Extracting Higher-Order Goals from the Mizar Mathematical Library

Chad E. Brown, Josef Urban

Czech Technical University in Prague

July 2016

Outline

Introduction

- Extension of Urban's MPTP translation of Mizar's MML from FO to HO
- Constructs fitting HO:
  - Schemes
  - Global Choice Operator
  - Fraenkel Terms
- Provides problems for HO ATPs
- Evaluation of Satallax and LEO-II on some of these problems.

Outline

Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

$$\forall A. \forall P.$$
$$(\forall x, y, z. Pxy \wedge Pxz \to y = z) \to$$
$$\exists X. \forall x. x \in X \leftrightarrow \exists y. y \in A \wedge Pyx$$

Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

$$\forall A. \forall P.$$
$$(\forall x, y, z. Pxy \land Pxz \to y = z) \to$$
$$\exists X. \forall x. x \in X \leftrightarrow \exists y. y \in A \land Pyx$$

- Problem: Can't write $\forall P$ in FO

Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

$$\forall A. \forall P. \\ (\forall x, y, z. Pxy \wedge Pxz \to y = z) \to \\ \exists X. \forall x. x \in X \leftrightarrow \exists y. y \in A \wedge Pyx$$

- Problem: Can't write $\forall P$ in FO
- For *proving* schemes, not an issue.

Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

$$\forall A.\forall P.$$
$$(\forall x, y, z.Pxy \wedge Pxz \to y = z) \to$$
$$\exists X.\forall x.x \in X \leftrightarrow \exists y.y \in A \wedge Pyx$$

- Problem: Can't write $\forall P$ in FO
- For *proving* schemes, not an issue.
- The issue with *using* schemes.

Using Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

can be used to prove

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

Proof:

Using Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

can be used to prove

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

Proof:

- Let A and P be given.

Using Schemes

```
:: Fraenkel's scheme
scheme Replacement { A() -> set, P[object, object] }:
  ex X st for x holds x in X iff ex y st y in A() & P[y,x]
 provided
  for x,y,z being object st P[x,y] & P[x,z] holds y = z
```

can be used to prove

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

Proof:

- Let A and P be given.
- Instantiate the A and P from Replacement with

$$A := A$$

$$P[x, y] := x = y \land P[x]$$

MPTP and Schemes

- How does MPTP handle schemes?
- Solution staying in FO: Give some FO instances of the scheme
- HO Solution: Simply quantify over P
- HO ATP must find the instance as part of the search, e.g.,

$$\lambda xy.x = y \wedge Px$$

MPTP and Schemes

- How does MPTP handle schemes?
- Solution staying in FO: Give some FO instances of the scheme
- HO Solution: Simply quantify over P
- HO ATP must find the instance as part of the search, e.g.,

$$\lambda xy.x = y \wedge Px$$

- Finding these instantiations is nontrivial.

MPTP and Schemes

- How does MPTP handle schemes?
- Solution staying in FO: Give some FO instances of the scheme
- HO Solution: Simply quantify over P
- HO ATP must find the instance as part of the search, e.g.,

$$\lambda xy.x = y \wedge Px$$

- Finding these instantiations is nontrivial.
- Redundancy. These also work:

$$\lambda xy.x = y \wedge Py$$

$$\lambda xy.y = x \wedge Px$$

$$\lambda xy.y = x \wedge Py$$

MPTP and Schemes

- How does MPTP handle schemes?
- Solution staying in FO: Give some FO instances of the scheme
- HO Solution: Simply quantify over P
- HO ATP must find the instance as part of the search, e.g.,

$$\lambda xy.x = y \wedge Px$$

- Finding these instantiations is nontrivial.
- Redundancy. These also work:

$$\lambda xy.x = y \wedge Py$$

$$\lambda xy.y = x \wedge Px$$

$$\lambda xy.y = x \wedge Py$$

- Many other instantiations of roughly this size don't work.

Global Choice

$$\texttt{the } A$$

where A is a Mizar type.
Examples of Mizar types:

- `set`

Global Choice

$$\texttt{the } A$$

where A is a Mizar type.

Examples of Mizar types:

- `set`

```
theorem Th1: ex X being set st X = X
proof
  take the set;
  thus thesis;
end;
```

Global Choice

$$\texttt{the}\ A$$

where A is a Mizar type.

Examples of Mizar types:

- `set`

```
theorem Th1: ex X being set st X = X
proof
  take the set;
  thus thesis;
end;
```

- `Element of` X whenever X has type `set`

Global Choice

$$\mathtt{the}\ A$$

where A is a Mizar type.
Examples of Mizar types:

- set

```
theorem Th1: ex X being set st X = X
proof
  take the set;
  thus thesis;
end;
```

- Element of X whenever X has type set

```
theorem Th2: for X being set holds ex y being Element of X st y = y
proof
  let X be set;
  take the Element of X;
  thus thesis;
end;
```

Global Choice

$$\texttt{the}\ A$$

where A is a Mizar type.

- All Mizar types are nonempty.

Global Choice

$$\texttt{the } A$$

where A is a Mizar type.

- All Mizar types are nonempty.
- `Element of` X means $\in X$ if X is nonempty, and means $= \emptyset$ if X is empty.

Global Choice

$$\texttt{the } A$$

where A is a Mizar type.

- All Mizar types are nonempty.
- `Element of` X means $\in X$ if X is nonempty, and means $= \emptyset$ if X is empty.
- If X is empty `the Element of` X is $\emptyset$ when X is empty.

Global Choice

$$\texttt{the}\ A$$

where A is a Mizar type.

- All Mizar types are nonempty.
- `Element of` X means $\in X$ if X is nonempty, and means $= \emptyset$ if X is empty.
- If X is empty `the Element of` X is $\emptyset$ when X is empty.

```
theorem Th2: for X being set holds ex y being Element of X st y = y
proof
  let X be set;
  take the Element of X;
  thus thesis;
end;
```

Global Choice and MPTP

- MPTP in FO?

Global Choice and MPTP

- MPTP in FO?
- Deanonymize:
  - Suppose A is a Mizar type depending on x, y.

Global Choice and MPTP

- MPTP in FO?
- Deanonymize:
  - Suppose A is a Mizar type depending on x, y.
  - Translate `the` A as $h(x, y)$ where

$$\forall x, y.\phi_A(h(x, y), x, y)$$

  where ϕ_A is the FO translation of A.

Global Choice and MPTP

- MPTP in FO?
- Deanonymize:
  - Suppose A is a Mizar type depending on x, y.
  - Translate $\mathtt{the}$ A as $h(x, y)$ where

$$\forall x, y.\phi_A(h(x, y), x, y)$$

    where ϕ_A is the FO translation of A.
- MPTP in HO?
- Translate $\mathtt{the}$ A as $\Phi_A\,(\varepsilon(\lambda u.\Phi_A\, u\, x\, y)\, x\, y)$ where ε is a choice operator and Φ_A is the HO translation of A.

Global Choice Example

```
scheme Sch { F(set) -> set, G(set) -> set } :
  (for x being set holds F(x) = G(x))
  implies
  (for x being set holds the Element of F(x) = the Element of G(x))
```

Global Choice Example

```
scheme Sch { F(set) -> set, G(set) -> set } :
  (for x being set holds F(x) = G(x))
  implies
  (for x being set holds the Element of F(x) = the Element of G(x))
```

FO MPTP translation. Let $x \hat{\in} Y$ be translation of x `Element of` Y.

- $\forall x. F(x) = G(x)$
- $\forall x. h(x) \hat{\in} F(x)$
- $\forall x. k(x) \hat{\in} G(x)$

Conjecture: $\forall x. h(x) = k(x)$

Global Choice Example

```
scheme Sch { F(set) -> set, G(set) -> set } :
  (for x being set holds F(x) = G(x))
  implies
  (for x being set holds the Element of F(x) = the Element of G(x))
```

FO MPTP translation. Let $x \hat{\in} Y$ be translation of x `Element of` Y.

- $\forall x.F(x) = G(x)$
- $\forall x.h(x) \hat{\in} F(x)$
- $\forall x.k(x) \hat{\in} G(x)$

Conjecture: $\forall x.h(x) = k(x)$

Not actually a theorem!

Global Choice Example

```
scheme Sch { F(set) -> set, G(set) -> set } :
  (for x being set holds F(x) = G(x))
  implies
  (for x being set holds the Element of F(x) = the Element of G(x))
```

FO MPTP translation. Let $x \hat{\in} Y$ be translation of x `Element of` Y.

- $\forall x.F(x) = G(x)$
- $\forall x.h(x) \hat{\in} F(x)$
- $\forall x.k(x) \hat{\in} G(x)$

Conjecture: $\forall x.h(x) = k(x)$

Not actually a theorem!

HO MPTP translation. Axioms:

- $\forall qx.qx \to q(\varepsilon q)$ (Choice)
- $\forall x.Fx = Gx$

Conjecture: $\forall x.\varepsilon(\lambda u.u \hat{\in} (Fx)) = \varepsilon(\lambda u.u \hat{\in} (Gx))$

Easy theorem.

Fraenkel Terms

$$\{t \text{ where } x \text{ is } A : P\}$$

where

- t is a Mizar term,
- A is a "setlike" Mizar type and
- P is a Mizar proposition.

Fraenkel Terms

$$\{t \text{ where } x \text{ is } A : P\}$$

where

- t is a Mizar term,
- A is a "setlike" Mizar type and
- P is a Mizar proposition.

$$\{t \text{ where } x_1 \text{ is } A_1, \ldots, x_n \text{ is } A_n : P\}$$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

FO MPTP version, deanonymize:

- $h(Y)$ such that

$$\forall z.z \in h(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge P(x)$$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

FO MPTP version, deanonymize:

- $h(Y)$ such that

$$\forall z.z \in h(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge P(x)$$

- $k(Y)$ such that

$$\forall z.z \in k(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge Q(x)$$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

FO MPTP version, deanonymize:

- $h(Y)$ such that

$$\forall z.z \in h(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge P(x)$$

- $k(Y)$ such that

$$\forall z.z \in k(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge Q(x)$$

- Assume $\forall x.P(x) \leftrightarrow Q(x)$.

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

FO MPTP version, deanonymize:

- $h(Y)$ such that

$$\forall z.z \in h(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge P(x)$$

- $k(Y)$ such that

$$\forall z.z \in k(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge Q(x)$$

- Assume $\forall x.P(x) \leftrightarrow Q(x)$.
- Prove $\forall Y.h(Y) = k(Y)$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

FO MPTP version, deanonymize:

- $h(Y)$ such that

$$\forall z.z \in h(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge P(x)$$

- $k(Y)$ such that

$$\forall z.z \in k(Y) \leftrightarrow \exists x.x \hat{\in} Y \wedge z = \{x\} \wedge Q(x)$$

- Assume $\forall x.P(x) \leftrightarrow Q(x)$.
- Prove $\forall Y.h(Y) = k(Y)$
- Set extensionality and information about $\hat{\in}$ is also given, which is enough to make the FO problem provable.

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

HO MPTP version, use a declared constant `replSep1`

- $\mathtt{replSep}_1 : (\iota o)(\iota\iota)(\iota o)\iota.$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

HO MPTP version, use a declared constant `replSep1`

- $\texttt{replSep}_1 : (\iota o)(\iota\iota)(\iota o)\iota$.
- Assume $\forall x. Px \leftrightarrow Qx$.

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

HO MPTP version, use a declared constant $\mathtt{replSep1}$

- $\mathtt{replSep}_1 : (\iota o)(\iota\iota)(\iota o)\iota$.
- Assume $\forall x.Px \leftrightarrow Qx$.
- Conjecture:

$$\begin{aligned} & \mathtt{replSep}_1\ (\lambda x.x\hat{\in}Y)\ (\lambda x.\{x\})\ (\lambda x.Px) \\ = {} & \mathtt{replSep}_1\ (\lambda x.x\hat{\in}Y)\ (\lambda x.\{x\})\ (\lambda x.Qx) \end{aligned}$$

Fraenkel Example

```
scheme sch1 {P[set],Q[set]} :
  (for x holds P[x] iff Q[x])
  implies
  for Y holds
  {{x} where x is Element of Y : P[x]}
  = {{x} where x is Element of Y : Q[x]}
```

HO MPTP version, use a declared constant replSep1

- $\mathtt{replSep}_1 : (\iota o)(\iota\iota)(\iota o)\iota$.
- Assume $\forall x. Px \leftrightarrow Qx$.
- Conjecture:

$$\begin{aligned} &\mathtt{replSep}_1\ (\lambda x. x \hat{\in} Y)\ (\lambda x. \{x\})\ (\lambda x. Px) \\ = &\mathtt{replSep}_1\ (\lambda x. x \hat{\in} Y)\ (\lambda x. \{x\})\ (\lambda x. Qx) \end{aligned}$$

- Easy HO theorem (using extensionality rules).

Outline

Simple Type Theory

Simple types:

- ι - individuals (Mizar sets/objects)
- o - propositions
- $\alpha\beta$ - functions from α to β

Simple Type Theory

Simple types:

- ι - individuals (Mizar sets/objects)
- o - propositions
- $\alpha\beta$ - functions from α to β

Simply typed terms:

- Typed variables: $x : \alpha$
- Typed constants: $c : \alpha$
- Application: $st : \beta$ where $s : \alpha\beta$ and $t : \alpha$
- Abstraction: $\lambda x.t : \alpha\beta$ where $x : \alpha$ and $t : \beta$
- Implication: $s \to t : o$ where $s, t : o$
- Quantification: $\forall x.t : o$ where $x : \alpha$ and $t : o$

Simple Type Theory

Simple types:

- ι - individuals (Mizar sets/objects)
- o - propositions
- $\alpha\beta$ - functions from α to β

Simply typed terms:

- Typed variables: $x : \alpha$
- Typed constants: $c : \alpha$
- Application: $st : \beta$ where $s : \alpha\beta$ and $t : \alpha$
- Abstraction: $\lambda x.t : \alpha\beta$ where $x : \alpha$ and $t : \beta$
- Implication: $s \to t : o$ where $s, t : o$
- Quantification: $\forall x.t : o$ where $x : \alpha$ and $t : o$

Proofs: Usual rules, including extensionality

Outline

Variables and Constants

Inherit variables and constants from STT.

- $x : \iota$ *object variables*, $c : \iota$ *object constants*

Variables and Constants

Inherit variables and constants from STT.

- $x : \iota$ *object variables*, $c : \iota$ *object constants*
- $F : \underbrace{\iota \ldots \iota}_{n}\iota$ *function variables of arity n*
- $f : \underbrace{\iota \ldots \iota}_{n}\iota$ *function constants of arity n*

Variables and Constants

Inherit variables and constants from STT.

- $x : \iota$ *object variables*, $c : \iota$ *object constants*
- $F : \underbrace{\iota \ldots \iota}_{n}\iota$ *function variables of arity n*
- $f : \underbrace{\iota \ldots \iota}_{n}\iota$ *function constants of arity n*
- $P : \underbrace{\iota \ldots \iota}_{n}o$ *predicate variables of arity n*
- $p : \underbrace{\iota \ldots \iota}_{n}o$ *predicate constants of arity n*

Variables and Constants

Inherit variables and constants from STT.

- $x : \iota$ *object variables*, $c : \iota$ *object constants*
- $F : \underbrace{\iota \ldots \iota}_{n}\iota$ *function variables of arity* n
- $f : \underbrace{\iota \ldots \iota}_{n}\iota$ *function constants of arity* n
- $P : \underbrace{\iota \ldots \iota}_{n}o$ *predicate variables of arity* n
- $p : \underbrace{\iota \ldots \iota}_{n}o$ *predicate constants of arity* n
- Predicate constants also play the role of Mizar *modes* or *attributes*.

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-types:

- $\mathtt{set}$
- $p(\cdot, T_1, \ldots, T_n)$ where p $n+1$-ary predicate constant (p mode)
- $q\ A$ and $\mathtt{non}\ q\ A$ where q is a unary predicate constant (q attribute)

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-terms:

- object variables and object constants
- $F(T_1, \ldots, T_n)$
- $f(T_1, \ldots, T_n)$

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-terms:

- object variables and object constants
- $F(T_1, \ldots, T_n)$
- $f(T_1, \ldots, T_n)$
- $(\mathtt{the}\ A)$ (global choice)

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-terms:

- object variables and object constants
- $F(T_1, \ldots, T_n)$
- $f(T_1, \ldots, T_n)$
- $(\mathtt{the}\ A)$ (global choice)
- $\{T\ \mathtt{where}\ x_1\ \mathtt{is}\ A_1, \ldots x_n\ \mathtt{is}\ A_n : \Phi\}$ (Fraenkel terms)

M-types, M-terms and M-propositions

Mutually recursive definitions of

- M-types $A, B, \ldots$
- M-terms $S, T, \ldots$
- M-propositions $\Phi, \Psi, \ldots$

M-propositions

- $P[T_1, \ldots, T_n]$
- $p[T_1, \ldots, T_n]$
- $(S = T)$ and $(S\ \mathtt{in}\ T)$
- $(\mathtt{not}\ \Phi)$
- $(\Phi\ \&\ \Psi)$, $(\Phi\ \mathtt{or}\ \Psi)$, $(\Phi\ \mathtt{implies}\ \Psi)$ and $(\Phi\ \mathtt{iff}\ \Psi)$
- $(\mathtt{for}\ x\ \mathtt{being}\ A\ \mathtt{holds}\ \Phi)$
- $(\mathtt{ex}\ x\ \mathtt{being}\ A\ \mathtt{st}\ \Phi)$

M-statements

A prefix Γ is a list of variable declarations:

- $x : A$
- $F(A_1, \ldots, A_n) : B$
- $P[A_1, \ldots, A_n]$

An M-statement is a prefix and an M-proposition.

M-statements

A prefix Γ is a list of variable declarations:

- $x : A$
- $F(A_1, \ldots, A_n) : B$
- $P[A_1, \ldots, A_n]$

An M-statement is a prefix and an M-proposition.

Example:

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

M-statements

A prefix Γ is a list of variable declarations:

- $x : A$
- $F(A_1, \ldots, A_n) : B$
- $P[A_1, \ldots, A_n]$

An M-statement is a prefix and an M-proposition.

Example:

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

- Γ is the prefix

$$A : \mathtt{set}, P[\mathtt{set}]$$

M-statements

A prefix Γ is a list of variable declarations:

- $x : A$
- $F(A_1, \ldots, A_n) : B$
- $P[A_1, \ldots, A_n]$

An M-statement is a prefix and an M-proposition.

Example:

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

- Γ is the prefix

$$A : \mathtt{set}, P[\mathtt{set}]$$

- Φ is the M-proposition

$$\mathtt{ex}\ X\ \mathtt{being\ set\ st\ for}\ x\ \mathtt{being\ set\ holds}\\ x\ \mathtt{in}\ X\ \mathtt{iff}\ x\ \mathtt{in}\ A\ \&\ P(x)$$

M-statements

A prefix Γ is a list of variable declarations:

- $x : A$
- $F(A_1, \ldots, A_n) : B$
- $P[A_1, \ldots, A_n]$

An M-statement is a prefix and an M-proposition.

Example:

```
scheme Separation { A()-> set, P[set] } :
  ex X being set st for x being set holds x in X iff x in A() & P[x]
```

- Γ is the prefix

$$A : \mathtt{set}, P[\mathtt{set}]$$

- Φ is the M-proposition

$$\mathtt{ex}\ X\ \mathtt{being\ set\ st\ for}\ x\ \mathtt{being\ set\ holds}$$
$$x\ \mathtt{in}\ X\ \mathtt{iff}\ x\ \mathtt{in}\ A\ \&\ P(x)$$

- The M-statement is (Γ, Φ)

Outline

Introduction

Three Constructs

Simple Type Theory

Idealized Mizar

Translation

Results

Conclusion

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι

- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner (\Gamma, \Phi) \urcorner$ of type o (schemes)

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner (\Gamma, \Phi) \urcorner$ of type o (schemes)
- $\ulcorner \texttt{the } A \urcorner = \varepsilon \ulcorner A \urcorner$

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner(\Gamma, \Phi)\urcorner$ of type o (schemes)
- $\ulcorner \texttt{the } A \urcorner = \varepsilon \ulcorner A \urcorner$
- $\ulcorner \{T \text{ where } x \text{ is } A : \Phi\} \urcorner =$ $\texttt{replSep}_1 \ulcorner A \urcorner\ (\lambda x. \ulcorner T \urcorner)\ (\lambda x. \ulcorner \Phi \urcorner)$

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner (\Gamma, \Phi) \urcorner$ of type o (schemes)
- $\ulcorner \mathtt{the}\ A \urcorner = \varepsilon \ulcorner A \urcorner$
- $\ulcorner \{T \text{ where } x \text{ is } A : \Phi\} \urcorner =$ $\mathtt{replSep}_1\ \ulcorner A \urcorner\ (\lambda x. \ulcorner T \urcorner)\ (\lambda x. \ulcorner \Phi \urcorner)$
- $\ulcorner (\cdot, \Phi) \urcorner = \ulcorner \Phi \urcorner.$
- $\ulcorner ((x : A, \Gamma), \Phi) \urcorner = \forall x. \ulcorner A \urcorner\ x \to \ulcorner (\Gamma, \Phi) \urcorner.$

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner (\Gamma, \Phi) \urcorner$ of type o (schemes)
- $\ulcorner \mathtt{the}\ A \urcorner = \varepsilon \ulcorner A \urcorner$
- $\ulcorner \{T \text{ where } x \text{ is } A : \Phi\} \urcorner =$ $\mathtt{replSep}_1\ \ulcorner A \urcorner\ (\lambda x. \ulcorner T \urcorner)\ (\lambda x. \ulcorner \Phi \urcorner)$
- $\ulcorner (\cdot, \Phi) \urcorner = \ulcorner \Phi \urcorner.$
- $\ulcorner ((x : A, \Gamma), \Phi) \urcorner = \forall x. \ulcorner A \urcorner\ x \to \ulcorner (\Gamma, \Phi) \urcorner.$
- $\ulcorner ((F(A_1, \ldots, A_n) : B, \Gamma), \Phi) \urcorner = \forall F. (\forall x_1. \ulcorner A_1 \urcorner\ x_1 \to \ldots \to \forall x_n. \ulcorner A_n \urcorner\ x_n \to \ulcorner B \urcorner\ (F x_1 \cdots x_n)) \to \ulcorner (\Gamma, \Phi) \urcorner.$

Translation

- M-types A translate to terms $\ulcorner A \urcorner$ of type ιo
- M-terms T translate to terms $\ulcorner T \urcorner$ of type ι (choice, Fraenkels)
- M-propositions Φ translate to terms $\ulcorner \Phi \urcorner$ of type o
- M-statements (Γ, Φ) also translate to terms $\ulcorner(\Gamma, \Phi)\urcorner$ of type o (schemes)
- $\ulcorner \mathtt{the}\ A \urcorner = \varepsilon \ulcorner A \urcorner$
- $\ulcorner \{T \text{ where } x \text{ is } A : \Phi\} \urcorner =$ $\mathtt{replSep}_1\ \ulcorner A \urcorner\ (\lambda x. \ulcorner T \urcorner)\ (\lambda x. \ulcorner \Phi \urcorner)$
- $\ulcorner(\cdot, \Phi)\urcorner = \ulcorner \Phi \urcorner.$
- $\ulcorner((x : A, \Gamma), \Phi)\urcorner = \forall x. \ulcorner A \urcorner\ x \to \ulcorner(\Gamma, \Phi)\urcorner.$
- $\ulcorner((F(A_1, \ldots, A_n) : B, \Gamma), \Phi)\urcorner = \forall F. (\forall x_1. \ulcorner A_1 \urcorner\ x_1 \to \ldots \to \forall x_n. \ulcorner A_n \urcorner\ x_n \to \ulcorner B \urcorner\ (F x_1 \cdots x_n)) \to \ulcorner(\Gamma, \Phi)\urcorner.$
- $\ulcorner((P[A_1, \ldots, A_n], \Gamma), \Phi)\urcorner = \forall P. \ulcorner(\Gamma, \Phi)\urcorner.$

Translation (General Fraenkel Case)

$$\ulcorner\{T \text{ where } x_1 \text{ is } A_1, \ldots x_n \text{ is } A_n : \Phi\}\urcorner$$
$$= \mathtt{replSep}_n \ulcorner A_1 \urcorner\, (\lambda x_1.A_2) \cdots\, (\lambda x_1 \cdots x_{n-1}.\ulcorner A_n \urcorner)$$
$$(\lambda x_1 \cdots x_n.\ulcorner T \urcorner)\, (\lambda x_1 \cdots x_n.\ulcorner \Phi \urcorner)$$

Translation (General Fraenkel Case)

$$\ulcorner\{T \text{ where } x_1 \text{ is } A_1, \ldots x_n \text{ is } A_n : \Phi\}\urcorner$$
$$= \mathtt{replSep}_n \ulcorner A_1 \urcorner \, (\lambda x_1.A_2) \cdots \, (\lambda x_1 \cdots x_{n-1}.\ulcorner A_n \urcorner)$$
$$(\lambda x_1 \cdots x_n.\ulcorner T \urcorner) \, (\lambda x_1 \cdots x_n.\ulcorner \Phi \urcorner)$$

- $\mathtt{replSep}_n : (\iota o) \cdots (\iota \cdots \iota o)(\iota \cdots \iota\iota)(\iota \cdots \iota o)$

Translation (General Fraenkel Case)

$$\ulcorner\{T \text{ where } x_1 \text{ is } A_1, \ldots x_n \text{ is } A_n : \Phi\}\urcorner$$
$$= \texttt{replSep}_n\ \ulcorner A_1 \urcorner\ (\lambda x_1.A_2) \cdots\ (\lambda x_1 \cdots x_{n-1}.\ulcorner A_n \urcorner)$$
$$(\lambda x_1 \cdots x_n.\ulcorner T \urcorner)\ (\lambda x_1 \cdots x_n.\ulcorner \Phi \urcorner)$$

- $\texttt{replSep}_n : (\iota o) \cdots (\iota \cdots \iota o)(\iota \cdots \iota\iota)(\iota \cdots \iota o)$
- Axioms for $\texttt{replSep}_n$:

Translation (General Fraenkel Case)

$$\ulcorner\{T \text{ where } x_1 \text{ is } A_1, \ldots x_n \text{ is } A_n : \Phi\}\urcorner$$
$$= \mathtt{replSep}_n \ulcorner A_1\urcorner\ (\lambda x_1.A_2) \cdots\ (\lambda x_1 \cdots x_{n-1}.\ulcorner A_n\urcorner)$$
$$(\lambda x_1 \cdots x_n.\ulcorner T\urcorner)\ (\lambda x_1 \cdots x_n.\ulcorner\Phi\urcorner)$$

- $\mathtt{replSep}_n : (\iota o) \cdots (\iota \cdots \iota o)(\iota \cdots \iota\iota)(\iota \cdots \iota o)$
- Axioms for $\mathtt{replSep}_n$:
- $\mathtt{replSepE}_n$: If $z \in \mathtt{replSep}_n A_1 \cdots A_n F P$, then there exist $x_1, \ldots, x_n$ such that $A_1 x_1$, …, $A_n x_1 \cdots x_n$, $z = F x_1 \cdots x_n$ and $P x_1 \cdots x_n$.

Translation (General Fraenkel Case)

$$\ulcorner\{T \text{ where } x_1 \text{ is } A_1, \ldots x_n \text{ is } A_n : \Phi\}\urcorner$$
$$= \mathtt{replSep}_n\ \ulcorner A_1\urcorner\ (\lambda x_1.A_2) \cdots\ (\lambda x_1 \cdots x_{n-1}.\ulcorner A_n\urcorner)$$
$$(\lambda x_1 \cdots x_n.\ulcorner T\urcorner)\ (\lambda x_1 \cdots x_n.\ulcorner \Phi\urcorner)$$

- $\mathtt{replSep}_n : (\iota o)\cdots(\iota\cdots\iota o)(\iota\cdots\iota\iota)(\iota\cdots\iota o)$
- Axioms for $\mathtt{replSep}_n$:
- $\mathtt{replSepE}_n$: If $z \in \mathtt{replSep}_n A_1 \cdots A_n F P$, then there exist $x_1, \ldots, x_n$ such that $A_1 x_1$, ..., $A_n x_1 \cdots x_n$, $z = F x_1 \cdots x_n$ and $P x_1 \cdots x_n$.
- $\mathtt{replSepI}_n$: If A_1 is "setlike", $A_1 x_1$, ..., $A_n x_1 \cdots x_{n-1}$ is "setlike", $A_n x_1 \cdots x_n$ and $P x_1 \cdots x_n$, then $F x_1 \cdots x_n \in \mathtt{replSep}_n A_1 \cdots A_n F P$.

Outline

Problems Sets

- Top level justifications (`by`)

Problems Sets

- Top level justifications (`by`)
  - involving global choice: 47

Problems Sets

- Top level justifications (`by`)
  - involving global choice: 47
  - involving Fraenkels: 245

Problems Sets

- Top level justifications (`by`)
  - involving global choice: 47
  - involving Fraenkels: 245
- Scheme justifications (`from`): 10192

Problems Sets

- Top level justifications (`by`)
  - involving global choice: 47
  - involving Fraenkels: 245
- Scheme justifications (`from`): 10192
- Scheme Proofs: 610

Global Choice Justification Problems

Ran Satallax and LEO-II for 5 minutes with default settings.

- 47 total
- Satallax: 24 (51%)
- LEO-II: 28 (60%)
- Either: 30 (64%)

Fraenkel Justification Problems

Ran Satallax and LEO-II for 5 minutes with default settings.

- 245 total
- Satallax: 126 (52%)
- LEO-II: 88 (36%)
- Either: 165 (67%)

Fraenkel Justification Problems

Ran Satallax and LEO-II for 5 minutes with default settings.

- 245 total
- Satallax: 126 (52%)
- LEO-II: 88 (36%)
- Either: 165 (67%)

Idea: Use E to indicate which FO axioms it needs to do the FO version. "Prune" the HO version.

Fraenkel Justification Problems

Ran Satallax and LEO-II for 5 minutes with default settings.

- 245 total
- Satallax: 126 (52%)
- LEO-II: 88 (36%)
- Either: 165 (67%)

Idea: Use E to indicate which FO axioms it needs to do the FO version. "Prune" the HO version.

- 245 total
- Satallax: 159 (65%)
- LEO-II: 155 (63%)
- Either: 192 (78%)

Scheme Justification Problems

Ran Satallax and LEO-II for 5 minutes with default settings.

- 10192 total
- Satallax: 5608 (55%)
- LEO-II: 1524 (15%)
- Either: 6072 (60%)

Full Schemes

Ran Satallax and LEO-II for 5 minutes with default settings.

- 610 total
- Satallax: 31 (5%)
- LEO-II: 67 (11%)
- Either: 81 (13%)

Outline

Conclusion

- Some Mizar constructs translate in a "higher-order" way.
- The translation gives higher-order problem sets

Conclusion

- Some Mizar constructs translate in a “higher-order” way.
- The translation gives higher-order problem sets
- ...which are challenging for HO ATPs.

Conclusion

- Some Mizar constructs translate in a "higher-order" way.
- The translation gives higher-order problem sets
- ...which are challenging for HO ATPs.
- Future Work:
- Should theorem provers always prove "Mizar-obvious" problems?

Conclusion

- Some Mizar constructs translate in a “higher-order” way.
- The translation gives higher-order problem sets
- ...which are challenging for HO ATPs.
- Future Work:
- Should theorem provers always prove “Mizar-obvious” problems?
- Should Mizar’s type system be preserved by the translation and used by theorem provers?

Conclusion

- Some Mizar constructs translate in a "higher-order" way.
- The translation gives higher-order problem sets
- ...which are challenging for HO ATPs.
- Future Work:
- Should theorem provers always prove "Mizar-obvious" problems?
- Should Mizar's type system be preserved by the translation and used by theorem provers?
- Translating proofs back to Mizar