A Brief Overview of MIZAR

Adam Naumowicz
adamn@mizar.org

Institute of Informatics,
University of Bialystok, Poland

August 19th, 2009

Outline

Outline

- 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

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
- Examples: formalizing the friendship puzzle

What is 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)

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 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

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
  - S. Jaśkowski, On the rules of supposition formal logic. *Studia Logica*, 1, 1934.
  - F. B. Fitch, *Symbolic Logic. An Introduction*. The Ronald Press Company, 1952.
  - K. Ono, On a practical way of describing formal deductions. *Nagoya Mathematical Journal*, 21, 1962.

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. Jaśkowski, On the rules of supposition formal logic. *Studia Logica*, 1, 1934.
  - F. B. Fitch, *Symbolic Logic. An Introduction*. The Ronald Press Company, 1952.
  - K. Ono, On a practical way of describing formal deductions. *Nagoya Mathematical Journal*, 21, 1962.
- 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
  - S. Jaśkowski, On the rules of supposition formal logic. *Studia Logica*, 1, 1934.
  - F. B. Fitch, *Symbolic Logic. An Introduction*. The Ronald Press Company, 1952.
  - K. Ono, On a practical way of describing formal deductions. *Nagoya Mathematical Journal*, 21, 1962.
- 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
  - S. Jaśkowski, On the rules of supposition formal logic. *Studia Logica*, 1, 1934.
  - F. B. Fitch, *Symbolic Logic. An Introduction*. The Ronald Press Company, 1952.
  - K. Ono, On a practical way of describing formal deductions. *Nagoya Mathematical Journal*, 21, 1962.
- 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)
- ...

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.)

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

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
- Current MML version - 4.117.1046
  - includes 1047 articles written by 219 authors
  - 48199 theorems
  - 9262 definitions
  - 757 schemes
  - 8573 registrations

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
- Current MML version - 4.117.1046
  - includes 1047 articles written by 219 authors
  - 48199 theorems
  - 9262 definitions
  - 757 schemes
  - 8573 registrations
- The library is based on the axioms of Tarski-Grothendieck set theory

Basic kinds of MIZAR formulas

$\perp$	`contradiction`
$\neg\alpha$	`not` α
$\alpha \wedge \beta$	α `&` β
$\alpha \vee \beta$	α `or` β
$\alpha \rightarrow \beta$	α `implies` β
$\alpha \leftrightarrow \beta$	α `iff` β
$\forall_x \alpha(x)$	`for` x `holds` $\alpha(x)$
$\exists_x \alpha(x)$	`ex` x `st` $\alpha(x)$

MIZAR's main logical module - the CHECKER

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")

MIZAR as a disprover

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

$$\frac{([\neg]\alpha^{1,1} \wedge \cdots \wedge [\neg]\alpha^{1,k_1}) \vee \cdots \vee ([\neg]\alpha^{n,1} \wedge \cdots \wedge [\neg]\alpha^{n,k_n})}{\bot}$$

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

$$\frac{[\neg]\alpha^{1,1} \wedge \cdots \wedge [\neg]\alpha^{1,k_1}}{\bot}$$

$$\ldots$$

$$\frac{[\neg]\alpha^{n,1} \wedge \cdots \wedge [\neg]\alpha^{n,k_n}}{\bot}$$

The system of MIZAR's semantic correlates

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 _ 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
- α `implies` β is changed into `not(`α `& not` β`)`
- α `iff` β is changed into α `implies` β `&` β `implies` α, i.e. `not(`α `& not` β`) & not(`β `& not` α`)`
- conjunction is associative but not commutative

Basic proof strategies – Propositional calculus

Basic proof strategies – Propositional calculus

- Deduction rule

```
A implies B                 :: thesis = A implies B
 proof
  assume A;                 :: thesis = B
  ...
  thus B;                   :: thesis = {}
 end;
```

Basic proof strategies – Propositional calculus

- Deduction rule

```
A implies B                    :: thesis = A implies B
 proof
  assume A;                    :: thesis = B
  ...
  thus B;                      :: thesis = {}
 end;
```

- Adjunction rule

```
A & B                          :: thesis = A & B
 proof
  ...
  thus A;                      :: thesis = B
  ...
  thus B;                      :: thesis = {}
 end;
```

Basic proof strategies – Quantifier calculus

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;
```

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;
```

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

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 explicitely

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

Types in MIZAR – ctd.

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 Hom(a,b)<>{} & Hom(b,c)<>{};
   func g*f -> Morphism of a,c equals
:: CAT_1:def 13
  g*f;
...correctness...
end;
```

Types in MIZAR – ctd.

Types in MIZAR – ctd.

- Structural types (with a sort of polimorfic 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

Recently implemented features

Recently implemented features

- Identifying (formally different, but equal) constructors

Recently implemented features

- Identifying (formally different, but equal) constructors
- Support for global choice in the language

Recently implemented features

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

Recently implemented features

- 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.

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

Enhancing MIZAR texts

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

Enhancing MIZAR texts

- Utilities detecting irrelevant parts of proofs
  - `relprem`
  - `relinfer`
  - `reliters`
  - `trivdemo`
  - ...
- Checking new versions of system implementation

Importing notions from the library

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

Exemplary students' tasks

```
reserve R,S,T for Relation;

R is transitive implies R*R c= R
proof
  assume a: R is transitive;
  let a,b;
  assume [a,b] in R*R; then
  consider c such that
  c: [a,c] in R & [c,b] in R
                       by RELATION:def 7;
  thus [a,b] in R by c,a,RELATION:def 12;
end;
```

Exemplary students' tasks

```
reserve R,S,T for Relation;

R is transitive implies R*R c= R
proof
  assume a: R is transitive;
  let a,b;
  assume [a,b] in R*R; then
  consider c such that
  c: [a,c] in R & [c,b] in R
                       by RELATION:def 7;
  thus [a,b] in R by c,a,RELATION:def 12;
end;
```

```
ex R,S,T st not R*(S \ T) c= (R*S) \ (R*T)
proof
  reconsider R={[1,2],[1,3]} as Relation
             by RELATION:2;
  reconsider S={[2,1]} as Relation
             by RELATION:1;
  reconsider T={[3,1]} as Relation
             by RELATION:1;
  take R,S,T;
  b: [1,2] in R by ENUMSET:def 4;
  d: [2,1] in S by ENUMSET:def 3;
  [2,1] <> [3,1] by ENUMSET:2; then
  not [2,1] in T by ENUMSET:def 3; then
  [2,1] in S \ T by d,RELATION:def 6; then
  a: [1,1] in R*(S \ T) by b,RELATION:def 7;
  e: [1,3] in R by ENUMSET:def 4;
  [3,1] in T by ENUMSET:def 3; then
  [1,1] in R*T by e,RELATION:def 7; then
  not [1,1] in (R*S) \ (R*T) by RELATION:def 6;
  hence not R*(S \ T) c= (R*S) \ (R*T)
            by RELATION:def 9,a;
end;
```

Exemplary students' tasks

```
reserve i,j,k,l,m,n for natural number;

i+k = j+k implies i=j;
proof
  defpred P[natural number] means
          i+$1 = j+$1 implies i=j;
  A1: P[0]
  proof
   assume B0: i+0 = j+0;
   B1: i+0 = i by INDUCT:3;
   B2: j+0 = j by INDUCT:3;
   hence thesis by B0,B1,B2;
  end;
  A2: for k st P[k] holds P[succ k]
  proof
    let l such that C1: P[l];
    assume C2: i+succ l=j+succ l;
    then C3: succ(i+l) = j+succ l by C2,INDUCT:4
    .= succ(j+l) by INDUCT:4;
    hence thesis by C1,INDUCT:2;
  end;
  for k holds P[k] from INDUCT:sch 1(A1,A2);
  hence thesis;
end;
```

Formalizing the friendship puzzle

In any cocktail party with two or more people, there must be at least two people who have the same number of friends. Assume that friend is symmetric - if x is a friend of y, then y is a friend of x.

Formalizing the friendship puzzle

In any cocktail party with two or more people, there must be at least two people who have the same number of friends. Assume that friend is symmetric - if x is a friend of y, then y is a friend of x.

```
scheme Friendship {P()->finite non trivial set,Friends[set,set]}:
  ex x,y being Element of P() st x<>y &
  card {z where z is Element of P() : Friends[x,z]} =
  card {z where z is Element of P() : Friends[y,z]}
provided
  for x holds not Friends[x,x] and
  for x,y st Friends[x,y] holds Friends[y,x]
```

Miscelanea

- Formalized Mathematics - FM (http://mizar.org/fm)
- XML-ized presentation of MIZAR articles (http://mizar.uwb.edu.pl/version/current/html)
- MMLQuery - search engine for MML (http://mmlquery.mizar.org)
- MIZAR TWiki (http://wiki.mizar.org)
- MIZAR mode for GNU Emacs (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)
- MIZAR Proof Advisor (http://wiki.mizar.org/twiki/bin/view/Mizar/MizarProofAdvisor)

Recommended reading

- P. Rudnicki, To type or not to type, QED Workshop II, Warsaw 1995. (ftp://ftp.mcs.anl.gov/pub/qed/workshop95/by-person/10piotr.ps)
- A. Trybulec, Checker (a collection of e-mails compiled by F. Wiedijk). (http://www.cs.ru.nl/~freek/mizar/by.ps.gz)
- M. Wenzel and F. Wiedijk, A comparison of the mathematical proof languages Mizar and Isar. (http://www4.in.tum.de/~wenzelm/papers/romantic.pdf)
- F. Wiedijk, Mizar: An Impression. (http://www.cs.ru.nl/~freek/mizar/mizarintro.ps.gz)
- F. Wiedijk, Writing a Mizar article in nine easy steps. (http://www.cs.ru.nl/~freek/mizar/mizman.ps.gz)
- F. Wiedijk (ed.), The Seventeen Provers of the World. LNAI 3600, Springer Verlag 2006. (http://www.cs.ru.nl/~freek/comparison/comparison.pdf)