Practical Computer Formalization of Mathematics Using Mizar

Adam Naumowicz
adamn@mizar.org

Institute of Informatics,
University of Bialystok, Poland

July 31, 2011
Adam Naumowicz: Introduction to the Mizar proof-assistant
About this tutorial

- Adam Naumowicz: Introduction to the Mizar proof-assistant
- Artur Korniłowicz: Practical presentation of Mizar
About this tutorial

- Adam Naumowicz: Introduction to the Mizar proof-assistant
- Artur Korniłowicz: Practical presentation of Mizar
- Jesse Alama: The dependency structure of Mizar Mathematical Library
About this tutorial

- Adam Naumowicz: Introduction to the Mizar proof-assistant
- Artur Korniłowicz: Practical presentation of Mizar
- Jesse Alama: The dependency structure of Mizar Mathematical Library
- Josef Urban: Artificial intelligence methods in automated reasoning
About this tutorial

- Adam Naumowicz: Introduction to the Mizar proof-assistant
- Artur Korniłowicz: Practical presentation of Mizar
- Jesse Alama: The dependency structure of Mizar Mathematical Library
- Josef Urban: Artificial intelligence methods in automated reasoning
- Piotr Rudnicki: On-line mini-tutorial
  http://mizar.uwb.edu.pl/CADE23-tutorial/MiniTut/New/
What is MIZAR?

- A bit of history
- Language – system – database
- Related projects
Outline

- What is MIZAR?
  - A bit of history
  - Language – system – database
  - Related projects
- Theoretical foundations
  - The system of semantic correlates in MIZAR
  - Proof strategies
  - Types in MIZAR
  - More advanced language constructs
  - Recently implemented features
Outline

- What is MIZAR?
  - A bit of history
  - Language – system – database
  - Related projects
- Theoretical foundations
  - The system of semantic correlates in MIZAR
  - Proof strategies
  - Types in MIZAR
  - More advanced language constructs
  - Recently implemented features
- Practical usage
  - Running the system
  - Importing notions from the library (building the environment)
  - Enhancing MIZAR texts
What is MIZAR?

The MIZAR project started around 1973 as an attempt to reconstruct mathematical vernacular in a computer-oriented environment.

- A formal language for writing mathematical proofs
- A computer system for verifying correctness of proofs
- The library of formalized mathematics – MIZAR Mathematical Library (MML)

For more information see:
- The language's grammar
- The bibliography of the MIZAR project
- Free download of binaries for several platforms
- Discussion forum(s)
- MIZAR User Service - e-mail contact point

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland

Practical Computer Formalization of Mathematics Using Mizar
What is MIZAR?

- The MIZAR project started around 1973 as an attempt to reconstruct mathematical vernacular in a computer-oriented environment
  - A formal language for writing mathematical proofs
  - A computer system for verifying correctness of proofs
  - The library of formalized mathematics – MIZAR Mathematical Library (MML)

For more information see:
- The language's grammar
- The bibliography of the MIZAR project
- Free download of binaries for several platforms
- Discussion forum(s)
- MIZAR User Service - e-mail contact point

Adam Naumowicz adamn@mizar.org
Institute of Informatics, University of Białystok, Poland
What is MIZAR?

- The MIZAR project started around 1973 as an attempt to reconstruct mathematical vernacular in a computer-oriented environment
  - A formal language for writing mathematical proofs
  - A computer system for verifying correctness of proofs
  - The library of formalized mathematics – MIZAR Mathematical Library (MML)
- For more information see http://mizar.org
  - The language’s grammar
  - The bibliography of the MIZAR project
  - Free download of binaries for several platforms
  - Discussion forum(s)
  - MIZAR User Service - e-mail contact point
The MIZAR language

The proof language is designed to be as close as possible to "mathematical vernacular". It is a reconstruction of the language of mathematics, forming a subset of standard English used in mathematical texts. It is based on a declarative style of natural deduction. There are 27 special symbols, 110 reserved words. The language is highly structured to ensure producing rigorous and semantically unambiguous texts. It allows prefix, postfix, infix notations for predicates as well as parenthetical notations for functors.

Similar ideas:
- MV (Mathematical Vernacular - N. G. de Bruijn)
- CML (Common Mathematical Language)
- QED Project (http://www-unix.mcs.anl.gov/qed/) - The QED Manifesto from 1994

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
The MIZAR language

- The proof language is designed to be as close as possible to "mathematical vernacular"
  - It is a reconstruction of the language of mathematics
  - It forms "a subset" of standard English used in mathematical texts
  - It is based on a declarative style of natural deduction
  - There are 27 special symbols, 110 reserved words
  - The language is highly structured - to ensure producing rigorous and semantically unambiguous texts
  - It allows prefix, postfix, infix notations for predicates as well as parenthetical notations for functors
The MIZAR language

- The proof language is designed to be as close as possible to “mathematical vernacular”
  - It is a reconstruction of the language of mathematics
  - It forms “a subset” of standard English used in mathematical texts
  - It is based on a declarative style of natural deduction
  - There are 27 special symbols, 110 reserved words
  - The language is highly structured - to ensure producing rigorous and semantically unambiguous texts
  - It allows prefix, postfix, infix notations for predicates as well as parenthetical notations for functors

- Similar ideas:
  - MV (Mathematical Vernacular - N. G. de Bruijn)
  - CML (Common Mathematical Language)
  - QED Project (http://www-unix.mcs.anl.gov/qed/) - The QED Manifesto from 1994
Key features of the MIZAR system

The system uses classical first-order logic. Statements with free second-order variables (e.g., the induction scheme) are supported. The system uses natural deduction for doing conditional proofs.

S. Ja´skowski, On the rules of supposition formal logic. Studia Logica, 1, 1934.

The system uses a declarative style of writing proofs (mostly forward reasoning) - resembling mathematical practice. A system of semantic correlates is used for processing formulas (as introduced by R. Suszko in his investigations of non-Fregean logic). The system as such is independent of the axioms of set theory.

Adam Naumowicz adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
Key features of the MIZAR system

- The system uses classical first-order logic
Key features of the MIZAR system

- The system uses classical first-order logic
- Statements with free second-order variables (e.g. the induction scheme) are supported
Key features of the MIZAR system

- The system uses classical first-order logic
- Statements with free second-order variables (e.g. the induction scheme) are supported
- The system uses natural deduction for doing conditional proofs

Key features of the MIZAR system

- The system uses classical first-order logic
- Statements with free second-order variables (e.g. the induction scheme) are supported
- The system uses natural deduction for doing conditional proofs
- The system uses a declarative style of writing proofs (mostly forward reasoning) - resembling mathematical practice
Key features of the MIZAR system

- The system uses classical first-order logic
- Statements with free second-order variables (e.g. the induction scheme) are supported
- The system uses natural deduction for doing conditional proofs
- The system uses a declarative style of writing proofs (mostly forward reasoning) - resembling mathematical practice
- A system of semantic correlates is used for processing formulas (as introduced by R. Suszko in his investigations of non-Fregean logic)
Key features of the MIZAR system

- The system uses classical first-order logic
- Statements with free second-order variables (e.g. the induction scheme) are supported
- The system uses natural deduction for doing conditional proofs
- The system uses a declarative style of writing proofs (mostly forward reasoning) - resembling mathematical practice
- A system of semantic correlates is used for processing formulas (as introduced by R. Suszko in his investigations of non-Fregean logic)
- The system as such is independent of the axioms of set theory
Related systems

Systems influenced by MIZAR comprise:

- Mizar mode for HOL (J. Harrison)
- Declare (D. Syme)
- Isar (M. Wenzel)
- Mizar-light for HOL-light (F. Wiedijk)
- MMode/DPL - declarative proof language for Coq (P. Corbineau)
- ...

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
“A good system without a library is useless. A good library for a bad system is still very interesting... So the library is what counts.”
(F. Wiedijk, Estimating the Cost of a Standard Library for a Mathematical Proof Checker.)
“A good system without a library is useless. A good library for a bad system is still very interesting... So the library is what counts.”

(F. Wiedijk, Estimating the Cost of a Standard Library for a Mathematical Proof Checker.)

- A systematic collection of articles started around 1989

MIZAR Mathematical Library - MML
“A good system without a library is useless. A good library for a bad system is still very interesting... So the library is what counts.”
(F. Wiedijk, Estimating the Cost of a Standard Library for a Mathematical Proof Checker.)

- A systematic collection of articles started around 1989
- Recent MML version - 4.160.1126
  - includes 1122 articles written by 236 authors
  - 51244 theorems
  - 9993 definitions
  - 789 schemes
  - 10593 registrations
"A good system without a library is useless. A good library for a bad system is still very interesting... So the library is what counts."
(F. Wiedijk, Estimating the Cost of a Standard Library for a Mathematical Proof Checker.)

- A systematic collection of articles started around 1989
- Recent MML version - 4.160.1126
  - includes 1122 articles written by 236 authors
  - 51244 theorems
  - 9993 definitions
  - 789 schemes
  - 10593 registrations
- The library is based on the axioms of Tarski-Grothendieck set theory
## Basic kinds of MIZAR formulas

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>contradiction</td>
</tr>
<tr>
<td>¬α</td>
<td>not α</td>
</tr>
<tr>
<td>α ∧ β</td>
<td>α &amp; β</td>
</tr>
<tr>
<td>α ∨ β</td>
<td>α or β</td>
</tr>
<tr>
<td>α → β</td>
<td>α implies β</td>
</tr>
<tr>
<td>α ↔ β</td>
<td>α iff β</td>
</tr>
<tr>
<td>∀ₓα(x)</td>
<td>for x holds α(x)</td>
</tr>
<tr>
<td>∃ₓα(x)</td>
<td>ex st α(x)</td>
</tr>
</tbody>
</table>
MIZAR’s main logical module - the CHECKER

There is no set of inference rules - M. Davis’s concept of “obviousness w.r.t an algorithm”
The de Bruijn criterion of a small checker is not preserved
The deductive power is still being strengthened (CAS and DS integration)
new computation mechanisms added
more automation in the equality calculus
experiments with more than one general statement in an inference (“Scordev’s device”)
MIZAR’s main logical module - the CHECKER

- There is no set of inference rules - M. Davis’s concept of “obviousness w.r.t an algorithm”
MIZAR’s main logical module - the CHECKER

- There is no set of inference rules - M. Davis’s concept of “obviousness w.r.t an algorithm”
- The de Bruijn criterion of a small checker is not preserved
MIZAR’s main logical module - the CHECKER

- There is no set of inference rules - M. Davis’s concept of “obviousness w.r.t an algorithm”
- The de Bruijn criterion of a small checker is not preserved
- The deductive power is still being strengthened (CAS and DS integration)
  - new computation mechanisms added
  - more automation in the equality calculus
  - experiments with more than one general statement in an inference (“Scordev’s device”)

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
MIZAR as a disprover

An inference of the form \( \alpha_1, \ldots, \alpha_k \beta \) is transformed to \( \alpha_1, \ldots, \alpha_k, \neg \beta \). A disjunctive normal form (DNF) of the premises is then created and the system tries to refute it:

\[
\neg \alpha_1, 1 \land \cdots \land \neg \alpha_k, 1 \lor \cdots \lor \neg \alpha_{n}, 1 \land \cdots \land \neg \alpha_{n}, k_n \end{equation}

where \( \alpha_i, j \) are atomic or universal sentences (negated or not) - for the inference to be accepted, all disjuncts must be refuted. So in fact \( n \) inferences are checked.

Adam Naumowicz  
admn@mizar.org  
Institute of Informatics, University of Bialystok, Poland  
Practical Computer Formalization of Mathematics Using Mizar
MIZAR as a disprover

An inference of the form

\[ \frac{\alpha^1, \ldots, \alpha^k}{\beta} \]

is transformed to

\[ \frac{\alpha^1, \ldots, \alpha^k, \neg \beta}{\bot} \]

A disjunctive normal form (DNF) of the premises is then created and the system tries to refute it

\[ \left( (\neg \alpha^{1,1} \land \cdots \land \neg \alpha^{1,k_1}) \lor \cdots \lor (\neg \alpha^{n,1} \land \cdots \land \neg \alpha^{n,k_n}) \right) \]

\[ \bot \]

where \( \alpha^{i,j} \) are atomic or universal sentences (negated or not) - for the inference to be accepted, all disjuncts must be refuted. So in fact \( n \) inferences are checked

\[ \left( (\neg \alpha^{1,1} \land \cdots \land \neg \alpha^{1,k_1}) \lor \cdots \lor (\neg \alpha^{n,1} \land \cdots \land \neg \alpha^{n,k_n}) \right) \]

\[ \bot \]
The system of MIZAR’s semantic correlates

Internally, all MIZAR formulas are expressed in a simplified “canonical” form - their semantic correlates using only \textsc{verum}, not, and for holds together with atomic formulas. \textsc{verum} is the neutral element of the conjunction Double negation rule is used de Morgan’s laws are used for disjunction and existential quantifiers \( \alpha \) implies \( \beta \) is changed into \( \neg(\alpha \& \neg \beta) \) \( \alpha \) iff \( \beta \) is changed into \( \alpha \) implies \( \beta \) & \( \beta \) implies \( \alpha \), i.e. \( \neg(\alpha \& \neg \beta) \) & \( \neg(\beta \& \neg \alpha) \) conjunction is associative but not commutative
The system of MIZAR’s semantic correlates

Internally, all MIZAR formulas are expressed in a simplified “canonical” form - their semantic correlates using only VERUM, not, & and for _ holds _ together with atomic formulas.
The system of MIZAR’s semantic correlates

Internally, all MIZAR formulas are expressed in a simplified “canonical” form - their semantic correlates using only VERUM, not, & and for which holds together with atomic formulas.

- VERUM is the neutral element of the conjunction
- Double negation rule is used
- de Morgan’s laws are used for disjunction and existential quantifiers
- $\alpha$ implies $\beta$ is changed into not($\alpha$ & not $\beta$)
- $\alpha$ iff $\beta$ is changed into $\alpha$ implies $\beta$ & $\beta$ implies $\alpha$, i.e. not($\alpha$ & not $\beta$) & not($\beta$ & not $\alpha$)
- conjunction is associative but not commutative
Basic proof strategies – Propositional calculus
Basic proof strategies – Propositional calculus

- Deduction rule

\[ A \implies B \quad \therefore \thesis = A \implies B \]

proof

assume A;

...\n
thus B;

end;

\[ \therefore \thesis = \emptyset \]
Basic proof strategies – Propositional calculus

- **Deduction rule**
  
  \[ A \text{ implies } B \]  
  
  **proof**  
  
  assume \( A \);  
  
  ...  
  
  thus \( B \);  
  
  end;  
  
  :: thesis = \( A \) implies \( B \)

- **Adjunction rule**  
  
  \( A \& B \)  
  
  **proof**  
  
  ...  
  
  thus \( A \);  
  
  ...  
  
  thus \( B \);  
  
  end;  
  
  :: thesis = \( A \& B \)

Adam Naumowicz  
adamn@mizar.org  
Institute of Informatics, University of Bialystok, Poland  
Practical Computer Formalization of Mathematics Using Mizar
Basic proof strategies – Quantifier calculus

Generalization rule for \( x \) holds \( A(x) \) :: thesis = for \( x \) holds \( A(x) \)
proof
let \( a \); :: thesis = A(a)
...
thus A(a); :: thesis = {}
end;

Exemplification rule ex \( x \) st \( A(x) \) :: thesis = ex \( x \) st \( A(x) \)
proof
take \( a \); :: thesis = A(a)
...
thus A(a); :: thesis = {}
end;
Basic proof strategies – Quantifier calculus

- **Generalization rule**
  
  for x holds A(x) :: thesis = for x holds A(x)
  
  proof
  
  let a;
  
  \ldots
  
  thus A(a); :: thesis = {} 
  
  end;
Basic proof strategies – Quantifier calculus

- Generalization rule
  for x holds A(x) :: thesis = for x holds A(x)
  proof
  let a;
  ...
  thus A(a); :: thesis = {} end;

- Exemplification rule
  ex x st A(x) :: thesis = ex x st A(x)
  proof
  take a;
  ...
  thus A(a); :: thesis = {} end;

Adam Naumowicz
adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
More proof strategies

A :: thesis = A
proof
  assume not A; :: thesis = contradiction
  ...
  thus contradiction; :: thesis = {}
end;

... :: thesis = ...
proof
  assume not thesis; :: thesis = contradiction
  ...
  thus contradiction; :: thesis = {}
end;
More proof strategies – ctd.

... :: thesis = ...
proof
  assume not thesis; :: thesis = contradiction
  ...
  thus thesis; :: thesis = {}
end;

A & B implies C :: thesis = A & B implies C
proof
  assume A; :: thesis = B implies C
  ...
  assume B; :: thesis = C
  ...
  thus C; :: thesis = {}
end;
More proof strategies – ctd.

A implies (B implies C):: thesis = A implies (B implies C)
proof
  assume A; :: thesis = B implies C
  ...
  assume B; :: thesis = C
  ...
  thus C; :: thesis = {}
end;

A or B or C or D :: thesis = A or B or C or D
proof
  assume not A :: thesis = B or C or D
  ...
  assume not B; :: thesis = C or D
  ...
  thus C or D; :: thesis = {}
Types in MIZAR

A hierarchy based on the "widening" relation with:
- set
- Function of X,Y $\succ$ PartFunc of X,Y $\succ$ Relation of X,Y $\succ$ Subset of [:X,Y:] $\succ$ Element of bool [:X,Y:] $\succ$ set

MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn).

One-to-one Function of X,Y
finite non empty proper Subset of X

Adjectives are processed to enable automatic deriving of type information (so called "registrations").

Types also play a syntactic role - e.g. enable overloading of notations.

The type of a variable can be "reserved" and then not used explicitly.

MIZAR types are required to have a non-empty denotation (existence must be proved when defining a type).
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  
  Function of $X,Y \succ PartFunc$ of $X,Y \succ Relation$ of $X,Y \succ$
  
  Subset of $[:X,Y:] \succ Element$ of $bool$ $[:X,Y:] \succ set$
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  Function of $X,Y \succ$ PartFunc of $X,Y \succ$ Relation of $X,Y \succ$
  Subset of $[:X,Y:] \succ$ Element of bool $[:X,Y:] \succ$ set

- MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn)
  one-to-one Function of $X,Y$
  finite non empty proper Subset of $X$
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  Function of $X,Y \succ$ PartFunc of $X,Y \succ$ Relation of $X,Y \succ$
  Subset of $[:X,Y:] \succ$ Element of bool $[:X,Y:] \succ$ set

- MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn)
  one-to-one Function of $X,Y$
  finite non empty proper Subset of $X$

- Adjectives are processed to enable automatic deriving of type information (so called "registrations")
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  Function of $X,Y \succ$ PartFunc of $X,Y \succ$ Relation of $X,Y \succ$
  Subset of $[:X,Y:] \succ$ Element of bool $[:X,Y:] \succ$ set

- MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn)
  one-to-one Function of $X,Y$
  finite non empty proper Subset of $X$

- Adjectives are processed to enable automatic deriving of type information (so called “registrations”)

- Types also play a syntactic role - e.g. enable overloading of notations
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  Function of $X,Y \succ PartFunc$ of $X,Y \succ Relation$ of $X,Y \succ Subset$ of $[[:X,Y:] \succ Element$ of $bool$ $[[:X,Y:] \succ set$

- MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn)
  one-to-one Function of $X,Y$
  finite non empty proper Subset of $X$

- Adjectives are processed to enable automatic deriving of type information (so called “registrations”)

- Types also play a syntactic role - e.g. enable overloading of notations

- The type of a variable can be “reserved” and then not used explicitly
Types in MIZAR

- A hierarchy based on the “widening” relation with set being the widest type
  Function of $X,Y \succ$ PartFunc of $X,Y \succ$ Relation of $X,Y \succ$
  Subset of [:X,Y:] \succ Element of bool [:X,Y:] \succ set

- MIZAR types are refined using adjectives ("key linguistic entities used to represent mathematical concepts" according to N.G. de Bruijn)
  one-to-one Function of $X,Y$
  finite non empty proper Subset of $X$

- Adjectives are processed to enable automatic deriving of type information (so called “registrations”)

- Types also play a syntactic role - e.g. enable overloading of notations

- The type of a variable can be “reserved” and then not used explicitly

- MIZAR types are required to have a non-empty denotation (existence must be proved when defining a type)
Types in MIZAR – ctd.

Dependent types

**definition**

let \( C \) be Category, \( a, b, c \) be Object of \( C \), \( f \) be Morphism of \( a, b \), \( g \) be Morphism of \( b, c \);

assume \( \text{Hom}(a, b) \neq \emptyset \) & \( \text{Hom}(b, c) \neq \emptyset \);

func \( g \cdot f \) -> Morphism of \( a, c \) equals

\[
\text{:: CAT}_1: \text{def 13}
\]

\( g \cdot f \);

...correctness...

end.

Adam Naumowicz  adamm@mizar.org

Institute of Informatics, University of Bialystok, Poland

Practical Computer Formalization of Mathematics Using Mizar
Dependent types

\textbf{definition}

\texttt{let C be Category}\n\texttt{a, b, c be Object of C,}\n\texttt{f be Morphism of a, b,}\n\texttt{g be Morphism of b, c;}\n\texttt{assume Hom(a, b)<>{} & Hom(b, c)<>{};}\n\texttt{func g*f -> Morphism of a, c equals}\n\texttt{:: CAT_1: def 13}\n\texttt{g*f;}\n\texttt{...correctness...}\n\texttt{end;}

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Structural types (with a sort of polymorphic inheritance) - abstract vs. concrete part of MML definition

let F be 1-sorted;
struct(LoopStr) VectSpStr over F
(\#carrier \rightarrow \text{set},
add \rightarrow \text{BinOp of the carrier},
ZeroF \rightarrow \text{Element of the carrier},
lmult \rightarrow \text{Function of } [:\text{the carrier of } F,\text{the carrier}],\text{the carrier} \#);
end;

Adam Naumowicz adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
Structural types (with a sort of polymorphic inheritance) - abstract vs. concrete part of MML

definition
let F be 1-sorted;
struct(LoopStr) VectSpStr over F
(
  carrier -> set,
  add -> BinOp of the carrier,
  ZeroF -> Element of the carrier,
  lmult -> Function of
    [:the carrier of F, the carrier:], the carrier
);
end;
More advanced language constructs
More advanced language constructs

- Iterative equalities
- Schemes
- Redefinitions
- Synonyms/antonyms
- “properties”
  - E.g. commutativity, reflexivity, etc.
- “requirements”
  - E.g. the built-in arithmetic on complex numbers
More advanced language constructs - ctd.

- Identifying (formally different, but equal) constructors
- Support for global choice in the language
- Adjective completion in equality classes
- Adjectives with visible arguments
  - E.g. n-dimensional, NAT-valued, etc.

Adam Naumowicz
adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
More advanced language constructs - ctd.

- Identifying (formally different, but equal) constructors

Adam Naumowicz  
adamn@mizar.org

Institute of Informatics, University of Bialystok, Poland
More advanced language constructs - ctd.

- Identifying (formally different, but equal) constructors
- Support for global choice in the language
More advanced language constructs - ctd.

- Identifying (formally different, but equal) constructors
- Support for global choice in the language
- Adjective completion in equality classes
More advanced language constructs - ctd.

- Identifying (formally different, but equal) constructors
- Support for global choice in the language
- Adjective completion in equality classes
- Adjectives with visible arguments
  - E.g. n-dimensional, NAT-valued, etc.
Running the system

- Logical modules (passes) of the MIZAR verifier
- Communication with the database
Running the system

- Logical modules (passes) of the MIZAR verifier
  - parser (**tokenizer** + identification of so-called “long terms”)

- Communication with the database
Running the system

- Logical modules (passes) of the MIZAR verifier
  - parser (tokenizer + identification of so-called “long terms”)
  - analyzer (+ reasoner)

- Communication with the database
Running the system

- Logical modules (passes) of the MIZAR verifier
  - parser (tokenizer + identification of so-called “long terms”)
  - analyzer (+ reasoner)
  - checker (preparator, prechecker, equalizer, unifier) + schematizer
- Communication with the database
Running the system

- Logical modules (passes) of the MIZAR verifier
  - **parser** (tokenizer + identification of so-called “long terms”)
  - **analyzer** (+ reasoner)
  - **checker** (preparator, prechecker, equalizer, unifier) + schematizer
- Communication with the database
  - **accommodator**
Running the system

- Logical modules (passes) of the MIZAR verifier
  - **parser** (**tokenizer** + identification of so-called “long terms”)
  - **analyzer** (+ **reasoner**)
  - **checker** (**preparator**, **prechecker**, **equalizer**, **unifier**) + **schematizer**

- Communication with the database
  - **accommodator**
  - **exporter** + **transferer**
Running the system – ctd.
The interface (CLI, Emacs Mizar Mode by Josef Urban, “remote processing”)
Running the system – ctd.

- The interface (CLI, Emacs Mizar Mode by Josef Urban, “remote processing”)
  - The way Mizar reports errors resembles a compiler’s errors and warnings
  - Top-down approach
  - Stepwise refinement
  - It’s possible to check correctness of incomplete texts
  - One can postpone a proof or its more complicated part
Enhancing MIZAR texts

Utilities detecting irrelevant parts of proofs
relprem
relinfer
reliters
trivdemo

Checking new versions of system implementation

Adam Naumowicz adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Enhancing MIZAR texts

- Utilities detecting irrelevant parts of proofs
  - relprem
  - relinfer
  - reliters
  - trivdemo
  - ...

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland
Practical Computer Formalization of Mathematics Using Mizar
Enhancing MIZAR texts

- Utilities detecting irrelevant parts of proofs
  - relprem
  - relinfer
  - reliters
  - trivdemo
  - ...

- Checking new versions of system implementation
Importing notions from the library

The structure of MIZAR input files

- `environ`
- `begin`

Library directives
- `vocabulary`
- `(using symbols)`
- `constructors`
- `(using introduced objects)`
- `notations`
- `(using notations of objects)`
- `theorems`
- `(referencing theorems)`
- `schemes`
- `(referencing schemes)`
- `definitions`
- `(automated unfolding of definitions)`
- `registrations`
- `(automated processing of adjectives)`
- `requirements`
- `(using built-in enhancements for certain constructors, e.g. complex numbers)`

Using a local database

Adam Naumowicz  adamn@mizar.org
Institute of Informatics, University of Bialystok, Poland

Practical Computer Formalization of Mathematics Using Mizar
Importing notions from the library

The structure of MIZAR input files

- environ
  .....  
- begin
  .....
Importing notions from the library

- The structure of MIZAR input files
  - environ
    - ..... begin
    - ..... 

- Library directives
  - vocabularies (using symbols)
  - constructors (using introduced objects)
  - notations (using notations of objects)
  - theorems (referencing theorems)
  - schemes (referencing schemes)
  - definitions (automated unfolding of definitions)
  - registrations (automated processing of adjectives)
  - requirements (using built-in enhancements for certain constructors, e.g. complex numbers)
Importing notions from the library

- The structure of MIZAR input files
  ```
  environ
  .....  
  begin
  .....  
  ```

- Library directives
  - vocabularies (using symbols)
  - constructors (using introduced objects)
  - notations (using notations of objects)
  - theorems (referencing theorems)
  - schemes (referencing schemes)
  - definitions (automated unfolding of definitions)
  - registrations (automated processing of adjectives)
  - requirements (using built-in enhancements for certain constructors, e.g. complex numbers)

- Using a local database
Miscelanea

- Formalized Mathematics - FM ([http://mizar.org/fm](http://mizar.org/fm))
- MMLQuery - search engine for MML ([http://mmlquery.mizar.org](http://mmlquery.mizar.org))
- MIZAR TWiki ([http://wiki.mizar.org](http://wiki.mizar.org))
- MIZAR mode for GNU Emacs ([http://wiki.mizar.org/twiki/bin/view/Mizar/MizarMode](http://wiki.mizar.org/twiki/bin/view/Mizar/MizarMode))
- MoMM - interreduction and retrieval of matching theorems from MML ([http://wiki.mizar.org/twiki/bin/view/Mizar/MoMM](http://wiki.mizar.org/twiki/bin/view/Mizar/MoMM))
- MIZAR Proof Advisor ([http://wiki.mizar.org/twiki/bin/view/Mizar/MizarProofAdvisor](http://wiki.mizar.org/twiki/bin/view/Mizar/MizarProofAdvisor))
Recommended reading

- F. Wiedijk, Writing a Mizar article in nine easy steps. (http://www.cs.ru.nl/~freek/mizar/mizman.ps.gz)