Matching Logic and Lean

Horațiu Cheval [1,2] (jww Bogdan Macovei [1,2])

[1]University of Bucharest

[2]Institute for Logic and Data Science

Timișoara, February 9, 2023

Matching Logic and Lean

Lean: An ITP based on dependent type theory; since version 4 a full functional programming language

Matching Logic: A formal system aimed at program verification; the foundation of the $\mathbb{K}$ framework

Matching Logic and Lean

The formalization consists so far of:

- A deep embedding of the syntax and proof system
- Named representation for variables and binders
- A definition of the set-valued interpretation of patterns
- Soundness theorem (WIP)
- Other metatheoretical results, e.g. the deduction theorem

Syntax

We say that a *signature* is a set $\mathbb{S}$, intended to represent the set of *symbols*.

Let *EVar* and *SVar* be two countable and disjoint sets of so-called *element variables* resp. *set variables*.

Then, the *patterns over the signature* $\mathbb{S}$ are defined by:

$$\varphi, \psi ::= x \in \mathit{EVar} \mid X \in \mathit{SVar} \mid \bot \mid \sigma \in \mathbb{S} \mid \varphi \cdot \psi \mid \varphi \to \psi \mid \exists x \varphi \mid \mu X \varphi$$

In the usual formulation, φ is required to be *positive* for X when constructing $\mu X \varphi$, but we do not require that.

Syntax

```
inductive Pattern (𝕊 : Type) where
| evar        : EVar → Pattern 𝕊
| svar        : SVar → Pattern 𝕊
| bottom      : Pattern 𝕊
| symbol      : 𝕊 → Pattern 𝕊
| application : Pattern 𝕊 → Pattern 𝕊 → Pattern 𝕊
| implication : Pattern 𝕊 → Pattern 𝕊 → Pattern 𝕊
| existential : EVar → Pattern 𝕊 → Pattern 𝕊
| mu          : SVar → Pattern 𝕊 → Pattern 𝕊
```

where `EVar` and `SVar` are simple wrappers over $\mathbb{N}$ (for typeclass reasons). Derived connectors are defined as usual, and we use the notations $\bot$, $\Rightarrow$, $\cdot$, $\exists\exists$, $\sim$, $\bigwedge$, $\bigvee$ for bottom, implication, application, existential, negation, conjunction and disjunction, respectively.

Proof system

For any premises Γ and pattern φ, we define the inductive type $\Gamma \vdash \varphi$ of the proofs of φ from Γ.

```
def pushConjInExist (not_fv : ¬ψ.isFreeEvar x) :
  Γ ⊢ ∃∃ x φ ⋀ ψ ⇒ ∃∃ x (φ ⋀ ψ) :=
let l₁ : Γ ⊢ φ ⋀ ψ ⇒ ∃∃ x (φ ⋀ ψ) := implExistSelf
let l₂ : Γ ⊢ φ ⇒ ψ ⇒ ∃∃ x (φ ⋀ ψ) := exportation l₁
let l₃ : Γ ⊢ ∃∃ x φ ⇒ ψ ⇒ ∃∃ x (φ ⋀ ψ) :=
  existGen (by simp [*, Pattern.exists_binds]) l₂
let l₄ : Γ ⊢  ∃∃ x φ ⋀ ψ ⇒ ∃∃ x (φ ⋀ ψ) :=
  importation l₃
l₄
```

Semantics

Patterns are interpreted as subsets of M, for some fixed set M. Given a signature $\mathbb{S}$, interpretations are parametrized by:

- an interpretation of symbols $e_{symb} : \mathbb{S} \to 2^M$;
- an interpretation of application $e_{app} : M \times M \to 2^M$;
- interpretations of the element and set variables $e_E : EVar \to M$ and $e_S : SVar \to 2^M$.

```
structure Interpretation (𝕊 : Type) (M : Type) where
  evalSymb : 𝕊 → Set M
  evalApp : M → M → Set M

structure Valuation (M : Type) where
  evalEvar : EVar → M
  evalSvar : SVar → Set M
```

Semantics

Fixing an interpretation of symbols and one of application, the interpretation of $\mathbb{S}$-patterns is a mapping $\|\cdot\|_{e_E,e_S} : \mathit{Pattern}\ \mathbb{S} \to 2^M$

- $\|x\|_{e_E,e_S} := \{e_E(x)\}$
- $\|X\|_{e_E,e_S} := e_S(X)$
- $\|\sigma\|_{e_E,e_S} := e_{symb}(\sigma)$
- $\|\varphi \cdot \psi\|_{e_E,e_S} := e_{app}(\|\varphi\|_{e_E,e_S}, \|\psi\|_{e_E,e_S})$
- $\|\varphi \to \psi\|_{e_E,e_S} := \{a \in M \mid a \in \|\varphi\|_{e_E,e_S} \to a \in \|\psi\|_{e_E,e_S}\}$
- $\|\exists x \varphi\|_{e_E,e_S} := \bigcup_{a \in M} \|\varphi\|_{e_{E x \mapsto a}, e_S}$
- $\|\mu X \varphi\|_{e_E,e_S} := \bigcap \{B \subseteq M \mid \|\varphi\|_{e_E, e_{S X \mapsto B}} \subseteq B\}$

```
def Pattern.interpret
  (I : Interpretation 𝕊 M)
  (val : Valuation M)
  (φ : Pattern 𝕊) : Set M :=
  match φ with
  ...
```

Future work

The end goal is to build a verified ITP for Matching Logic on top of Lean. To this end:

- a user-friendly interface; automatic handling of variables, binders, etc.
- a tactic framework
- interoperability with the $\mathbb{K}$ framework
- generation of proof objects based on a Metamath formalization of Matching Logic (WIP)

`https://gitlab.com/ilds/aml-lean/MatchingLogic`