Lean 4: State of the U

Sebastian Ullrich
A brief history of Lean

- Lean 0.1 (2014)
- Lean 2 (2015)
  - first official release
  - fixed tactic language
- Lean 3 (2017)
  - make Lean a meta-programming language: build tactics in Lean
  - backed by a bytecode interpreter
- Lean 4 (201X)
  - make Lean a general-purpose language: native back end, FFI, ...
  - reimplement Lean in Lean
A brief history of Lean

- Lean 0.1 (2014)
- Lean 2 (2015)
  - first official release
  - fixed tactic language
- Lean 3 (2017)
  - make Lean a meta-programming language: build tactics in Lean
  - backed by a bytecode interpreter
- Lean 4 (201X)
  - make Lean a general-purpose language: native back end, FFI, ...
  - reimplement Lean in Lean

\[ X \geq 10 \]
The Lean dream team

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter
The Lean dream team

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter
- Daniel Selsam: new typeclass resolution
The Lean dream team

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter
- Daniel Selsam: new typeclass resolution
- Simon Hudon: language server
Lean 4 progress: Jan 2019

- C++
- Lean
- other

Diagram:
- Editor
- Language server
- Parser
- Macro expander
- Elaborator
- Tactic framework
- Kernel
- Compiler
- FFI
- Module system
Lean 4 progress: Jun 2019

- C++
- Lean
- other

Diagram:
- Editor
- Language server
- Parser
- Macro hygiene
- Elaborator
- Tactic framework
- Kernel
- Compiler
- FFI
- Module system
Lean 4 progress: Dec 2019

- C++
- Lean
- other

- editor
- language server
- parser
- meta monad
- module system
- typeclass resolution
- elaborator
- tactic framework
- kernel
- compiler
- FFI
- interpreter

Interpreted in C++ and Lean, other components in the system include:

- typeclass resolution
- meta monad
- module system
- tactic framework
- kernel
- compiler
- FFI
- interpreter

The diagram illustrates the components of the Lean 4 system and their interconnections.
Lean 4 progress: Jan 2020

- C++
- Lean
- other

- editor
- language server
- parser
- macro hygiene
- elaborator
- tactic framework
- kernel
- compiler
- FFI
- interpreter

- meta monad
- module system
- typeclass resolution
Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: TypeName, ModuleName, termName
Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: TypeName, ModuleName, termName
- lemma convention termName_property_of_assumption?
Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: TypeName, ModuleName, termName
  - lemma convention termName_property_of_assumption?
- consistent pattern syntax

```lean
def hiThere : ...
  | pat1, ... => ...
  | ...

match ... with
  | pat1, ... => ...
  | ...
```
Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: TypeName, ModuleName, termName
  - lemma convention termName_property_of_assumption?
- consistent pattern syntax
  ```lean
def hiThere : ...
  | pat1, ... => ...
  | ...

match ... with
  | pat1, ... => ...
  | ...
```
- etc...
  ```lean
  fun x =>
  let y := 1;
do a; b
  ```
New typeclass resolution

Performance issues with the old implementation:
- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination
New typeclass resolution

Performance issues with the old implementation:
- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination

Typeclass resolution follows a “Prolog-like search”
New typeclass resolution

Performance issues with the old implementation:
- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination

Typeclass resolution follows a “Prolog-like search”
⇒ adapt known Prolog optimization, *tabled resolution*, to Lean!

 Guarantees termination if size of typeclass problems is bounded
notation "∪ " binders ", " r:(scoped f, Union f) := r
notation "⋃" binders "," r:(scoped f, Union f) := r

expected ':='
State of the $\bigcup$


```lean
notation "$\bigcup" b "," r := Union (fun b => r)
```

```lean
#check $\bigcup$ x, x = x
#check $\bigcup$ (x : Set Unit), x = x
#check $\bigcup$ x ∈ univ, x = x   -- error
```
State of the $\bigcup$

```lean
notation "\bigcup" b "," r := Union {b | r}

#check \bigcup x, x = x
#check \bigcup (x : Set Unit), x = x
#check \bigcup x ∈ univ, x = x  -- works!
```
State of the $\bigcup$

- **Syntax**: 
  
  ```lean
  syntax "\( \bigcup \)" term "," term : term
  macro `(\( \bigcup \) $b, $r) => `(Union {b | r})
  ```

- **Check Examples**:
  
  ```lean
  #check \( \bigcup x \), x = x
  #check \( \bigcup (x : Set Unit) \), x = x
  #check \( \bigcup x \in \text{univ} \), x = x  -- works!
  ```
syntax "∪" term "," term : term
macro `∪ $b, $r)` => `(Union `{b | $r})

#check ∪ x, x = x
#check ∪ (x : Set Unit), x = x
#check ∪ x ∈ univ, x = x -- works!

syntax "{" term " | " term "}" : term
macro
| `(\{x ∈ s | $p\}) => `(setOf (fun $x => $x ∈ s ∧ $p))
| `(\{x ≤ $e | $p\}) => `(setOf (fun $x => $x ≤ $e ∧ $p))
| `(\{b | $r\}) => `(setOf (fun $b => $r))
State of the $\bigcup$

```
syntax "\bigcup" setIdx "," term : term
macro `(\bigcup $b, $r) => `(Union {($b | $r)})

#check \bigcup x, x = x
#check \bigcup x : Set Unit, x = x  -- works!
#check \bigcup x ∈ univ, x = x
```

```
syntax "{" term " | " term "}" : term
macro
| `({($x ∈ $s | $p})) => `(setOf (fun $x => $x ∈ $s ∧ $p))
| `({($x ≤ $e | $p})) => `(setOf (fun $x => $x ≤ $e ∧ $p))
| `({$b | $r}) => `(setOf (fun $b => $r))
```
syntax "\(\text{setIdx}\) " term : term
macro `(\(\bigcup\) $b$, $r)$ => `(Union {$b \mid r}$)

#check \(\bigcup\) x, x = x
#check \(\bigcup\) x : Set Unit, x = x -- works!
#check \(\bigcup\) x ∈ univ, x = x

```lean
declare_syntax_cat setIdx
syntax term : setIdx
syntax ident " : " term : setIdx
syntax "{" setIdx " | " term "}" : term
macro |
| `{($x ∈ $s \mid p)`} => `(setOf (fun $x => $x ∈ $s ∧ p))
| `{($x ≤ $e \mid p)`} => `(setOf (fun $x => $x ≤ $e ∧ p))
| `{($x : $t \mid r)`} => `(setOf (fun ($x : $t) => $r))
| `{($b \mid r)`} => `(setOf (fun $b => $r))
```
Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable
Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable
  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

$ lean --plugin lean3-compat mathlib/src/...
Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable
  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

$ lean --plugin lean3-compat mathlib/src/...

- elaborator changes: probably not too drastic
Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable
  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

    $ lean --plugin lean3-compat mathlib/src/...

- elaborator changes: probably not too drastic
- library changes: mostly *missing* API, needs to be reimplemented
  - but not necessarily in the stdlib