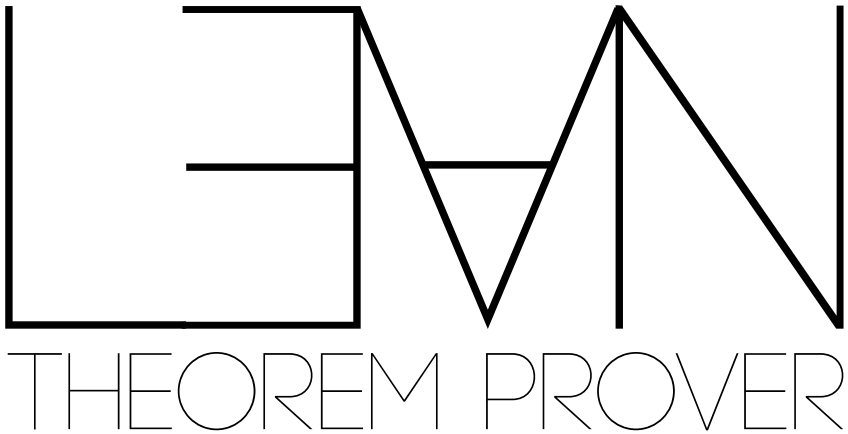
Karlsruhe Institute of Technology

Metaprogramming in Lean 4

Leonardo de Moura[1], Sebastian Ullrich[2]

[1]Microsoft Research, USA [2]KIT, Germany



The Lean 4 Frontend Pipeline

Karlsruhe Institute of Technology

- parser: `≈ String → Syntax`
- macro expansion: `Syntax → MacroM Syntax`
  - actually interleaved with elaboration
- elaboration
  - terms: `Syntax → TermElabM Expr`
  - commands: `Syntax → CommandElabM Unit`
  - universes: `Syntax → TermElabM Level`
  - tactics: `Syntax → TacticM Unit`

The Lean 4 Frontend Pipeline

Karlsruhe Institute of Technology

- parser: `≈ String → Syntax`
- macro expansion: `Syntax → MacroM Syntax`
  - actually interleaved with elaboration
- elaboration
  - terms: `Syntax → TermElabM Expr`
  - commands: `Syntax → CommandElabM Unit`
  - universes: `Syntax → TermElabM Level`
  - tactics: `Syntax → TacticM Unit`
- pretty printer
  - delaborator: `Expr → DelaboratorM Syntax`
  - parenthesizer: `Syntax → ParenthesizerM Syntax`
  - formatter: `Syntax → FormatterM Format`

Notations

```
infixl:65 " + "  => HAdd.hAdd  -- left-associative
infix:65  " - "  => HSub.hSub  -- ditto
infixr:80 " ^ "  => HPow.hPow  -- right-associative
prefix:100 "-"   => Neg.neg
postfix:max "⁻¹" => Inv.inv
```

Notations

```
infixl:65 " + "  => HAdd.hAdd  -- left-associative
infix:65  " - "  => HSub.hSub  -- ditto
infixr:80 " ^ "  => HPow.hPow  -- right-associative
prefix:100 "-"   => Neg.neg
postfix:max "⁻¹" => Inv.inv
```

These are just macros.

```
notation:65 lhs " + " rhs:66 => HAdd.hAdd lhs rhs
notation:65 lhs " - " rhs:66 => HSub.hSub lhs rhs
notation:70 lhs " * " rhs:71 => HMul.hMul lhs rhs
notation:80 lhs " ^ " rhs:80 => HPow.hPow lhs rhs
notation:100 "-" arg:100 => Neg.neg arg
notation:1000 arg "⁻¹" => Inv.inv arg
```

Notations

```
infixl:65 " + "  => HAdd.hAdd  -- left-associative
infix:65  " - "  => HSub.hSub  -- ditto
infixr:80 " ^ "  => HPow.hPow  -- right-associative
prefix:100 "-"   => Neg.neg
postfix:max "⁻¹" => Inv.inv
```

These are just macros.

```
notation:65 lhs " + " rhs:66 => HAdd.hAdd lhs rhs
notation:65 lhs " - " rhs:66 => HSub.hSub lhs rhs
notation:70 lhs " * " rhs:71 => HMul.hMul lhs rhs
notation:80 lhs " ^ " rhs:80 => HPow.hPow lhs rhs
notation:100 "-" arg:100 => Neg.neg arg
notation:1000 arg "⁻¹" => Inv.inv arg
```

```
set_option trace.Elab.command true in
...
```

Mixfix Notations

```
notation:max "(" e ")" => e
notation:10 Γ " ⊢ " e " : " τ => Typing Γ e τ
```

No other “special” forms of **notation**

Mixfix Notations

```
notation:max "(" e ")" => e
notation:10 Γ " ⊢ " e " : " τ => Typing Γ e τ
```

No other "special" forms of **notation**

```
notation:65 a " + " b:66 " + " c:66 => a + b - c
#eval 1 + 2 + 3  -- 0
```

Overlapping notations are parsed with a (local) "longest parse" rule

Syntax

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

`term` is a *syntax category*

Syntax

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

`term` is a *syntax category*

```
declare_syntax_cat index
syntax term : index
syntax term "≤" ident "<" term : index
syntax ident ":" term : index

syntax "{" index " | " term "}" : term
```

More Syntax

```
syntax binderIdent                 := ident <|> "_"
syntax unbracketedExplicitBinders := binderIdent+ (" : " term)?

syntax "begin " tactic,*,? "end" : tactic
```

Lower-Level Syntax in Lean

Karlsruhe Institute of Technology

```
def fromTerm := parser! " from " >> termParser
@[builtinTermParser] def «show» := parser!:leadPrec "show " >> termParser >> (fromTerm <|>
↪  byTactic)
```

is roughly equivalent to

```
syntax fromTerm := " from " term
syntax:leadPrec "show " term (fromTerm <|> byTactic) : command
```

Summary: Parsing

Each syntax category is

- a precedence (Pratt) parser composed of a set of leading and trailing parsers
- with per-parser precedences
- following the longest parse rule

Summary: Parsing

Karlsruhe Institute of Technology

Each syntax category is

- a precedence (Pratt) parser composed of a set of leading and trailing parsers
- with per-parser precedences
- following the longest parse rule

on the lower level: a combinatoric, non-monadic, lexer-less, memoizing recursive-descent parser
`https://github.com/leanprover/lean4/blob/master/src/Lean/Parser/Basic.lean#L7`

Macros

Karlsruhe Institute of Technology

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

Macros

Karlsruhe Institute of Technology

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

which can also be written as

```
macro:max "(" e:term ")" : term => `($e)
```

Macros

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

which can also be written as

```
macro:max "(" e:term ")" : term => `($e)
```

or, in this case

```
macro:max "(" e:term ")" : term => pure e
```

Macros

```
notation:max "(" e ")" => e
```

This is just a macro.

```
syntax:max "(" term ")" : term
macro_rules
| `(($e)) => `($e)
```

which can also be written as

```
macro:max "(" e:term ")" : term => `($e)
```

or, in this case

```
macro:max "(" e:term ")" : term => pure e
```

since it's really just

```
@[macro «term(_)»] def myMacro† : Macro†
  | `(($e)) => pure e
  | _       => throw† Macro.Exception.unsupportedSyntax†
```

Macros

Karlsruhe Institute of Technology

Macros are extensible

```
syntax ident "|" term : index
macro_rules
| `(_big [$op, $idx] ($i:ident | $p) $F) => `(bigop $idx (Enumerable.elems _) (fun $i:ident =>
↪  ($i:ident, $op, $p, $F)))
#check Σ i | myPred i => i+i
#check Π i | myPred i => i+i
```

(*Beyond Notations* supplement,
https://github.com/leanprover/lean4/blob/master/tests/lean/run/bigop.lean)

Macros

Karlsruhe Institute of Technology

Macros are extensible

```
syntax ident "|" term : index
macro_rules
| `(_big [$op, $idx] ($i:ident | $p) $F) => `(bigop $idx (Enumerable.elems _) (fun $i:ident =>
↪  ($i:ident, $op, $p, $F)))
#check Σ i | myPred i => i+i
#check Π i | myPred i => i+i
```

(*Beyond Notations* supplement,
https://github.com/leanprover/lean4/blob/master/tests/lean/run/bigop.lean)

The newest macro is tried first, absent specific priorities

```
macro (priority := high) ...
```

Quotations

```
`(let $id:ident $[$binders]* $[: $ty?]? := $val; $body)
```

- has type Syntax in patterns
- has type m Syntax given MonadQuotation m in terms
- id , val , body have type Syntax
- binders has type Array Syntax
- ty? has type Option Syntax

Quotations

```
`(let $id:ident $[$binders]* $[: $ty?]? := $val; $body)
```

- has type Syntax in patterns
- has type m Syntax given MonadQuotation m in terms
- id , val , body have type Syntax
- binders has type Array Syntax
- ty? has type Option Syntax
- ts in $ts,* has type SepArray

Quotations

```
`(let $id:ident $[$binders]* $[: $ty?]? := $val; $body)
```

- has type `Syntax` in patterns
- has type `m Syntax` given `MonadQuotation m` in terms
- `id`, `val`, `body` have type `Syntax`
- `binders` has type `Array Syntax`
- `ty?` has type `Option Syntax`
- `ts` in `$ts,*` has type `SepArray`

`syntax foo := ...` introduces a new *antiquotation kind* `$e:foo`

`declare_syntax_cat index` introduces a new antiquotation kind `$e:index` and a new *quotation kind* `` `(index|...) ``

Scope of Hygiene

Karlsruhe Institute of Technology

```
macro "foo" : term => do
  let a ← `(rfl)
  `(fun rfl => $a)
```

Scope of Hygiene

Karlsruhe Institute of Technology

```
macro "foo" : term => do
  let a ← `(rfl)
  `(fun rfl => $a)
```

This unfolds to the identity function. Hygiene works *per-macro*

Scope of Hygiene

Karlsruhe Institute of Technology

```
macro "foo" : term => do
  let a ← `(rfl)
  `(fun rfl => $a)
```

This unfolds to the identity function. Hygiene works *per-macro*

Nested scopes can be opened with `withFreshMacroScope`

```
def expandMatchAltsIntoMatchAux (matchAlts : Syntax) (discrs : Array Syntax) :
  Nat → MacroM Syntax
  | 0   => `(match $[$discrs:term],* with $matchAlts:matchAlts)
  | n+1 => withFreshMacroScope do
    let x ← `(x)
    let body ← expandMatchAltsIntoMatchAux matchAlts n (discrs.push x)
    `(@fun $x => $body)
```

Summary: Macros

Karlsruhe Institute of Technology

Macros are syntax-to-syntax translations

- applied iteratively and recursively
- associated with a specific parser and tried in a specific order
- with “well-behaved” (hygienic) name capturing semantics