On a Dependently Typed Encoding of Matching Logic

Ádám Kurucz, Péter Bereczky, Dániel Horpácsi

Eötvös Loránd University

2025.09.17.

Pi Squared

Motivation

- $\mathbb{K}$ framework can be used to define programming languages

Motivation

- $\mathbb{K}$ framework can be used to define programming languages
- Inconsistent theories can be defined

Motivation

- $\mathbb{K}$ framework can be used to define programming languages
- Inconsistent theories can be defined
- Specifications can be expressed in matching logic

Motivation

- $\mathbb{K}$ framework can be used to define programming languages
- Inconsistent theories can be defined
- Specifications can be expressed in matching logic
- Consistency may be proven in it

Background

Matching logic, dependent type theory

Matching logic

- Base unit is the *pattern*

$$\varphi ::= \widehat{x} \mid \widehat{\widehat{X}} \mid \neg\varphi \mid \varphi \wedge \varphi' \\ \mid \exists x . \varphi \mid \mu X . \varphi \mid \sigma(\varphi, \ldots, \varphi)$$

Matching logic

- Base unit is the *pattern*
  - Sorted

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \qquad \mid \neg \varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists \ x : s'. \ \varphi_s \mid \mu \ \ X : s. \ \varphi_s \mid \sigma(\varphi_{s_1}, \ \ldots, \ \varphi_{s_n})$$

Matching logic

- Base unit is the *pattern*
  - Sorted
  - Polyadic

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \qquad \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists \; x : s'.\ \varphi_s \mid \mu \; X : s.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n})$$

Matching logic

- Base unit is the *pattern*
  - Sorted
  - Polyadic
  - Using locally nameless representation

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s'$$
$$\mid \exists_{s'} \quad . \ \varphi_s \mid \mu \quad . \ \varphi_s \mid \sigma(\varphi_{s_1}, \ \ldots, \ \varphi_{s_n})$$

Matching logic

- Base unit is the *pattern*
  - Sorted
  - Polyadic
  - Using locally nameless representation
  - Theories may be built in

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists_{s'} \quad . \ \varphi_s \mid \mu \quad . \ \varphi_s \mid \sigma(\varphi_{s_1}, \ \ldots, \ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}$$

Matching logic

- Base unit is the *pattern*
  - Sorted
  - Polyadic
  - Using locally nameless representation
  - Theories may be built in
  - Can be extended with other connectives

$$
\begin{aligned}
\varphi_s ::= {} & \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s \\
& \mid \exists_{s'} \quad . \, \varphi_s \mid \mu \quad . \, \varphi_s \mid \sigma(\varphi_{s_1}, \ \ldots, \ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'} \\
& \color{red}{\mid \top_s \mid \bot_s \mid \varphi_s \vee \varphi'_s \mid \varphi_s \rightarrow \varphi'_s \mid \varphi_s \leftrightarrow \varphi'_s} \\
& \color{red}{\mid \forall_{s'} . \, \varphi_s \mid \nu . \, \varphi_s \mid \lfloor \varphi_s \rfloor_s^{s'} \mid \varphi_s =_s^{s'} \varphi'_s \mid \varphi_s \subseteq_s^{s'} \varphi'_s}
\end{aligned}
$$

Matching logic

- Base unit is the *pattern*

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg \varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}$$

Matching logic

- Base unit is the *pattern*

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s'$$
$$\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n}) \mid \lceil\varphi_s\rceil_s^{s'}$$

- Substitutions

$$(x \wedge y)[\psi/x] = (\psi \wedge y) \qquad (\exists.\ \underline{0} \wedge \underline{1})[\psi/\underline{0}] = \exists.\ \underline{0} \wedge \psi$$

Matching logic

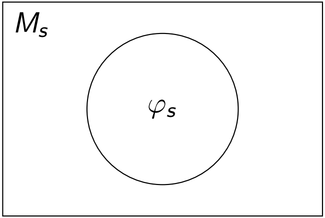
- Base unit is the *pattern*

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}$$

- Substitutions

$$(x \wedge y)[\psi/x] = (\psi \wedge y) \qquad (\exists.\ \underline{0} \wedge \underline{1})[\psi/\underline{0}] = \exists.\ \underline{0} \wedge \psi$$

- Semantics
  - Pattern matching
  - Over sorted carrier sets: M_s
  - Valuations for free variables



Matching logic

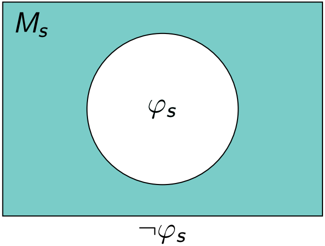
- Base unit is the *pattern*

$$
\begin{aligned}
\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s \\
\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1}, \ \ldots, \ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}
\end{aligned}
$$

- Substitutions

$$
(x \wedge y)[\psi/x] = (\psi \wedge y) \qquad (\exists.\ \underline{0} \wedge \underline{1})[\psi/\underline{0}] = \exists.\ \underline{0} \wedge \psi
$$

- Semantics
  - Pattern matching
  - Over sorted carrier sets: M_s
  - Valuations for free variables



Matching logic

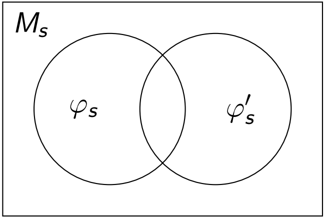
- Base unit is the *pattern*

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}$$

- Substitutions

$$(x \wedge y)[\psi/x] = (\psi \wedge y) \qquad (\exists.\ \underline{0} \wedge \underline{1})[\psi/\underline{0}] = \exists.\ \underline{0} \wedge \psi$$

- Semantics
  - Pattern matching
  - Over sorted carrier sets: M_s
  - Valuations for free variables



Matching logic

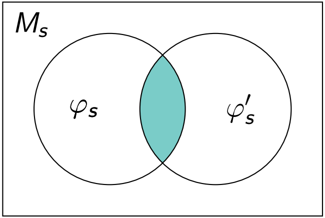
- Base unit is the *pattern*

$$\varphi_s ::= \widehat{x} : s \mid \widehat{\widehat{X}} : s \mid \underline{n} : s \mid \underline{\underline{N}} : s \mid \neg\varphi_s \mid \varphi_s \wedge \varphi'_s$$
$$\mid \exists_{s'}.\ \varphi_s \mid \mu.\ \varphi_s \mid \sigma(\varphi_{s_1},\ \ldots,\ \varphi_{s_n}) \mid \lceil \varphi_s \rceil_s^{s'}$$

- Substitutions

$$(x \wedge y)[\psi/x] = (\psi \wedge y) \qquad (\exists.\ \underline{0} \wedge \underline{1})[\psi/\underline{0}] = \exists.\ \underline{0} \wedge \psi$$

- Semantics
  - Pattern matching
  - Over sorted carrier sets: M_s
  - Valuations for free variables



$\varphi_s \wedge \varphi'_s$

Matching logic

- Formalizations of matching logic exist
  - Can be applicative, unsorted, fully named
  - Well-sortedness and well-formedness are predicates

Matching logic

- Formalizations of matching logic exist
  - Can be applicative, unsorted, fully named
  - Well-sortedness and well-formedness are predicates
- Term algebras can solve this
  - No ill-typed terms
  - Substitutions are not computable

Dependent type theory

Dependent type theory

- Regular types

$$\begin{aligned}
\mathtt{List} &: \mathit{Type} \to \mathit{Type} \\
\mathit{nil} &: \forall (A : \mathit{Type}).\ \mathtt{List}\ A \\
\mathit{cons} &: \forall (A : \mathit{Type}).\ A \to \mathtt{List}\ A \to \mathtt{List}\ A
\end{aligned}$$

Dependent type theory

- Regular types

$$\begin{aligned} \texttt{List} &: \mathit{Type} \to \mathit{Type} \\ \mathit{nil} &: \forall (A : \mathit{Type}).\ \texttt{List}\ A \\ \mathit{cons} &: \forall (A : \mathit{Type}).\ A \to \texttt{List}\ A \to \texttt{List}\ A \end{aligned}$$

- Dependence on $\mathbb{N}$ gives more control

$$\begin{aligned} \texttt{Vec} &: \mathit{Type} \to \mathbb{N} \to \mathit{Type} \\ \mathit{vnil} &: \forall (A : \mathit{Type}).\ \texttt{Vec}\ A\ 0 \\ \mathit{vcons} &: \forall (A : \mathit{Type})\ (n : \mathbb{N}).\ A \to \texttt{Vec}\ A\ n \to \texttt{Vec}\ A\ (S\ n) \end{aligned}$$

Dependent type theory

- Regular types

$$\begin{aligned} \texttt{List} &: \mathit{Type} \to \mathit{Type} \\ \mathit{nil} &: \forall (A : \mathit{Type}).\ \texttt{List}\ A \\ \mathit{cons} &: \forall (A : \mathit{Type}).\ A \to \texttt{List}\ A \to \texttt{List}\ A \end{aligned}$$

- Dependence on $\mathbb{N}$ gives more control

$$\begin{aligned} \texttt{Vec} &: \mathit{Type} \to \mathbb{N} \to \mathit{Type} \\ \mathit{vnil} &: \forall (A : \mathit{Type}).\ \texttt{Vec}\ A\ 0 \\ \mathit{vcons} &: \forall (A : \mathit{Type})\ (n : \mathbb{N}).\ A \to \texttt{Vec}\ A\ n \to \texttt{Vec}\ A\ (S\ n) \end{aligned}$$

- Dependence – and therefore functionality – can be complex

$$\begin{aligned} \texttt{HList} &: \texttt{List}\ \mathit{Type} \to \mathit{Type} \\ \mathit{hnil} &: \texttt{HList}\ [] \\ \mathit{hcons} &: \forall (A : \mathit{Type})\ (As : \texttt{List}\ \mathit{Type}).\ A \to \texttt{HList}\ As \to \\ &\qquad \texttt{HList}\ (A :: As) \end{aligned}$$

Dependent type theory

- Regular types

$$\begin{aligned}
\texttt{List} &: \mathit{Type} \to \mathit{Type} \\
\mathit{nil} &: \forall (A : \mathit{Type}).\ \texttt{List}\ A \\
\mathit{cons} &: \forall (A : \mathit{Type}).\ A \to \texttt{List}\ A \to \texttt{List}\ A
\end{aligned}$$

- Dependence on $\mathbb{N}$ gives more control

$$\begin{aligned}
\texttt{Vec} &: \mathit{Type} \to \mathbb{N} \to \mathit{Type} \\
\mathit{vnil} &: \forall (A : \mathit{Type}).\ \texttt{Vec}\ A\ 0 \\
\mathit{vcons} &: \forall (A : \mathit{Type})\ (n : \mathbb{N}).\ A \to \texttt{Vec}\ A\ n \to \texttt{Vec}\ A\ (S\ n)
\end{aligned}$$

- Dependence – and therefore functionality – can be complex

$$\begin{aligned}
\texttt{HList} &: \texttt{List}\ \mathit{Type} \to \mathit{Type} \\
\mathit{hnil} &: \texttt{HList}\ [] \\
\mathit{hcons} &: \forall (A : \mathit{Type})\ (\mathit{As} : \texttt{List}\ \mathit{Type}).\ A \to \texttt{HList}\ \mathit{As} \to \\
&\qquad \texttt{HList}\ (A :: \mathit{As})
\end{aligned}$$

$$\texttt{HList}\ [\mathbb{N}, \mathbb{B}]$$

Dependent type theory

- Regular types

$$\begin{aligned} \texttt{List} &: \mathit{Type} \to \mathit{Type} \\ \mathit{nil} &: \forall (A : \mathit{Type}).\ \texttt{List}\ A \\ \mathit{cons} &: \forall (A : \mathit{Type}).\ A \to \texttt{List}\ A \to \texttt{List}\ A \end{aligned}$$

- Dependence on $\mathbb{N}$ gives more control

$$\begin{aligned} \texttt{Vec} &: \mathit{Type} \to \mathbb{N} \to \mathit{Type} \\ \mathit{vnil} &: \forall (A : \mathit{Type}).\ \texttt{Vec}\ A\ 0 \\ \mathit{vcons} &: \forall (A : \mathit{Type})\ (n : \mathbb{N}).\ A \to \texttt{Vec}\ A\ n \to \texttt{Vec}\ A\ (S\ n) \end{aligned}$$

- Dependence – and therefore functionality – can be complex

$$\begin{aligned} \texttt{HList} &: \texttt{List}\ \mathit{Type} \to \mathit{Type} \\ \mathit{hnil} &: \texttt{HList}\ [] \\ \mathit{hcons} &: \forall (A : \mathit{Type})\ (As : \texttt{List}\ \mathit{Type}).\ A \to \texttt{HList}\ As \to \\ &\qquad \texttt{HList}\ (A :: As) \end{aligned}$$

$$[4, \mathit{true}] : \texttt{HList}\ [\mathbb{N}, \mathbb{B}]$$

Dependent type theory

- Equality

$$\begin{array}{ll} \mathsf{eq} & : \forall (A : \mathit{Type}).\ A \to A \to \mathit{Type} \\ \mathit{refl} & : \forall (A : \mathit{Type})\ (x : A).\ \mathsf{eq}\ A\ x\ x \end{array}$$

Dependent type theory

- Equality

$$\begin{array}{ll} \mathrm{eq} & : \forall (A : \mathit{Type}).\ A \to A \to \mathit{Type} \\ \mathit{refl} & : \forall (A : \mathit{Type})\ (x : A).\ \mathrm{eq}\ A\ x\ x \end{array}$$

- Transport

$$\mathit{transport} : \forall (A : \mathit{Type})\ (P : A \to \mathit{Type})\ (x\ y : A).\ x = y \to P\ x \to P\ y$$

Dependent type theory

- Equality

$$\begin{aligned} \mathrm{eq} &: \forall (A : \mathit{Type}).\ A \to A \to \mathit{Type} \\ \mathit{refl} &: \forall (A : \mathit{Type})\ (x : A).\ \mathrm{eq}\ A\ x\ x \end{aligned}$$

- Transport

$$\mathit{transport} : \forall (A : \mathit{Type})\ (P : A \to \mathit{Type})\ (x\ y : A).\ x = y \to P\ x \to P\ y$$

$$[1, 2] : \mathtt{Vec}\ \mathbb{N}\ 2$$

Dependent type theory

- Equality

$$\begin{aligned} \mathsf{eq} \quad &: \forall (A : \mathit{Type}).\ A \to A \to \mathit{Type} \\ \mathit{refl} \quad &: \forall (A : \mathit{Type})\ (x : A).\ \mathsf{eq}\ A\ x\ x \end{aligned}$$

- Transport

$$\begin{aligned} \mathit{transport} : \forall (A : \mathit{Type})\ (P : A \to \mathit{Type})\ (x\ y : A).\ x = y \to \\ P\ x \to P\ y \end{aligned}$$

$$[1, 2] : \mathtt{Vec}\ \mathbb{N}\ 2$$

$$[1, 2] : \mathtt{Vec}\ \mathbb{N}\ (1 + 1)$$

Dependent type theory

- Equality

$$\begin{aligned} \mathsf{eq} &: \forall (A : \mathit{Type}).\ A \to A \to \mathit{Type} \\ \mathit{refl} &: \forall (A : \mathit{Type})\ (x : A).\ \mathsf{eq}\ A\ x\ x \end{aligned}$$

- Transport

$$\begin{aligned} \mathit{transport} : \forall (A : \mathit{Type})\ (P : A \to \mathit{Type})\ (x\ y : A).\ x = y \to \\ P\ x \to P\ y \end{aligned}$$

$$\begin{aligned} [1,2] &: \mathtt{Vec}\ \mathbb{N}\ 2 \\ \mathit{transport}\ \mathbb{N}\ (\mathtt{Vec}\ \mathbb{N})\ 2\ (1+1)\ \mathit{refl}\ \ [1,2] &: \mathtt{Vec}\ \mathbb{N}\ (1+1) \end{aligned}$$

Implementation

Dependently typed syntax and semantics

Dependently typed syntax

$$
\begin{aligned}
\texttt{Pattern} &: \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\widehat{\widehat{\square}} &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\square} &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{ex} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\underline{\square}} &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{mu} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \cdot \square &: \forall (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
&\qquad \texttt{HList}\ ((\texttt{Pattern}\ \mathit{ex}\ \mathit{mu}) <\$> (\mathit{params}\ \sigma)) \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ (\mathit{return}\ \sigma) \\
\neg\square &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \wedge \square &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\exists_{\square}.\ \square &: \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: \mathit{ex})\ \mathit{mu}\ s \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\mu.\ \square &: \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ (s :: \mathit{mu})\ s \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\lceil \square \rceil^{\square} &: \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
&\qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s'
\end{aligned}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\texttt{Pattern} & : \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (EV\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\widehat{\widehat{\square}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (SV\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\square} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ ex \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\underline{\square}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ mu \to \texttt{Pattern}\ ex\ mu\ s \\
\square \cdot \square & : \forall (ex\ mu : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
 & \qquad \texttt{HList}\ ((\texttt{Pattern}\ ex\ mu) <\$> (\mathit{params}\ \sigma)) \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ (\mathit{return}\ \sigma) \\
\neg\square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\square \wedge \square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \to \texttt{Pattern}\ ex\ mu\ s \\
\exists_{\square}.\ \square & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: ex)\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\mu.\ \square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ (s :: mu)\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\lceil \square \rceil^{\square} & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s'
\end{array}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\mathtt{Pattern} & : \mathtt{List}\ \mathit{Sorts} \to \mathtt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\widehat{\widehat{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{In}\ s\ \mathit{ex} \to \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\underline{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{In}\ s\ \mathit{mu} \to \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \cdot \square & : \forall (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
 & \qquad \mathtt{HList}\ ((\mathtt{Pattern}\ \mathit{ex}\ \mathit{mu})\ <\$>\ (\mathit{params}\ \sigma)) \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ (\mathit{return}\ \sigma) \\
\neg\square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \wedge \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\exists_{\square}.\ \square & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{Pattern}\ (s' :: \mathit{ex})\ \mathit{mu}\ s \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\mu.\ \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{Pattern}\ \mathit{ex}\ (s :: \mathit{mu})\ s \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\lceil \square \rceil^{\square} & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \mathtt{List}\ \mathit{Sorts}).\ \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad\qquad \mathtt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s'
\end{array}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\texttt{Pattern} & : \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\Box} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\widehat{\widehat{\Box}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\Box} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ ex \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\underline{\Box}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ mu \to \texttt{Pattern}\ ex\ mu\ s \\
\overline{\Box} \cdot \Box & : \forall (ex\ mu : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
 & \qquad \texttt{HList}\ ((\texttt{Pattern}\ ex\ mu) <\$> (\mathit{params}\ \sigma)) \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ (\mathit{return}\ \sigma) \\
\neg\Box & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\Box \wedge \Box & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad \texttt{Pattern}\ ex\ mu\ s \to \texttt{Pattern}\ ex\ mu\ s \\
\exists_{\Box}.\ \Box & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: ex)\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\mu.\ \Box & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ (s :: mu)\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\lceil \Box \rceil^{\Box} & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s'
\end{array}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\texttt{Pattern} & : \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\widehat{\widehat{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{ex} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\underline{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{mu} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \cdot \square & : \forall (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
& \qquad \texttt{HList}\ ((\texttt{Pattern}\ \mathit{ex}\ \mathit{mu}) \mathbin{<\$>} (\mathit{params}\ \sigma)) \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ (\mathit{return}\ \sigma) \\
\neg \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \wedge \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\exists_{\square}.\ \square & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: \mathit{ex})\ \mathit{mu}\ s \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\mu.\ \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ (s :: \mathit{mu})\ s \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\lceil \square \rceil^{\square} & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
& \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s'
\end{array}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\texttt{Pattern} & : \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\widehat{\widehat{\square}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\square} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ ex \to \texttt{Pattern}\ ex\ mu\ s \\
\underline{\underline{\square}} & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ mu \to \texttt{Pattern}\ ex\ mu\ s \\
\square \cdot \square & : \forall (ex\ mu : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
 & \qquad\qquad \texttt{HList}\ ((\texttt{Pattern}\ ex\ mu) <\$> (\mathit{params}\ \sigma)) \to \\
 & \qquad\qquad\qquad\qquad \texttt{Pattern}\ ex\ mu\ (\mathit{return}\ \sigma) \\
\neg\square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad\qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\square \wedge \square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ ex\ mu\ s \to \texttt{Pattern}\ ex\ mu\ s \\
\exists_{\square}.\ \square & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: ex)\ mu\ s \to \\
 & \qquad\qquad\qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\mu.\ \square & : \forall (s : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ (s :: mu)\ s \to \\
 & \qquad\qquad\qquad\qquad \texttt{Pattern}\ ex\ mu\ s \\
\lceil \square \rceil^{\square} & : \forall (s\ s' : \mathit{Sorts})\ (ex\ mu : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ ex\ mu\ s \to \\
 & \qquad\qquad\qquad\qquad \texttt{Pattern}\ ex\ mu\ s'
\end{array}
$$

Dependently typed syntax

$$
\begin{array}{ll}
\texttt{Pattern} & : \texttt{List}\ \mathit{Sorts} \to \texttt{List}\ \mathit{Sorts} \to \mathit{Sorts} \to \mathit{Type} \\
\widehat{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{EV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\widehat{\widehat{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ (\mathit{SV}\ s) \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\square} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{ex} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\underline{\underline{\square}} & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{In}\ s\ \mathit{mu} \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \cdot \square & : \forall (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts})\ (\sigma : \Sigma). \\
 & \qquad \texttt{HList}\ ((\texttt{Pattern}\ \mathit{ex}\ \mathit{mu}) <\$> (\mathit{params}\ \sigma)) \to \\
 & \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ (\mathit{return}\ \sigma) \\
\neg\square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\square \wedge \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\exists_{\square}.\ \square & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ (s' :: \mathit{ex})\ \mathit{mu}\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\mu.\ \square & : \forall (s : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ (s :: \mathit{mu})\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \\
\lceil \square \rceil^{\square} & : \forall (s\ s' : \mathit{Sorts})\ (\mathit{ex}\ \mathit{mu} : \texttt{List}\ \mathit{Sorts}).\ \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s \to \\
 & \qquad\qquad \texttt{Pattern}\ \mathit{ex}\ \mathit{mu}\ s'
\end{array}
$$

Dependently typed syntax

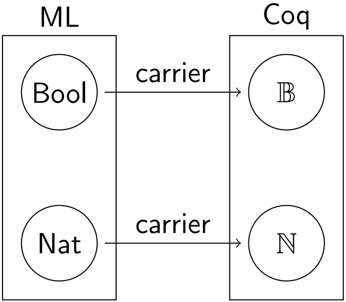
- No positivity
  - Rarely needed
  - Introduces more difficulty than ease-of-use

Dependently typed syntax

- No positivity
  - Rarely needed
  - Introduces more difficulty than ease-of-use
- Substitutions
  - Computable function
  - Defined using recursive descent on all datatypes
  - Dependent equality checks using transport
  - Complications with induction

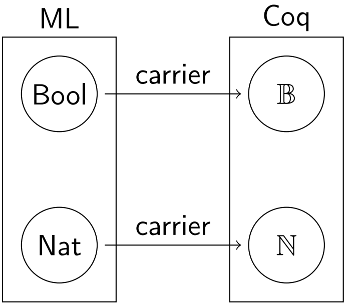
$$
\begin{aligned}
\forall ex\ ex'\ mu\ s\ s'.\ & \mathtt{Pattern}\ ex'\ mu\ s' \to \\
& \mathtt{Pattern}\ (ex \mathbin{+\!\!+} s' :: ex')\ mu\ s \to \\
& \mathtt{Pattern}\ (ex \mathbin{+\!\!+} ex')\ mu\ s
\end{aligned}
$$

Dependently typed semantics



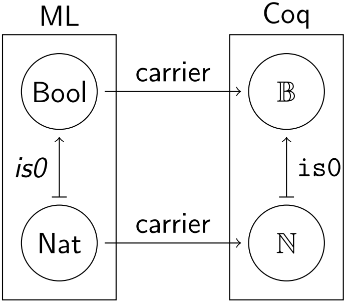
- We need a carrier for each sort
  - Sorts can correspond to meta-theoretic types

Dependently typed semantics



- We need a carrier for each sort
  - Sorts can correspond to meta-theoretic types
- Valuations follow this as well
  - Similar difficulties as with substitutions

Dependently typed semantics



- We need a carrier for each sort
  - Sorts can correspond to meta-theoretic types
- Valuations follow this as well
  - Similar difficulties as with substitutions
- Symbol interpretation can be delegated to meta-theory
  - Shallow embedding
  - Makes proof writing simpler

Future work

- Type level support for subsorting

Future work

- Type level support for subsorting
- Proof system for this implementation

Future work

- Type level support for subsorting
- Proof system for this implementation
- Automatic model generation

Conclusion

- Established a dependently typed description of matching logic
- Defined substitutions in a computable manner
- Created semantics that can map to Coq's types and functions
- Enabled writing proofs over Coq's types using set reasoning

Contact us at:

- Ádám Kurucz: cphfw1@inf.elte.hu
- Péter Bereczky: berpeti@inf.elte.hu
- Dániel Horpácsi: daniel-h@elte.hu

Thank you for your attention!