and output

Once atom-typed, the `OpenMM Topology`, now containing atom types for all particles in the system, is used to create an `OpenMM System` object and bonded parameters of systems determined. This step can be accomplished by simply searching the list of bonded parameters for the appropriate pair, triplet, quartet of atom types for bonds, angles, and dihedrals, respectively; such routines exist within OpenMM and are utilized in this context, where again we note these interactions undergo a sorting to ensure more specific definitions appear first in the file. Additionally, validation checks are performed at this time to ensure all triplets and quartets of interaction sites have had angle and dihedral (proper

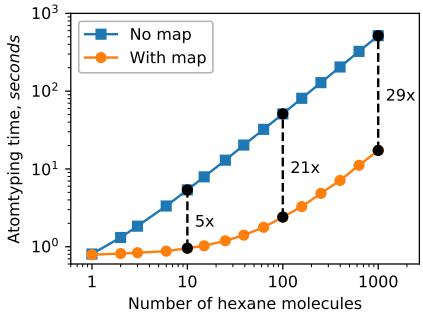


Figure 4: Comparison of atom-typing cost with and without the use of a residue templates. Without a residue template map, the scaling is approximately linear with system size. With a map, the scaling is independent of system size for small systems and becomes approximately linear at larger system sizes due to operations other than the subgraph isomorphism. The speedup approaches a factor of approximately 30 as the system size becomes large. The times to atom type each system were obtained with a 2013 MacBook Pro, 3GHz Core i7, 8 GB RAM.

and improper) parameters assigned (checks for bond parameterization of interaction site pairs are performed by `OpenMM` in the prior step). These validation checks provide the user with an error (that can optionally be overridden) to help prevent the return of incorrectly parameterized `Structures`. To output the atom-typed system into a usable format for a simulation engine, the fully atom-typed and parameterized system is returned as a `ParmEd Structure` object. Through the use of the `ParmEd Structure`, `Foyer` has access to various additional functionality, such as I/O routines that properly parse the `ParmEd Structure` into common chemical file formats (MOL2, PDB, NAMD and GROMACS formats, among others[22]). For file formats not natively supported by `ParmEd`, custom I/O routines for outputting to these formats (e.g., the LAMMPS data file format) have been developed within the `mBuild` package.

It should be noted that by utilizing `ParmEd` to take advantage of the extensive I/O routines, the force fields that `Foyer` currently supports must match functional forms supported by the internals of the `ParmEd Structure` object. For example, non-bonded interactions are currently limited to a 12-6 Lennard-Jones functional form. However, due to the large amount of force fields that utilize this functional form (e.g., Amber[1], GAFF[39], OPLS[3], TraPPE[28, 29], CHARMM[2]), the current version of `Foyer` is still widely applicable; planned future development will include support additional functional forms to better accommodate the diverse force field landscape that exists.

3.4. Validating/verifying output

`Foyer` provides scripts to validate its output files by comparing against systems with known atom types (e.g., those determined by hand or reference molecules provided by a force field developer). Output validation requires (1) system(s) with known, validated atom types and (2) the force field XML file.

The known systems are read into `Foyer` and atom types are determined using the rules in the XML file. The atom types generated by `Foyer` are then compared against the known atom-typed system(s). The `pytest`[40] library is used to provide a clear, descriptive output of the results of these validation tests. Implementing output validation tests is particularly useful to force field developers as they ensure that the desired output is retained if a force field file is evolved through the addition of new atom types definitions or merged with a separate force field file. The utility of these validation checks relies not only upon providing accurate reference systems, but also a sufficient variety of test systems that encompass all defined atom types, as discussed previously. An example of such a validation test suite is provided as part of the `Foyer` template repository freely available on GitHub[41] .

4. Usage Examples

At the time of publication, `Foyer` includes example force field XML files with `def`, `overrides`, and `doi` statements for 110 OPLS atom types as well as parameters and atom types for the simulation of alkanes and primary alcohols using the TraPPE force field. As discussed later in Sec. 5, separate repositories have been created to demonstrate how to reproducibility distribute force fields. These include OPLS compatible sets of parameters for perfluoropolyethers[42, 43, 44], perfluoroalkanes[45], and alkylsilanes grafted silica substrates [46]. Implementation of additional atom types for OPLS and TraPPE force fields and the implementation of other `ParmEd` compatible force fields is an active area of work.

Here, a basic overview of the usage of `Foyer` is provided, although we direct readers to the GitHub project repository[34] and tutorial repository [35] for additional usage examples. Consider constructing a bulk system of ethane molecules and applying the OPLS force field. Listing 6 shows a simple `mBuild` script to load an ethane molecule and fill a 2nm x 2nm x 2nm box with 100 molecules. This defines the system's chemical topology to which the force field will be applied. As input, the force field file is identical to Listing 1 but with the `<AtomType>` information from Listing 2, as Listing 2 includes the usage rule definitions. Listing 6 demonstrates two different syntaxes for applying a force field using `Foyer` and saving the output, in this case to the file format required by GROMACS. The second option allows different forcefields to be applied to different topologies in the system. Listing 7 shows an example of creating two separate chemical topologies in the system, and applying two different force field files to each. The two atom-typed structures that result (`ethane_fluid` and `silica_substrate`) are then combined using a simple + operator and saved to any format supported by `ParmEd` (this assumes that cross interactions between ethane and silica are defined using standard mixing rules). Note that if the surface and polymers were bonded together (e.g., to create a surface-bound monolayer), the force field files would need to be combined into a single XML document.

Listing 6: Script to fill a box with ethane and apply the OPLS-AA force field to the system.

```python
import mbuild as mb
from mbuild.examples import Ethane
from foyer.test.utils import get_fn
from foyer import Forcefield

### Approach 1 ###
# Create the chemical topology
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=100, box=[2, 2, 2])
# Apply and save the topology
ethane_fluid.save('ethane-box.top', forcefield_files=get_fn('oplsaa_alkane.xml'))
ethane_fluid.save('ethane-box.gro')

### Approach 2 ###
# Create the chemical topology
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=100, box=[2, 2, 2])
# Load the forcefield
opls_alkane = Forcefield(forcefield_files=get_fn('oplsaa_alkane.xml'))
# Apply the forcefield to atom-type
ethane_fluid = opls_alkane.apply(ethane_fluid)

# Save the atom-typed system
ethane_fluid.save('ethane-box.top', overwrite=True)
ethane_fluid.save('ethane-box.gro', overwrite=True)
```

Listing 7: Script to build a system with an amorphous silica substrate in contact with a bulk ethane system and apply a different force field to the substrate and fluid respectively.

```python
from foyer import Forcefield
from foyer.test.utils import get_fn
import mbuild as mb
from mbuild.examples import Ethane
from mbuild.lib.atoms import H
from mbuild.lib.bulk_materials import AmorphousSilica

# Create a silica substrate, capping surface oxygens with hydrogen
silica = mb.SilicaInterface(bulk_silica=AmorphousSilica())
silica_substrate = mb.Monolayer(surface=silica, chains=H(), guest_port_name='up')
# Determine the box dimensions dictated by the silica substrate
box = mb.Box(mins=[0, 0, max(silica.xyz[:,2])], maxs=silica.periodicity + [0, 0, 4])
# Fill the box with ethane
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=200, box=box)
# Load the forcefields
opls_silica = Forcefield(forcefield_files=get_fn('opls-silica.xml'))
opls_alkane = Forcefield(forcefield_files=get_fn('oplsaa_alkane.xml'))
# Apply the forcefields
silica_substrate = opls_silica.apply(silica_substrate)
ethane_fluid = opls_alkane.apply(ethane_fluid)
# Merge the two topologies
system = silica_substrate + ethane_fluid
# Save the atom-typed system
system.save('ethane-silica.top')
system.save('ethane-silica.gro')
```

5. Promoting Reproducible Force Field Dissemination

Force field files and associated documentation, examples, and validation tests, can be readily developed and distributed using standard software de-

velopment approaches to improve quality and reproducibility. For example, the common git + GitHub/Bitbucket based distribution process allows force field creators to disseminate their force field files and associated content to the public via a version controlled repository that can be referenced from relevant publications. In this approach, a specific version of the force field used in a publication can be tagged in the git repository and a reference to this tagged version provided in the manuscript, allowing for a clear reference to the exact parameters and usage rules employed in the work. Other services, such as Zenodo [47], can additionally provide a digital object identifier (DOI) for the tagged record and a snapshot of the content of the archive. A variety of other features of this standard software development process translate well to force field development. Version control systems like git are designed to facilitate distributed, collaborative software development and allow for changes to the files in the repository to be easily tracked in a transparent manner. For example, as a force field is evolved or corrected, revisions can be easily tracked, including the author(s) responsible for the changes, and the specific differences between force field versions clearly identified using standard tools such as DIFF and through the use of descriptive "commit" statements as the content of the repository is changed. The support for tracking issues in services such as GitHub/Bitbucket additionally allow the community to provide feedback, request clarification, or identify errors in a file in a transparent manner. Whenever the developers wish to they can create a new release of the force field that, as noted above, can be tagged or provided with a citable DOI. Verification and validation of a force field can also be simplified by using this software design approach, by implementing automated testing tools that can perform checks on every new iteration (i.e., commit) of the force field content, to ensure errors are not introduced as the force field is changed.

To promote these practices, we have created a template git repository on GitHub which contains the basic framework needed to create, test, and publish a new force field as well as a guided tutorial that introduces users to the SMARTS based atom-typing scheme[41]. This process was successfully used in recent work that derived force field parameters for perfluoropolyethers[42], a novel lubricant class. The force field was published in conjunction with the manuscript and made freely available on GitHub[43]. The specific version of the force field at time of publication is citable via a separate DOI[44]. Any adjustments or improvements to the force field could now be released under a new DOI while the old one would still exist and point to the originally published force field in order to maintain provenance.

5.1. Atom type DOI labels

While automated atom-typing and the containment of atom types and force field parameters within a single file helps reduce user error and promotes reproducibility, users also require knowledge of the original source of parameters, in order to ensure proper citation and validation that the parameters are appropriate for their system of interest. `Foyer` achieves this goal by adding a `doi` attribute to each `Type` definition within the `AtomTypes` block of a force field

XML. Listing 8 shows the same atom-type definition for the OPLS-AA methyl carbon as in Listing 2 with the additional `doi` attribute providing the DOI to the original source where parameters for this atom type were derived.

Listing 8: Atom type definition for a methyl carbon tagged with the source DOI

```
<ForceField>
 <AtomTypes>
  <Type name="opls_135" class="CT" element="C" mass="12.01100"\\
    def="[C;X4](C)(H)(H)H" desc="alkane CH3" \\
    doi="10.1021/ja9621760"/>
 </AtomTypes>
</ForceField>
```

This feature eliminates ambiguity concerning the origin of parameters for a particular atom type. Furthermore, `Foyer` automatically logs associations between DOIs and atom types during the atom-typing process, providing a BibTeX file featuring the full citation for the sources of all parameters applied to a particular system, along with additional notes detailing precisely which atom types are contained within each source. For example, Listing 9 shows the BibTeX file generated for a nitropropane molecule using the OPLS-AA force field, which includes all reference information as well as notes describing which atom type parameters were obtained from each reference.

Listing 9: BibTeX file generated during atom-typing of nitropropane using the OPLS-AA force field (modified with line breaks for readability).

```
@article{Price_2001,
    doi = {10.1002/jcc.1092},
    url = {https://doi.org/10.1002%2Fjcc.1092},
    year = 2001,
    publisher = {Wiley-Blackwell},
    volume = {22},
    number = {13},
    pages = {1340--1352},
    author = {Melissa L. P. Price and Dennis Ostrovsky and William L. Jorgensen},
    title = {Gas-phase and liquid-state properties of esters, nitriles, and nitro
        compounds with the {OPLS}-{AA} force field},
    journal = {Journal of Computational Chemistry},
    note = {Parameters for atom types: opls_761, opls_760, opls_764, opls_763}
}
@article{Jorgensen_1996,
    doi = {10.1021/ja9621760},
    url = {https://doi.org/10.1021%2Fja9621760},
    year = 1996,
    month = {jan},
    publisher = {American Chemical Society ({ACS})},
    volume = {118},
    number = {45},
    pages = {11225--11236},
    author = {William L. Jorgensen and David S. Maxwell and Julian Tirado-Rives},
    title = {Development and Testing of the {OPLS} All-Atom Force Field
        on Conformational Energetics and Properties of Organic Liquids},
    journal = {Journal of the American Chemical Society},
    note = {Parameters for atom types: opls_135, opls_140, opls_136}
}
```

6. Conclusions

The `Foyer` Python library and annotation scheme for defining force field usage has been presented in this work. `Foyer` defines a general applicable approach for the specification of classical force fields in a format that is both human and machine readable and provides tools for automated atom-typing and validation. The force field annotation scheme used in `Foyer` defines parameters and their usage within an XML formatted force field, allowing force fields to be developed and evolved without the need to modify the source code used to evaluate them. This annotation scheme uses SMARTS to define the chemical context of an atom type and defines rule precedence via explicit `override` statements, allowing rules to appear in any order in a force field file. The `Foyer` software treats chemical topologies as graphs and rules as subgraphs, to identify atom types, using an iterative approach. This iterative approach allows force fields to be automatically evaluated for completeness and identify under specified force fields, helping to prevent the dissemination and usage of ambiguously defined force fields. `Foyer` is designed to be compatible with several common simulation engines and utilizes/extends several existing open-source Python tools and file

formats, in order to maximize flexibility and general applicability. Collectively, the approach utilized by `Foyer` can used to improve the clarity of force field usage and dissemination within the molecular simulation community. The `Foyer` software is open-source and freely available via GitHub[34].

7. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. 1535150 and 1835874. We also acknowledge the National Energy Research Supercomputing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, P. Weinerl, A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins, Journal of the American Chemical Society 106 (1984) 765–784 (1984). doi:10.1021/ja00315a051.

[2] A. D. MacKerell, N. Banavali, N. Foloppe, Development and current status of the CHARMM force field for nucleic acids., Biopolymers 56 (4) (2000) 257–65 (2000). doi:10.1002/1097-0282(2000)56:4¡257::AID-BIP10029¿3.0.CO;2-W.

[3] W. L. Jorgensen, D. S. Maxwell, 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 118 (45) (1996) 11225–11236 (jan 1996). doi:10.1021/ja9621760.

[4] J. I. Siepmann, S. Karaborni, B. Smit, Simulating the critical behaviour of complex fluids, Nature 365 (6444) (1993) 330 (1993). doi:10.1038/365330a0.

[5] J. J. Potoff, J. I. Siepmann, Vapor-liquid equilibria of mixtures containing alkanes, carbon dioxide, and nitrogen, AIChE Journal 47 (2001) 1676–1682 (2001). doi:10.1002/aic.690470719.

[6] 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 5647 (1998) 7338–7364 (1998). doi:10.1021/jp980939v.

[7] C. Oostenbrink, A. Villa, A. E. Mark, 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 25 (2004) 1656–1676 (2004). doi:10.1002/jcc.20090.

[8] Gromacs OPLS Atom Types.
URL https://github.com/gromacs/gromacs/blob/e131e1d16c589fded5cad47bbd52b010d59c80a7/share/top/oplsaa.ff/atomtypes.atp

[9] G. K. Sandve, A. Nekrutenko, J. Taylor, E. Hovig, Ten simple rules for reproducible computational research, PLOS Computational Biology 9 (10) (2013) 1–4 (10 2013). doi:10.1371/journal.pcbi.1003285.

[10] M. W. Thompson, R. Matsumoto, R. L. Sacci, N. C. Sanders, P. T. Cummings, Scalable screening of soft matter: A case study of mixtures of ionic liquids and organic solvents, The Journal of Physical Chemistry B ASAP (2019). doi:10.1021/acs.jpcb.8b11527.

[11] B. L. Bush, 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 33 (1993) 756–762 (1993). doi:10.1021/ci00015a015.

[12] A. W. Schüttelkopf, D. M. F. Van Aalten, PRODRG: A tool for high-throughput crystallography of protein-ligand complexes, Acta Crystallographica Section D: Biological Crystallography 60 (2004) 1355–1363 (2004). doi:10.1107/S0907444904011679.

[13] J. Wang, W. Wang, P. a. Kollman, D. a. Case, Automatic atom type and bond type perception in molecular mechanical calculations., Journal of Molecular Graphics & Modelling 25 (2) (2006) 247–60 (oct 2006). doi:10.1016/j.jmgm.2005.12.005.

[14] A. A. S. T. Ribeiro, B. A. C. Horta, R. B. De Alencastro, MKTOP: A program for automatic construction of molecular topologies, Journal of the Brazilian Chemical Society 19 (7) (2008) 1433–1435 (2008). doi:10.1590/S0103-50532008000700031.

[15] A. K. Malde, L. Zuo, M. Breeze, M. Stroet, D. Poger, P. C. Nair, C. Oostenbrink, A. E. Mark, An Automated force field Topology Builder (ATB) and repository: Version 1.0, Journal of Chemical Theory and Computation 7 (2011) 4026–4037 (2011). doi:10.1021/ct200196m.

[16] K. Vanommeslaeghe, a. D. MacKerell, Automation of the CHARMM General Force Field (CGenFF) I: bond perception and atom typing., Journal of Chemical Information and Modeling 52 (12) (2012) 3144–54 (dec 2012). doi:10.1021/ci300363c.

[17] J. D. Yesselman, D. J. Price, J. L. Knight, C. L. Brooks, MATCH: an atom-typing toolset for molecular mechanics force fields., Journal of Computational Chemistry 33 (2) (2012) 189–202 (jan 2012). doi:10.1002/jcc.21963.

[18] D. Weininger, Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules, Journal of Chemical Information and Computer Sciences 28 (1) (1988) 31–36 (1988). doi:10.1021/ci00057a005.

[19] D. L. Mobley, C. C. Bannan, A. Rizzi, C. I. Bayly, J. D. Chodera, V. T. Lim, N. M. Lim, K. A. Beauchamp, D. R. Slochower, M. R. Shirts, M. K. Gilson, P. K. Eastman, Escaping atom types in force fields using direct chemical perception, Journal of Chemical Theory and Computation 14 (11) (2018) 6076–6092 (2018). doi:10.1021/acs.jctc.8b00640.

[20] Daylight Theory: SMARTS - A Language for Describing Molecular Patterns.
URL `http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html`

[21] P. J. in't Veld. EMC: Enhanced Monte Carlo; A multi-purpose modular and easily extendable solution to molecular and mesoscale simulations [online, cited 31 July 2018].

[22] ParmEd [cited 1 Feb 2019].
URL `http://parmed.github.io/ParmEd/html/index.html`

[23] P. Eastman, M. S. Friedrichs, J. D. Chodera, R. J. Radmer, C. M. Bruns, J. P. Ku, K. A. Beauchamp, T. J. Lane, L. P. Wang, D. Shukla, T. Tye, M. Houston, T. Stich, C. Klein, M. R. Shirts, V. S. Pande, OpenMM 4: A reusable, extensible, hardware independent library for high performance molecular simulation, Journal of Chemical Theory and Computation 9 (1) (2013) 461–469 (2013). doi:10.1021/ct300857j.

[24] OpenMM User Guide, Creating Force Fields [cited 1 Feb 2019].
URL `http://docs.openmm.org/7.0.0/userguide/`

[25] mBuild Software Repository on GitHub [cited 1 Feb 2019].
URL `http://github.com/mosdef-hub/mbuild`

[26] J. Sallai, G. Varga, S. Toth, C. Iacovella, C. Klein, C. McCabe, A. Ledeczi, P. T. Cummings, Web- and Cloud-based Software Infrastructure for Materials Design, Procedia Computer Science 29 (2014) 2034–2044 (2014). doi:10.1016/j.procs.2014.05.187.

[27] C. Klein, J. Sallai, T. J. Jones, C. R. Iacovella, C. McCabe, P. T. Cummings, A Hierarchical, Component Based Approach to Screening Properties of Soft Matter, Springer Singapore, Singapore, 2016, pp. 79–92 (2016).

[28] M. G. Martin, J. I. Siepmann, Transferable Potentials for Phase Equilibria. 1. United-Atom Description of n-Alkanes, The Journal of Physical Chemistry B 5647 (97) (1998) 2569–2577 (1998). doi:10.1021/jp972543+.

[29] TraPPE Force Fields.
URL http://www.chem.umn.edu/groups/siepmann/trappe/

[30] E. K. Watkins, W. L. Jorgensen, Perfluoroalkanes: Conformational Analysis and Liquid-State Properties from ab Initio and Monte Carlo Calculations, The Journal of Physical Chemistry A 105 (2001) 4118–4125 (2001). doi:10.1021/jp004071w.

[31] T. E. Oliphant, Guide to NumPy, 2nd Edition, CreateSpace Independent Publishing Platform, USA, 2015 (2015).

[32] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python.
URL http://www.scipy.org/

[33] D. A. Schult, P. Swart, Exploring network structure, dynamics, and function using networkx, in: Proceedings of the 7th Python in Science Conferences (SciPy 2008), Vol. 2008, 2008, pp. 11–16 (2008).

[34] Foyer Software Repository on GitHub [cited 1 Feb 2019].
URL http://github.com/mosdef-hub/foyer

[35] Foyer Tutorial Repository on GitHub [cited 1 Feb 2019].
URL https://github.com/mosdef-hub/foyer_tutorials

[36] Foyer Website [cited 1 Feb 2019].
URL http://mosdef-hub.github.io/foyer

[37] M. R. Shirts, C. Klein, J. M. Swails, J. Yin, M. K. Gilson, D. L. Mobley, D. A. Case, E. D. Zhong, Lessons learned from comparing molecular dynamics engines on the sampl5 dataset, Journal of Computer-Aided Molecular Design 31 (1) (2017) 147–161 (2017). doi:10.1007/s10822-016-9977-1.

[38] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub) graph isomorphism algorithm for matching large graphs, IEEE transactions on pattern analysis and machine intelligence 26 (10) (2004) 1367–1372 (2004).

[39] J. Wang, R. M. Wolf, J. W. Caldwell, P. A. Kollman, D. A. Case, Development and testing of a general amber force field., Journal of Computational Chemistry 25 (9) (2004) 1157–74 (jul 2004). doi:10.1002/jcc.20035.

[40] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, F. Bruhin. pytest [online] (2004).

[41] FoyerTemplate Repository on GitHub [cited 1 Feb 2019].
URL github.com/mosdef-hub/foyer_template

[42] J. E. Black, G. M. Silva, C. Klein, C. R. Iacovella, P. Morgado, L. F. Martins, E. J. Filipe, C. McCabe, Perfluoropolyethers: Development of an all-atom force field for molecular simulations and validation with new experimental vapor pressures and liquid densities, The Journal of Physical Chemistry B 121 (27) (2017) 6588–6600 (2017).

[43] Foyer Formatted Perfluoroethers Force Field [cited 1 Feb 2019].
URL https://github.com/mosdef-hub/forcefield_perfluoroethers

[44] J. Black, G. Silva, C. Klein, C. Iacovella, P. Morgado, L. Martins, E. Filipe, C. McCabe, Opls-aa compatible parameters for perfluoroethers. doi:10.5281/zenodo.583310.

[45] OPLS-AA compatible parameters for perfluoalkanes [cited 25 April 2019].
URL https://github.com/chrisiacovella/OPLSaa_perfluoroalkanes

[46] OPLS-AA compatible parameters for alkylsilanes and silica [cited 25 April 2019].
URL https://github.com/summeraz/OPLSaa_alkylsilanes

[47] Zenodo.
URL https://zenodo.org/