分子模板手册

安德鲁·朱维特
jewett.aij@gmail.com

2022 年 9 月 10 日

内容

警告:本手册不解释如何构建分子
使用全原子力场,或如何运行反应模拟。

然而,许多示例和 README 文件可用于演示如何运行这些类型的模拟。下载这些示例

强烈推荐。 （见第 2 节。）

1 简介

Moltemplate [1] 是用于 LAMMPS 的通用分子构建器和力场数据库系统。已经创建了一个简单的文件格式来存储分子定义和力场（LAMMPS 模板格式，“LT”）。 LT 文件是包含与特定分子相关的所有文本的模板（包括坐标、键拓扑、角度、力场参数、约束、组和修复）。然后，Moltemplate 可以复制分子，对其进行定制，并将其用作构建更大、更复杂分子的构件。 （这些分子可用于构建更大的分子。）

一旦建成，就可以定制单个分子和亚基（原子和键，以及亚基可以插入、移动、删除和/或替换）。

Moltemplate 非常灵活。它支持所有 LAMMPS 力场样式和几乎所有原子样式（现在和将来）。

全原子模拟

Moltemplate 最初是为准备粗粒度模拟而设计的。完全原子模拟通常需要选择力场特定的原子类型（“原子类型”），并计算原子电荷。 Moltem 板不这样做。 Moltemplate 也不会修复不完整的 PDB 文件。

为了绕过这些限制，用户可以使用 3rd 方分子构建器工具，例如： ATB数据库（LT 格式）、 AmberTools （MOL2 格式）、 EMC （DATA 格式）、 LigParGen （DATA 格式)和OpenBabel （DATA 格式）。这些工具可以生成 MOL2 或 LAMMPS-DATA 文件，这些文件可以使用mol22lt.py或ltemplify.py转换为 moltemplate 格式（LT 格式）（如果您想尝试手动选择原子类型而不是使用 3rd-party 工具，请阅读这些建议。)或者，用户可以使用具有相对简单的原子类型规则的DREIDING 力场。

除了上面列出的工具外，对于原子部分电荷， RED 服务器也很有用（可以生成 MOL2 文件）。或者，用户可以尝试使用 LAMMPS 的内置修复 qeq功能。

要求

Moltemplate 需要 Bourne-shell 和最新版本的 python（2.7 或 3.0 或更高版本），并且可以在 OS X、linux 或 windows 上运行（如果已安装合适的 shell 环境）。运行 moltemplate 需要大量内存。例如，构建一个包含 1000000 个原子的系统通常需要 3 到 12 GB 的可用内存。 （这取决于键、分子和角度相互作用的数量。有关详细信息，请参阅第 11 节。）

1.1 将 LT 文件转换为 LAMMPS 输入/数据文件

moltemplate.sh 程序将 LT 文件（包含分子定义)转换为完整的 LAMMPS 输入脚本和数据文件：

moltemplate.sh -atomstyle “完整” system.lt

或者

moltemplate.sh -xyz coords.xyz -atomstyle full -vmd system.lt

在第一个示例中,系统中原子的坐标是根据“system.lt”文件中的命令构建的。在第二个示例中,原子的坐标从 XYZ 文件中读取,然后调用 VMD 以可视化刚刚创建的系统。 （还支持 PDB 文件和其他坐标格式。注意:本示例中使用了“完整”原子样式,但支持其他 LAMMPS 原子样式,包括混合样式。）

这些命令中的任何一个都将构建一个 LAMMPS 数据文件和一个 LAMMPS 输入脚本（可能还有一个或多个辅助输入文件）,它们可以直接在 LAMMPS 中运行,只需最少的编辑。

1.2 将 LAMMPS 输入/数据文件转换为 LT 文件

现有的 LAMMPS 输入/数据文件可以使用“ltemplify.py”实用程序转换为“.LT”文件。 （可能需要一些额外的手动编辑。
见附录 B。）

附加工具

前面提到的 ATB 服务 [2] (https://atb.uq.edu.au)是数十万个经过仔细参数化的分子的存储库。用户可以提交对尚未在存储库中的分子的请求。这些分子的电荷和力场参数均已使用量子化学计算进行了单独优化。这些分子以 moltemplate 的原生“LT”格式提供。

Moltemplate 还可以读取其他工具生成的 LAMMPS DATA 文件文件,包括 VMD/topotools [3] 和 OpenBabel [4]。可以使用“ltemplify.py”实用程序将这些数据文件转换为“LT”格式。 VMD [5]、topotools [3] 和 OVITO [6] 也可用于可视化由 moltemplate.sh 和 LAMMPS 创建的数据文件和轨迹文件。 （见第 4.3 节。）

PACKMOL [7] 程序可用于生成分子密集异质混合物的坐标,这些坐标可由 moltemplate 读取。

（VMD“solvate”插件也可能有帮助。）

例子

本手册详细解释了如何使用 moltemplate.sh 从头开始构建 LAMMPS 文件。您还需要学习如何运行 LAMMPS 并可视化您的结果。第 4 节包含一个简短的教程,它解释了如何使用 moltemplate 构建一箱水并可视化初始构象,运行 LAMMPS,然后可视化轨迹。可以从以下网址在线下载和修改几个完整的工作示例（带有图像和自述文件）： http://moltemplate.org/visual_examples.html更全面的示例列表包含在

“examples/” 子目录与 moltemplate 一起分发。这些示例是学习 LAMMPS 和 moltemplate 的良好起点。

执照

Moltemplate 是开源的,可在http://moltemplate 上公开获得。组织。除了一个文件 (ttree lex.py) 之外,moltemplate 可根据 MIT 许可条款使用。其余文件 (ttree lex.py) 是 (shlex.py) 的修改版本,可在 PSF 许可(https://docs.python.org/3/license.html) 下使用。

2 安装

moltemplate的三种安装方式：

安装方法一（pip）

如果您熟悉 pip 或 pip3,则可以通过在终端/shell 中键入以下命令来安装 moltemplate：

pip3 install moltemplate # 或“pip”如果失败

如果您收到有关权限的错误,请以这种方式运行 pip/pip3：

pip3 install moltemplate --user # 或者 pip 如果失败

确保您的默认 pip 安装 bin 目录位于您的 PATH 中。 （这通常类似于 ~/.local/bin/ 或 ~/anaconda3/bin/。如果你安装了 anaconda,你的 PATH 应该会自动为你更新。）稍后,你可以使用以下命令卸载 moltemplate：

pip3 uninstall moltemplate # 或“pip”如果失败

如果您在使用 pip/pip3 时遇到困难,请尝试安装 moltemplate
通过运行以下命令进入临时虚拟环境：

python -m venv ~/venv #(或者 virtualenv venv 如果使用 python2) source ~/venv/bin/activate
pip3 install moltemplate # 或者 pip 如果失败 #(现在用 moltemplate 做一些有用的事情...)

（每次您想运行 moltemplate 时,您都必须预先在终端中输入“source ~/venv/bin/activate”。）
如果这一切都失败了,请尝试通过手动更新 $PATH 环境变量来安装 moltemplate。执行此操作的说明如下。

注意:如果您以这种方式安装 moltemplate,则会省略大量详细的 moltemplate 示例。强烈建议下载示例。您可以使用 git 或从网页下载它们来执行此操作。 （见下文。）（示例位于下载中包含的示例/子目录中。）

安装方式二（git/webpage + pip）

获取 Moltemplate

可以使用 git 下载最新版本的 moltemplate。

git 克隆 https://github.com/jewettaij/moltemplate ~/moltemplate

稍后,您可以使用以下命令将下载更新到最新版本的 moltemplate：

git 拉

（这是下载 moltemplate 的推荐方式。）
或者,如果您没有安装 git,您可以从http://www.moltemplate.org下载 moltem plate 作为 .tar.gz 存档,然后使用以下命令解压缩：

tar -xzvf moltemplate_2020-2-22.tar.gz

（日期因版本而异。）
然后将解压后的 moltemplate 目录移动到您的主目录 (~/) 并运行：

cd ~/moltemplate pip3 安
装。 # （或“pip3 install --user”,如果失败,则为“pip”）

如前所述,如果您遇到困难,请尝试在虚拟环境中安装 Moltem Plate。使用虚拟环境更干净,并且可以避免在具有复杂 python 安装的计算机中出现的许多常见陷阱。为此,请按照上述说明（在方法 1 中)启动虚拟环境,并按照上述使用 pip 或 pip3 的说明进行操作。

安装方法 3（更新你的 PATH）

或者,您可以下载 moltemplate 文件（如上所述)并编辑 PATH 变量。 PATH 变量有助于 shell（例如。

BASH) 找到您在终端中键入的程序。您必须编辑 PATH 变量以包含 moltemplate.sh 脚本所在的子目录（例如“~/moltemplate/moltemplate/scripts/”）,以及包含大多数 Python 脚本的目录（例如“~ / moltem 板/moltemplate/”)。

如果您使用 BASH shell,通常您会编辑 /.bashrc 文件（或您的 /.bash 配置文件或 /.profile 文件)并将以下行附加到该文件：

导出 PATH= $PATH:$HOME/moltemplate/moltemplate 导出
PATH= $PATH:$HOME/moltemplate/moltemplate/scripts

如果改为使用 TCSH shell,通常您会编辑 /.cshrc、 /.tcshrc 或 /.login 文件并附加以下行：

```
setenv PATH    $PATH:$HOME/moltemplate/moltemplate
setenv PATH    $PATH:$HOME/moltemplate/moltemplate/scripts
```

注意:您可能需要注销然后重新登录以进行更改
生效。

警告:如果您打算从 python 环境中调用 moltemplate,或者如果您正在使用

“vipster”、“cellpack2moltemplate”或其他具有 moltemplate 的软件
蟒蛇依赖。为了能够像这些程序一样运行“import moltem plate”,必须使用 pip 安装
moltemplate（或
安装工具）。

WINDOWS安装建议

您可以在 windows 中同时安装 moltemplate 和 LAMMPS,但您将
首先需要在你的电脑上安装 BASH shell 环境。我建议在 Windows 中安装virtualbox(https://
www.virtualbox.org)
以及带有轻量级桌面的（基于 debian 的)linux 发行版,例如xubuntu（https://
xubuntu.org）。或者,如果您是
使用 Windows 10 或更高版本,您可以尝试安装适用于 Linux (WSL) 的 Windows 子系统（仅
限文本， https://docs.microsoft。
com/en-us/windows/wsl)或 Hyper-V (https://www.nakivo.com/blog/
运行-linux-hyper-v/)。否则,如果您使用的是旧版本的 Windows,请尝试安装 CYGWIN (https://
www.cygwin.com/) 。

要使用 LAMMPS 和 moltemplate,您还需要安装（和
学习如何使用)文本编辑器。 （Word、写字板和记事本不会
工作。)如果您不使用 WSL,那么您可以使用流行的图形
文本编辑器,例如 Atom、Sublime、Notepad++、VSCode,以及 emacs 和 vim 的图形版本。
（注意:如果您是
使用 WSL 环境。在 WSL 下,这些编辑器可能会导致文件
系统腐败。暂时避开它们。 （https://www.reddit.com/r/
bashonubuntuonwindows/comments/6bu1d1/since_we_shouldnt_edit_files_
stored_in_wsl_with/)如果您正在使用 WSL,那么您仅限于
使用可以安全安装和运行的非图形文本编辑器
在 WSL 终端内。其中包括:nano、ne、emacs（文本版）、
vim（文本版本）,和 jove。

3 快速参考（初读时跳过）

注意:新用户请跳至第 4 节

3.1 Moltemplate 命令

命令	含义 定义一

分子类型 {
内容 ...
}
	种名为 MolType 的新型分子（或命名空间）。用大括号括起来的文本（内容）通常包含多个 write()、write once() 命令来定义原子、键、角度、系数等...（如果该分子类型已经存在,那么这将附加附加内容到它的定义。）（参见第 4.1、6.1、10.7 和 10.4.1 节中定义的 SPCE、单体和丁烷分子,以及 TraPPE 命名空间。

mol 名称 = 新的 MolType	创建（实例化)类型为 MolType 的分子的副本并将其命名为 mol name。（见第 4.1 节。）
分子名称 = 新的 MolType.xform()	创建分子的副本并将坐标变换 xform() 应用于其坐标。 （参见第 4.2 和 3.3 节。)分子 = new MolType [N].xform() 创建 MolType 类型分子的 N 个副本,并将它们命名为分子[0]、分子[1]、分子[2]...
	每个连续副本中的坐标都根据 xform() 进行累积转换。 （参见第 4.2、8.1 和 3.3 节。)也允许使用多维数组。 （见第 8.6 节。）
分子 = 新 MolType.xform1() [N].xform2()	将坐标变换 (xform1() 应用于 MolType,然后再制作 N 个副本,同时累积应用 xform2()。（参见第 8.1 和 8.3 节。）
分子=新随机（[M1.xf1 （），M2.xf2（），M3.xf2 （），...], [p1， p2， p3，...],种子） [N].xform()	使用（可选)初始坐标变换 xf1(), xf2(), xf3, ...生成从 M1,M2,M3,... 中随机选择的 N 个分子数组,概率为p1, p2, p3... ,并在此后累积应用转换 xform()。这也适用于多维数组。您可以通过替换概率列表[p1,p2,p3,...]来有效地指定每种分子的数（见第 8.4 和 8.8 节。）
新摩尔 = 旧摩尔	基于现有的分子类型创建新的分子类型。稍后可以使用 NewMol {更多内容...}将其他原子（或键等)添加到新分子中。 （见第 9.4 节。）
NewMol = OldMol.xform()	基于现有的分子类型创建一个新的分子类型,并对其应用坐标变换 xform()。（见第 9.4 节。）

NewMol 继承 Mol1 Mol2 ... {
附加内容...
}
	基于一个或多个现有的对象类型定义新的对象类型
	对象类型。原子类型、键类型、角度类型
	（等）在 Mol1 或 Mol2, ... 中定义，在新对象中可用。附加内容（包括更多
	write() 或 write once() 或新命令）
	跟在大括号内。（见第 9.4 节，
	10.7 和 10.7.1）

MolType.xform()	将坐标变换 xform() 应用于 MolType 类型的所有分子中的原子坐标。 （见第 9.4 节。）
分子.xform()	将坐标变换 xform() 应用于分子中的坐标。（这里的分子是指特定的 特定分子类型的实例或副本。看 第 9 节和第 4.2 节。）
分子[范围].xform()	将坐标变换 xform() 应用于由分子 [范围] 指定的分子的坐标。（这个 也适用于多维数组。见章节 8.5 和 9。）
删除分子	删除分子实例。（删除也可以 用于删除原子、键、角、二面角和 不正确的交互。）见第 9.3 节。
删除分子[范围]	删除由指定的一系列分子 分子[范围]。（这也适用于多维数组。见 9.3 和 8.10 节。）

写一次（文件）{
文本 ...
}
	写出用大括号 { 括起来的文本。 . .} 归档
	文件。文本可以包含由整数替换的@variables。（见第 5.1 和 5.2 节。）

写（文件）{
文本 ...
}
	写出用大括号 { 括起来的文本。 . 。} 至
	文件文件。每次创建新副本时都会执行此操作
	分子是使用“new”命令创建的。这
	text 可以包含 @variables 或 $variables
	„将被整数替换。（见第 5.1 和 5.2 节。）

注意：以“Data”或“In”开头的文件名（例如“Data Atoms”或“In Settings”）是 插入到 LAMMPS 数据文件或输入脚本的相关部分。（见第 5.3 节。）	
include file	在此处插入文件文件的内容。（引号可选。）
import file 在此处插入file file的内容，防止循环	（自我参照）包含。（推荐的）
类别 $catname(i0, Δ) 或者 类别@catname(i0, Δ)	创建一个新类别的计数器变量，从 i0并以 Δ 递增。详见 D.2 节。
创建变量 {$variable } 或者 创建静态变量 { @variable }	创建一个特定于该分子对象的变量。 （create var 通常用于创建分子 ID 较大的子单元之间共享的数字 分子。请参阅第 6.1 节。 create static var 通常用于定义共享的@atom 类型。）

替换 { 旧变量 新变量 }	允许相同变量的备用名称。这个 用 newvariable 替换 oldvariable 的所有实例。 两个变量名都必须有一个“@”前缀。这是 通常用于减少长变量的长度， 例如,允许简写“@atom:C2” 请参阅“@atom:C2 bC2 aC dC iC”（参见第 7.4 节。）
#注释文字	“#”字符后面的所有文本都被视为注释并被忽略。

3.2 常用$和@变量

（有关详细信息,请参阅第 5.2 节。）

变量类型	意义
$atom:name	分配给该分子中原子名称的唯一 ID 号。 （注意:如果分子中的 :name 后缀可以省略 这个变量似乎只包含一个原子。）
@原子:类型	表示原子类型的数字（通常用于查找 配对互动。）
$债券:名称	分配给债券名称的唯一 ID 号（注意：:name 如果出现这个变量的分子可以省略后缀 只包含一个键。）
@bond: 类型	表示债券类型的数字
$angle:name	分配给角度名称的唯一 ID 号（注意：:name 如果出现这个变量的分子可以省略后缀 仅包含单角度交互。）
@角度:类型	表示角度类型的数字
$二面体:名称	分配给二面体名称的唯一 ID 号（注意：:name 如果出现这个变量的分子可以省略后缀 只包含一个二面角相互作用。）
@二面体:输入	表示二面角类型的数字
$improper:name	分配给不正确名称的唯一 ID 号（注意：:name 如果出现这个变量的分子可以省略后缀 仅包含一个不正确的交互。）
@不正确:键入	表示不当类型的数字
$mol:名称	唯一的分子 ID 号。通常分子中的所有原子 具有相同的分子 ID。 （“:name”后缀是可选的。如果 省略,默认名称为“.”。）
美元摩尔：...	分配给该对象所属的分子的 ID 号（如果适用）。请参见第 6.1、10.6.2 节和附录 G.1。
分配给每个变量的数字保存在输出 ttree/ttree assignments.txt 文件中 _ _	
高级变量使用	
$类别:查询（）	查询此 $category 中计数器的当前值,无需 增加它。 （“$category”通常是 $atom、$bond、 $angle、$dihedral、$improper 或 $mol。）这对于计数很有用 原子、键、角度、分子等的数量..... 远的。
@类别:查询（）	查询此@category 中计数器的当前值而不增加它。 （“@category”通常是 @atom， @bond、@angle、@dihedral 或 @improper。）这对于计算到目前为止声明的原子类型、键类型、角度类型等的数量很有用。）
@{category:variable} 或 ${类别:变量}	花括号 {} 用于引用具有非标准分隔符或空白字符的变量。 （见第 5.6 节。）

@{category:type.rjust(n)} 或 @{category:type.ljust(n)} 或 ${category:name.rjust(n)} 或 ${category:name.ljust(n)}	在固定宽度 n 个字符的右对齐或左对齐文本字段中打印计数器变量。（这对于生成需要固定宽度列的文本文件很有用。）

3.3 坐标变换

（详见第 4.2 节和第 8.1 节）。）

后缀	意思是将数
.move(x,y,z) .rot(θ, x, y, z)	字 (x,y,z) 添加到每个原子的坐标 将原子坐标绕轴 (x,y,z) 旋转角度 θ 通过原点。 （偶极子方向也旋转。）
.rot(θ, x, y, z, x0, y0, z0)	围绕指向方向 (x,y,z) 的轴将原子坐标旋转角度 θ,通过点(x0, y0, z0)。 （这个点将是一个固定点。）
.rotvv（v1x， v1y， v1z ， v2x， v2y， v2z）	以一个角度旋转原子坐标,该角度将矢量v1旋转到v2 （围绕垂直于v1和v2 的轴）。如果您提供 3 个额外的数字x0、 y0、 z0,则旋转轴将通过该位置。
.scale（比率）	将所有原子坐标乘以比率。 （重要提示:scale() 命令不会更新力场参数,例如原子半径或键长。偶极子大小会受到影响。）
.scale(xr, yr, zr) .scale(ratio,x0, y0, z0) 或 .scale(xr, yr, zr, x0, y0, z0)	分别将 x、y、z 坐标乘以xr、 yr、 zr您可以提供 3 个可选的附加参数x0、 y0、 z0 ,它们指定您希望缩放发生的点。 （此点将是一个固定点。如果省略,则使用原点。）
.quat(a, b, c, d) .quat(a, b, c, d, x0, y0, z0)	通过对应于四元数 a + bi + cj + bk 的旋转来旋转原子坐标（如果指定,则围绕(x0, y0, z0)）。
.matrix(M1,1,M1,2,M1,3,M2,1, M2,2 ,M2,3,M3,1,M3,2,M3,3)	应用一般线性坐标变换。 $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} M1,1 & M1,2 & M1,3 \\ M2,1 & M2,2 & M2,3 \\ M3,1 & M3,2 & M3,3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$

注意:多个转换可以链接到一个复合操作中。
（例如：“.scale(2.0).rot(45.2, 1, 0, 0).move(25.0, 0, 0)”）
这些是从左到右评估的。 （见第 8.1 节。）

推（腐烂（152.3,0.79,0.43,-0.52）） 单体1 = 新单体 推（移动（0.01,35.3，-10.1））单体 2 =新单体pop（）pop（）	使用 push() 命令引入的坐标变换应用于稍后实例化的分子（使用 new)命令,并且在使用 pop() 命令删除它们之前一直有效。 （并且出现在数组中的转换也会累积,但不需要使用 pop() 删除。)在此示例中,第一个转换“rot()”同时应用于“monomer1”和“monomer2”。最后一个转换“move()”在“rot()”之后应用,并且只作用于“monomer2”。

3.4 moltemplate.sh 命令行参数：

论点	意思是通知
-atomstyle 风格	moltemplate 你正在使用哪种原子样式。 （样式默认为“完整”）。支持“分子”或“混合全偶极子”等其他样式。对于自定义原子样式，您还可以手动指定列名列表。例如： -atomstyle ”molid xyz atomid atomtype mux muy muz” Atom 样式应该用引号 (”) 括起来。
-raw coords.raw	从外部 RAW 文件中读取所有原子坐标。 （RAW 文件是简单的 3 列 ASCII 文件，包含每个原子的 XYZ 坐标，用空格分隔。）
-xyz 坐标.xyz	从外部 XYZ 文件中读取所有原子坐标（XYZ 文件是 ATOMTYPE XYZ 格式的 4 列 ascii 文件。第一列 ATOMTYPE 被跳过。第一行应包含原子数。第二行被跳过。请参阅第 4.2 节。）
-pdb 坐标.pdb	从外部 PDB 文件中读取所有原子坐标（如果存在，还会读取周期性边界条件。 PDB 文件中原子的顺序必须与原子在数据文件中出现的顺序相匹配，这与它们在 LT 文件中出现的顺序相匹配。请参阅第 4.2 节。） -dump coords.lammpstrj 从 LAMMPS 转储（轨迹）文件中读取所有原子坐标。 （如果存在椭球，转储文件必须
按以下顺序格式化：“id type xu yu zu cq[1] cq[2] cq[3] cq[4] ...” 其中“q”定义为“计算 q 所有属性/原子 quatw quati quatj quatk”。）	– – – –
-a 变量值	将变量分配给值。 （变量应该以 @ 字符或 $ 字符开头。需要单引号和空格分隔符。请参阅附录 D.1。）
-一个绑定文件 –	绑定文件（文本文件）第 1 列中的变量将分配给该文件第 2 列中的值。 （这在需要进行许多变量赋值时很有用。参见附录 D.1。）
-b 变量值 或者 -b 绑定文件 –	将变量分配给值。与使用“-a”进行的分配不同，使用“-b”进行的分配是非排他性的。 （它们可能与同一类别中的其他变量重叠。见附录 D.1。）
-overlay-bonds -overlay-angles -overlay-dihedrals -overlay-impropers	默认情况下，moltemplate 会覆盖涉及相同原子集的重复键合相互作用。这些标志禁用该行为。当您想在同一组原子上叠加多个角力或二面力时（例如，启用更复杂的力场），这可能很有用。 注意：每个副本仍然必须有一个唯一的 $bond、$angle、$dihedral、$improper 样式变量名称。

-nocheck	不检查常见的 LAMMPS/moltemplate 语法 错误。 （这在将 moltemplate 与 LAMMPS 以外的仿真软件，或构建需要新的非标准 LAMMPS 功能的系统。）
-checkff	这迫使 moltemplate.sh 检查系统中每 3 或 4 个连续键合的原子是否定义了有效的角和二面体相互作用（在“数据 按类型划分的角度”和“按类型划分的数据二面体”部分）。
-vmd	运行moltemplate后调用VMD查看系统 你刚刚创建。 （必须安装 VMD。参见章节 4.3, C 了解详情。）
-二面体符号文件.py -不正确的符号文件.py -键对称文件.py -角对称文件.py	通常 moltemplate.sh 会重新排列每个键中的原子， 写作前的角度、二面角和不适当的交互 将它们添加到 DATA 文件中，以帮助避免相同原子之间的重复相互作用，如果列在不同但 等价的订单。有时这是不可取的。要禁用此行为，请将“file.py”设置为“None”。你可以 也手动为异常选择备用对称规则 力场。 （如 class2 力场、二面体式球面等……有关“file.py”的文件格式示例，请参见 “nbody Impropers.py”文件。）
-molc	对 MOLC 用户有用的附加后处理 粗粒度模型
-短评论名称 -完整的评论名称	Moltemplate 在 LAMMPS 数据文件的“质量”部分后面的注释中写入原子类型名称。这些 两个参数控制是否打印原子类型名称的短版本或完整版本。 （默认： 短的。有关详细信息，请参阅第 5.2.3 节。）
-禁止通配符	禁止使用“*”和“?” “pair coeff”中的字符， “bond coeff”、“angle coeff”、“dihedral_coeff”和“improper_coeff”命令。 （例如：“债券系数@bond:CH?? ...”，“对系数 @atom:C* @atom:C* ...”。这些是允许的 默认。）

4 入门教程

概括

Moltemplate 基于一个非常简单的文本生成器（包装器），它重复地将短文本片段复制到一个（或多个)文件中并跟踪各种计数器。

LAMMPS 是一个功能强大但复杂的程序，有许多贡献者。Moltemplate 是 LAMMPS 的前端。 Moltemplate 用户将不得不解决其他 LAMMPS 用户必须面对的同样陡峭的学习曲线（以及偶尔出现的错误）。 Moltemplate 文件（LT 文件)与 LAMMPS DATA 文件和 INPUT 脚本共享相同的文件格式和语法结构。 Moltem plate 会尝试纠正用户的错误，但用户仍然必须学习 LAMMPS 语法并编写遵循它的 LT 文件。对于刚接触 LAMMPS 的用户，最简单的方法是修改现有示例（例如本节中的水箱示例）。 （官方 LAMMPS 文档https://docs.lammps.org是查找您在这些示例中看到的您不熟悉的 LAMMPS 命令的绝佳参考。）

4.1 使用 moltemplate 和 LAMMPS 模拟一箱水

图 1:我们示例中单个水分子的坐标。 （原子半径不按比例。）

这里我们展示了一个用于水的 lammps 模板文件的示例。 （此处显示的设置是从简单点电荷 [8] SPC/E 模型中借用的。)除了坐标、拓扑和力场设置之外，“LT”文件还可以选择包含任何其他类型的 LAMMPS 设置，包括 RATTLE约束、k 空间设置，甚至组定义。

```
# （注意：  #  字符后面的文本是注释)#

# 文件“spce_simple.lt”#

#       h1          h2
#          \      /
#             .
#

SPCE {

  ## Atom 属性和键拓扑位于各个“数据...”部分

  # 我们选择了“atom_style full”。这意味着我们使用这种列格式：
```

```
# atomID molID atomType charge coordX                                    协调                    坐标Z

写（“数据原子”）{
    $atom:o $mol:w @atom:O -0.8476 0.0000000 0.000000 0.00000
    $atom:h1 $mol:w @atom:H 0.4238 0.8164904 0.5773590 0.00000
    $atom:h2 $mol:w @atom:H 0.4238 -0.8164904 0.5773590 0.00000
}

# 以 $ 或 @ 开头的变量将被替换为数字 LAMMPS 将
# 最终阅读。三个原子中的每一个“都将被分配唯一的
# atomIDs（这里用“$atom:o”、“$atom:h1”、“$atom:h2”表示）,即使
# 它们属于不同的分子。但是,原子类型
#（表示为“@atom:O”、“@atom:H”）为所有分子中的原子共享。
# 所有 3 个原子共享相同的 molID 编号（这里用   $mol:w   表示）
# 但是对于不同的水分子,这个数字是不同的。

write_once(   数据量   ) {
    # atomType 质量
    @原子:O 15.9994
    @原子:H                 1.008
}

写（“数据债券”）{
    #bondIDbondType atomID1 atomID2
    $bond:oh1 @bond:OH $atom:o $atom:h1
    $bond:oh2 @bond:OH $atom:o $atom:h2
}

写（“数据角度”）{
    # angleID angleType atomID1 atomID2 atomID3
    $angle:hoh @angle:HOH $atom:h1 $atom:o $atom:h2
}

# --- 力场参数进入“设置”部分：---

write_once(   在设置中   ) {
    # -- 非键（对)相互作用 --
    #                       atomType1 atomType2 参数列表（epsilon,sigma）
    pair_coeff @atom:O @atom:O pair_coeff                   0.1553 3.166
    @atom:H @atom:H #（混合规则决定了@atom:O                0.0             2.058
    和@atom:H 类型之间的交互）

    # -- 绑定交互 --
    #                       bondType 参数列表 (k_bond, r0)
    债券系数@债券:哦                                        1000.00 1.0
    #                       angleType 参数列表 (k_theta, theta0)
    角度系数@角度:HOH 1000.0 109.47
```

```
    # 组定义和约束也可以进入“In Settings”部分 group spce type @atom:O @atom:H fix fRATTLE spce rattle 0.0001 10
    100 b @bond:OH a @angle:HOH # (lammps quirk: Remember to在最小化过程中“取消固定 fRATTLE”。）


}

# LAMMPS 支持大量的力场样式。我们必须选择我们需要的#。此信息属于“In Init”部分。


write_once(    In Init    ) { 单位实数
                                                    # 埃、千卡/摩尔、道尔顿、开尔文 # 选择原子部分的列格式
    atom_style full pair_style
    lj/charmm/coul/long 9.0 10.0 10 # 需要的参数：epsilon sigma bond_styleharmonic # 需要的参数：k_bond, r0
    angle_styleharmonic # 需要的参数：k_theta, theta0 kspace_style pppm 0.0001 # 混合规则 pair_modify 混合算
```

} # SPCE

以“$”或“@”字符开头的单词是计数器变量，将被整数替换。 （有关详细信息，请参阅第 5.2 节。）用户可以使用以下命令在模拟中包含 SPCE 水：

```
# -- 文件    system.lt    -- 导入
  spce_simple.lt    wat = new SPCE
[1000]
```

您现在可以使用“moltemplate.sh”为 LAMMPS 创建模拟输入文件

```
moltemplate.sh -pdb coords.pdb -atomstyle “完整” system.lt
```

此命令将为“system.lt”中描述的分子系统创建 lammps 输入文件，使用所需的原子样式（默认为“完整”）。在此示例中，moltemplate 依靠外部文件（“coords.pdb”）来提供水分子的原子坐标以及周期性边界条件。注意：PDB 文件中的原子顺序必须与原子在 LT 文件中出现的顺序一致。 （因此在本例中，PDB 文件中每个水分子中的氧原子必须位于该分子中的两个氢原子之前。）使用“-xyz coords.xyz”也支持 XYZ 格式的坐标。

细节

请注意，由于 XYZ 文件缺少边界信息，因此您还必须在“.lt”文件中包含“边界”部分，如第 4.2 节所示。在这两种情况下，PDB 或 XYZ 文件中原子类型的顺序（排序后）应该与它们由 moltemplate 创建的顺序相匹配（由 LT 文件中“新”命令的顺序决定）。不幸的是，这可能需要仔细手动编辑 PDB 或 XYZ 文件。

4.2 坐标生成

没有必要提供带有原子坐标的单独文件。更常见的是使用 LT 文件本身中的“.move()”和“.rot()”命令手动指定系统中分子的位置（和方向）（在第 6 节中讨论）。例如,您可以替换以下行：

```
wat = 新 SPCE [1000]
```

从上面的例子中,有 1000 行：

```
wat1      = 新的 SPCE
wat2      = 新 SPCE.move(3.1034, 0.00, 0.00) = 新
wat3      SPCE.move(6.2068, 0.00, 0.00) = 新 SPCE.move(9.3102,
wat4      0.00, 0.00)
  :                :
wat1000 = 新 SPCE.move(27.9306, 27.9306, 27.9306)
```

以这种方式指定几何图形很乏味。或者,moltemplate 具有简单的命令,用于将分子的多个副本排列成周期性、结晶、环形和螺旋形 1-D、2-D 和 3-D 晶格。例如,您可以使用单个命令（在本例中我们拆分为多行)生成一个简单的 10×10×10 水分子立方晶格（间距为 3.1034 埃）

```
wat = 新 SPCE [10].move(0.0, 0.0, 3.1034) [10].move(0.0, 3.1034,
                                    0.0) [10].move(3.1034, 0.0, 0.0)
```

（有关更多详细信息和示例,请参阅第 6 节。)这将创建 1000 个分子,其名称如“wat[0][0][0]”、“wat[0][0][1]”。 . .,“wat[9][9][9]”。对于 LT 文件中其他位置的任何分子,您始终可以使用以下注释访问单个 atomID、molID、bondID、angleID 和 dihe dralID（如果存在）：“$atom:wat[2][3][4]/h1 ”，“$bond:wat[0][5][1]/oh1”，“$angle:wat[2][8][3]/hoh”，“$mol:wat[0][1][2]/w”。这允许您定义将不同分子连接在一起的相互作用（参见第 6 节）。

3.3 节提供了可用坐标变换的列表。

边界条件：

LAMMPS 模拟具有有限的体积并且通常是周期性的。我们必须使用“write once(“Data Boundary”)”命令指定模拟边界的尺寸。

```
write_once(  数据边界  ) {
    0.0 31.034 xlo xhi
    0.0 31.034 ylo yhi 0.0 31.034
    zlo zhi
}
```

这通常在最外层的 LT 文件（本例中的“system.lt”)中指定。 （注:边界条件不必是矩形甚至

周期性的。对于三斜晶胞,可以添加额外的“xy”、“xz”和“yz”倾斜参数。有关详细信息,请在官方 LAMMPS 文档中查找“读取数据”和“边界”命令。）

该系统如图 2a) 所示。指定几何图形后,您可以这样运行 moltemplate.sh：

moltemplate.sh -atomstyle “完整” system.lt

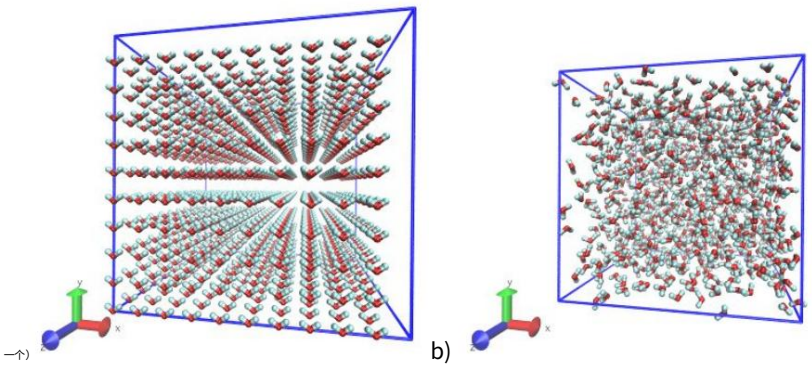


图 2:一盒 1000 个水分子（压力平衡前后）,由 moltemplate 生成并由 VMD 使用 topo 工具插件进行可视化。（用于可视化的 VMD 控制台命令是：“topo readlammpsdata system.data full”、“animate write psf system.psf”、“pbc wrap -compound res -all”和“pbc box”。参见第 4.3 节和 ap详见附录 C。

4.3 使用 VMD 和拓扑工具进行可视化

当您运行 moltemplate 时,它会生成一个 LAMMPS 数据文件。该文件通常称为“system.data”。几何信息和绑定拓扑存储在此文件中。运行 moltemplate 后,您应该查看系统以检查是否有错误。重叠的原子（缺少分子）、周期性边界、错误键合的原子、错误的旋转和运动很容易出现问题。有时需要多次运行 moltemplate 和可视化迭代。

可选:如果您安装了 VMD,您可以通过使用 -vmd 命令行参数调用 moltemplate 来自动可视化您刚刚创建的系统。（换句话说,使用 moltemplate.sh -vmd 而不是 moltemplate.sh 调用 moltemplate.sh。必须安装 VMD。）如果您不使用 -vmd 命令行参数,您可以稍后在 VMD 中手动查看系统.有关如何执行此操作的说明,请继续阅读...

下面提供了一些如何使用 VMD 的非常基本的说明：（注意：这些说明是为 VMD 1.9 和 topotools 1.2 编写的）
查看数据文件：

a) 启动 VMD b) 从

菜单中选择 Extensions→Tk Console

中央：

```
topo readlammpsdata system.data full animate
write psf system.psf
```

第一个命令将在 VMD 的 3-D 窗口中显示系统中的所有原子和键。 （我们使用“full”是因为我们在这个特定的示例中使用了“full”原子样式。如果您使用不同的原子样式，则相应地更改上面的命令。）

第二个命令将创建一个 PSF 文件（“system.psf”），稍后用于查看在 LAMMPS 仿真期间创建的轨迹文件。 （见第 4.5 节。）

默认情况下，原子和键很可能由难看的点和线表示。要更改分子的显示方式、控制它们的颜色、显示周期性边界和包裹原子坐标，请阅读附录 C 中的简短 VMD 教程。

（注意：截至 2022 年 8 月 11 日，VMD 没有内置支持外来原子样式，例如椭圆体和偶极子。VMD 也不能用于显示包含创建和销毁的原子或键的模拟。我建议使用 OVITO [6] 而不是 VMD 进行此类模拟。）

4.4 运行 LAMMPS 模拟（使用 moltem 盘子）

要运行一个或多个分子的模拟，LAMMPS 需要一个输入脚本和一个数据文件。输入脚本通常包含力场样式、参数和运行设置。 （它们有时也包含原子坐标。）数据文件通常包含原子坐标和键合拓扑数据。 （它们有时还包含力场参数。）

Moltemplate 将创建以下文件：“system.data”、“run.in.EXAMPLE”、“system.in.init”、“system.in.settings”（可能还有其他文件，包括“system.in.coords”）。这些是 LAMMPS 输入/数据文件，只需极少修改即可在 LAMMPS 中运行（见下文）。主要输入脚本文件名为“run.in.EXAMPLE”，通常只包含三行：

包括“system.in.init” read_data
“system.data” 包括“system.in.settings”

要运行模拟，您必须编辑此文件以添加几个运行命令。 （您也可以根据需要重命名文件。）这些命令告诉 LAMMPS 您要使用的模拟条件（温度、压力）、运行模拟的时间、如何整合运动方程以及如何编写结果到一个文件（文件格式，频率等）。 Moltemplate.sh 无法为您执行此操作。可以从https://moltemplate.org/下载的在线示例中提供了一些简单的示例（您可以将其粘贴到输入脚本中）

示例.html。 （这些示例输入脚本通常具有类似“run.in.nvt”和“run.in.npt”的名称。）

除了示例之外，以下链接还提供了对 LAMMP 输入脚本的介绍： https://docs.lammps.org/Commands_input。 html、 https : //docs.lammps.org/Commands_structure.html 和 https://docs.lammps.org/Commands_all.html。

以下是 moltemplate 示例中使用的基本输入脚本命令列表（以及指向其文档的链接）：

运行https://docs.lammps.org/run.html时间步
https://docs.lammps.org/timestep.html

热https://docs.lammps.org/thermo.html

转储https://docs.lammps.org/dump.html

读取数据https://docs.lammps.org/read_data.html

重启https://docs.lammps.org/restart.html

包括https://docs.lammps.org/include.html

修复 nve https://docs.lammps.org/fix_nve.html

修复 nvt https://docs.lammps.org/fix_nh.html

修复 npt https://docs.lammps.org/fix_nh.html

修复 langevin https://docs.lammps.org/fix_langevin.html

修复https://docs.lammps.org/fix.html

组https://docs.lammps.org/group.html

计算https://docs.lammps.org/compute.html

打印https://docs.lammps.org/print.html

变量https://docs.lammps.org/variable.html

重新运行https://docs.lammps.org/rerun.html

修复拨浪鼓https://docs.lammps.org/fix_shake.html

修复震动https://docs.lammps.org/fix_shake.html

修复刚性https://docs.lammps.org/fix_rigid.html

此外，所有用户都应该熟悉以下命令：（这些命令出现在大多数 LT 文件的“In Init”部分。）

原子风格https://docs.lammps.org/atom_style.html对风格https://docs.lammps.org/pair_style.html

债券风格https://docs.lammps.org/bond_style.html

角度样式https://docs.lammps.org/angle_style.html

4.5 可视化轨迹

在 LAMMPS 中运行模拟后,有几个程序可以可视化系统。如果您已将轨迹保存为 LAMMPS “转储”格式,稍后您可以在 VMD [5] 中查看它。为了在 LAMMPS 中查看轨迹,我建议在运行 LAMMPS 时使用的 LAMMPS 输入脚本中使用以下样式的“转储”命令:

转储 1 所有自定义 1000 DUMP_FILE.lammpstrj id mol 类型 xyz ix iy iz

(“all”和“1000”是指原子选择和保存间隔,可能会因您运行的模拟类型而异。有关详细信息,请参阅 https://docs.lammps.org/dump.html 。)

有了转储文件后,您可以使用以下命令在 VMD 中查看它:

a) 启动 VMD 从左上角的菜单中,选择文件→新分子 b) 浏览以选择您在上面创建的 PSF 文件,然后加载它。(暂时不要关闭窗口。)

c) 浏览选择轨迹文件。如有必要,为“文件类型”选择:“LAMMPS 轨迹”。单击确定。

d) 单击加载按钮。

同样,要自定义分子外观、显示周期性边界条件和包裹分子坐标,请参见附录 C 中讨论的命令。

(注意:VMD 可能无法正确可视化不保留原子数和键数的模拟,例如使用 fix bond/create、fix bond/break 或 fix gcmc 运行的模拟。)

5 概述

5.1 基础:write() 和 write once() 命令

每个 LT 文件通常包含一个或多个“写入”或“写入一次”命令。这些命令具有以下语法

write_once(文件名) {text_block}

这将创建一个新文件并用大括号 {} 括起来的文本填充它。文本块通常跨越多行并包含计数器变量(以“@”或“$”开头)。替换为数字。然而,每次生成或复制分子(出现写入命令的位置)时,“write()”命令将重复地将相同的文本块附加到文件中(使用“new”命令,在增加适当的计数器后,如5.2.2 中解释)。

(也可以通过这种方式创建文件的部分。参见第 5.3 章。),

5.2 基础:计数器变量

“@”或“$”字符后面的单词是计数器变量。（不要将这些与 LAMMPS 变量混淆https://docs.lammps.org/variable.html）。默认情况下,所有计数器变量在写入文件之前都会被数字计数器替换。这些计数器从 1 开始（默认情况下）,并随着系统大小和复杂性的增长而递增（见下文）。

这些词通常包含一个冒号 (:),后跟更多文本。此冒号前面的文本是类别名称。（例如：“$atom:”、“$bond:”、“$angle:”、“@atom:”、“@bond:”、“@angle:”)属于不同类别的变量独立计算。

用户可以覆盖这些分配规则并创建自定义类别。
（详见附录 D.1 和 D.2。）

5.2.1 静态计数器以“@”开头

“@”变量一般对应类型:如原子类型、键类型、角度类型、二面体类型、不当类型。这些是简单的变量,它们按照从 LT 文件中读取的顺序分配给唯一的整数。每个类别中每个唯一命名的变量都分配给不同的整数。例如，“@bond:”类型变量从“1”编号到键类型的数量。（成对的键合原子被分配一个键类型。稍后,LAMMPS 将使用这个整数来查找描述这两个原子之间的力的键长和胡克定律弹性常数。）

5.2.2 实例计数器以“$”开头

另一方面，“$”变量对应于唯一的 ID 编号:原子 ID、键 ID、角度 ID、二面体 ID、不正确 ID 和分子 ID。

每当创建分子的副本时（使用“new”命令）,就会创建这些变量。如果你使用一个水分子创建 1000 个副本

命令之类的

wat = 新 SPCE[10][10][10]

然后 moltemplate 创建 3000 个“$atom”变量,名称如下

$atom:wat[0][0][0]/o
$atom:wat[0][0][0]/h1
$atom:wat[0][0][0]/h2
$atom:wat[0][0][1]/o
$atom:wat[0][0][1]/h1
$atom:wat[0][0][1]/h2

⋮

$atom:wat[9][9][9]/o
$atom:wat[9][9][9]/h1
$atom:wat[9][9][9]/h2

5.2.3 变量名:短名与全名

在上面的例子中,$ 变量的全名是“$atom:wat[8][3][7]/h1”,而不是“$atom:h1”。但是在水分子的定义中,您没有指定全名。您可以将此原子称为“$atom:h1”。同样,@atom 变量的全名实际上是“@atom:SPCE/H”,而不是“@atom:H”。然而,在水分子的定义中,您通常使用简写符号“@atom:H”。

5.2.4 数字替换

在写入文件之前,每个具有唯一全名的变量（$ 或 @)都将被分配给一个唯一的整数,默认情况下从 1 开始。

水示例中的各种 $atom 变量将替换为从 1 到 3000 的整数（假设不存在其他分子）。

但是“@atom:O”和“@atom:H”变量（“@atom:SPCE/O”和“@atom:SPCE/H”的简写)将分配给“1”和“2””（再次假设不存在其他分子类型）。

因此,总而言之,@ 变量会随着系统的复杂性而增加（即分子类型或力场参数的数量）,但 $ 变量会随着系统的大小而增加。

5.2.5 变量范围

这实际上意味着所有变量都特定于定义它们的局部分子。换句话说，“SPCE”分子一侧名为“@atom:H”的原子类型将被分配给与名为“ @atom:H”在“精氨酸”分子中。这是因为这两个变量将具有不同的全名（“@atom:SPCE/H”和“@atom:Arginine/H”）。

在分子之间共享原子类型或其他变量

有几种方法可以在两个分子之间共享原子类型。推荐的方法是在单独的文件中定义它们,并在需要时引用它们。这种方法在 6.1 节中演示。

（或者,您可以在当前分子定义之外定义它们,并使用类似文件系统路径的语法（“../”或“../../”或“/”)来访问原子（或分子)在当前分子之外。例如,两个不同的分子类型可以共享相同类型的氢原子,使用以下语法引用它：“@atom:../H”。详细请参见第 10.6 节。附录 G.1.)

5.3 “数据”和“输入”

同样,LAMMPS 需要输入脚本和数据文件才能运行。 Moltem plate 的工作就是生成这些文件。输入脚本通常包含力场样式、参数和运行设置。数据文件通常包含原子坐标、键、角度、二面角和不正确。

如果您熟悉 LAMMPS,您可能已经注意到上面的文件名（在 4.1 节的示例中)听起来很像 LAMMPS 数据文件或输入脚本中的部分,例如“Data Boundary”、“Data Atoms”、“Data Bonds” ”、“数据量”、“数据角度”、“数据二面体”、“数据不正确”、“初始化”、“设置”）。所有名称开头的文件都是特殊的。为方便用户,moltem plate.sh 脚本将这些文件的内容复制到 DATA 文件的相应部分（“Atoms”„、“Bonds”、„“Angles”等)或 moltemplate（“ system.data”、“system.in.settings”内 或“数据 等）。（然后将原始文件移动到“output ttree/”目录,以便清理并隐藏它们。)用户可以为 LAMMPS 数据文件创建自己的自定义部分。（见第 5.6 节。）

更一般地说，“write()”和“write once()”命令可用于创建运行模拟可能需要的任何其他文件,而不仅仅是 DATA 文件和 INPUT 脚本。（示例见第 5.5 节。）

5.4 使用输出 ttree 目录进行故障排除

不幸的是,当您第一次尝试在 moltemplate 创建的 DATA 文件上运行 LAMMPS 时,它崩溃并不少见。如果您未能始终如一地拼写原子类型和其他变量,则通常会发生这种情况。发生这种情况时,LAMMPS 将打印一条错误消息（位于由 LAMMPS 创建的“log.lammps”文件的末尾）,有时可以帮助您确定您犯了什么类型的错误。如果您只查看 LAMMPS 错误消息,可能很难找出导致此错误的原因。

幸运的是,当您运行 moltemplate 时,它会创建一个名为“output ttree”的目录。通过查看 LAMMPS 错误消息并查看“输出 ttree”目录中的文件,您通常可以找出错误在 moltemplate (.LT) 文件中的位置。

此处显示了详细示例： https://
moltemplate.org/troubleshooting.html。

例如:假设您在运行 LAMMPS 时收到一条错误消息：“未设置所有对系数”或“未设置所有质量”。在这种情况下,您应该打开“ttree assignments.txt”文件（位于“out put ttree”目录中)并查找拼写错误的原子类型名称（以“@/atom:”开头）。另一方面,如果您收到错误消息“Invalid atom ID in Angles section of data file”,则在“ttree assignments.txt”文件中查找拼写错误的原子 ID 名称（这些名称以“$/atom:”开头）.

“ttree assignments.txt”文件是一个简单的 2 列文本文件,其中包含您在一列中创建的所有变量的列表,以及在第二列中分配给它们的数字。每行还有一个注释,以“#”字符开头,表示第一次使用该变量的文件和行号。（“输出 ttree”目录还包含所有文件,以及您在将它们合并在一起以创建“system.data”文件之前创建的文件部分。具有“.template”扩展名的文件包含散布有完整变量的文本数字替换之前的名称。)如果您检查,拼写错误,例如当您打算说“$atom:h1”或“@atom:H”时使用“$atom:h”将显示在这些文件中

他们仔细。这可以帮助您确定 moltemplate (LT) 文件中发生错误的位置。

一旦分子系统被调试并工作,用户可以忽略或丢弃“输出 ttree”目录的内容。

5.5（进阶)使用moltemplate生成辅助文件

LT 文件的以下摘录创建了一个名为“system.in.sw”的文件。
（它包含“sw”对样式的参数。这种奇异的多体对样式需要大量参数,这些参数是从一个单独的文件中读取的。)这个“system.in.sw”文件文件将在稍后读取你运行模拟。 （下面的 pair coeff 命令告诉 LAMPS 读取该文件。）

```
write_once( system.in.sw ) {
  毫瓦 毫瓦 毫瓦 6.189 2.3925 1.8 23.15 1.2 -0.33333 7.04956 0.602224 4 0 0
}
write_once( 在设置中 ) { pair_coeff *
  * sw system.in.sw mW NULL NULL NULL
}
```

随着新的力场样式和/或修复被添加到 LAMMPS,它们所依赖的文件可以以这种方式嵌入到 LT 文件中。

注意:辅助文件可以包含对原子类型（@atom 变量)和原子 ID（$atom 变量)的引用,以及键合相互作用,尽管它们没有出现在本示例中。

5.6（高级)制作自定义 DATA 部分

假设将来,LAMMPS DATA 文件的格式发生变化,因此现在需要提供一个名为“Foo Fee Fum”的新部分。您可以使用以下命令执行此操作：

```
write_once( Data Foo Fee Fum ) {
  文件内容放在这里。 （这些文件可以包含原子计数器和/或其他计数器变量）。

}
```

这样 moltemplate 会将此文本复制到它正在构建的 DATA 文件末尾的“Foo Fee Fum”部分。这允许用户适应 LAMMPS 数据文件格式的未来变化。

“@atom:H”是否与“$atom:H”冲突?

不可以。static(@) 和 instance($) 变量可以使用相同的名称。 （Moltemplate 将它们视为不同的变量,它们将被独立分配。）

附加细节

变量和分子名称可以包含 Unicode 字符。它们还可以使用反斜杠和花括号包含一些空白字符和其他特殊字符，例如：“@{atom: CA }”和“@atom:\ CA\”。花括号有助于阐明变量名称何时开始和结束，例如在此示例中：“@{atom:C}*@{atom:H}”。这可以防止将“*”字符附加到“C”变量名称的末尾。 （请注意，不鼓励在 moltemplate 中的任何 coeff 命令中使用“*”字符。请参阅第 11 节。）

（支持 Unicode。）

6 对象组合和坐标生成

物体可以连接在一起形成更大的分子物体。这些对象可用于形成更大的对象。例如,我们定义了一个名为“单体”的小 2 原子分子,并用它来构建一个短聚合物（“聚合物”）。

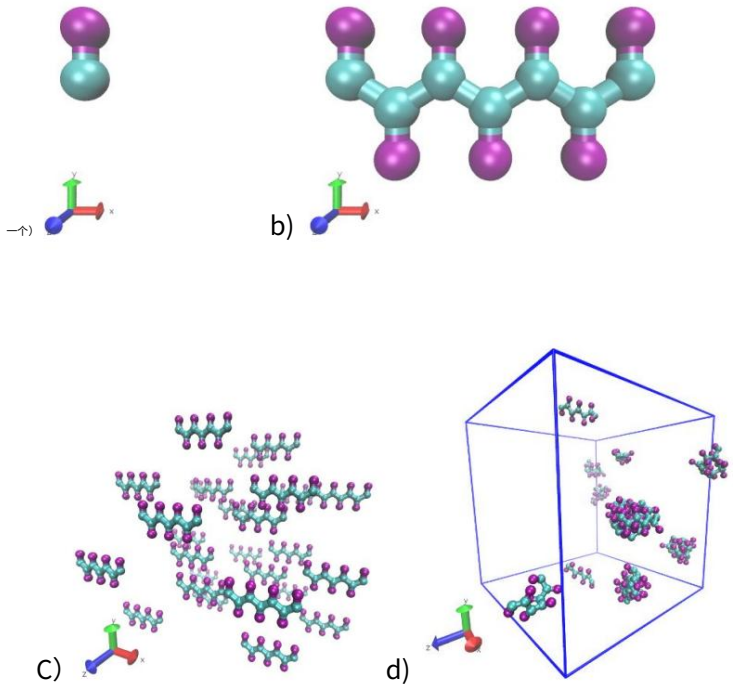


图 3:a)-b) 从小块构建复杂系统:使用组合和刚体转换从较小的（2 个原子)亚基 (a) 构建聚合物 (b)。将不同单体连接在一起的键（蓝色)必须明确声明,但角度和二面角相互作用将自动生成。有关详细信息,请参阅第 6.1 节。 c) 短聚合物的不规则晶格。 （见第 8.6 节。） d) 使用朗之万动力学在 100000 时间步后的同一系统。 （用于可视化的 VMD 控制台命令是:“topo readlammpsdata system.data full”、“animate write psf system.psf”、“pbc wrap -compound res -all”和“pbc box”。参见第 4.3 节和 C详情。

6.1 从小块构建大分子

考虑以下简单的 2 原子哑铃形分子（“单体”）

```
# -- 文件 monomer.lt --

import forcefield.lt # 包含力场参数

单体继承 ForceField {

  write( Data Atoms ) { # atomId
    molId atomType 电荷 xy $atom:ca $mol:... @atom:CA 0.0 0.000 1.0000 z
    0.0000000 $atom:r $mol:... @atom:R
                                                0.0 0.000 4.4000 0.0000000
  }
  写（“数据债券”）{
    #bond-id 债券类型 $bond:cr atom-id1 atom-id2
    @bond:Sidechain $atom:ca $atom:r
  }
}
```

很快将使用它来构建聚合物（“Polymer”） 注意:此处使用的省略号表示法“$mol:...”。 moltemplate 警告说，“单体”分子可能是更大分子的一部分。 （这在 10.6.2 节中有更详细的解释。）（注：“继承 ForceField”的含义将在下面的 7.1 节中解释）

在这个例子中,我们将定义两种分子对象：“单体”，
和“聚合物”（稍后定义）。

构建简单的聚合物

我们通过制作 7 个“单体”副本,旋转和移动每个副本来构建一个短聚合物：

```
# -- 文件 polymer.lt --

导入“monomer.lt”#（定义“单体”和“力场”）

Polymer 继承 ForceField {

  # 下一行是可选的： create_var {$mol} #（强制所
  有单体共享相同的分子 ID）

  # 现在创建一些单体

  mon1 = 新单体 #(无需移动第一个单体) mon2 = new Monomer.rot(180.0,
  1,0,0).move(3.2,0,0) mon3 = new Monomer.rot(360.0, 1,0,0)。 move(6.4,0,0) mon4 = new
  Monomer.rot(540.0, 1,0,0).move(9.6,0,0) mon5 = new Monomer.rot(720.0, 1,0,0).move( 12.8,0,0)
```

```
    mon6 = new Monomer.rot(900.0, 1,0,0).move(16.0,0,0) mon7 = new
    Monomer.rot(1080.0, 1,0,0).move(19.2,0,0)

    # 现在,以这种方式将单体连接在一起:write(    Data Bonds    ) {

        $bond:backbone1 @bond:Backbone $atom:mon1/ca $atom:mon2/ca $bond:backbone2
        @bond:Backbone $atom:mon2/ca $atom:mon3/ca $bond:backbone3 @bond:Backbone
        $atom :mon3/ca $atom:mon4/ca $bond:backbone4 @bond:Backbone $atom:mon4/ca
        $atom:mon5/ca $bond:backbone5 @bond:Backbone $atom:mon5/ca $atom:mon6/ca
        $bond:backbone6 @bond:Backbone $atom:mon6/ca $atom:mon7/ca

    }
}
```

“单体”的每个副本的位置和方向在“新”声明之后指定。每个“新”语句通常后面跟着一个由点分隔的移动/旋转/缩放函数链,从左到右评估（可选地后跟方括号,然后是更多点）。例如，“mon2”是“Monomer”的副本,它首先围绕 X 轴（用“1,0,0”表示)旋转 180 度,然后沿 (3.2,0,0) 方向移动。 （“rot()”命令的最后三个参数表示旋转轴,不必对其进行归一化。）（可用的坐标变换列表在第 3.3 节中提供。）

（注意:虽然我们在这里没有这样做,但有时将聚合物表示为一维数组很方便。示例见第 8 节和第 8.4 节。）

为了将不同分子或分子亚基中的原子结合在一起,我们使用 write(“Data Bonds”) 命令将额外的键附加到系统中。

7个力场

这是一个简短的章节,描述了 moltemplate 语言的特性,使用户能够在 LT 中创建和存储他们自己的自定义力场
格式。

这不是解释如何在模拟中使用流行的全原子力场（如 OPLSAA 或 GAFF2)的教程。 （也许稍后,我会在这个有价值的主题上添加更多信息。目前,学习构建逼真的全原子模拟的最佳方法是下载 moltemplate 及其示例。然后修改一个与您想要模拟的示例相似的示例.)

7.1 按类型划分的键相互作用

在前一章的聚合物示例中,我们没有列出所有作用在聚合物中的原子的角度、二面角和不适当的力。我们可以列出它们,但通常有很多。向 moltemplate 提供帮助它的说明通常更方便

自动确定哪些原子参与三体和四体角度相互作用,以及分配给它们的力场参数。我们将

使用以下命令在下面执行此操作:“write once(”Data Angles By
Type”)”、“write once(”Data Dihedrals By Type”)”和“write once(”Data
不当类型”)”
此外,由于许多不同种类的分子通常共享相同的
创建三体和四体角度交互的规则,很方便
将所有这些信息组织到一个地方(例如,一个
名为“ForceField”的对象)。 “ForceField”对象通常包括
一长串“只写一次(”数据角度/二面体/按类型划分的不当行为“)”
命令,以及力场参数和相关的原子类型属性。将力场对象存储在单独的文件中也是一个好主意

(例如“forcefield.lt”、“oplsaa.lt”、“gaff2.lt”、“compass.lt”等...),这样这些
规则和参数可以更容易地应用于不同的分子
(通过使用 import 命令在需要时加载文件)。

```
# -- 文件    forcefield.lt    --

力场 {

    # 有 2 种原子类型:“CA”和“R”
    write_once(    数据量    ) {
        @原子:CA              13.0
        @原子:R               50.0
    }

    # 力场参数(“coeffs”)进入“设置”部分:

    write_once(    在设置中    ) {
        # 成对(非键合)交互:
        #                     atomType1 atomType2 epsilon sigma
        pair_coeff @atom:CA @atom:CA pair_coeff                0.10 2.0
        @atom:R @atom:R #(不同原子之间的相互作用               0.50 3.6
        由混合规则决定。)
    }

    # 2-body (bonded) 相互作用:
    #
    # Ubond(r) = k*(r-r0)^2
    #
    write_once(    在设置中    ) {
        #                     键型                                r0
        bond_coeff @bond:Sidechain 15.0 3.4
        bond_coeff @bond:Backbone                  15.0 3.7
    }

    # 虽然我们上面定义的简单的“单体”对象只有
    # 两个原子,稍后,我们将创建具有许多键的分子。
```

```
# 按照惯例,在这个文件中,我们跟踪所有可能的
# 这些原子之间可能存在的相互作用:

# 通过原子和键类型确定三体（角度)相互作用的规则： # angle-type atomType1 atomType2 atomType3 bondType1
bondType2

write_once(　数据角度按类型　)
　{ @angle:Backbone @atom:CA @atom:CA @atom:CA @bond:* @bond:* @angle:Sidechain
　@atom:CA @atom:CA @atom:R @债券:* @债券:*
}

# 三体（角度)相互作用的力场参数:
#
# Uangle(theta) = k*(theta-theta0)^2
#
write_once(　在设置中　) { 角度类型
　#                                                    θ0
　angle_coeff @angle:Backbone 30.00 114 angle_coeff
　@angle:侧链 30.00 132
}

# 本例中的四体相互作用按原子类型和（可选)按键类型列出。 （您可以使用通配符 * 来接受所有债券类型。）

#
# dihedralType atmType1 atmType2 atmType3 atmType4 bondType1 bType2 bType3
write_once(　Data Dihedrals By Type　) {
　@二面体:CCCC @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
　@二面体:RCCR @atom:R @atom:CA @atom:CA @atom:R @bond:* @bond:* @bond:*
}

# 这个例子中使用的论坛是 :#

# Udihedral(phi) = K * (1 + cos(n*phi - d)) #

#　　d 参数以度为单位,K 以 kcal/mol/rad^2 为单位。
#
# 对应的命令是#dihedral_coeff dihedralType
                                                    K ndw（忽略）

write_once(　在设置中　) { dihedral_coeff
　@dihedral:CCCC -0.5 1 -180 0.0 dihedral_coeff @dihedral:RCCR -1.5 1
　-180 0.0
}

write_once(　在初始化中　) {
　# --　ForceField　中使用的样式 --
　# --（改变这些样式会改变上面的公式） -- 单位
                        真实的
```

```
        atom_style      满的
        bond_style      谐波
        angle_style     谐波
        dihedral_style charmm lj/
        pair_style      cut 11.0
    }
}
```

任何想要访问此信息的分子都可以使用“继承 ForceField”关键字。 （...正如我们在上面示例中的“monomer.lt”和“polymer.lt”文件中所做的那样。注意：“import forcefield.lt”语句也是必要的，因为信息位于单独的文件中：“force field.lt”。您可以通过更改键拓扑搜索规则和原子类型对称性来进一步自定义这些“按类型”规则。有关详细信息，请参见附录 F.2。）

7.2 通配符

星号 (*) 和问号 (?) 字符是多字符和单字符通配符。它们可以出现在任何“pair coeff”、“bond coeff”、“angle coeff”、“dihedral coeff”或“improper coeff”命令中。例如，以下摘录告诉 moltemplate 将非键对参数“0.10 2.0”分配给原子类型字符串以字母“C”开头的所有原子对

```
write_once(    在设置中    ) { pair_coeff
    @atom:C* @atom:C* 0.10 2.0
}
```

通配符也可以出现在“按类型”部分中的任何@atom 或@bond 类型名称中。如果最后一个原子的类型名称以“R”结尾，则以下规则将在任何三元组原子之间生成二面角相互作用（类型为“@angle:CCR”）。

```
write_once(    数据角度按类型    ) {
    @angle:CCR @atom:* @atom:* @atom:*R
}
```

这样，少量的这些规则就可以描述大量的原子类型组合。 OPLSAA 和 COMPASS 等力场使用此策略来减小“按类型划分的角度”部分（以及其他地方）中条目的大小。请注意，带有通配符的规则通常出现在“数据角度按类型”部分（以及其他“按类型”部分）的开头附近。描述特定原子类型组合之间角度的规则应该出现在列表的后面，以便它们可以覆盖更早出现的包含通配符的更一般的规则。

（注：基于正则表达式的原子类型名称匹配已经实现，但是此功能尚未经过测试。见 A.1 节。 -安德鲁 2020-8-09）

7.3 “按类型划分的数据绑定”和“数据绑定列表”

也可以通过指定原子类型来确定键类型
“按类型划分的数据键”部分：

```
write_once(  数据绑定按类型  ) {
   @bond:骨干 @atom:CA @atom:CA
   @bond:侧链 @atom:CA @atom:R
}
```

我们也可以将这些信息放在“ForceField”对象中。稍后，在“Monomer”和“Polymer”对象中，我们可以省略@bond 类型。例如，我们可以将“Polymer”中的“Data Bonds”部分替换为“Data Bond List”部分：

```
write_once(  数据绑定列表  ) {
   $bond:backbone1 $atom:mon1/ca $atom:mon2/ca $bond:backbone2
   $atom:mon2/ca $atom:mon3/ca
            :                    :                    :
}
```

请注意，我们没有指定“@bond:SideChain”和“@bond:Backbone”债券类型信息，这些信息通常会出现在“数据债券”部分的第二列中。虽然在此示例中不是很有用，但许多全原子力场（例如 OPLSAA、GAFF、COMPASS）按原子类型分配键类型。因此，使用这些力场的分子通常包含“数据键列表”部分，而不是“数据键”部分。

7.4 替换命令

“替换”命令对于想要以 moltemplate (LT) 格式发布自己的自定义力场的人很有用。此命令通常用于简化现代力场中原子类型的名称。在某些力场（例如，COMPASS、OPLSAA）中，原子类型名称因其用于查找力场参数的方式而变得复杂。 @atom 类型名称可能很长且难以输入（例如“@atom:h1h ph1h bh1h ah1 dh1 ih1”）。

“替换”命令允许使用你的力场的人在他们的分子中用短名称而不是长名称来引用这些原子。

考虑以下示例：

```
替换{ @atom:C4 @atom:C4_aC4 } 替换{ @atom:H1
@atom:H1_aH1 } 替换{ @atom:C4o @atom:C4o_aC4 }
替换{ @atom:H1o @atom:H1o_aH1 }
```

在此示例中，“@atom:C4”和“@atom:H1”分别是指“@atom:C4 aC4”和“@atom:H1 aH1”的简写。（“replace”命令适用于所有 @-style 变量，而不仅仅是 @atom 类型名称。它不适用于 $-style 变量。）

这些较长的原子名称通常与“按类型”规则结合使用，以生成角度和二面体相互作用。例如，考虑使用以下规则生成角度相互作用的力场：

```
write_once(    数据角度按类型    ) {
    @angle:HCH @atom:*_ah1 @atom:*_ac4 @atom:*_ah1 @angle:CCH
    @atom:*_ac4 @atom:*_ac4 @atom:*_ah1
}
```

通过使用通配符 (*),这两个规则仅取决于原子类型名称中“a”字符（“h1”或“c4”）之后的部分。由于多个原子类型共享相同的“h1”和“c4”名称后缀,这显着减少了此示例中碳和氢原子所需的“按类型划分的角度”规则的数量。 （随着原子类型数量的增加,这种减少变得更加显着。）

“替换”命令很有用,因为它允许用户在创建分子时通过它们的短名称（“@atom:C4”和“@atom:H1”）来引用这些长原子。例如：

```
# atom-id mol-id atom-type charge x write(    Data                    是的          z
Atoms    ) { $atom:c $mol:me @atom:C4 -0.18 0.0000
    0.0000 0.0000 $atom:h1 $mol:me @atom:H1 0.06 0.0000 0.6310 0.8924 $atom:h2
    $mol:me @atom:H1 0.06 0.0000 0.6310 -0.8924 $atom:h3 $mol:me @atom:H1 0.06
    -0.8924 -0.6310 0.0000

                :              :              :
}
```

7.5 “负责”

回想一下,moltemplate 是为模拟粗粒度分子模型而设计的。粗粒度模型使用包含许多原子、完整蛋白质甚至更大复合物的大粒子并不少见。将电荷分配给此尺寸范围内的粒子通常没有意义,并且 moltemplate 不会强制您使用带电荷的粒子表示。但如果你想为粒子分配电荷,有几种方法可以做到这一点。

现代分子构建工具通常使用复杂的从头算物理计算将部分电荷分配给原子。 Moltemplate 不这样做。但是 moltemplate 确实支持几种可选方法,允许您根据原子类型 (@atom) 分配粒子电荷。例如,以下是“loplsaa.lt”文件（2020-8-09)的摘录：

```
write_once（“负责”）{
    set type @atom:80L charge -0.222 #    烷烃 CH3- (LOPLS)    set type @atom:81L
    charge -0.148 #    烷烃 -CH2- (LOPLS)
    :
}
```

这会将“@atom:80L”类型的所有原子的电荷设置为 -0.222（无论它们与谁结合）。

令人困惑的是,收费信息并未写入 moltemplate.sh 创建的 DATA 文件中。而是将费用写入名称以“.in.charges”结尾的文件（例如“system.in.charges”）。稍后当你运行

LAMMPS,你必须记住在读取数据文件后使用 LAMMPS “include”命令来加载这个文件。例如：

```
读取数据“system.data”包括
“system.in.charges”
```

请注意,分子的“数据原子”部分中指定的任何电荷信息（以“system.data”结尾)都将被忽略。如果您想自定义分子中单个原子的电荷,请在分子定义的末尾添加“In Charges”部分。对于烷烃聚合物中的碳原子（例如）,您可以使用如下内容：

```
写（“负责”）{
    设置原子 $atom:monomer[0]/c 电荷 -0.222 设置原子
    $atom:monomer[1]/c 电荷 -0.148
                                :
}
```

（作为参考,这里解释了 LAMMPS 的“set”命令： https: //docs.lammps.org/set.html。）

7.6 “按数据收费”

“键的数据电荷”部分允许根据原子键合的其他原子及其类型将部分电荷分配给原子。

以下是“compass published.lt”文件的摘录（稍作修改）：

```
write_once(  数据绑定按类型  ) {
    @atom:h1 @atom:si4                          -0.1260 0.1260
    @atom:*bc4* @atom:*bsi4* -0.1350 0.1350
}
```

原子之间的每个键都有助于这些原子的部分电荷。
第一行告诉 moltemplate，“@atom:h1”和“@atom:si4”类型的原子之间的任何键分别将它们的电荷修改了 -0.1260 和 0.1260。
如果第一个原子类型包含字符串“bc4”并且第二个原子类型包含字符串“bsi4”。每个原子的最终电荷是通过将与其键合的每个原子对其电荷的贡献相加来确定的。

同样,这些费用不会写入您的 DATA 文件。如上所述,加载 DATA 文件后,必须记住使用 LAMMPS “include”命令加载包含电荷信息的文件。（例如：“include system.in.charges”。)同样,如果需要,您可以自定义分子中单个原子的电荷,方法是在末尾添加“write(”In Charges”)”部分您的分子定义（如上所示）。

7.7 力场目录 –

包含流行力场（如“oplsaa.lt”和“gaff2.lt”）和分子模型（如“spce.lt”）的文件存储在与 moltemplate 一起分发的“force fields/”子目录中。 （当您使用 git 或 pip 下载 moltemplate 时，会包含此目录。）您可以使用 import 语句加载任何这些文件的内容。它们不需要位于您的本地目录中。您也可以在那里添加自己的 LT 文件。

这是许多不同示例共享相同力场的便捷方式。如果定义了名为“TTREE PATH”的 shell 环境变量，则还会搜索那里列出的目录（由“:”分隔）。 （注意：如果您要导入的文件可以在多个位置找到，则本地目录中的文件具有优先权。）

8 数组、切片和坐标变换

Moltemplate 支持一维和多维数组。这些可用于制造直的（或螺旋的）聚合物片材、管材、环面。它们还用于用分子或原子填充固体 3 维体积。 （见第 4.2 和 8.6 节。）

在这里，我们展示了一种更简单的方法来创建部分所示的短聚合物 6.1。您可以通过这种方式制作 7 个单体分子的副本：

```
单体 = 新单体[7]
```

这将创建 7 个新的单体分子（命名为单体 [0]、单体 [1]、单体 [2]、单体 [3]、...单体 [6]）。您可以像我们之前所做的那样将它们与债券联系起来：

```
write( 数据键 )
  { $bond:backbone1 @bond:Backbone $atom:monomers[0]/ca $atom:monomers[1]/ca $bond:backbone2
  @bond:Backbone $atom:monomers[1]/ ca $atom:monomers[2]/ca $bond:backbone3 @bond:Backbone
  $atom:monomers[2]/ca $atom:monomers[3]/ca
          :                    :                    :                    :
}
```

不幸的是，默认情况下，每个分子的坐标是相同的。为了防止原子坐标重叠，您有几种选择：

8.1 新语句中括号 [] 后面的转换

在新命令中的每个方括号 [] 之后，您可以指定要应用的转换列表。例如，我们可以使用以下命令为 6.1 节中的短聚合物生成原子坐标：

```
单体 = 新单体 [7].rot(180, 1,0,0).move(3.2,0,0)
```

这将产生 7 个分子。第一个分子单体[0] 的坐标不会被修改。然而，每个连续的分子将通过命令“rot(180, 1,0,0)”和“move(3.2,0,0)”累积修改其坐标。

可选:初始自定义（前面的 [] 括号）

您还可以在复制分子之前和应用任何数组转换之前对分子的初始坐标进行调整。

例如：

```
单体 = new Monomer.scale(1.5) [7].rot(180, 1,0,0).move(3.2,0,0)
```

在这个例子中，“scale(1.5)”变换被应用一次来放大每个单体对象。这将在应用任何旋转和移动命令来构建聚合物之前发生（因此每个单体之间的 3.2 埃间距不会受到影响）。

8.2 实例化后的转换

或者,您可以在创建分子后对其应用转换（即使它们是数组的一部分）。

```
单体 = 新单体 [7]

# 同样,第一行创建了名为#“monomers[0]”、“monomers[1]”、
“monomers[2]”、...“monomers[6]”的分子。
# 以下行将它们移动到位。单体[1].rot(180.0, 1,0,0).move(3.2,0,0) 单体
[2].rot(360.0, 1,0,0).move(6.4,0,0) 单体[ 3].rot(540.0, 1,0,0).move(9.6,0,0)
单体[4].rot(720.0, 1,0,0).move(12.8,0,0) 单体[5] .rot(900.0,
1,0,0).move(16.0,0,0) 单体[6].rot(1080.0, 1,0,0).move(19.2,0,0)
```

8.3 变换顺序（一般情况）

一个典型的分子数组可以这样实例化：

```
mols = new Molecule.XFORMS1() [N].XFORMS2() mols[*].XFORMS3()
```

本例中由“XFORMS1”表示的转换列表首先应用于分子。然后将“XFORMS2”中的转换多次应用于分子的每个副本。（对于索引为“i”的分子,名为“Molecule[i]”,XFORMS2 将被应用 i 次。）

最后,在创建了所有分子之后,将应用 XFORMS3 中的转换列表。例如,要创建一个由 10 个半径为 30.0 的聚合物组成的环,以位置 (0,25,0) 为中心,请使用以下符号：

```
polymer_ring = new Polymer.move(0,30,0) [10].rot(36,1,0,0)
    # 创建后,我们可以移动整个环 # （这些命令最后应用。）
    polymer_ring[*].move(0,25,0)
```

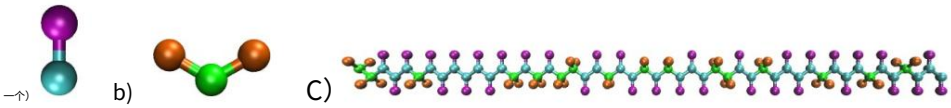


图 4:随机杂聚物 (c),由 Monomer 和 Monomer3 单体亚基 (a 和 b) 组成,(目标) 概率为 0.6 和 0.4。 （但由于随机波动,本案例实际比例为 68% 和 32%。为避免此问题,请参见第 8.4.1 节。）

8.4 随机数组

可以使用新的 random() [] 语法生成随机分子数组。例如,下面我们定义了一个由 50 个 Monomer 和 Monomer3 单体亚基组成的无规聚合物。 （见图 4。）

```
RandPoly50 继承 ForceField {
    # 制作一个随机选择的单体链：

    单体 = 新随机（[单体,单体 3],[0.6,0.4],123456)[50].rot(180,1,0,0).move(2.95,0,0)


    # 现在,以这种方式将单体连接在一起:write(    Data Bonds    ) {

        $bond:bb1 @bond:Backbone $atom:monomers[0]/ca $atom:monomers[1]/ca $bond:bb2
        @bond:Backbone $atom:monomers[1]/ca $atom:monomers[2] /ca $bond:bb3 @bond:Backbone
        $atom:monomers[2]/ca $atom:monomers[3]/ca $bond:bb4 @bond:Backbone $atom:monomers[3]/
        ca $atom:monomers[ 4]/ca

            ⋮

        $bond:bb50 @bond:Backbone $atom:monomers[48]/ca $atom:monomers[49]/ca
    }
    #(注意：“单体”和“单体3”亚基都包含名为“$atom:ca”的原子。
    #
} #RandPoly50
```

也可以用分子随机填充 2 维或 3 维体积。
这将在第 8.8 节中讨论。

新的 random() 函数接受 2 或 3 个参数:分子类型列表（本例中为 Monomer 和 Monomer3）,以及用方括号 [] 括起来的概率列表（0.6 和 0.4）。

8.4.1 具有精确分子类型计数的随机数组

回想一下,我们要求 60% 的分子属于“单体”类型,40% 属于“单体 3”类型（分别对应于 30 和 20）。然而，所得聚合物（如图 4 所示)包含 34 个“单体”和 16 个“单体 3”单体（分别为 68% 和 34%）。这是因为每次创建单体时,都会生成一个随机数来决定将创建哪种类型的单体。不能保证总

单体的最终部分将与目标概率完全匹配（分别为 60% 和 40%）。要精确指定分子类型的数量，您可以将概率列表“[0.6,0.4]”替换为整数列表“[30,20]”。

```
单体 = 新随机（[单体，单体 3]，[30, 20]，123456)[50].rot(180,1,0,0).move(2.95, 0, 0)
```

这将恰好产生 30 个“单体”和 20 个“单体 3”单体。 （您也可以对多维数组执行此操作。请参阅第 8.9.1 节。）

有关新随机命令的详细信息：

注意：您可以告诉 moltemplate 自定义键类型和角度，具体取决于每个键连接的单体（类型）。可在 www.moltemplate.org下载的“随机杂聚物”示例演示了如何执行此操作。

–

注意：虽然在这个例子中，只有两种单体类型（“单体”
和“Monomer3”），对出现在这些列表中的分子类型的数量没有限制（例如“[Monomer, Monomer3, 4bead],[0.2,0.3,0.2]”）
注意：还可以包含一个可选的随机种子参数。 （例如上面显示的“123456”。如果省略这个数字，那么每次运行 moltemplate 都会得到不同的结果。）

注意：这些列表也可以包含空缺/空白。见第 8.9 节。）
注意：一旦定义了包含随机单体的分子，（“RandPoly50”
在此示例中），该分子的每个副本（使用新命令创建)都是相同的。

可选：在 random() 数组中自定义分子位置

在构造数组之前，您可以自定义数组中每种分子的位置。为此，您可以在列表中每个分子的类型名称之后添加额外的移动命令（例如“单体”和“单体3”）：

```
单体 = 新随机（[Monomer.move(0,0.01,0), Monomer3.move(0,-0.01,0)],
                                                [30,20], 123456) [50].rot(180,1,0,
                                               0).move(2.95, 0, 0)
```

.move(0,0.01,0) 和 .move(0,-0.01,0) 后缀使这些单体靠近或远离聚合物轴（本例中的 x 轴）。

这不仅限于 .move() 命令。 （您也可以使用 .rot() 和 .scale() 命令。)这些移动将在稍后出现的任何 .move() 和 .rot() 命令之前应用（按从左到右的顺序） （在“[50]”之后)被执行。

8.5 [*] 和 [ij] 切片符号

您可以使用“[*]”符号移动整个分子阵列：

```
单体[*].move(0,0,40)
```

（请注意，“monomers.move(0,0,40)”不起作用。您必须包含“[*]”。）您还可以使用范围限制来仅移动一些单体：

```
单体[2-4].move(0,0,40)
```

这只会移动第三、第四和第五个单体。如果您更熟悉 python 的切片表示法，您可以使用以下方法完成相同的操作：

```
单体[2:5].move(0,0,40)
```

（在这种情况下，第二个整数（例如“5”）被解释为严格的上限。）

（如果这些定制不足以满足您的需求，您还可以随时从外部 PDB 或 XYZ 文件加载原子坐标。此类文件可以由 PACKMOL 或各种高级图形分子建模程序生成。对于复杂系统，这可能成为最佳选择。）

8.5.1 一次构建一个间隔的数组（使用切片表示法）

对于更复杂的示例，您可以使用切片符号构建聚合物。下面的示例演示了如何构建聚合物，指定哪个部分是随机的，哪个部分不是：

```
单体[0] 单体 = 新单体3
[1-48] = 新随机([单体, 单体3], [30, 18], 123456)
[48].rot(180,1,0,0).move(2.95, 0, 0) 单体[49] = new
Monomer3 # 移动这些单体以防止它们重叠单体是个好主意[0].rot (180,1,0,0) 单体
[1-48].move(2.95,0,0) 单体[49].move(144.55,0,0)

#（注：144.55=49*2.95）
```

在此示例中，我们确保单体 [0] 和单体 [49] 都属于“单体 3”类型（同时保持“单体”和“单体 3”单体的总数分别为 30 和 20）。

（注意：如果您更喜欢这种语法风格，可以将“monomers[1-48]”替换为“monomers[1:49]”或“monomers[1*48]”。您可以使用切片表示法构建多维数组同样，例如“分子[3][10-19][4-6] = 新分子[10][3]”）

8.6 多维数组

相同的技术适用于多维数组。坐标变换可以应用于多维阵列中的每一层。例如，要创建 3x3x3 聚合物的立方晶格：您可以使用以下语法：

```
分子 = 新聚合物 [3].move(30.0, 0, 0) [3].move(0, 30.0, 0) [3].move(0,
                                                    0, 30.0)
```

（类似的命令可以与旋转一起使用，以生成具有圆柱、螺旋、圆锥或环形对称的对象。）

8.7 自定义单独的行、列或层

同样，您可以使用上述方法自定义单个聚合物、层或列的位置：

```
分子[1][*][*].move(0,20,0) 分子[*][1][*].move(0,0,20)
分子[*][*][1].move (20,0,0)
```

参见图 3c)（您也可以使用切片表示法，例如“molecules[1][0-2][0-1].move(20,0,0)”）

您可以使用切片表示法删除数组的一部分并将其替换为其他内容（例如“Lipid”）：

```
删除分子[0-1][1][1-2] #（删除分子[0][1][1]的简写）删除分子[0][1][2]删除分子[1][1][ 1] 删除分子[1][1][2])
                                                    #
                                                    #
                                                    #

# 现在替换我们删除的数组元素：分子[0-1][1][1-2] = new Lipid [2].move(30,
0.0, 0.0) [2].move(0.0, 0.0, 30.0)

# ...并将它们移回我们创建的空缺位置
分子[0-1][1][1-2].move(0, 30.0, 30.0)
```

此示例中的“脂质”一词并不重要。它是一些其他分子类型的名称。

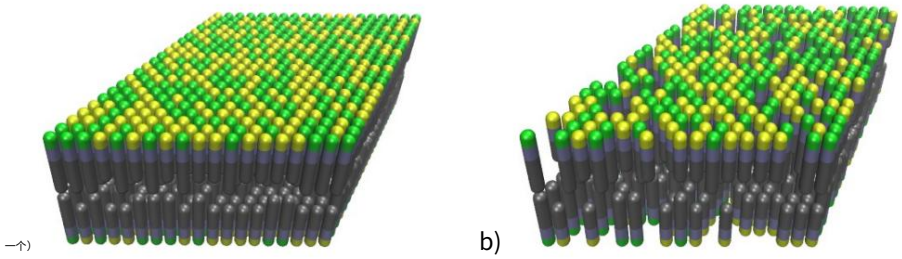
8.8 使用多维 ar 创建随机混合

光线

您可以使用“new random()”用分子的随机混合物填充空间。
以下二维示例创建了由 DPPC 和 DLPC 脂质的等量混合物组成的脂质双层（如图 5 所示）。 （.....我们在这里省略了谁的定义。有关详细信息，请参阅在线示例。）

```
进口“脂质”脂质=新
随机（[DPPC,DLPC],[0.5,0.5],123)[19].move（7.5,0)0,[22].move
                                        （3.75,6.49519,0)[2] .rot(180, 1, 0, 0)
```

```
# 定义 DPPC 和 DLPC
#    123    =random_seed
# 格子间距 7.5
# 六边形格子
# 2 单层
```



一个）

b)

图 5:由随机等量混合物组成的脂质双层膜
以 1:1 的比例混合两种不同的脂质类型。 （参见第 8.8 节。)在 b) 中,其中一个
分子类型留空,留下空位。 （见第 8.9 节。）

8.9 插入随机空位

传递给 random() 函数的分子类型列表可能包含
空白。在下一个示例中,缺少 30% 的脂质：

```
脂质 = new random([DPPC, ,DLPC], [0.35,0.3,0.35], 123) # 第二个元素为空白
                                    [19].move(7.5, 0)            0,
                                    [22].move(3.75, 6.49519, 0)
                                      [2].rot(180, 1, 0, 0)
```

结果如图 5b) 所示。 （注意:发生这种情况时,数组将包含缺失的元素。任何访问内部原子的尝试

这些缺失的分子会产生错误信息,无论移动或
使用 [*] 或 [ij] 表示法删除数组条目应该是安全的。）

8.9.1 具有精确类型计数的随机多维数组

由于随机波动,产生的 DPPC 和 DLPC 脂质的数量
可能不完全等于数组中条目数的 0.35 倍，
　　或者,您可以指定 DPPC 和 DLPC 的确切数量
您想要的分子（而不是概率列表）。去做这个，
用整数替换概率列表：

```
脂质 = 新随机 ([DPPC, ,DLPC], [293,250,293], 123)
                                    [19].move(7.5, 0, 0)
                                    [22].move(3.75, 6.49519, 0)
                                      [2].rot(180, 1, 0, 0)
```

这将准确生成 293 个 DPPC 和 DLPC 分子（以及 250 个空白
条目,因为第二种分子类型未指定）。总和（即
293+250+293) 必须等于您正在创建的数组中的条目数（即 19x22x2）。

8.10 使用删除切割矩形孔

删除命令可用于在 1、2 和 3 维对象上切割大孔。例如，考虑一个简单的 3 维 12x12x12 分子立方体。 （为简单起见，本例中的每个“分子”仅包含一个原子。这些原子在图 6 中显示为蓝色球体。）

```
分子 = 新的 OneAtomMolecule [12].move(3.0,0,0) [12].move(0,3.0,0)
                                                    [12].move(0,0,3.0)
```

然后，我们剪掉一些矩形的空位：

```
删除分子[*][*][2] 删除分子[*][*][8] 删除
分子[6-7][0-8][5-6]
```

这些操作的结果如图 6 所示。（注意：您可以多次移动或删除先前删除的数组元素，和/或删除重叠的矩形区域而不会出错。）

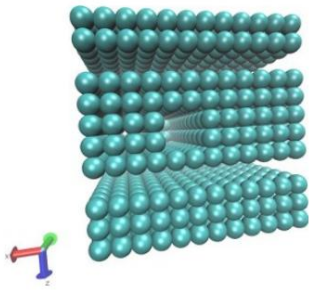


图 6：可以使用“删除”命令从分子阵列（此处用蓝色球体表示）中雕刻出矩形孔。三个删除命令用于删除两个平面区域和中心的矩形孔。

9 自定义分子位置和拓扑

默认情况下，使用新命令创建的分子的每个副本都是相同的。不必如此。

如第 8.2 节所述，最近创建的单个分子可以移动、旋转和缩放。您还可以覆盖或删除分子或其子单元中的单个原子、键和其他相互作用。 （参见第 9.3.2、9.1.1、9.1.2 和 9.2 节。）您对分子的某些副本进行任何这些修改，而不会影响其他副本。

此外，如果这些分子是复合对象（如果它们在其中包含单独的分子亚基），那么您也可以重新排列它们的亚基的位置。所有这些都可以从 LT 文件中的任何其他位置完成。

例如，假设我们使用了上面定义的“聚合物”分子
创造一个更大、更复杂的“分子复合物”分子。

```
MolecularComplex 继承 ForceField {
    聚合物[0] = 新聚合物聚合物[1] = 新聚合
    物.rot(180,1,0,0).move(0, 12.4, 0)
}
mol_complex = 新分子复合物
```

MolecularComplex 分子如图 7a) 所示。可选:如果您希望“MolecularComplex”中的所有原子共享相同的分子 ID,则以这种方式定义“MolecularComplex”：

```
MolecularComplex 继承 ForceField {
    create_var { $mol } 聚合物 [0]
    = 新聚合物聚合物 [1] = 新聚合
    物.rot(180,1,0,0).move(0, 12.4, 0)
}
```

为此,您还必须从 Polymer 分子的定义中删除“create var {$mol}”行。请参阅第 6.1 节。

我们可以随后自定义第二个聚合物（“polymers[1]”）的第三个单体（“monomers[2]”）的位置,这样：

```
mol_complex/polymers[1]/monomers[2].move(0,0.2,0.6)
```

这不会影响聚合物[0] 中单体[2] 的位置。 （或在任何其他“聚合物”或“分子复合物”分子中。）如果您想对两种聚合物做同样的事情,您可以使用通配符“*”

```
mol_complex/polymers[*]/monomers[2].move(0,0.2,0.6)
```

如果要移动两种聚合物,可以使用：

```
mol_complex/polymers[*].move(0,0.2,0.6)
```

（您也可以使用范围（切片）表示法,例如“polymers[0-1]”,作为“polymers[*]”的替代。参见第 8.5 节。

进行适用于随后创建的每个“聚合物”的更改或“MolecularComplex”分子,见第 9.4 节。）

9.1 自定义单个原子或键

9.1.1 自定义单个原子位置

不幸的是，“move”或“rot”命令不能用于控制单个原子的位置。而是简单地以这种方式覆盖它们的坐标：

```
写（“数据原子”）{
    $atom:mol_complex/polymers[0]/monomers[2]/ca $mol:mol_complex @atom:R 0 6.4 8.2 0。
}
```

这种情况我们必须使用原子的全名（“$atom:mol complex/polymers[0]/mon”来表示我们要覆盖哪个“ca”原子（在这种情况下,我们要修改“ ca”原子属于“聚合物[0]”中的“单体[2]”

“摩尔络合物”）。在此示例中,X、Y、Z 数字（6.4、8.2、0.6)位于全局坐标系中。相反,如果“write(”Data Atoms”)”语句出现在“MolecularComplex”对象的定义中,我们可以这样指定原子坐标：

```
分子复合体继承 ForceField {
    ：
   写（“数据原子”）{
       $atom:polymers[0]/monomers[2]/ca $mol:... @atom:R 0 6.4 8.2 0.6
   }
}
```

在这种情况下,原子坐标将在“MolecularComplex”对象的坐标系中定义,并受制于任何“.move()”、“.rot()”或“.scale()”命令应用于该对象（或其副本）。

9.1.2 自定义单个键、角度、二面角、...

同样,您可以通过覆盖包含对该特定键（或角度，...)的引用的行来自定义现有键、角度等。例如,您可以通过将类型“@bond:Backbone”更改为“@bond:LongBond”来增加代表第一个聚合物中第三个主链键的弹簧的剩余长度

```
写（“数据债券”）{
   $bond:mol_complex/polymers[0]/backbone3 @bond:ForceField/LongBond &
         $atom:mol_complex/polymers[0]/monomers[2]/ca & $atom:mol_complex/
         polymers[0]/monomers[3]/约
}
```

注意:可选的“&”字符在 LAMMPS 中用于将长命令拆分为多行。这个命令很长,因为我们超出了聚合物分子的定义,我们必须为每个变量提供完整的路径（例如“mol complex/polymers[0]”和“ForceField”）。

或者,如果您愿意,您可以删除键并定义连接同一对原子的新键（参见第 9.3 节）。

当然,在某些时候,您还必须创建一个新的“键系数”命令来定义这种新型键的属性（例如,增加新弹簧的长度）。您可以编辑“forcefield.lt”文件,也可以简单地将以下文本行添加到您的一个 .LT 文件（例如“system.lt”)中的某处

```
# 注意:这将增加“ForceField”对象,而不是覆盖它:ForceField { write_once(  In Settings  ) { new bond-type k r0 bond_coeff @bond:LongBond 10.0 4.8

      #

   }
}
```

9.2 给单个分子添加键和角

在分子中添加额外的键可以通过在“数据键”部分写入额外的文本行来完成。 （这就是我们在 6.1 节中添加单体之间的键以创建聚合物时所做的。）同样，键和原子名称必须用它们的全名来引用。

可以使用“删除”命令删除键和键相互作用。
（见第 9.3 节。）

9.3 删除命令

9.3.1 删除分子或分子亚基

通过删除单个原子、键、键合相互作用和整个亚基，可以进一步定制分子。我们可以删除第二个聚合物的第三个单体，使用“删除”命令：

删除 mol_complex/polymers[1]/monomers[2]

9.3.2 删除原子、键、角、二面角和不正确

可以用类似的方式删除单个原子或键：

delete mol_complex/polymers[1]/monomers[3]/ca #<-- 删除“ca”原子 delete mol_complex/polymers[1]/monomers[4]/cr #<-- 删除“cr”键

每当一个原子或一个分子被删除时，涉及这些原子的键、角、二面体和不正确的相互作用也会被删除。注意：删除原子、键或角度时必须省略“$”字符，就像我们在上面两行中所做的那样）。

当一个键被删除时，任何由 moltemplate 自动生成的角、二面角或不正确的相互作用也会被删除。
（但是，用户在其“数据角度”、“数据二面体”或“数据不当”部分中明确列出的其他绑定交互不会被删除。这些需要手动删除。）

多个分子可以在单个命令中移动或删除。例如，以下命令从聚合物[0] 和聚合物[1] 中删除第三、第四和第五个单体：

删除 mol_complex/polymers[*]/monomers[2-4]

有关范围（“[2-4]”）数组表示法和通配符（“*”）的解释，请参见第 8.5 节。

小错误通知：删除原子或分子可能会导致“ttree assignments.txt”文件的 $atoms、$bonds、$angles、$dihedrals 和 $impropers 部分不准确。 （如果这是一个问题，请给我发电子邮件。-Andrew 2019-9-03。）幸运的是，这不会损害生成的 LAMMPS 数据文件或 moltemplate 生成的输入脚本。他们仍然应该与 LAMMPS 一起工作。

警告：删除功能是实验性的。删除命令中存在一些错误，但到 2019-9-03 应该会修复这些错误。请报告您发现的任何问题。与往常一样，一定要可视化您的结构

以确保它们看起来合理。 （...例如,通过使用“-vmd”命令行选项运行 moltemplate.sh。有关详细信息,请参阅第 4.3 节,C 节。）

9.4 自定义分子类型

您可以创建现有分子类型的修改版本,而无需重新定义整个分子。例如：

```
MolecularComplex0 = MolecularComplex.move(-9.6,-6.2, 0).scale(0.3125)
```

或等效地：

```
MolecularComplex0 = MolecularComplex
MolecularComplex0.move(-9.6,-6.2, 0).scale(0.3125)
```

这将创建一种名为“MolecularComplex0”的新型分子,其坐标已居中并重新缩放。 （请注意，“scale()”命令只影响原子坐标。（您必须覆盖早期的力场设置,例如原子半径和键长才能使其正常工作。）如果我们想要额外的自定义（例如添加原子、键或分子亚基）,我们可以使用以下语法：

```
分子复合物0 = 分子复合物

# 在 mol_complex 中添加一些连接两个聚合物的新原子

MolecularComplex0 继承 ForceField {
   写（“数据原子”）{
      $atom:t1 $mol:... @atom:CA 0.0 23.0 0.0 $atom:t2 $mol:... @atom:CA      0.0
      0.0 24.7 4.0 $atom:t3 $mol:... @atom:CA 0.0 24.7 8.4 $atom:t4           0.0
      $mol:... @atom:CA 0.0 23.0 12.4                                        0.0
                                                                             0.0
   }

   write( 数据键 ) { $bond:b1
      @bond:Backbone $atom:polymers[0]/res7/CA $atom:t1 $bond:b2
      @bond:Backbone $atom:t1 $atom:t2 $bond: b3 @bond:Backbone $atom:t2
      $atom:t3 $bond:b4 @bond:Backbone $atom:t3 $atom:t4 $bond:b5
      @bond:Backbone $atom:t4 $atom:polymers[1]/res7 /ca

   }
}

# 将所有“MolecularComplex0”中的原子居中并重新缩放
MolecularComplex0.move(-9.6,-6.2, 0).scale(0.3125)
```

这些修改的结果如图 7b) 所示。

注意:这些坐标变换将在分子构建后应用。 （如果您将原子添加到分子中,这些将在应用坐标变换之前添加,即使您发出命令

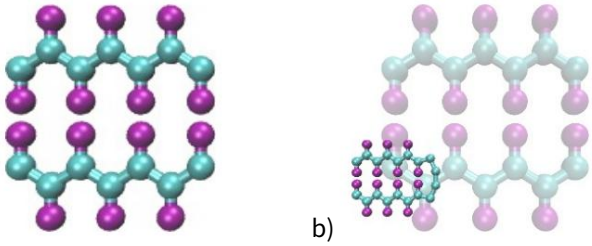


图 7:a) “MolecularComplex”分子。这是一个由两个“聚合物”组成的人为示例。请参阅第 6.1 b) 节“MolecularComplex”分子的定制版本。 （原始的“MolecularComplex”在背景中显示为褪色以进行比较。）

稍后。)因此,为了清楚起见,我建议在声明所有内部细节（键、原子、亚基)之后,将坐标变换应用于整个分子类型,就像我们在这里所做的那样。

（高级)继承

MolecularComplex0 分子是 MolecularComplex 分子的一种。
对于熟悉编程的人来说,这样的关系类似于面向对象编程语言中父对象和子对象之间的关系。
moltemplate 支持更一般的继承类型,并在第 10.7 节中讨论。

（高级)多重继承

如果我们愿意,我们可以创建一种新的分子类型（如“Molecular Complex0”） ,其中包括来自多种不同类型分子的原子类型和特征。第 10.7 节提到了这样做的一种方法,第 10.7.3 节讨论了替代方法。

10 高级 moltemplate 使用

本章包括对 moltemplate 语法的更详细讨论。
本章的目标是帮助您以可移植的方式编写 moltemplate (“.LT”) 文件,以便与他人共享。
“.LT”格式是一种灵活的文件格式,用于在 LAMMPS 中存储分子和力场参数。有多种方法可以做到这一点。为了完整起见,本章包括对这些不同方法的讨论以及它们之间的比较。然而,在使用力场时,鼓励大多数用户遵循第 10.7 章和第 7 章中解释的语法。

10.1 混合分子类型

在 4.1 节中,我们提供了一个 SPCE 水分子模型的例子。
这个例子很容易理解。但是 LAMMPS 能够在同一个模拟中使用不同类型的力场将分子组合在一起。如果您想与其他人共享您的“.LT”文件,它不是

可以安全地假设所有相互作用都使用相同的简谐键或
我们在该示例中使用的角度样式。例如，我们使用谐波
恢复力以将水角保持在 109.47。 ，但其他用户可能
想要将这种 SPCE 水与少量使用的分子混合
一个更复杂的角势公式，或表格角势。
使用“混合”关键字，您可以避免此限制。更健壮的
下面包括使用“混合”样式的示例。

```
# 文件“spce.lt”
#
#         h1            h2
#             \       /
#                 .

SPCE {

    write_once(    在初始化中    ) {
        # -- 默认样式（用于独奏“SPCE”水）--
        单位                        真实的
        atom_style 全
        pair_style 混合 lj/charmm/coul/long 9.0 10.0 10.0
        bond_style 混合谐波
        angle_style 混合谐波
        kspace_style pppm 0.0001
        pair_modify 混合算法
    }

    # AtomID MolID AtomType 电荷 coordX coordY coordZ
    写（“数据原子”）{
        $atom:o $mol:w @atom:O -0.8476 0.0000000 0.00000 0.000000
        $atom:h1 $mol:w @atom:H 0.4238 0.8164904 0.00000 0.5773590
        $atom:h2 $mol:w @atom:H 0.4238 -0.8164904 0.00000 0.5773590
    }

    # 原子类型质量
    write_once(    数据量    ) {
        @原子:O              15.9994
        @原子:H                1.008
    }

    # -- 原子之间的力（非键合） --

    #                       atomTypeI atomTypeJ                 对样式名称                          参数列表
    write(    在设置中    ) {
        pair_coeff @atom:O @atom:O lj/charmm/coul/long 0.1553 3.166
        pair_coeff @atom:H @atom:H lj/charmm/coul/long 0.0 2.058
    }
```

```
# -- 原子之间的力（键合） --

#bond-id bond-type atom-id1 atom-id2 write(    Data
Bonds    ) {
    $bond:oh1 @bond:OH $atom:o $atom:h1 $bond:oh2
    @bond:OH $atom:o $atom:h2
}

#                          键型键型名称参数列表
写（“在设置中”）{
    债券系数@债券:哦                                  谐波                          200.0 1.0
}

# 角度 id 角度类型 atom-id1 atom-id2 atom-id3 write(    Data Angles    ) {

    $angle:hoh @angle:HOH $atom:h1 $atom:o $atom:h2
}

#                                角度类型角度样式名称参数列表写入（“在设置中){

    角度系数@角度:HOH                                 谐波                          200.0 109.47
}

# 各种各样的
write_once(    In Settings    ) { group
    spce type @atom:O @atom:H fix fRATTLE
    spce rattle 0.0001 10 100 b @bond:OH a @angle:HOH # （记住在最小化过程中“unfix”
    fRATTLE。）
}

} # SPCE
```

此分子定义与 4.1 节中的“spce simple.lt”示例有两个不同之处：

混合力场样式

对于有经验的 LAMMPS 用户来说，在这个例子中我们选择了“混合”样式，然后只选择了一种力场样式（“谐波”），这可能看起来很奇怪。然而，这将使您的分子更容易与他人分享。当其他人使用您的 LT 文件时，他们可以覆盖这些样式，如第 10.2 节所述。

10.2 结合不同力场风格的分子

稍后，如果用户想要将 SPCE 水分子与另一个使用表格对样式的分子组合（例如），他们必须在其 LT 文件的“Init”部分中指定完整的混合对样式。

例如：

```
导入“spce.lt” 导入
 “other_molecule.lt”

write_once(  在初始化中  ) {
   pair_style 混合 lj/charmm/coul/long 9 10 10 表样条 1000
}
```

注意:通过在“import”spce.lt”之后放置“write once(”In Init”){ }”语句,这确保了这里发出的 pair 样式命令将覆盖之前发出的 pair 样式命令“spce.lt” ”。这允许 moltem 板用户用户将此处显示的分子“spce.lt”文件与其他模板文件组合在一起,而无需修改(假设原子样式匹配)。

警告:力场参数属于“设置”,而不是“数据”

LAMMPS 允许用户将力场参数(“Coeffs”)存储在两个位置:DATA 文件或 INPUT 脚本。同样,moltemplate 技术允许您将这些参数存储在 .LT 文件的“数据”部分中:

一次写入(“数据对系数”) 一次写入
(“数据键系数”)

一次写入(“数据角度系数”)

一次写入(“数据二面角系数”)

一次写入(“数据不正确系数”)

但是,出于便携性的原因,不鼓励这样做。相反,像我们在本手册中所做的那样,使用相应的输入脚本命令来声明您的力场参数。 (例如,“pair coeff”、“bond coeff”、“an gle coeff”、“dihedral coeff”和“improper coeff”命令。在示例中,这些命令通常应位于 “write once(”在 .LT 文件的“设置”部分。)

10.3 嵌套

诸如“溶剂”(甚至“水”)之类的分子名称简短且易于键入,但含糊不清且不便于携带。如果您使用常见的通用分子名称,您将无法将您的分子模板与其他人编写的模板组合起来(无需仔细检查命名冲突)。LT 文件旨在用于存储和交换不同分子类型的库。

例如,假设您要运行由不同分子类型组成的模拟,每个分子类型都属于不同的 LT 文件。假设两个 LT 文件都恰好包含“水”的定义。

Moltemplate 不会自动检测这些名称冲突,而是

尝试将两个版本的“水”合并在一起，（很可能创建一个具有 6 个原子而不是 3 个原子的分子）。这可能不是你的

想。

随着分子类型数量的增加，命名冲突的可能性也在增加。由于可以使用许多不同的力场来近似同一分子的行为，因此必须小心避免发生冲突的分子

名字。

为了缓解这个问题，您可以将您的分子“嵌套”在其他分子或命名空间对象的定义中。这缩小了定义分子的范围。有关示例，请参见第 10.5 节。

10.4 一个简单的力场例子

力场参数可以由相关分子组共享。在下面的示例中，我们创建了一个名为“TraPPE”的对象。后来我们用它来定义一个名为“环戊烷”的新分子。

以下示例定义了“环戊烷”分子的粗粒度（联合原子）版本。 （氢原子已被省略。）在这个例子中，只需要指定原子类型（和位置）和连接它们的键。它们之间的相互作用由力场文件“trappe1998.lt”中的设置自动确定。

```
导入“trappe1998.lt”

环戊烷{

  # AtomID MolID(  .  =this) AtomType charge coordX coordY coordZ write(  Data
  Atoms  ) {
    $atom:c1 $mol:cp @atom:TraPPE/CH2 0.0 0.0000 0.000000000 1.0000000 $atom:c2
    $mol:cp @atom:TraPPE/CH2 0.0 0.0000 0.951056516 0.3090170 $atom:c3 $mol:cp
    @atom:TraPPE/CH2 0.0 0.0000 0.587785252 -0.809017 $atom:c4 $mol:cp @atom:TraPPE/
    CH2 0.0 0.0000 -0.587785252 -0.809017 $atom:c5 $mol:cp @atom:TraPPE/CH2 0.0 0.0000
    -0.95105650160.0.
  }

  写（“数据债券”）{
    $bond:bond1 @bond:TraPPE/CC $atom:c1 $atom:c2 $bond:bond2
    @bond:TraPPE/CC $atom:c2 $atom:c3 $bond:bond3 @bond:TraPPE/
    CC $atom:c3 $atom:c4 $bond:bond4 @bond:TraPPE/CC $atom:c4
    $atom:c5 $bond:bond5 @bond:TraPPE/CC $atom:c5 $atom:c1

  }
}
```

（“TraPPE/”符号解释如下。）

注意：虽然概念上很简单，但目前不推荐在 moltemplate 中使用力场。 （请参阅第 10.7 节了解更好的方法。）

...然后我们可以像使用 SPCE 一样创建这个分子的副本：

125 个环戊烷分子的立方晶格（12 埃间距)mols = new Cyclopentane [5].move(0,0,12) [5].move(0,12,0) [5].move(12,0,0)

与 SPCE 示例不同,我们不必指定这些原子之间的所有相互作用,因为原子和键类型（CH2、CC）。匹配“trappe1998.lt”文件中定义的类型名称。该文件包含粗粒烃链的原子类型和力场参数的集合。 （详见[9]。)这样,环戊烷中的“CH2”原子将与任何其他使用 TraPPE 力场的分子中的任何“CH2”原子相互作用,并且行为相同。 （对于其他原子类型和“TraPPE”特有的相互作用类型也是如此,例如“@atom:TraPPE/CH3”、“@bond:TraPPE/CC”等.....另一个使用 TraPPE 力场将在后面的 10.5 节中讨论。）“trappe1998.lt”文件的重要部分如下所示：

10.4.1 命名空间示例

```
# -- 文件    trappe1998.lt    --

TraPPE
   { write_once（“数据量”){
      @原子:CH2 14.1707
      @原子:CH3 15.2507
   }
   write_once(   在设置中   ) { bond_coeff
      angle_coeff dihedral_coeff @dihedral:CCCC    谐波 120.0 1.54
      opls 1.411036 -0.271016 3.145033 0.000       谐波 62.0022 114
      pair_coeff @atom:CH2 @atom:CH2 lj/charmm/coul/charmm 0.091411522 3.95 pair_coeff @atom:CH3
      @atom:CH3 lj /charmm/coul/charmm 0.194746286 3.75 #（不同原子类型之间的交互使用混合规则。）


      # （混合样式用于便携性。）
   }
   write_once(   数据角度按类型   ) {
      @angle:CCC @atom:C* @atom:C* @atom:C* @bond:CC @bond:CC
   }
   write_once(   数据二面体按类型   ) {
     @二面体:CCCC @atom:C* @atom:C* @atom:C* @atom:C* @bond:CC @bond:CC @bond:CC
   }
}
```

除了原子类型名称和质量之外,该文件还存储了它们之间相互作用的力场参数（系数）。

按类型划分的键合相互作用

同样，“数据角度按类型”和“数据二面体按类型”部分告诉 moltemplate.sh 在任何 3 或 4 个连续键合的碳原子（CH2、CH3、

或 CH4)假设它们使用“CC”(饱和)键键合。 “*”字符是通配符。 “C*”匹配“CH2”、“CH3”和“CH4”。

(债券类型可以省略或替换为通配符“@bond:*”。)

命名空间和嵌套:

“CH2”和“CC”之类的名称非常常见。为了避免将它们与其他分子中类似命名的原子和键混淆,我们将它们(“嵌套”它们)包含在一个命名空间(在本例中为“TraPPE”)中。与“SPCE”和“Cyclopentane”不同,“TraPPE”不是分子。

它只是其他分子共享的原子类型、键类型和力场参数的容器。我们这样做是为了将它们与其他具有相同名称但含义不同的原子和键区分开来。在其他地方,我们可以将这些原子/键类型称为“@atom:TraPPE/CH2”和“@bond:TraPPE/CC”。 (您还可以避免为在 TraPPE 命名空间中定义的分子重复繁琐的“TraPPE/”前缀。例如,请参阅第 10.5 节。)

10.5 嵌套分子

在前面的 10.4.1 节中,我们创建了一个名为“TraPPE”的对象,并使用它创建了一个名为“Cyclopentane”的分子。这里我们用它来演示嵌套。假设我们定义了一个新分子“丁烷”,它由 4 个粗粒(联合原子)类碳珠组成,其类型分别命名为“CH2”和“CH3”。

```
# -- 文件 trappe_butane.lt --

导入“trappe1998.lt”

丁烷 { 写 (“数
  据原子”){
    $atom:c1 $mol:bt @atom:TraPPE/CH3 0.0 0.419372 0.000 -1.937329 $atom:c2 $mol:bt
    @atom:TraPPE/CH2 0.0 -0.419372 0.000 -0.645776 $atom:c3 $mol:bt @atom: TraPPE/CH2 0.0
    0.419372 0.000 0.645776 $atom:c4 $mol:bt @atom:TraPPE/CH3 0.0 -0.419372 0.0000 1.937329

  }
  写 (“数据债券”){
    $bond:b1 @bond:TraPPE/CC $atom:c1 $atom:c2 $bond:b2
    @bond:TraPPE/CC $atom:c2 $atom:c3 $bond:b3 @bond:TraPPE/
    CC $atom:c3 $原子:c4
  }
}
```

注意:同样,这不是使用力场的推荐方式。 (有关更清洁、更简单的方法,请参见第 10.7 节。)

如上所述,将我们的“丁烷”嵌套在“TraPPE”中可能更简单,这样它就不会与丁烷分子的其他(可能是全原子)表示混淆。在这种情况下,我们会

利用:

```
# -- 文件 trappe_butane.lt --

导入“trappe1998.lt”

陷阱{
  丁烷 { 写（“数
    据原子”）{
      $atom:c1 $mol:bt @atom:../CH3 0.0 0.419372 0.000 -1.937329 $atom:c2 $mol:bt @atom:../
      CH2 0.0 -0.419372 0.000 -0.645776 $atom:c3 $mol:bt @原子：../CH2 0.0 0.419372 0.000
      0.645776 $atom:c4 $mol:bt @atom:../CH3 0.0 -0.419372 0.0000 1.937329

    }
    写（“数据债券”）{
      $bond:b1 @bond:../CC $atom:c1 $atom:c2 $bond:b2
      @bond:../CC $atom:c2 $atom:c3 $bond:b3 @bond:../CC
      $原子:c3 $原子:c4
    }
  }
}
```

注意:同样,尽管此处解释的“嵌套”方法有效,但第 10.7 节演示了一种更简单、更好的使用力场的方法。

另请注意,将丁烷包装在“Trappe1998.lt”文件（之前已包含)中定义的“Trappe1998.lt”中定义的“Trappe1998.lt”对象中仅附加附加内容。它不会覆盖它。

再次“../”告诉 moltemplate 使用 TraPPE 环境上下文中定义的“CH2”原子（即上一级）。这确保了熔体板不会产生新的“CH2”原子类型,该原子类型位于丁烷分子的局部。 （同样,默认情况下,所有原子类型和其他变量都是本地的。

请参阅第 5.2.5 节。）

要在模拟中使用此丁烷分子,您将导入包含丁烷定义的文件,并使用“新建”命令创建一个或多个丁烷分子。

```
导入“trappe_butane.lt”新丁烷 =
TraPPE/丁烷
```

（在本例中,您不需要导入“trappe1998.lt”,因为它是在“trappe butane.lt”中导入的。）“Bu tane”之前的“TraPPE/”前缀让 moltemplate/ttree 知道丁烷是在本地定义的在 TraPPE 内。

注意:存在使用继承的替代过程,它可能是处理这些类型关系的更简洁的方法。请参阅第 10.7 和 10.7.1 节。

10.6 路径语法：“../”、“.../”和“$mol:.”

通常,可以使用多个斜线（“/”）以及（“../”）构建指示对象中任何其他分子的（相对)位置的路径

等级制度。 （“.”、“/”和“..”符号在这里的使用方式与它们用于在类 unix 文件系统中指定路径的方式相同。例如，“$mol”中的“.”：.”指的是当前分子（实例），就像“./”指的是当前目录一样。（注：“$mol”是“$mol:.”的简写）

斜线本身，“/”，指的是全局环境。这是
定义/创建所有分子的最外层环境。

10.6.1（高级）省略号“.../”

如果您正在使用多层嵌套，并且您不知道（或者如果您不想指定）特定分子类型或原子类型（例如“CH2”）是在哪里定义的，则可以参考它使用“.../CH2”而不是“../CH2”。 “...”省略号语法搜索嵌套分子树以找到目标（“/”斜杠后面的文本）。

10.6.2（高级）$mol:... 符号

回想一下，LAMMPS 允许用户选择为每个原子分配分子 ID。 （在水的例子（第 4.1 节）中，每个水分子中的原子都被分配了一个分子 ID，表示为“$mol:w”。在那个例子中，“w”是该分子 ID 的名称。）

如果您想使用较小的块作为构建块来构建大分子，moltemplate 有一种方法可以让所有原子共享相同的分子 ID。要引用您所属分子的 ID，请使用“$mol:...”。 （如果没有实例化当前分子对象的分子对象定义 $mol 类别中的变量，则将自动创建一个新的局部 $mol 变量。）这意味着“数据”的每一行的第二列Atoms”部分应包含“$mol:...”（假设使用“atom style full”或“molecular”）。

“...”语法在第 6.1 节中解释，更正式地在第 6.1 节中解释
10.6 和附录 G.1

10.7 继承

我们可以这样定义丁烷：

导入“trappe1998.lt”

```
丁烷继承 TraPPE { write(    Data
   Atoms   ){
      $atom:c1 $mol:bt @atom:CH3 0.0 0.419372 0.000 -1.937329 $atom:c2 $mol:bt
      @atom:CH2 0.0 -0.419372 0.000 -0.645776 $atom:c3 $mol:bt @atom:CH2 0.0 0.419372
      0.000 0.645776 $atom:c4 $mol:bt @atom:CH3 0.0 -0.419372 0.0000 1.937329

   }
   write(   数据键   ){ $bond:b1
      @bond:CC $atom:c1 $atom:c2 $bond:b2 @bond:CC
      $atom:c2 $atom:c3
```

```
        $bond:b3 @bond:CC $atom:c3 $atom:c4
    }
}
```

“继承”关键字使丁烷具有“父”对象（TraPPE)的所有属性（例如原子类型、键类型、分子定义）。以这种方式定义丁烷意味着您可以直接访问这些分子类型、原子类型和键类型。例如,您可以参考 CH3 原子类型“@atom:CH3”而不是“@atom:TraPPE/CH3”或“@atom:../CH3”。 （消除不必要的冗余文本,例如“TraPPE/”和“../”,减少了拼写错误或拼写错误的机会。)这是在 moltemplate 中使用力场的首选方式。

10.7.1 多重继承：

一个分子可以从多个父母那里继承。这是允许丁烷分子从多个不同的力场母体借用原子、键、角度、二面体和不正确类型的一种方法：

```
进口“trappe1998.lt”进口
“oplsaa.lt”

丁烷继承 TraPPE OPLSAA {
    ...
}
```

详细信息:如果在双亲中都找到重复的原子类型或分子类型,Moltemplate 会尝试解析它们,优先考虑“继承”关键字后面的双亲列表中的第一个父代。 （本例中为“TraPPE”。）

10.7.2 继承与嵌套

如果两个分子以这种方式相互关联：“A 是 B 的特定类型”,那么请考虑使用继承而不是嵌套（或对象组合）。

在此示例中（使用 Butane 和 TraPPE）,嵌套或继承都可以工作（但继承更简洁、更安全）。

同样,将丁烷嵌套在 TraPPE 中的一个非常小的优势是,它可以防止丁烷名称与其他地方定义的任何其他版本的丁烷分子混淆或冲突。 （通常这不是考虑因素。）

10.7.3 继承与对象组合

另一方面,如果两个分子以这种方式相互关联：“A 由 B 和 C 组成”,那么您可能会考虑使用对象组合而不是继承。例如：

```
import    B.lt    # <-- 定义分子类型    B

import    C.lt    # <-- 定义分子类型    C
```

```
一个 {
    b = 新 B
    c = 新 C
}
```

11 已知错误和限制

请将您发现的任何错误通过电子邮件报告给 lammps- jewett.aij@gmail.com，或到 users 邮件列表。

1) 不支持 LAMMPS 样式的分子模板。由 moltemplate 创建的 DATA 文件的格式不正确，无法被 LAMMPS 分子命令读取。 （这是因为该命令是在编写 moltemplate 之后添加的。）但是格式相似，可以使用文本编辑器提取相关信息并转换为其他格式。 （使用文本编辑器和 awk 或电子表格程序。

有关这些文件格式的更多信息，请访问https://docs.lammps.org/read_data.html https://docs.lammps.org/molecule.html 。）同样，请随时联系以请求对 LAMMPS 式分子的支持模板。
jewett.aij@gmail.com

2) Moltemplate 消耗大量内存（RAM）

内存使用与系统大小成正比增长。截至 2019 年 9 月 3 日，使用 moltemplate 建立一个包含 1000000 个原子的系统目前需要 2.7 到 12 GB 的可用内存。 （具有许多键和角度的系统会消耗更多的内存，以及具有高分子数的系统。）

不幸的是，这段代码没有仔细编写以尽量减少内存使用时间。 （此外，python 程序需要的内存是用 C/C++ 编写的类似程序的 10 倍以上。）

这个问题可以通过使用其他具有较低内存占用的 Python 解释器来缓解。或者，可能需要将大型系统拆分为多个部分，在每个部分上运行 moltemplate，然后将生成的数据文件合并为一个大型数据文件。

此外，可以租用具有适量 RAM 的计算机
便宜。 （例如，请参阅https://cloud.google.com/compute/。）

使用 moltemplate 设置大型模拟时，请考虑使用“ulimit”命令来防止系统崩溃。 （如果您在共享计算机上，请让管理员执行此操作。）如果这些选项不可用，您始终可以在启动 moltem 板之前运行资源监视器（如“top”）并在内存使用情况下终止进程超过 80%。

3) 对非点状原子的有限支持：截至 2019 年 9 月 3 日，仅支持“完

整”、“角度”、“原子”、“电荷”、“球体”、“偶极子”、“椭球体”、和“分子”风格已经过测试。其他非点状原子，如“tri”、“line”也应该与 moltemplate 一起使用。这些对象如何无法通过“.rot()”命令正确旋转（或通过“.scale()”命令正确缩放）。更奇特的奇异原子样式，例如“wavepacket”、“electron”、“sphere”和“peri”尚未经过测试。

此外，不支持 atom style body 和 atom style template。 –

随意联系风格。 jewett.aij@gmail.com 请求支持外来原子

4) 当放置在行尾时,LAMMPS 将“&”字符解释为将两行合并在一起的请求。在 moltemplate write() 或 write once() 命令中使用这个字符通常是安全的。但是在极少数情况下,使用“&”字符将两行连接在一起可能会使 moltemplate 感到困惑。例如,在 lammps 输入脚本命令中（如“pair coeff”或“dihedral coeff”），“&”字符不应出现在最后一个“@”或“$”变量被引用之前。还要避免在“数据原子”、“数据键”、“数据角度”、“数据二面体”、“数据不正确”、“数据角度按类型”、“数据二面体按类型”中的任何地方使用“&”字符,和“按类型划分的数据不当”部分。

5) 三斜边界条件尚未测试:截至 2019 年 9 月 3 日,对具有三斜单元的 PDB 文件的支持是实验性的。
如果它不起作用,请告诉我。

6) 对通配符（“*”和“?”)的支持不一致 通配符“*”在 LT 文件的不同部分有不同的解释。
通配符在“键系数”、“角度系数”、“二面体系数”、“不正确系数”和大多数“配对系数”命令以及任何“按类型”命令中可靠地工作并用于字符串模式匹配” LT 文件中的部分（例如“按类型划分的数据角度”、“按类型划分的数据二面体”和“按类型划分的数据不正确”）。然而,这些通配符不在需要超过 2 个原子类型作为参数的 pair coeff 命令中。（例如“pair style hbond/dreiding/lj”。但是许多使用“pair coeff * *”符号的体对样式工作正常。）

附录

A Bonded 交互“按类型”

LAMMPS 中未键合的原子之间的相互作用（即“非键合”或“配对”相互作用)由原子类型指定。
LAMMPS 中的键合相互作用（包括三体角、四体二面角和不正确的相互作用)由唯一的原子 ID 号指定。 （典型分子中通常有大量的角度和键,这些信息占据了典型 LAMMPS 数据文件的大部分内容。）

这在 moltemplate.sh 中发生了变化。 moltemplate.sh 包含一个实用程序,它可以根据原子和键类型自动生成角度、二面体和不正确。 （此实用程序在 F 节中描述。)moltemplate.sh 将检查系统中存在的键网络,检测所有 3-body 和 4-body 相互作用,并确定它们的类型。 （用户也可以定义更高的 n 体相互作用。)以这种方式指定相互作用可以消除显着的冗余,因为许多原子共享相同的类型。

要利用此功能,您将创建一个名为“Data Angles By Type”、“Data Dihedrals By Type”或“Data Impropers By Type”的新部分,其语法模仿“Angles”、“Dihedrals”和“Impropers”” LAMMPS 数据文件的部分。语法最好通过示例来解释：

```
write(  数据角度按类型  )
  { @angle:XCXgeneral @angle:CCCgeneral  *C*  *
  @angle:CCCaturated @atom:C @atom:C @atom:C @atom:C  *  *
  @atom:C @bond:SAT @bond:SAT
}
```

第一行将生成一个三体角度交互（类型为“@angle:XCXgeneral”）
只要第二个原子的类型名称包含字母“C”,任何 3 个连续键合的原子之间。 （原子和键类型名称可以包含通配符 *）

第二行将生成“@angle:CCCgeneral”类型的三体交互
在任何 3 个“@atom:C”类型的原子之间,无论连接它们的键类型如何。 （最后两列都是通配符 *，告诉 moltemplate.sh 忽略这两种键类型。由于这是默认行为,这两列是可选的,可以省略。）

第三行将生成“@angle:CCCaturated”类型的三体交互
在“@atom:C”类型的任何 3 个原子之间,如果它们通过“@bond:SAT”类型的键连接。

注意 :本例中的第 2 行和第 3 行将生成新的 interac
可能会覆盖之前分配的任何角度交互的选项。

A.1 正则表达式

正则表达式也可用于匹配潜在的原子和键类型。要使用正则表达式，“:”后面的前 3 个字符应该是“re.”,变量名应该用大括号 {} 括起来。例如:@{atom:re.C[1-5]},应该匹配@atom:C1 到 @atom:C6。

（注意:Moltemplate 无法理解正则表达式,除非它们中的 { 和 } 字符的数量是平衡的。（或者如果你在它们前面加上反斜杠字符）。Moltemplate 也无法理解包含空格的正则表达式。如果你违反这些规则,moltemplate 会变得如此混乱,以至于它可能会为您提供令人困惑或误导性的错误消息。）

以类似的方式,可以在“Dihedrals By Type”和“Impropers By Type”中使用正则表达式。

注意:这是一个实验性功能。截至 2020 年 11 月 4 日,正则表达式尚未经过测试。

B 使用 ltemplify.py 创建一个 LT 文件

“ltemplify.py”脚本用于将 LAMMPS 数据文件和输入脚本转换为单个 MOLTEMPLATE (“LT”) 文件。

通常,由“ltemplify.py”生成的 LT 文件包含数据文件中存在的单一类型分子（或分子复合物)的定义。这样,moltemplate 用户以后可以使用该分子作为构建块（可能与其他分子一起)构建复杂的模拟。

用户可以使用“-mol”、“-id”或“-type”参数选择他们想要的分子（或多个分子）,并使用“-name”参数为分子命名。 （见下文。)生成的 LT 文件将包含与该分子相关的所有信息,包括原子类型、电荷、坐标、键合相互作用、力场参数、力场样式、基团和影响分子的固定。 （其他信息将省略。）

但是,默认情况下，“ltemplify.py”会将这些文件中的所有信息复制到描述整个系统的 LT 文件中。（稍后,当 moltemplate.sh 在该 LT 文件上运行时,它会尝试重新生成所有原始 LAMMPS 文件。)通常，这不是很有用。

典型用法

ltemplify.py -name MoleculeName -mol MolID INPUT_SCRIPT DATA_FILE > FILE.lt

...其中 MoleculeName 是一个字符串,MolID 是一个整数,INPUT SCRIPT 和 DATA FILE 是 LAMMPS 输入脚本和包含感兴趣分子的数据文件的名称,FILE.lt 是由 ltemplify 创建的结果 MOLTEM PLATE 文件.py。 （有关详细信息,请参阅第 B.1 节。）

注意:令人厌烦的细节。

初次阅读的读者可能应该跳到第 B.4 节中的示例。

必需的参数

“ltemplify.py”只需要一个参数:LAMMPS 数据文件的名称。但是（如上面的示例所示）,它还读取 LAMMPS 输入脚本。 （注意:如果包含 LAMMPS 输入脚本,它们必须

出现在参数列表中的 DATA 文件之前。请参见下面的示例。）“ltemplify.py”还接受许多参数来选择属于感兴趣分子的原子并自定义输出。

LAMMPS 输入脚本

除了 LAMMPS DATA 文件，用户还可以提供一个或多个 LAMMPS 输入脚本（通常包含与该分子相关的信息，例如力场参数和修复）。如果包含 LAMMPS 输入脚本，它们必须出现在参数列表中的 DATA 文件之前。

（见下面的例子。）

B.1 可选参数

参数 -name	意义
NAME	指定输出中描述的分子的名称 文件。 （可选的“继承”关键字可用于选择 一个力场。）
-atomstyle 风格	可选:强制 ltemplify.py 使用特定的原子样式。 （混合样式应该用引号括起来,例如 -atomstyle “混合全偶极子”。）
-columns “列列表”	对于自定义原子样式,您可以在 手动“原子”部分。 （例如：-列 “molid xyz atomid atomtype mux muy muz”）
-mol “molID 列表”	选择要出现在结果中的分子 LT 文件。要选择多个分子,请提供一个列表 用引号括起来的数字。
-id “身份证”	通过原子 ID 选择原子。使用引号包围 原子 ID 号列表。未选择的原子将被省略。
-type“类型列表”	按类型选择原子。使用引号包围列表 原子类型数。
-数据系数	将力场信息放在数据文件中,而不是在输入中 脚本。 （默认情况下,力场参数会放在 最终将被写入的“In Settings”部分 LAMMPS 输入脚本。）
-忽略评论	不要推断原子、键、角度、二面体和不正确的类型 数据文件中注释的名称。
-忽略系数 -忽略角	忽略所有力场参数（系数）。
度	忽略角度、二面角和不正确。从输出中省略 文件。 （这在使用外力场时很有用,例如 OPLSAA.)
-忽略键类型	忽略“债券”部分的第 2 列 LAMMPS 数据文件,并在 生成的 MOLTEMPLATE LT 文件省略了键 类型。 （这在使用外力场时很有用,例如 作为 OPLSAA。）
-忽略质量	忽略数据文件中的所有质量。从输出文件中省略。 （这在使用外力场时很有用,例如 OPLSAA.)
-prepend-atom-type STR 将字符串	从 STR 参数添加到开头 所有原子类型名称。

显示参数用法的示例包含在 B.4 节中。

默认行为

请注意,默认情况下（如果省略“-mol”、“-id”或“-type”参数），“ltemplify.py”会将 LAMMPS 文件中的所有信息复制到描述整个文件的 LT 文件中系统。通常,这不是很有用。

细节

所有原子、键、角、二面体和不正确及其相关类型都将转换为 moltemplate “$” 或 “@” 计数器变量（并且每个文件的相关部分将移动到具有正确标题名称的部分）。系数、原子样式和大多数力场样式和设置也应包含在生成的 .LT 文件中。 ltemplify.py 还可以理解简单的组命令（使用“id”、“molecule”或“type”样式)和“修复抖动”、“修复嘎嘎声”和“修复刚性”（未经测试 2019-9-03）。

但是,大多数其他修复程序和复杂的组命令都无法理解。
这些命令必须手动添加到生成的 .LT 文件中。 （有关详细信息,请参阅第 B.5 节。）

B.2 修正和组

ltemplify.py 对“修复”和“组”命令的支持有限,包括“修复抖动”、“修复嘎嘎声”、“修复刚性”和“修复诗歌”。其他修复必须手动添加到 ltemplify.py 生成的文件中。（如修复“约束”、“绑定/创建”、“绑定/中断”、“绑定/反应”、“ttm”等...）

ltemplify.py 可以理解简单的（静态）“组”命令,如果它可以确定它们包含任何相关的原子,则将它们包含在输出文件中。 （根据不相关组的修复也被删除。）

注意:此功能未经仔细测试。因此,请检查所有组并修复 ltemplify.py 生成的命令,以确保它们引用正确的原子。请报告您发现的任何错误。 (-安德鲁 2019-9-03)

自动生成原子、键、角、二面角、不正确
名字

默认情况下,ltemplify.py 会自动生成原子、键、角度、二面体和不正确的类型名称和 id 名称。这导致原子的类型像“@atom:type3”,ID 像“$atom:type3 7”（即类型 3 的第 7 个原子。）

从注释中推断原子类型名称

但是,ltemplify.py 使用 LAMMPS 数据文件（如果存在)的“质量”部分中的注释来确定每个原子类型的名称。考虑以下假设数据文件的摘录：

群众

1 12.01 #c3

2 1.008 # h3
3 1.008 # 豪
4 16.00#哦

这意味着类型 1、2、3 和 4 的原子将在 moltemplate 中分别称为“@atom:c3”、“@atom:h3”、“@atom:ho”和“@atom:oh”（LT）文件由 ltemplify.py 创建。

忽略评论

“-ignore-comments”参数将禁用此行为并以通常的方式为原子类型分配数字名称（例如“@atom:type1”、“@atom:type2”、“@atom:type3”、“@atom :type4”）。

键、角、二面角和不正确的类型名称

同样，默认情况下，键和角度会自动分配给类型名称，例如“@bond:type4”、“@angle:type7”。

但是，如果注释直接出现在头文件“Nbond types”的行之后，那么这些注释将被解释为键类型名称的列表（可选地以整数开头）。 （角度、二面体和不正确的类型名称也是如此。）考虑以下来自 LAMMPS 数据文件的摘录：

2 种原子类型 #c3

h3

2种债券类型
#CC乙烷
c3_h3

2 种角度类型 #
c3_c3_h3 # h3_c3_h3

在此示例中，类型 1 和 2 的键将被称为“@bond:CCethane”
和 moltemplate 文件中的“@bond:c3 h3”。类似地，类型 1 和 2 的角度将分别称为“@angle:c3 c3 h3”和“@angle:h3 c3 h3”。 （与前面的示例一样，类型 1 和 2 的原子将分别称为“@atom:c3”和“@atom:h3”。您可以在此处或在质量部分中指定原子类型字符串。）

（和以前一样，“-ignore-comments”参数将禁用此行为。）
如果您忘记在运行 ltemplify.py 之前向 LAMMPS 数据文件添加注释，您始终可以使用文本编辑器（或 sed）手动查找“@atom:type1”的所有实例并将其替换为更有意义的内容，例如“ @atom:c3”，例如。

B.3 力场

一些数据文件包含角度、二面角或不正确的键合相互作用的列表。如果是这样，则默认情况下 ltemplify.py 将在它创建的 moltemplate (LT) 文件中包含此信息。有时，数据文件缺少此信息。

无论哪种方式，力场（包括“OPLSAA”、“GAFF2”和“COM PASS”）都包含自动生成这些交互的规则。因此，当此信息包含在他们想要使用的力场中时，用户可能有意从 ltemplify.py 生成的熔印板文件中排除此信息。（他们可以使用下面解释的“-ignore-coeffs”、“-ignore-angles”和“-ignore-bond-types”参数来做到这一点。）

使用继承关键字指定力场

Moltemplate 提供了几种不同的力场可供选择（例如 OPLSAA、GAFF2 或 COMPASS）。此外，用户可以创建自己的自定义力场。要使用这些力场，您必须使用带有 inherits 关键字的 -name 参数指定要使用的力场（“-name ”MOLECULE NAME 继承 FORCE FIELD”），例如：

```
ltemplify.py -name Ethane 继承 GAFF2 \ -ignore-coeffs
                \ ethane.data > ethane.lt
```

这将要求 ltemplify.py 创建一个定义名为“Ethane”的分子的文件。
稍后当使用 moltemplate 读取此文件时，“GAFF2”力场将用于生成角度、二面角和不正确，并查找它们的力场参数。

此外，在 ltemplify.py 完成后，用户必须在 ltemplify.py 创建的文件的开头手动插入以下行。例如：

```
导入“gaff2.lt”                    #<-- 定义 GAFF2 力场

# --- 下面的文本是由 ltemplify.py 生成的 --- Ethane 继承了 GAFF2 {

  ...
}
```

ltemplify.py 不会为您执行此操作。可以在 github 上随 moltem 板分发的“moltemplate/force fields/”目录中找到可用力场的列表。

-忽略系数

可选的“-ignore-coeffs”参数将强制 ltemplify.py 忽略它在用户输入脚本或 DATA 文件中遇到的力场参数。生成的 LT 文件将省略此信息。如果您计划对该分子使用力场，那么此信息将出现在您正在使用的力场中，因此无需将其包含在生成的 LT 中

您现在正在创建的文件。 （稍后当您在 ltemplify.py 创建的 LT 文件上运行 moltemplate.sh 时，它将使用力场来查找这些力场参数。）

-忽略角度

如果原始 DATA 文件有“Angles”、“Dihedrals”或“Impropers”，如果您想强制 ltemplify.py 从 ltemplify 的 LT 文件中忽略/删除这些交互，可以使用“-ignore-angles”参数创建。

（这样做将允许稍后我们在该文件上运行 moltemplate.sh 时力场规则优先。）

-忽略键类型

类似地，当使用力场时，您只需要指定一个列表，其中列出了哪些原子对结合在一起。力场将根据原子类型名称和力场规则确定每个键的类型和性质（例如，平衡静止长度、刚度等）。

为此，您必须强制 ltemplify.py 使用“-ignore-bond-types”参数忽略数据文件中存在的现有键类型信息。这将强制 ltemplify.py 忽略您提供的 LAMMPS 数据文件的（第二列）“债券”部分中的债券类型。

这样，以后可以通过 moltemplate.sh 以与您选择的力场一致的方式确定键类型。

有关示例，请参见第 B.4 节。

免责声明

ltemplify.py 是实验软件。 lemplify.py 脚本对 LAMMPS 中所有可用功能的理解有限。请查看生成的“.LT”文件并检查错误。 （如有必要，将任何剩余的原子、键、角度、二面体或不正确的 id 或类型数字转换为相应的 $ 或 @ 变量。）一些具有自己特殊语法的奇异对样式无法理解。这些系数必须手动转换。对“组”和“修复”命令的支持也受到限制。 （参见 B.2 节。）请报告 ltemplify.py 行为中的错误。

B.4 示例

示例 1

```
ltemplify.py -name Ethane -molid    1    FILE.in FILE.data > ethane.lt
```

此示例创建一个新文件（“ethane.lt”），其中包含一种新型分子（名为“Ethane”），由分子 ID 编号等于 1 的所有原子组成。（推测 FILE.data 中的第一个分子是乙烷分子。） ltemplify.py 从“FILE.data”中读取原子坐标和键合相互作用。与该分子相关的其他信息（包括

原子样式、力场样式和参数、组和修复)从“FILE.in”（大概是一个 LAMMPS 输入脚本文件)中读取。

（注意:同样,没有必要在参数列表中包含 LAMMPS 输入脚本。但是重要信息通常包含在 LAMMPS 输入脚本文件中,因此如果您有,建议包含它。

但是一个数据文件就足够了。）

注意:仅当您使用“分子”原子样式之一（例如“原子样式完整”）时,按分子 ID 选择原子才有效。如果您使用不同的原子样式（例如“原子样式角度”或“原子样式键”）,您可以通过类型或 ID 号选择所需的原子。 （见下文。）

示例 2

有时,描述您的分子的信息将分为多个灯输入脚本。 （例如,一个输入脚本可能包含各种样式命令。下一个输入脚本可能包含 coeff 命令。)在这种情况下,这些输入脚本应该出现在数据文件之前的参数列表中,并且按照它们出现的顺序由 LAMMPS 读取。

```
ltemplify.py -name Ethane \ -molid
                    1    \
                FILE1.in FILE2.in FILE.data > ethane.lt
```

示例 3

```
ltemplify.py -name Ethane -molid    1    \ -id    13
                14 15 61*69    \
                FILE.in FILE.data > ethane.lt
```

在此示例中,仅包括 id 为 13、14、15 和 61 到 69 的原子。

示例 4

```
ltemplify.py -name Ethane \ -atomtype
                    1 2 3    \ FILE.in FILE.data
                > ethane.lt
```

在此示例中,仅包括类型为 1、2 或 3 的原子。

示例 5

```
ltemplify.py -name EntireSystem FILE.in FILE.data > whole_system.lt
```

这将为一个新的分子对象（名为“EntireSystem”)创建一个模板,该模板由您包含的 lammps 文件中的所有原子组成,并将此数据保存在单个 LT 文件（“entire system.lt”)中。该文件可以与 moltemplate.sh（和/或 ttree.py)一起使用来定义包含该分子的大型系统。

注意:同样,输入脚本（本例中为“FILE.in”)应出现
在参数列表中的数据文件（“FILE.data”)之前。

您还可以使用 ltemplify.py 创建使用 3rd-party 力量的分子
OPLSAA、GAFF2、COMPASS 等字段.....

例 6

此示例演示如何使用“GAFF2”力场构建分子。以下示例从“FILE.in”和“FILE.data”中提取分子 1。

此示例创建一个新文件“ethane.lt”,其中将包含 # 使用“GAFF2”构建“Ethane”分子的说明。首先#指定哪个文件包含“GAFF2”力场的定义：

回声“进口 gaff2.lt”> 乙烷.lt

然后使用ltemplify.py从FILE.in、FILE.data中提取信息

ltemplify.py -name Ethane 继承 GAFF2 \ -molid 1 \
-ignore-angles -ignore-bond-types -ignore-
coeffs \
FILE.in FILE.data >> ethane.lt

注意:如果你想建立一个包含这些分子的模拟,#你必须创建一个引用“ethane.lt”的“system.lt”文件#然后在这个文件上运行moltemplate.sh。

如前所述，“file.data”中的注释将决定名称
每个原子类型的名称,并且应该匹配力场中的原子类型名称。
在这个例子中,角度、二面角、不正确和键型信息从原始文件.data 中删除（稍后将根据“GAFF 2 力场中定义的规则生成）。分子的名称（“乙烷继承 GAFF2”）包括对力场（“GAFF2”）的引用,该力场将用于查找此信息。 （注意：“GAFF2”力场参数通常在名为“gaff2.lt”的文件中定义。因此在本例中,我们使用“echo”在“ethane”开头插入“gaff2.lt”链接。 lt” 文件,以便 moltemplate.sh 知道在哪里可以找到它们。或者,这可以由用户手动完成。）

B.5 已知错误和限制 (ltemplify.py)

不支持异国风格

ltemplify.py 不理解奇异多体对样式的语法,例如 tersoff、sw、meam、reax、dpd、edip、dipole、lubricate、hbond/dreiding（即使 moltemplate 支持这些样式）。运行 ltemplify.py 后,用户必须手动编辑生成的“.lt”文件。例如:ltemplify.py 将无法理解通配符（“*”字符）,这些通配符（“*”字符)在使用这些多体对样式时通常出现在“pair coeff”命令或“Pair Coeffs”部分。您将不得不删除

ltemplify.py 自动生成额外的行并手动放回通配符（例如“pair coeff * * ...”）。 （稍后用户可能需要使用适当的“-a”命令行参数运行 moltemplate,以确保将各种原子类型分配给正确的编号。

这通常是为了使它们与相应对样式的输入文件中的参数顺序保持一致。见 D.1 节。）

此外,辅助原子类型（例如 hbond/dreiding 所需的“氢”原子类型)甚至不会被解析。如果您使用“hbond/dreiding”对样式,则必须在运行 ltemplify.py 后在每个“pair coeff”命令中手动指定氢原子介体的原子类型

通配符（“*”)扩展

如第 11 节所述,当通配符（“*”字符)出现在任何“coeff”命令（或数据文件的“Coeff”部分）中时,moltemplate 经常被混淆。因此 ltemplify.py 尝试删除这些字符并扩展这些命令,生成多行输出,并明确列出每个原子类型。 （这也适用于键类型、角度类型、二面角类型和不正确的类型。)这可能不是您想要的。 （例如,如果您对原子类型使用多个“* *”,这可能会出现问题,例如需要您指定为 tersoff、eam 或 sw 的主体对样式。）

VMD 中的 C 可视化

本附录仅旨在为您提供使用 VMD 显示分子所需了解的快速、最少的功能列表。这些说明是为 VMD 1.9 和 topotools 1.2 编写的。有关高级 VMD 功能、分析和渲染选项,请参阅位于http://www.ks.uiuc.edu/Research/vmd/current/docs.html的官方 VMD 文档

C.1 在 VMD 中自定义外观

默认情况下,VMD 可能会用点和线来显示你的分子,这可能很难看而且很难看。要改变分子的外观,请选择 Graphics→Representations... 菜单,然后从 Drawing Method 下拉菜单中选择一个选项。默认情况下,原子按原子类型着色。您可以通过图形→颜色自定义每种原子类型的颜色... 截至 2019-9-03,VMD 任意允许您仅将颜色分配给前 9 种原子类型。但是,您可以使用多个表示自定义剩余原子类型的外观来解决此限制（如下所述）。

您可能希望对不同的分子或原子类型使用不同的表示。为此,请选择 Graphics→Representations... 菜单,然后单击 Selections 选项卡。然后单击 Create Rep 按钮以创建系统的多个“表示”。对于每个表示,您可以选择不同的集合原子,并使用不同的绘制样式,对于那些

原子。例如,您可以通过从 Coloring Method 下拉菜单中选择 ColorID 手动自定义这些原子的颜色。然后,在此菜单的右侧,您可以选择颜色(由数字表示)。这将影响当前表示中的所有原子。您还可以选择不同的绘制样式并更改原子和键半径。

您可以通过单击“创建代表”按钮下的列表从已创建的表示列表中进行选择。 (双击暂时隐藏视图中的表示。)

同样,每个表示通常被分配给系统中不同的原子子集。要指定每个表示中的原子,请单击“选择”选项卡。默认情况下选择“所有”原子,但是您可以根据原子类型、指数、分子、电荷、质量、x、y、z 选择原子。这会将当前显示设置限制为系统中存在的原子/键的子集。选择原子时,可以使用复杂的布尔表达式(包含一个或多个 and 和 or 运算符和括号)。

有关更多信息和一些示例,请参阅http://www.ks.uiuc.edu/Research/vmd/vmd-1.9/ug/node19.html和http://www.ks.uiuc.edu/Research/vmd/ vmd-1.9/ug/node87.html#ug:topic:selections。

注意:在 VMD/topotools 中,每个原子的类型、索引和 molid 属性对应于 moltemplate 中每个原子的@atom、$atom 和 $mol 变量。不幸的是,VMD 不理解 moltem 板变量命名语法(在 5.2 节中讨论)。相反,在 VMD 中,变量必须由它们的等效数字来指定。您可以通过阅读输出 ttree/ttree assignments.txt 文件来确定这些数字。 (有关详细信息,请参阅 D.1 节。)该文件包含一个表格,其中包含分配给每个 @atom(类型)、$atom(id)和 $mol(molecule-id)变量的数字列表。

C.2 可视化周期性边界

要查看周期框边界,请选择 Extensions→Tk Console 菜单,然后在 Tk Console 窗口中输入:

```
pbc盒子
```

请注意,您系统中的分子可能不在此框内。您可以使用以下命令将它们包装在盒子内:

```
pbc 换行 - 复合 res -all
```

您可能希望将框围绕一个分子居中。有几种方法可以做到这一点。您可以通过以下方式手动移动框:

```
pbc wrap -compound res -all -shiftcenterrel {0.0 0.15 0.0} pbc box -shiftcenterrel {0.0
0.15 0.0}
```

这将使框的位置在 Y 方向上移动 15%。 (距离以箱长分数为单位,而不是埃。)

(高级用法:如果你有一个原子都是“1”型的溶质,周围是“2”型原子的溶剂,那么你也可以尝试使用以下方法将盒子围绕它居中:“pbc wrap -sel type =1 -all -centersel 类型=2

-center com”。 “1”和“2”是由 moltemplate 分配的@atom 类型编号。这可以在输出 ttree/ttree assignments.txt 文件中找到。如果您正在查看轨迹,那么这将修改轨迹中每一步的外观,使框以溶质原子为中心。）

有关可视化周期性边界的更多详细信息,请访问： http://www.ks。 uiuc.edu/Research/vmd/plugins/pbctools为了防止原子重叠,您还应该检查您的周期性边界是否

条件太小。要做到这一点：

a) 选择 Graphics→Representations 菜单选项 b) 单击“Periodic”选项卡,然后

c) 单击 +x、-x、+y、-y、+z、-z 和 self 复选框。

这样做时,检查系统以确保出现的原子在空间中占据不重叠的体积。

D 高级 moltemplate.sh 用法

moltemplate.sh 有几个可选的命令行参数。这些解释如下：

用法：

```
moltemplate.sh [-atomstyle 样式] \
                    [-pdb/-xyz/-raw coord_file] \ [-a
                    assignments.txt] file.lt
```

可选参数：

-atomstyle 样式 默认情况下,moltemplate.sh 假定您使用的是“完整”
LAMMPS 中的原子样式。您可以使用 -atomstyle dipole 将原子样式更改为“偶极子”。如果您使用混合样式,则必须将样式列表括在引号中。例如：-atomstyle 混合全偶极子

对于自定义原子样式,您还可以手动指定列名列表（用引号括起来）： -atomstyle
molid xyz atomid atomtype mux muy muz

小心将整个列表括在引号 ( ) 中。

-raw raw_file raw_file 文件应该包含 RAW 格式的原子坐标
RAW 文件是简单的 3 列 ASCII 文件,其中包含系统中原子的坐标。 （每个原子一行,每行 3 个数字。

原子必须以相同的顺序出现在数据文件中。）

-xyz xyz_file 应提供 xyz_file 参数作为参数
在“-xyz”之后。
该文件应包含 xyz 格式的原子坐标。
（原子必须以相同的顺序出现在数据文件中。）

-pdb pdb_file pdb_file 文件应包含 PDB 格式的原子坐标

该文件应包含每个原子一个 ATOM 或 HETATM 记录。原子

按chainID、resID、insertCode、atomID（按此顺序)排序。
此顺序必须与原子在数据文件中出现的顺序相匹配。

如果 PDB 文件包含周期性边界框信息（即，“CRYST1”记录），则该信息也会复制到 LAMMPS 数据文件中。

（截至 2019 年 9 月 3 日,对三斜细胞的支持是实验性的。
稍后可能会支持其他分子结构格式。

-a @atom:x 1

-a assignments.txt

用户可以使用 -a VARIABLE_NAME VALUE 自定义分配给原子、键、角度、二面体和不正确类型或 id 编号的数字

对于您要修改的每个变量。如果要修改的变量很多,可以将它们保存在一个文件中（每行一个变量）。有关文件格式的示例,请运行 moltemplate.sh 一次并搜索名为“ttree_assignments.txt”的文件。 （此文件通常位于“output_ttree/”目录中。)一旦分配,同一类别中的其余变量将自动分配给不与您选择的值重叠的值。

-b assignments.txt

“-b”类似于“-a”。但是,在这种情况下,不会尝试为每个变量分配独占（唯一)值。

-nocheck

通常 moltemplate.sh 会检查常见的错误和拼写错误,如果它认为找到了,就会停止。这迫使变量

和类别以及 write(file) 和 write_once(file) 命令遵守标准命名约定。 “-nocheck”参数绕过这些检查并消除了这些限制。

-checkff

这会导致 moltemplate.sh 检查以确保为系统中每 3 或 4 个连续键合的原子定义了有效的角度和二面体相互作用（在“角度/二面体类型”中定义）。

D.1 手动变量分配（“-a”或“-b”）

可以手动自定义分配给原子类型（或任何其他 ttree 样式变量)的值。例如,考虑前面显示的“spce.lt”文件。该文件定义了具有两种原子类型（氢和氧)的单个水分子。通常，“O”原子类型通常分配给整数“1”，而“H”将分配给“2”。这是因为在该文件中“O”出现在“H”之前。如果你想换

顺序,您可以交换它们首次出现的顺序。

或者,您可以使用一个或直接指定原子分配
更多“-a”标志后跟带引号的赋值字符串:

```
moltemplate.sh -a  @atom:SPCE/O 2  system.lt
```

这将氧原子类型分配给“2”。请注意,必须在 @atom:SPCE/O 2 字符串周围加上引号,这是一个单独的参数。
(另请注意,有必要在 O 之前包含 SPCE/,因为在该示例中,该原子出现(并因此被定义)在 SPCE 分子的环境中。或者,如果它已在外部全局定义,那么您可以参考使用“@atom:O”)

变量不需要分配给数字。如果出于某种原因,你想在这个原子类型出现的任何地方替换“一个字符串”,你可以这样做:

```
moltemplate.sh -a  @atom:SPCE/O  a string  system.lt
```

可以使用多个“-a”标志进行多个分配:

```
moltemplate.sh -a  @atom:SPCE/O 2  -a  @atom:SPCE/H 1  system.lt
```

但是,如果您要进行大量分配,将它们存储在文件中可能会更方便。您可以创建一个包含两列的文本文件(例如“new assignments.txt”)并以这种方式运行 moltemplate:

```
moltemplate.sh -a new_assignments.txt system.lt
```

本例中“new assignments.txt”文件的内容为:

```
@atom:SPCE/O 2
@atom:SPCE/H 1
```

此文件中的行顺序无关紧要。

分配 $angle, $dihedral, $improper 变量

一般来说,任何类型的变量都可以通过这种方式分配(不仅是原子类型),包括 $mol、$bond、@bond、@angle、$angle ……以及用户定义的变量类型。警告:唯一偶尔的例外是 $angle、$dihedral、$improper 变量。 (当用户选择“Angles By Type”交互,并与常规“Angles”混合时,所有 $angle 变量都会自动生成。“Dihedrals By Type”和“Impropers By Type”也是如此。参见F 部分解释了“按类型”交互。)

角度、二面角和不适当的交互是自动生成的,在这种情况下,用户没有分配这些变量的自由。

“-b”标志

请注意,当使用上面的“-a”标志时,将注意确保分配是独占的。没有任何原子类型（@atom:SPCE/O 除外)将被分配“2”。 （出于这个原因,使用“-a”标志来更改原子类型分配原则上可以更改分配给其他原子类型或变量的数字。)这通常是所需的行为。

但是,假设出于某种原因,您想强制一个变量作为符号,以便同一类别中的其他变量不受影响。在这种情况下,您可以使用“-b”标志：

```
moltemplate.sh -b   @atom:SPCE/O 2   system.lt
```

请记住,在此示例中,这可能会导致其他原子类型（例如“@atom:SPCE/H”)被分配给重叠数字。

“ttree assignments.txt”文件

通常,在运行 moltemplate.sh 后,将创建一个“ttree assignments.txt”文件（如果已经存在,则对其进行更新)以反映您所做的任何更改。 （该文件通常位于“output ttree/”目录中。它也可以位于当前目录“./”中。)您可以随时检查以确保原子类型（或任何其他 ttree 变量)已分配正确。

“ttree assignments.txt”文件与上面的“new assignments.txt”文件示例具有相同的格式。

注意:在这两个文件中,可选的斜杠“/”可以跟在“@”或“$”字符后面,如“@/atom:SPCE/O”。 （此斜杠是可选的,表示
定义计数器的环境。 “@atom”计数器是
全局定义。 D.2 节中描述的“$resid”反例不是。）

错误警告:使用“delete”命令可能会导致某些实例变量（特别是 $atom、$mol、$bond、$angle、$dihedral 和 $improper 变量)编号错误。然而静态变量（以@开头)应该总是准确的。 -安德鲁 2019-9-03。

lttree.py 和 ttree.py 也接受“-a”和“-b”标志

如果出于某种原因,您使用的是“lttree.py”或“ttree.py”而不是“moltem plate.sh”,那么此处说明的“-a”和“-b”标志也适用于这些脚本。它们并非特定于 moltemplate.sh。

D.2 使用类别自定义计数方法

“.lt”文件中的变量默认分配给整数,从 1 开始,递增 1。这可以使用“category”命令覆盖。

例如,要创建一个名为“distance”的新变量类别,它从 0 开始并以 0.5 递增,您可以在 LT 文件中包含以下命令：

```
类别 $distance(0.0, 0.5)
```

（此命令不应与传统的计数器类别一起使用,如 $atom、$bond、$angle、$dihedral、$improper、$mol、@atom、@bond、@angle、@dihedral 和 @improper。）

D.3 创建本地独立计数器

默认情况下,给定类别中的变量总是分配给唯一的整数。这可以使用“类别”命令覆盖。例如,您可能有一个变量来跟踪每种聚合物中的单体。

聚合物中的第一个单体被指定为“1”,第二个单体被指定为“2”,依此类推,无论系统中的聚合物数量如何。

为此,我们可以创建一个名为“monomerid”的新变量类别,该类别定义在“Polymer”分子的每个实例的范围内:

```
单体
   { write(  数据原子  ) {
      $atom:ca @atom:CA $monomerid:. 0.0 0.0 0.0 0.0 $atom:cb @atom:CB
      $monomerid:. 0.0 1.53 0.0 0.0
   }
}

聚合物 { 类别
   $monomerid(1,1) 单体 = 单体 [100]

}

聚合物=聚合物[10]
```

在这个例子中,有 10 种聚合物,每种聚合物含有 100 个单体。 “$monomerid” 计数器将被替换为 1 范围内的整数。 . . 100，（不是 1 ..... 1000,如您所料）。因为“$monomerid”计数器是它所定义的蛋白质的局部变量,所以其他蛋白质中的“$monomerid”变量不共享相同的计数器,并且可以重叠。

D.4 计数顺序

大多数变量是自动分配的。默认情况下,静态变量 (@) 按照它们在文件（或文件,如果包含多个 LT 文件)中出现的顺序分配。随后,实例变量 ($) 按照它们在实例化期间创建的顺序进行分配。但是,您可以自定义分配它们的顺序。

订购

LT 文件由 moltemplate.sh/lttree.py 分多个阶段解析。 “write once()”和“write()”命令分别在静态和实例阶段执行,如下所述。

静态阶段

在“静态”阶段，“write once()”语句按照它们从用户输入文件中读取的顺序执行（不管它们是否出现在嵌套类中）。任何“包含”命令都会影响此顺序。

在处理类定义并执行“write once()”命令后，lttree.py 开始实例化阶段。

实例化阶段

在此阶段，lttree.py 会复制（实例化）用户使用“new”命令请求的类。在此阶段，lttree.py 还使用“写入”命令将数据附加到文件中。（在本手册中，“write()”和“new”被称为实例命令。）“write()”和“new”命令按照它们在用户输入文件中出现的顺序交替排列。“新”命令为它们创建的每个类的副本递归调用任何实例命令。

同样，实例变量的计数（以“$”为前缀）不会干扰静态变量赋值。例如“@atom:x”和“$atom:x”对应不同的变量，属于不同的变量类别（“@atom”和“$atom”），它们分别被赋予数值。

E 直接使用 lttree.py 或 ttree.py

（绕过 moltemplate.sh）

“moltemplate.sh”只是一个简单的脚本，它调用“lttree.py”，然后将lttree.py生成的各种输出文件与坐标数据一起组合成一个LAMMPS输入脚本和一个数据文件。“lt tree.py”然后调用“ttree.py”。“ttree.py”缺乏读取或生成坐标的能力，但在其他方面与“lttree.py”和“moltemplate.sh”几乎相同。

如果将来 moltemplate.sh 不再适用于一些新的、最近添加的 LAMMPS 功能，您可以绕过 moltemplate.sh 并直接运行 lttree.py 或 ttree.py。moltemplate.sh 所做的一切基本上都可以使用 unix shell 和文本编辑器手动完成。此过程概述如下。

E.1 首先运行 ttree.py

运行“ttree.py”的语法与运行 moltemplate.sh 的语法相同。上面解释了 moltemplate.sh 语法。

不幸的是，ttree.py 不理解用于处理坐标数据的 -pdb、-xyz 或 -raw 参数。如果您直接运行“ttree.py”，那么您必须自己从这些文件中提取坐标数据并手动将其插入到您的 lammps 输入文件中。这在下面解释。

示例：进入examples/waterSPCE/目录并运行：ttree.py system.lt

这将为 32 个水分子的系统准备 LAMMPS 输入文件。
（在本例中,我们使用“SPCE”水模型。）

运行上面的命令可能会创建以下文件：“Data Atoms”（LAMMPS 数据文件的“Atoms”部分,无坐标）“Data Bonds”（LAMMPS 数据文件的“Bonds”部分）“ Data Angles”（LAMMPS 数据文件的“Angles”部分）“Data Masses”（LAMMPS 数据文件的“Masses”部分）“In Init”（LAMMPS 输入脚本的“Initialization”部分。）“在设置中”（LAMMPS 输入脚本的“设置”部分,通常包含力场参数、组定义和约束）“数据边界”（LAMMPS 数据文件的“周期性边界条件”部分。）“ttree assignments.txt”（可变分配。请参阅“自定义”部分。）

稍后可以使用文本编辑器或 unix “cat” 和 “paste” 命令轻松地将这些数据组合成单个 LAMMPS 数据文件和单个 lammps 输入脚本。

它还可以创建这些文件：“按类型划分的数据角度”、“按类型划分的数据二面体”、“按类型划分的数据不当”。这些文件告诉 moltemplate 如何按原子和键类型自动生成键合相互作用。必须使用“nbody by type.py”实用程序（如附录 A 中所述）将它们转换为角度、二面角和不正确的列表。

E.2 然后创建一个LAMMPS数据文件

创建一个新文件（本例中为“system.data”）,然后粘贴以下内容
文本到它：

创建“标题”部分

例子：

灯说明

96 个原子
64 债券
32个角0个二面角

2 原子类型 1 键类型 1
角度类型 0 二面体类型

0.000000 9.043 xlo xhi
0.000000 15.663 ylo yhi 0.000000 7.361
兹洛志

如果您使用 ttree.py,则必须自己计算原子数、键数和原子类型、键类型等。

注意：“xlo xhi”“ylo yhi”“zlo zhi”行中的数字决定了模拟框的大小,并且会因系统而异。如果创建了 ttree

一个名为“数据边界”的文件,您可以从那里复制此信息。
（三斜晶胞有第四行包含“xy xz yz”参数。）（如果
你有一个 .PDB 文件,这些边界框编号在“CRYST1”中
文件开头附近的行。）

创建数据文件的“标题”部分后,粘贴另一个
LAMMPS 数据文件末尾的部分（带有适当的部分
标题和空白行）。

```
回声 “” >> 系统数据
回声“原子”>> system.data
回声 “” >> 系统数据
cat 数据原子 >> system.data
回声 “” >> 系统数据
回声“债券”>> system.data
回声 “” >> 系统数据
cat 数据债券 >> system.data
回声 “” >> 系统数据
回声“角度”>> system.data
回声 “” >> 系统数据
cat 数据角度 >> system.data
回声 “” >> 系统数据
回声“群众”>> system.data
回声 “” >> 系统数据
cat “数据量” >> system.data
回声 “” >> 系统数据
```

根据您的系统,您可能还拥有以下文件：“数据
二面角” “数据不正确” “数据键系数” “数据角度系数”
“数据二面角系数”“数据不正确系数”。如果是这样,那么追加
也将它们添加到数据文件的末尾。 （还有许多其他可选
“class2”力场的部分。异国情调的原子风格也需要自己的
诸如“线”、“椭球”和“三角形”之类的部分。咨询 LAMMPS
有关这些的详细信息的文档。）

E.3 现在创建 LAMMPS 输入脚本

```
echo include \ In Init\ > run.in.EXAMPLE
echo read_data system.data >> run.in.EXAMPLE
echo include \ 在设置中\ >> run.in.EXAMPLE
```

最后,您必须担心提供原子坐标。 （不像
moltemplate,ttree.py 不处理原子坐标。）

以下命令对于从 PDB 中提取坐标很有用
或 XYZ 文件并将它们转换为 LAMMPS 输入脚本命令：

E.4 提取坐标

要从 .PDB 文件（“file.pdb”）中提取坐标,请使用：

```
awk    /^ATOM |^HETATM/{print substr($0,31,8) \        substr($0,39,8)
                                                \        substr($0,47,8)}    \

      <文件.pdb \ >
      tmp_atom_coords.dat
```

（注意:上面的“ATOM”后面应该有两个空格。）
要从 XYZ 文件（“file.xyz”）中提取坐标,请使用:

```
awk    函数是num(x){return(x==x+0)} \
        BEGIN{targetframe=1;framecount=0} \ {if (isnum($0))
        {framecount++} else \ {if (framecount==targetframe) { \

          如果 (NF>0) { \
            if ((NF==3) && isnum($1)) { \ print $1
                $2       $3} \ else if ((NF==4) && isnum($2))
            { \ print $2       $3       $4} }}}}    \ < file.xyz \ >
              tmp_atom_coords.dat
```

E.5 将坐标文件转换为 LAMMPS 输入脚本
格式

```
awk    {如果 (NF>=3) { \
          纳托姆++;打印    设置原子    natom    x    $1    y    $2    z    $3      }}    \ < tmp_atom_coords.dat
      \ >> system.in.coords
```

最后在您的 lammps 输入脚本中导入“system.in.coords”,使用:

```
echo    include \    system.in.coords\       >> run.in.EXAMPLE
```

F 使用 nbody by type.py 实用程序

（绕过 moltemplate.sh）

moltemplate.sh 使用“nbody by type.py”实用程序按原子类型生成键合原子之间的多体相互作用。如果 moltem plate.sh 崩溃或不是最新的 LAMMPS,您可以通过自己手动调用 type.py 来按类型分配交互。

例如,以下命令将生成一个文件“Angles”,其中包含最终应粘贴到 LAMMPS 数据文件的“Angles”部分的文本行:

```
nbody_by_type 角度 \
      -atoms    数据原子    \ -bonds
        数据键    \ -subgraph
        nbody_Angles.py    \ -nbodybytype    数据
      角度按类型    \ >    数据角度
```

对于二面角或不正确的交互,重复上述命令,然后
将“角度”替换为“二面体”或“不恰当”。
注意:上述说明的工作前提是您在“按类型划分的角度”部分中不使用任何通配符(“*”或“?”)或
正则表达式。如果使用通配符或正则表达式,则必须以这种方式运行程序:

```
nbody_by_type Angles \
      -atoms    Data Atoms.template    \
      -bonds    Data Bonds.template    \
      -subgraph    nbody_Angles.py    \
      -nbodybytype    Data Angles By Type.template    \ >
        Data Angles.template
```

之后,您必须将“Angles.template”文件中的每个变量替换为适当的整数,然后再将内容复制到
LAMMPS 数据文件中。 (ttree render.py 程序可能对此很有用。用文本编辑器打开 moltemplate.sh 文
件,看看它是如何完成的。)

请注意,“数据原子”和“数据键”是指通常由“ttree.py”或“lttree.py”创建的文件,它们分别包含
LAMMPS 数据文件格式的原子和键数据。类似地,“Data Angles By Type”是指包含如何按原子类型自
动生成角度的说明的文件。 (同样,这通常是通过在 LT 文件上运行“ttree.py”或“lttree.py”来生成
的,该文件包含包裹在“write once(Data Angles By Type)”命令中的文本块。)

注意:如果您已经有现有的“数据角度”,您可以通过 type.py 将它们添加到由 nbody 创建的角度
交互列表中。

```
nbody_by_type 角度 \
      -atoms    数据原子    \ -bonds
        数据键    \ -subgraph
        nbody_Angles.py    \ -nbodyfile    数据角
      度    \ -nbodybytype    数据角度按类型    \
      > extra_Angles.tmp

cat extra_Angles.tmp    数据角度    > new_Angles mv -f new_Angles
  数据角度    rm -f extra_Angles.tmp
```

F.1 用法

作为参考,下面包含“nbody by type.py”命令的完整手册页。

nbody_by_type.py 读取包含按原子类型(和键类型)的键合多体相互作用的 LAMMPS 数据文件(或 LAMMPS
的摘录)数据文件,并生成与这些类型一致的 LAMMPS 格式的附加相互作用列表(到标准输出)。

典型用法:

```
nbody_by_type.py X < old.data > new.data
```

- 或者 -

```
nbody_by_type.py X \
                -atoms atom.data \
                -bonds bonding.data \
                -subgraph   nbody_X.py   \
                -nbody X.data \ -nbodybytype
                X_by_type.data > new_X.data
```

在这两种情况下，“X”都表示交互类型，即“角度”、“二面体”或“不适当”。

用户可以添加对其他交互类型的支持。见下文。

注意：可选的“-subgraph”参数允许您自定义用于匹配和生成该类型交互的规则。

它是可选的，仅对使用非标准二面角或不正确的原子序对流的力场有用。）

-------- 示例 1 --------

```
nbody_by_type.py X < old.data > new.data
```

在此示例中，nbody_by_type.py 读取 LAMMPS 数据文件“orig.data”，并提取相关部分
（“Angles”、“Dihedrals”或“Impropers”）。它还查看名为“X By Type”的部分（例如
“Angles By type”、“Impropers By type”、“Impropers By type”），其中包含用于自动定义该类型的附加交互的
标准列表。例如，此文件可能包含：

角度类型

```
7 1 2 1 * *
8 2 2 * * *
9 3 4 3 * *
```

第一列是交互类型 ID。
接下来的 3 列是原子类型标识符。
最后两列是键类型标识符。
* 是一个通配符，表示在此示例中对债券类型没有偏好。 （可选地，正则表达式也可用于定义类型匹配，通过将原子或键类型
括在 / 斜杠中。）

第一行告诉我们，只要类型 1 的原子与原子键合，就应该存在类型为“7”的三体“角”相互作用

“2”型的,它再次与另一个“1”型的原子键合。
第二行告诉我们,只要三个原子键合在一起并且前两个原子的类型为“2”,就定义了一个角度。

（冗余角度交互被过滤。）

如果它们以该相互作用类型的相关方式（由 nbody_X.py 确定)键合在一起,则为符合这些标准的每组键合原子创建新的相互作用,并打印到标准输出。例如,假设您使用以下命令自动生成 3 体“角度”交互：

nbody_by_type 角度 < old.data > new.data

文件“new.data”将与“old.data”相同,但“Angles By Type”部分将被删除,以下文本行将添加到“Angles”部分：

394 7 5983 5894 5895
395 7 5984 5895 5896
396 7 5985 5896 5897
: : : : :
847 9 14827 14848 14849

第一列中的数字是计数器,它为该类型的每个交互分配一个 ID,并从原始“角度”数据停止的位置开始（新角度 ID 号不与旧 ID 号重叠）。

第二列中的文本（“7”、“9”、...)与输入文件的“按类型角度”部分的第一列中的文本相匹配。

-------- 示例 2 --------

```
nbody_by_type.py X \
                -atoms atom.data \
                -bonds bonding.data \
                -subgraph nbody_X.py \
                -nbody X.data \ -nbodybytype
                X_by_type.data \ > new_X.data
```

特别是对于角度交互：

```
nbody_by_type.py 角度\
                -atoms atom.data \
                -bonds bonding.data \
                -subgraph nbody_Angles.py \
                -nbody Angles.data \ -nbodybytype
                Angle_by_type.data \ > new_Angles.data
```

当以这种方式运行时,nbody_by_type.py 的行为方式与示例 1 中的完全相同,但是仅打印与新生成的交互相对应的文本行（而不是整个数据文件）。

另请注意,以这种方式运行时,nbody_by_type.py 不会从标准输入中读取 LAMMPS 数据。相反,它从“-atoms”、“-bonds”、“-nbody”和“-nbodybytype”标志后面的参数指示的不同文件中读取数据文件的每个部分。

“Angles”是一种三体交互风格。因此,当以这种方式运行时,nbody_by_type.py 将创建一个 5 (=3+2) 列文件 (new_Angles.data)。

注意:原子、键和其他 ID/类型不必是整数。

注意:该程序必须与多个 python 模块一起分发,包括:nbody_Angles.py、nbody_Dihedrals.py 和 nbody_Impropers.py。这些包含角、二面角和不正确相互作用的键定义。

F.2 自定义键拓扑

目前 nbody by type.py 可以检测并生成 3 到 4 个连续键合原子之间的“角度”和“迪赫德拉”相互作用。它还可以在以 T 形拓扑键合的 4 个原子（一个中心原子有 3 个分支)之间产生“不正确”的相互作用。 nbody by type.py 脚本导入名为“nbody Angles.py”、“nbody Dihedrals.py”和“nbody Impropers.py”的外部模块,以帮助它自动检测角度、二面角和不正确的交互。如果任何新的交互类型被添加到 LAMMPS,很容易通过提供新的“nbody X.py”python 模块来定义新的绑定交互类型。这些 python 文件通常只有几行长。复制现有模块之一“nbody Angles.py”、“nbody Dihedrals.py”或“nbody Impropers.py”)并将其修改为内部的子图以匹配您要搜索的绑定网络。

G Moltemplate 语法

Moltemplate.sh 是一个脚本,它调用 python 程序程序 (ttree.py) 来解析和解释 moltemplate 语言。下面提供了该语言语法的概述。这里显示的语法并不全面。一些较少使用的语言特性的语法（例如变量快捷方式,以及包含“../”和“...”的路径)在此省略（但稍后在附录 G.1 中讨论）。

请注意,当前的语言解析器 (ttree.py) 是手工编写的。 （它不是由编译器-编译器生成的。)因此很遗憾,可能存在一些满足语法规则的有效输入,但会导致 moltemplate 返回错误消息。然而,大多数 moltem 板命令都是简短而简单的。自从我收到此类错误的报告以来,已经过去了几年的大量使用。

端子说明：

终端是语言中不可分割的单词和符号。 moltemplate 语言中的终端分为几类。

终端 ::= 整数 |号码 |细绳

整数：非负整数

Number：浮点数

字符串：一个字符串。 （一系列字符。）

字符串分为几类：

字符串 ::= 实例名称 |对象类型名称 |文件名 |类别名称 |其他字符串

InstanceName：是一个字符串，引用实例树中当前级别的子节点之一。通常，它是您已实例化的分子之一的唯一名称，但它也可以是原子之一的名称或这些分子内的键合相互作用。

ObjectTypeName：是一个字符串，它引用静态树中当前级别的其中一个子节点的名称。该节点通常是指一种分子或一种原子或键合相互作用，或包含力场信息的对象（如“TraPPE”或“OPLSAA”）。

CategoryName：是一种计数器变量类别。典型的 cat egories 是“@atom”、“$atom”、“@bond”、“$bond”、“@angle”等...
然而，用户创建的类别也可以使用“category”命令。 （见下文。）原则上，每个类别都与静态树 (@) 或实例树 ($) 中的一个节点相关联，这将计数器的范围限制为该节点的子节点。然而在实践中，大多数类别（例如“@atom”、“$bond”、...）默认与根节点（“/”）相关联，因此在范围内是全局的。附录 D.2 中解释了创建自定义类别和稍后引用它们的语法。

FileName：引用文件名的字符串。

OtherStr：不属于任何这些类别的字符串

请参阅附录 G.1 以了解解释 $ 和 @ 样式变量的语法的详细信息，包括 CategoryNames 和节点引用。

生产规则

Moltemplate LT 文件包含命令列表。

命令列表 ::= |命令命令列表

...在哪里

命令 ::= ObjectDef |实例命令 |删除命令 |写C |写OC |
导入命令 |分类定义 |实例模组 |对象类型模块 |
对象定义 |替换命令 |使用命令 | PushXform

ObjectDef ::= ObjectTypeName ClassParents { CommandList } | “继承”父列表
类父母 ::=
父列表 ::= 静态节点 |静态节点父列表
静态节点 ::= ObjectTypeName | ObjectTypeName / 静态节点

InstanceCommand ::= InstanceName = new InstanceExpr InstanceExpr ::=
InstanceName RangeNDX | RangeX RangeNDX RangeNDX ::= RangeX ::= [IntRange
] Xforms IntRange ::= 整数 |整数 : 整数 | “*”

Xforms ::= | Xform Xforms Xform ::=
MoveXform | RotXform | ScaleXform MoveXform ::= .move(Number
, Number , Number)
RotXform ::= .rot(Number , Number , Number , Number)
ScaleXform ::= .scale(Number , Number , Number)

删除或修改现有实例

InstanceMod ::= InstanceSelection XformsNE
DeleteCommand ::= 删除 InstanceSelection

InstanceSelection ::= InstanceNode RangeND
实例节点 ::= 实例名称 | InstanceNode / InstanceName
范围ND ::= |范围范围ND
范围 ::= [IntRange]
XformsNE ::= Xform Xforms

编写包含计数器变量的文本

WriteC ::= write(FileName) { TemplateText }
模板文本 ::= |字符串模板文本 | StatVar TemplText | InstVar 模板文本
InstVar ::= $ CategoryName : InstanceNode
StatVar ::= @ CategoryName : StaticNode

WriteOC ::= write_once(FileName) { TemplateTextStat }
TemplTextStat ::= |字符串 TemplTextStat | StatVar TemplTextStat

各种各样的

ImportCommand ::= import 文件名

ReplaceCommand ::= replace { StatVar StatVar }

类别命令 ::= CatCommandStat | CatCommandInst
CatCommandStat ::= category @ CategoryName (Integer , Integer)
CatCommandInst ::= category $ CategoryName (Integer , Integer)

ObjectTypeMod ::= ObjectTypeName XformsNE
ObjectDefMod ::= ObjectTypeName = ObjectTypeName XformsNE

UsingCommand ::= using namespace ObjectTypeName

PushXform := push(XformsNE) CommandList pop()

G.1 计数器变量语法

计数器变量的名称如下：

$cpath/catname:lpath
或者
@cpath/catname:lpath （注意：
本附录中的所有变量示例都可以引用静态 @ 变量或实例 $ 变量。两种变量类型遵循相同的语法规则。为简洁起见，仅显示实例 $ 变量。）

所有计数器变量都有 3 个部分：

cpath，类别范围对象（通常省略） catname，类别名称

lpath，“叶路径”。这包括变量的名称和（可选）该变量在对象树中相对于引用该变量的对象（当前上下文对象）的位置

通常省略 cpath，在这种情况下，这意味着该类别具有全局范围。 （这适用于所有标准计数器变量类型：“@atom”、“$atom”、“$mol”、“@bond”、“$bond”、“@angle”、“$an gle”、“ @dihedral”、“$dihedral”、“@improper”和“$improper”。）但是可以显式指定 cpath，如下例所示：“$/atom:”（ “/”显式表示计数器具有全局范围）。另一个具有显式 cpath 的示例是名为“$/pro teins[5]/monomerid:”的自定义本地计数器变量。 （参见 D.3 节。）在本例中，cpath 为“$/proteins[5]”，catname 为“monomerid”，lpath 为“.”。 （在 D.3 节中，我们从未明确指定 cpath。这是造成混淆的根源。当省略 cpath 时，程序会在树上搜索包含具有匹配 catname 的类别的祖先节点。因此 cpath 很少需要明确说明。有关详细信息，请参阅第 G.3 节。）

G.2 一般变量语法

省略号（ “...”）通常出现在计数器变量中（或隐含）。
最复杂和通用的变量语法是： $cpath/.../catname:lpath 这意味着：找到包含名为“catname”的类别的 cpath 对象的最近祖先。这个祖先决定了类别的

范围。此类别中的计数器变量是该对象的祖先的本地变量。在这个使用示例中,lpath 标识了变量对应的“叶”对象相对于类别范围对象 (cpath) 的位置。

另一方面,如果用户没有明确说明类别的范围（cpath）（这是典型的）,则 lpath 标识叶对象相对于引用变量的对象的位置（当前-语境 ”。”）。

G.3 可变速记等价物

$catname:lpath 等价于“$.../catname:lpath”

这意味着 :找到当前对象的最近直接祖先,其中包含名称与 catname 匹配的类别。如果未找到,则创建一个新类别（在全局级别）。这是 LT 文件中最常用的语法。

如果冒号被省略,如 $lpath/catname,那么它等价于:$catname:lpath。同样,在这些情况下,lpath 是相对于引用变量的对象的路径。

如果省略了 $lpath,那么这相当于 $catname:。换句话说,叶子节点就是当前节点“.”。 （此语法通常用于计数跟踪分子 ID 号。您可以使用计数器变量“$mol”来跟踪当前分子 ID 号,因为它会计算定义此变量的分子对象。在此如果类别的名称是“mol”。在大多数示例中,没有指定类别对象 cpath。这意味着类别对象是自动全局的。全局类别对象意味着每个分子对象都有一个唯一的 ID 号这对于整个系统来说是唯一的,而不仅仅是在某些局部分子中是唯一的。作为反例,考虑氨基酸残基计数器。

可以为蛋白质中的每个氨基酸分配一个残基 ID 号,以在单个蛋白质链中识别它。然而,由于它们的类别是在蛋白质水平上局部定义的,因此这些残基 ID 编号不是全局的,并且如果存在多个蛋白质链,则不是唯一定义的。）（详见 D.3 节。）

$cpath/catname:lpath/...

（相当于简写）

查找与“$cpath/catname:”对应的类别名称和对象（见上文)如果 $cpath/ 为空,则搜索名称与 catname 匹配的类别的祖先,如上所述。要查找变量对应的“叶对象”,请从 CURRENT 对象（不是类别对象)开始。如果 lpath 不为空,则跟随 lpath 到树中的新位置。否则,从当前对象开始。 （一个空的 lpath 对应于当前对象。)从对象树中的这个位置搜索一个直接祖先,它恰好也是属于所需类别的一些其他变量的“叶对象”。如果没有找到这样的变量,则 ttree 创建一个新变量,其叶子对象是 lpath 位置的对象,并将其放入所需的类别中。

$lpath/.../catname 等价于 $catname:lpath/...

（相当于简写）
如果省略 lpath,则从当前节点开始。 （在分子示例中，“$.../mol”是一个类别名称为“mol”的变量。该变量的“叶对象”要么是定义此变量的当前对象,要么是这个对象已分配给属于名为“mol”类别的变量。这样,大对象（大分子)可以由较小的对象组成,而不会破坏跟踪我们属于哪个分子的“mol”计数器. 换句话说，“$.../mol”明确地指的是这个子分子所属的大分子的 ID#（不管它可能有多少层）。）

$cpath/catname:lpath

输出 ttree/ttree assignments.txt 文件中的变量使用此语法。
如果用户明确指定通向 cat 节点的路径,并避免使用“...” ,则 lpath 将相对于类别对象而非当前对象进行解释（但 cpath 相对于当前对象进行解释）。这恰好是“ttree assignments.txt”文件中使用的格式（尽管您可以在“.LT”文件的其他任何地方使用它）。在“ttree assignments.txt”文件中,cpath 是相对于全局对象定义的。该文件中的变量始终以“$/”或“@/”开头。开头的斜线将我们带到全局环境对象（所有其他对象都属于该对象）。 （由于“ttree assignments.txt”中的变量总是以“$/”或“@/”开头,这种区别通常并不重要,因为大多数变量的类别对象通常是“全局”根对象。）

-

参考

[1] AI Jewett, D. Stelter, J. Lambert, SM Saladi, OM Roscioni, M. Ricci, L. Autin, M. Maritan, SM Bashusqeh, T. Keyes, RT Dame, J.-E。 Shea、GJ Jensen 和 DS Goodsell。 Moltemplate:用于对复杂生物物质和软凝聚态物理进行粗粒度建模的工具。 J.摩尔。生物学,433（11）:166841,2021。https://doi.org/10 。 1016/j.jmb.2021.166841。

[2] AK Malde、L. Zuo、M. Breeze、M. Stroet、D. Poger、PC Nair、C. Oost enbrink 和 AE Mark。自动力场拓扑构建器 (atb) 和存储库:1.0 版。 J.化学。理论计算,7:4026–4037,2011。

[3] 阿克塞尔·科尔迈耶。 TopoTools VMD 插件。 http://sites.google。 com/site/akohlmey/软件/topotools/。

[4] NM O Boyle、M.Banck、CA James、C. Morley、T. Vandermeersch 和 GR Hutchison。 Open babel:一个开放的化学工具箱。 J.Cheminf.,3（33）,2011。

[5] 威廉·汉弗莱、安德鲁·达尔克和克劳斯·舒尔滕。 VMD 视觉分子动力学。分子图形学杂志，14：33-38，1996。http://www.ks.uiuc.edu/Research/vmd 。

[6] 亚历山大·斯图科夫斯基。材料科学和工程中的建模和模拟可视化以及使用 ovito（开放式可视化工具）分析原子模拟数据。建模模拟。母校。科学。工程，2010 年 18 月。

[7] L. Mart´ nez、R. Andrade、EG Brigin 和 JM Mart´ nez。 Pack mol：用于为分子动力学模拟构建初始配置的包。 J.比较。 Chem., 30(13):2157–2164, 2009. http://www.ime.unicamp.br/~martinez/packmol/ 。

[8] HJC Berendsen、JR Grigera 和 TP Straatsma。有效对势中的缺失项。 J.物理。化学，91（24）：6269–6271，1987。

[9] Marcus G. Martin 和 J. Ilja Siepmann。相平衡的可转移电位。 1. 正构烷烃的联合原子描述。 J.物理。化学。 B, 102(14):2569–2577, 1998。