the Mizar type system

Freek Wiedijk
Radboud University Nijmegen

TPHOLs 2007
University of Kaiserslautern

2007 09 12, 10:00
this talk

• Mizar

• Mizar types

• the paper
  – the Mizar type system in the form of typing rules
  – correctness with respect to first order predicate logic
## Mizar versus the HOLs

<table>
<thead>
<tr>
<th>Mizar</th>
<th>HOL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Isabelle</strong></td>
</tr>
<tr>
<td></td>
<td>PVS</td>
</tr>
<tr>
<td></td>
<td>Coq</td>
</tr>
</tbody>
</table>

- batch checking
- first order logic
- readable proofs
- untyped set theory
- very nice type system

interactive

higher order logic
tactic scripts
typed foundations
Mizar types

features

- dependent types
- no types built from types
  no function types $A \rightarrow B$ for types $A$ and $B$
  (‘Ordinal $\rightarrow$ Ordinal’?)
- subtyping
- ‘attributes’
- structure types (records)

Alternative Aggregates in Mizar
Gilbert Lee and Piotr Rudnicki
MKM 2007, LNAI 4573
attributes

\[
\begin{array}{c}
\text{non empty} & \text{finite} & \text{Subset of NAT} \\
\uparrow & \uparrow & \\
\text{adjectives} & \text{radix type} & \\
\text{attributes} & \text{mode} & \\
\end{array}
\]

overloading

meaning of an attribute depends on the radix type:

- connected Relation
- connected Graph
- connected TopSpace
definition let d be non zero Element of NAT;
func cyclotomic_poly d -> Polynomial of F_Complex means
ex s being non empty finite Subset of F_Complex
  st s = { y where y is Element of MultGroup F_Complex : ord y = d } &
  it = poly_with_roots((s,1)-bag);
end;

UNIROOTS Primitive Roots of Unity and Cyclotomic Polynomials
Broderick Arneson and Piotr Rudnicki
3293 lines, 135K
full Mizar library 985 ‘articles’, 1.97 million lines, 68.9M
typings also are formulas

for i being Integer holds $i \geq 0$ iff $i$ is Nat

\[ \forall i : \mathbb{Z}. i \geq 0 \Leftrightarrow (i : \mathbb{N}) \]

\[ \begin{align*}
\text{i is Nat} & \uparrow \not\equiv \text{i in NAT} \\
\text{type} & \uparrow \text{term} \\
\text{‘has type’} & \uparrow \text{‘is element of’} \\
\text{the type system} & \uparrow \text{the set theory}
\end{align*} \]

‘coerce’ a term to a more informative type by giving a proof

... then $n' - n \geq 0$ by XREAL_1:50;
then reconsider $d = n' - n$ as Nat by INT_1:16;
...
subtyping

variable

definition let C be Category;

    mode Subcategory of C -> Category means

:: CAT_2:def 4

    ...

end;

set

\[ \uparrow \]

Category

\[ \uparrow \]

Subcategory of C

supertype may depend on the types of the arguments of the type
for $X$ being set holds $X$ is empty implies $X$ is finite

cluster empty $\rightarrow$ finite set

any term with attribute empty automatically also gets attribute finite
typed or untyped logic?

what does it all mean?

foundations of Mizar:

Tarski-Grothendieck set theory

=  

ZFC + ‘there are arbitrarily large inaccessible cardinals’

set of axioms on top of untyped first order predicate logic
the axioms of Mizar

| TARSKI: def 3 | $X \subseteq Y \iff (\forall x. x \in X \Rightarrow x \in Y)$ |
| TARSKI: def 5 | $\langle x, y \rangle = \{\{x, y\}, \{x\}\}$ |
| TARSKI: def 6 | $X \sim Y \iff \exists Z. (\forall x. x \in X \Rightarrow \exists y. y \in Y \land \langle x, y \rangle \in Z) \land$
| | $(\forall y. y \in Y \Rightarrow \exists x. x \in X \land \langle x, y \rangle \in Z) \land$
| | $(\forall x \forall y \forall z \forall u. \langle x, y \rangle \in Z \land \langle z, u \rangle \in Z \Rightarrow (x = z \iff y = u))$ |
| TARSKI: def 1 | $x \in \{y\} \iff x = y$ |
| TARSKI: def 2 | $x \in \{y, z\} \iff x = y \lor x = z$ |
| TARSKI: def 4 | $x \in \bigcup X \iff \exists Y. x \in Y \land Y \in X$ |
| TARSKI:2 | $(\forall x. x \in X \iff x \in Y) \Rightarrow X = Y$ |
| TARSKI:7 | $x \in X \Rightarrow \exists Y. Y \in X \land \lnot \exists x. x \in X \land x \in Y$ |
| TARSKI:sch 1 | $(\forall x \forall y \forall z. P[x, y] \land P[x, z] \Rightarrow y = z) \Rightarrow$
| | $(\exists X. \forall x. x \in X \iff \exists y. y \in A \land P[y, x])$ |
| TARSKI:9 | $\exists M. N \in M \land (\forall X \forall Y. X \in M \land Y \subseteq X \Rightarrow Y \in M) \land$
| | $(\forall X. X \in M \Rightarrow \exists Z. Z \in M \land \forall Y. Y \subseteq X \Rightarrow Y \in Z) \land$
| | $(\forall X. X \subseteq M \Rightarrow X \sim M \lor X \in M)$ |
for $i$ being Integer holds $i \geq 0$ iff $i$ is Nat

$$\forall i: \text{Integer}. \ i \geq 0 \Leftrightarrow (i: \text{Nat})$$

$$\downarrow$$

$$\forall i. \text{Integer}(i) \Rightarrow [i \geq 0 \Leftrightarrow \text{Nat}(i)]$$

types are just predicates that the system manages automatically

dependent types with $n$ arguments are predicates with $n+1$ arguments
the type system as typing rules

symbolic notation

\[ t ::= x \mid f(t) \]  
\[ R ::= \ast \mid M(t) \]  
\[ a ::= \alpha \mid \bar{\alpha} \]  
\[ T ::= \vec{a}R \]  
\[ J ::= \cdot \mid t : T \mid T \leq T \mid \exists T \mid \alpha / T \]  
\[ D ::= x : T \]  
\[ \Delta ::= \vec{D} \]  
\[ \Gamma ::= [\Delta](J) \]  
\[ \Gamma; \Delta \vdash J \]

term variables  
function symbols  
mode symbols  
attribute symbols  
terms  
radius types  
adjectives  
types  
judgment elements  
declarations  
judgments
twenty-two typing rules

three examples:

\[
\Gamma; \vec{x} : \vec{T} \vdash \exists T'
\]

\[
\Gamma, [\vec{x} : \vec{T}] (M(\vec{x}) \leq T'), [\vec{x} : \vec{T}] (\exists M(\vec{x})); \vdash \cdot
\]

mode definition:

\[
\frac{\Gamma; \vec{x} : \vec{T} \vdash \exists T'}{\Gamma, [\vec{x} : \vec{T}] (M(\vec{x}) \leq T'), [\vec{x} : \vec{T}] (\exists M(\vec{x})); \vdash \cdot}
\]

conditional cluster:

\[
\frac{\Gamma; \Delta \vdash \vec{a}T' \leq T' \quad \Gamma; \Delta \vdash \vec{a}' T' \leq T'}{\Gamma, [\Delta] (\vec{a} T' \leq \vec{a}' T'); \vdash \cdot}
\]
correctness

translating judgments

type judgment → first order sequent

\[ \text{int} \leq \star, \exists \text{int}, \text{pos/int}, \exists \text{pos int}; x : \text{pos int} \vdash x : \star \]

→

\[ (\forall x. \text{int}(x) \Rightarrow \top), (\exists x. \text{int}(x)), \top, (\exists x. \text{pos}(x) \land \text{int}(x)), (\text{pos}(x) \land \text{int}(x)) \vdash \top \]

main theorem

‘the type system is correct’

derivable judgment → provable sequent
system in the paper is an idealization

definition let n be Nat;
  redefine mode Element of n → Element of n + 1;
end;

according to the rules from the paper we have

Element of n → Element of n + 1 → Element of n + 2 → ...

in the actual Mizar system we just have

Element of n → Element of n + 1
why not have something like the Mizar type system yourself?

- the Mizar proof language is well-known to be nice
- the Mizar type system is less known, but very nice too
- every system can have the Mizar type system as a layer on top