Electronic Supplementary Material (ESI) for Digital Discovery.
This journal is © The Royal Society of Chemistry 2023

5 Supporting Information

5.1 Additional Background

Lean is an open source theorem prover developed by Microsoft Research and Carnegie Mellon University, based on dependent type theory, with the goal to formalize theorems in an expressive way [13]. Lean supports user interaction and constructs axiomatic proofs through user input, allowing it to bridge the gap between interactive and automated theorem proving. Like Mizar [5] and Isabelle [6], Lean allows user to state definitions and theorems but also combines more imperative tactic styles as in Coq [7], HOL-Light [8], Isabelle [9] and PVS [10] to construct proofs. The ability to define mathematical objects, rather then just postulate them is where Lean gets its power [40]. It can be used to create an interconnected system of mathematics where the relationship of objects from different fields can be easily shown without loosing generality.

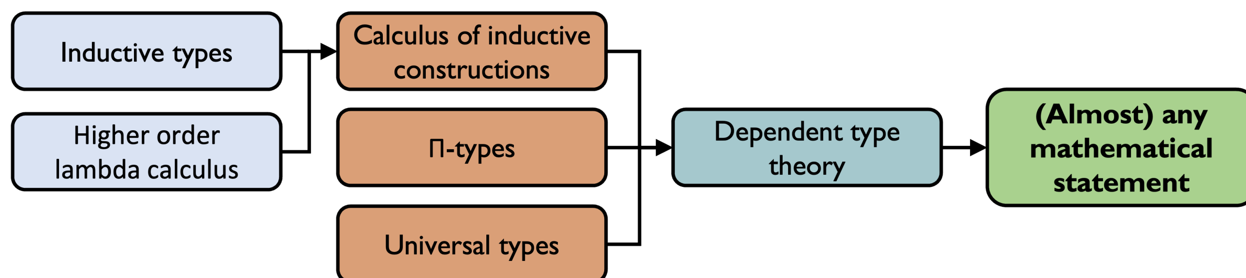


Figure S1: Overview of Lean Theorem Prover

As mentioned above, the power of Lean comes from the ability to define objects and prove properties about them. In Lean, there are three ways to define new Types: type universes, Pi types, and inductive types. The first two are used to construct the basis of dependent type theory, and are used for more theoretical, foundational stuff. Instead we will focus on the use of inductive types. Standard inductive types, known as just *inductive types*, are built from a set of constructors and well found *recursion*. Non-recursive inductive types that contain only one constructor are called *structures*.

Many mathematical objects in Lean can be constructed through inductive types, which is a type built from a set of constructors and proper recursion [104]. The natural numbers are an inductive type, defined using Peano's Encoding [105]. This requires two constructors, a constant element, 0 : nat, and a function called the successor function, S. Then one can be constructed as S(0), two can be constructed as S(S(0)), etc.

In Lean, the natural numbers are defined as:

```
inductive nat
| zero : nat
| succ (n : nat) : nat
```

Here, the type *nat* is defined through recursion by a constant element, zero, and a function. With this, the *def* command is used to define properties about the class, like addition or multiplication. For instance, the addition of the natural numbers is defined as:

```
protected def add : nat → nat → nat
| a  zero     := a
| a  (succ b) := succ (add a b)
```

Addition is defined as a function that takes in two natural numbers and outputs a natural number. Since the natural numbers are created from two constructors, there are two cases of addition that must be shown. The first is a general natural number plus zero which yields the general natural number, and the next is a general natural number plus the successor of a general natural number. The second case used recursion and calls *add* again until it reduces to zero.

The other way to define types is using *structure* which allows us to add constraints to a type variable. For instance, the *class has_add* constrains a type to have a function called *add* which represents addition.

```
class has_add (α : Type u) :=
(add : α → α →  α)
```

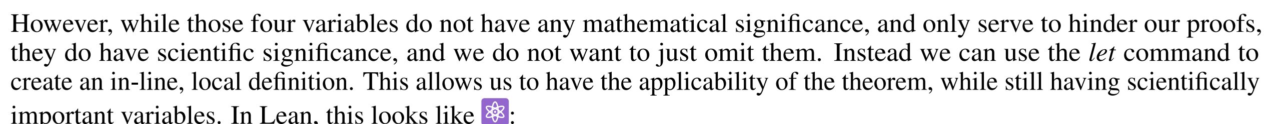
This can be used for more advanced ideas, like defining rings or abelian groups. We can use class to define areas of science as new types with constraints to follow certain rules.

5.2 Additional Proofs

5.2.1 Langmuir Adsorption

The first Langmuir proof introduced earlier states every premise explicitly but however we can condense that by rewriting *hrad* and *hrd* into *hreaction* to yield *k_ad*P*S = k_d*A* and we can then rewrite $h\theta$ and *hK* in the goal statement. While *hrad*, *hrd*, $h\theta$, and *hK* have scientific significance, they do not have any mathematical significance. In Lean it looks like:

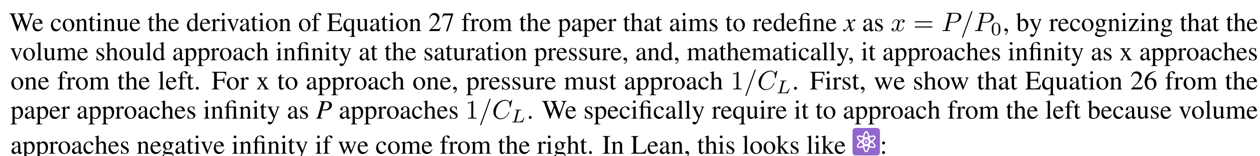
```
theorem Langmuir_single_site2
(P k_ad k_d A S: ℝ)
(hreaction : k_ad*P*S = k_d*A)
(hS : S ≠ 0)
(hk_d : k_d ≠ 0)
: A/(S+A) = k_ad/k_d*P/(1+k_ad/k_d*P) :=
```

However, while those four variables do not have any mathematical significance, and only serve to hinder our proofs, they do have scientific significance, and we do not want to just omit them. Instead we can use the *let* command to create an in-line, local definition. This allows us to have the applicability of the theorem, while still having scientifically important variables. In Lean, this looks like :

```
theorem Langmuir_single_site
(P k_ad k_d A S : ℝ)
(hreaction : let r_ad := k_ad*P*S, r_d := k_d*A in r_ad = r_d)
(hS : S ≠ 0)
(hk_d : k_d ≠ 0)
:
let θ := A/(S+A),
    K := k_ad/k_d in
θ = K*P/(1+K*P) :=
```

The first line after the *theorem* statement, gives the variables use in the proof. Notice that r_{ad}, r_d, K_{eq}, and θ are not defined as variables. Instead, the *let* statement defines those four variables in their respective premise or goal. Then, in the proof we can simplify the *let* statement to get local definitions of those variables, just like *hrad*, *hrd*, $h\theta$, and *hK*. While this version of proof follow the same proof logic minus the two initial rewrites from earlier version, however if we stick with the first proof, we will find it very difficult to use compared to using this proof above, because of all those hypothesises. Suppose we wanted to prove *langmuir_single_site2* and we already have proven *langmuir_single_site*. We would find it impossible to use *langmuir_single_site* because we are missing premises like *hrad* or *hrd*. Yet, we could prove the other way, ie. use *langmuir_single_site* to prove *langmuir_single_site2*. Having all of those extra premises that define the relation between variables only serves to hinder the applicability of our proofs.

5.2.2 BET Adsorption

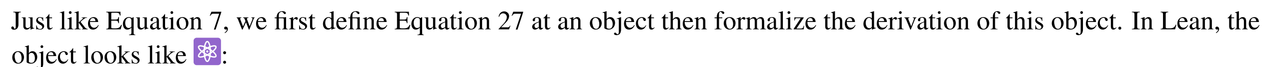
We continue the derivation of Equation 27 from the paper that aims to redefine *x* as $x = P/P_0$, by recognizing that the volume should approach infinity at the saturation pressure, and, mathematically, it approaches infinity as x approaches one from the left. For x to approach one, pressure must approach $1/C_L$. First, we show that Equation 26 from the paper approaches infinity as *P* approaches $1/C_L$. We specifically require it to approach from the left because volume approaches negative infinity if we come from the right. In Lean, this looks like :

```
lemma BET.tendsto_at_top_at_inv_CL
: filter.tendsto brunauer_26
(nhds_within (1/C_L) (set.Ioo 0 (1/C_L)))
filter.at_top:=
```

The function *filter.tendsto* is the generic definition of the limit. It has three inputs, the function, what the independent variable approaches, and what the function approaches, in that order. We split this into three lines to better visualize what is happening. First, we are using the object *brunauer_26*, which is the BET equation as a function of pressure in

terms of x. Next, *(nhds_within (1/C_L) (set.Ioo 0 (1/C_L)))* is how we say approaches $1/C_L$ from the left. *nhds_within* means the intersection of a neighborhood, abbreviated as *nhds*, and a set. A neighborhood of a point is the open set around that point. *set.Ioo* designates a left-open right-open interval. Here we have the interval $(0, 1/C_L)$. The intersections of the neighborhood and this set constrains us to approach the neighborhood from the left. The final part is *filter.at_top* which is a generalization of infinity, and just says our function approaches infinity.

In the original derivation done by Brunauer et al, they wish to show that $P_0 = 1/C_L$ because as pressure approaches each of these values, volume approaches infinity, these two values are equal. It should be noted that this idea is only true if *C*, the BET constant, is greater than or equal to one. If not, the function has two points where it hits infinity in the positive pressure region. We also have problems showing the congruence of such a fact in Lean, since such a relation has yet to be formalized and the congruence of two *nhds_within* has not been shown. For now, we use the lemma above to prove a simplier version of the theorem where we assume $P_0 = 1/C_L$, and show that with this assumption, V approaches infinity. In Lean, this looks like ⚛:



```
theorem brunauer_27
(h1 : P_0 = 1/C_L)
: filter.tendsto brunauer_26  (nhds_within (P_0) (set.Ioo 0 (P_0))) filter.at_top:=
```

The proof of this theorem involves rewriting h1, and then applying the lemma proved above. While we would prefer to prove that $P_0 = 1/C_L$, this proof will serve as a placeholder, until Mathlib builds out more math related to the congruence of this subject. This theorem does not use a lcoal definition, like Langmuir, because P_0 is already defined as a variable using *constant*.

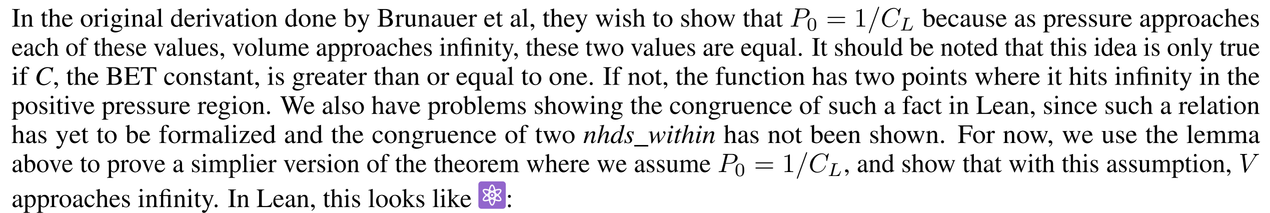
Finally, we formalize the derivation of Equation 28 from the paper, givne by Equation 27.

$$\frac{V}{A * V_0} = \frac{CP}{(P_O - P)(1 + (C-1)(P/P_0)} \tag{27}$$

Just like Equation 7, we first define Equation 27 at an object then formalize the derivation of this object. In Lean, the object looks like ⚛:

```
def brunauer_28 := λ P : ℝ, C*P/((P_0-P)*(1+(C-1)*(P/P_0)))
```

Now we can prove a theorem that formalizes the derivation of this object ⚛:



```
theorem brunauer_28_from_seq
{P V_0: ℝ}
(h27 : P_0 = 1/C_L)
(hx1: (x P) < 1)
(hx2 : 0 <  (x P))
: let Vads :=  V_0 * ∑' (k : ℕ), ↑k * (seq P k),
      A :=  ∑' (k : ℕ), (seq P k) in
  Vads/A = V_0*(brunauer_28 P) :=
```

Rather then explicitly solving the sequence ratio, like we did for Equation 7, we can now use the theorem that derived Equation 7 to solve the left hand side of our new goal. We then have a goal where we show that Equation 27 is just a rearranged version of Equation 7, which is done through algebraic manipulation.

5.2.3 The antiderivative in Lean

For a function, f, the antiderivative of that function, given by F, is a differentiable function, such that the derivative of F is the original function f. In Lean, we formalize the general antiderivative and show how it can be used for several specific applications, including the antiderivative of a constant, of a natural power, and of an integer power. We generalize our functions as a function from a general field onto a vector field, $f : \mathbb{K} \to E$. This allows us to apply the theorems to any parametric vector function, including scalar functions.

Our goal is to show, from the assumption that $f(t)$ is the derivative of $F(t)$ and $f(t)$ is the derivative of $G(t)$, then we have an equation $F(t) = G(t) + F(0)$, which is the antiderivative of $f(t)$. $G(t)$ is the variable portion of the equation. For example, if the antiderivative is of the form $F(t) = t^3 + t + 6$, then $G(t) = t^3 + t$ and $F(0) = 6$. $F(0)$ is the constant of integration, but written in a more explicit relation to the function. Since $G(t)$ is the function of just variables, we have as another premise $G(0) = 0$.

The first goal is to show that a linearized version of the antiderivative function holds. We can rewrite $F(t)$ so that is linear by moving $G(t)$ to the left hand side, leaving us with an equation that equals a constant.

$$F(t) - G(t) = C \tag{28}$$

Thus, we can relate any two points along this function, $\forall xy, F(x) - G(x) = F(y) - G(y)$. To show this holds, we recognize that if Equation 28 is constant, then the derivative of this function is equal to zero.

$$\frac{d}{dt}(F(t) - G(t)) = 0 \tag{29}$$

Next, we apply the linearity of differentiation to Equation 29 to get a new form: $\frac{d}{dt}F(t) - \frac{d}{dt}G(t) = 0$, and rearrange to get:

$$\frac{d}{dt}F(t) = \frac{d}{dt}G(t) \tag{30}$$

From the first premise, we assumed that $f(t)$ is the derivative of $F(t)$. From our second premise, we assumed that $f(t)$ is also the derivative of $G(t)$. Thus, applying both premises, we can simplify Equation 30 to:

$$f(t) = f(t) \tag{31}$$

which we recognize to be correct.

Now that we have a new premise to use, given by Equation 32, we can specialize this function to get our final form.

$$\forall xy, F(x) - G(x) = F(y) - G(y) \tag{32}$$

We specialize the universals by supplying two old names. For x, we use t (the variable we have been basing our differentiation around), and for y we use 0. Thus, Equation 32 becomes:

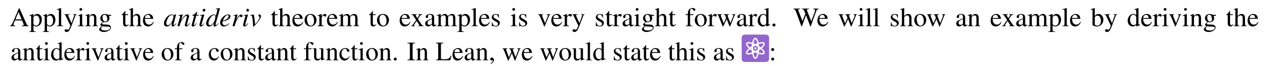
$$F(t) - G(t) = F(0) - G(0) \tag{33}$$

Our third premise was that $G(0) = 0$, so we can simplify and rearrange Equation 32, to get our final form:

$$F(t) = G(t) + F(0) \tag{34}$$

Which satisfies the goal we laid out in the beginning. In Lean, the statement of this theorem looks like :

```
theorem  antideriv
{E : Type u_2}  {𝕂: Type u_3} [is_R_or_C 𝕂] [normed_add_comm_group E]
[normed_space 𝕂 E]
{f F G: 𝕂 → E} (hf : ∀ t, has_deriv_at F (f t) t)
(hg : ∀ t, has_deriv_at G (f t) t)
(hg' : G 0 = 0)
: F = λ t, G t + F(0) :=
```



Applying the *antideriv* theorem to examples is very straight forward. We will show an example by deriving the antiderivative of a constant function. In Lean, we would state this as :

```
theorem antideriv_const
(F : 𝕂 → E) k : E
(hf : ∀ t, has_deriv_at F k t):
(F =  (x : 𝕂)), x·k + F 0) :=
```

Here we say that the derivative of $F(x)$ is the constant k, and want to show that $F(x) = x \cdot k + F(0)$, where the "·" operator stands for scalar multiplication. To use the *antideriv* theorem, we must show that its premises follow, meaning we must show:

```
∀ t, has_deriv_at F k t
∀ t, has_deriv_at x·k k t
0·k = 0
```

The first goal is explicitly given in our premises, *hf*. The next goal can be derived by taking out the constant, and showing that the function x has a derivative equal to 1. The final goal can be easily proven by recognizing zero multiplied by anything is zero. Thus, we have formalized antiderivative of a constant function, and can use this same process for any other function. The antiderivative is especially important for deriving the kinematic equations, as seen in the next section.