New features of the Mizar system

Artur Korniłowicz
artur@mizar.org

Project URL: http://mizar.org/

Institute of Informatics
University of Bialystok

9th Theorem Proving and Provers
Nagano, 21–22 November 2013

Outline

- Introduction
- Changes in syntax
- Changes in semantics
- Changes in MML
- Other changes

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)

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
- The language is highly structured – to ensure producing rigorous and semantically unambiguous texts
- It allows prefix, infix and postfix notations for predicates as well as parenthetical notations for functors

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
- The system as such is independent of the axioms of set theory

Mizar Mathematical Library MML

- A systematic collection of articles started in 1989
- The library is based on the axioms of Tarski-Grothendieck set theory
- Current MML version (5.20.1189) includes
  - 1198 articles
  - 53379 theorems
  - 10928 definitions
  - 829 schemes
  - 11652 registrations

Changes in syntax

- Collective formulas
- Fraenkel operator
- `sethood`

Changes in semantics

- Definitional expansions
- Reductions
- Ellipsis
- Order of adjectives

Changes in MML

- **object** type
- ...

Other changes

- XML
- Pragmas

Collective formulas

```
A c= B & B c= C implies A c= C;

A c= B c= C implies A c= C;
```

```
r < s & s <= t implies r < t;

r < s <= t implies r < t;
```

Fraenkel operator

```
{ f" where f is Element of Funcs(A,B):
     not contradiction }
```

```
the set of all f" where f is Element of Funcs(A,B)
```

Sethood

- Syntax
- Properties
- Applications

Sethood – example 1

```
definition
  let X;
  mode Element of X -> set means
  it in X if X is non empty otherwise it is empty;
  sethood;
end;
```

Sethood – example 2

```
registration
  sethood of Complex
  proof
    thus ex A being set st
      for c being Complex holds c in A;
  end;
end;
```

Sethood – properties

- Inheritance by subtypes
- Inheritance by redefined notions
- Inheritance by adjectives

Sethood – applications

```
{ A where A is Subset of X : card A = 2 }
```

```
{ i^2 where i is Integer : i is even }
```

Definitional expansions in Reasoner

```
P c= R implies P~ c= R~
proof
 assume
A1:  P c= R;
 let x;
 assume
A2:  x in P~;
 then consider a,b such that
A3: x = [a,b] by RELAT_1:def 1;
 [b,a] in P by A2,A3,RELAT_1:def 7;
 then [b,a] in R by A1;
 hence thesis by A3,RELAT_1:def 7;
end;
```

```
P c= R implies P~ c= R~
proof
 assume
A1:  P c= R;
 let a,b;
 assume [a,b] in P~;
 then [b,a] in P by RELAT_1:def 7;
 then [b,a] in R by A1;
 hence thesis by RELAT_1:def 7;
end;
```

Definitional expansions in Checker

The sign of each conjunct, say α, of a formula is analyzed. If it is positive, i.e. when α is a premise, then the conjunct is expanded to $\alpha \wedge \hat{\alpha}$, and if the conjunct is negative, i.e. when $\neg\alpha$ is a premise, then it is expanded to $\alpha \vee \hat{\alpha}$, where $\hat{\alpha}$ is the definitional expansion of the α.

Expanding a formula conjunctively or disjunctively depending on its sign creates more premises accessible to the Verifier.

The satisfiability of the original formula is preserved, since α is satisfiable if and only if $\alpha \wedge \alpha$ is satisfiable, and α is satisfiable if and only if $\alpha \vee \alpha$ is satisfiable.

Pros and cons

- ◦ less explicite references
- ◦ obvious theorems
- ◦ artificial proofs removed

- × time consuming

Obvious theorem – example

```
A c= B & B c= C implies A c= C;
```

$$\frac{\begin{array}{c} A \subseteq B \ \wedge\ (\forall_x x \in A \Rightarrow x \in B) \\ \wedge \\ B \subseteq C \ \wedge\ (\forall_x x \in B \Rightarrow x \in C) \end{array}}{A \subseteq C \ \vee\ \forall_x x \in A \Rightarrow x \in C}$$

$$\frac{\begin{array}{c} A \subseteq B \ \wedge\ (\forall_x x \in A \Rightarrow x \in B) \\ \wedge \\ B \subseteq C \ \wedge\ (\forall_x x \in B \Rightarrow x \in C) \\ \wedge \\ A \not\subseteq C \ \wedge\ (\exists_x x \in A \wedge x \notin C) \end{array}}{\lightning}$$

Artificial proof – example

```
{} c= A
proof
  let x be object;    :: starts expansion of c=
  thus thesis;
end;
```

Importing expansions to Checker

- `definitions` – definitional expansions to Reasoner
- `expansions` – definitional expansions to Checker
- `equalities` – expansions `equals`

Reductions – syntax

```
registration
 let x_1 be θ_1, x_2 be θ_2, ..., x_n be θ_n;
 reduce τ_1(x_1, x_2, ..., x_n) to τ_2(x_1, x_2, ..., x_n);
 reducibility
 proof
  thus τ_1(x_1, x_2, ..., x_n) = τ_2(x_1, x_2, ..., x_n);
 end;
end;
```

Reductions – examples

example

```
registration
 let r be Real;
 reduce r*' to r;
 reducibility
 proof
  thus r*' = r;
 end;
end;
```

counterexample

```
registration
 let r be Real;
 reduce r - r to 0;
 reducibility;
end;
```

Reductions – processing

- Each reduction generates an equality
- Equalities are processed by Equalizer
- Equalizer computes congruence closure

Ellipsis, flex connectives

- Syntax
- Expansion
- Bounds
- Example
- Profits

Ellipsis – syntax

```
P[i] & ... & P[j]
P[i] or ... or P[j]
```

```
( P[m,i] & ... & P[m,j] ) & ... & ( P[n,i] & ... & P[n,j] )
```

```
not, &, & ... &, or, or ... or, implies, iff
```

Ellipsis – expansions

In the case of a flexary conjunction $\phi(a) \wedge \cdots \wedge \phi(b)$ the formula is $\forall_{i \in \mathbb{N}} : a \leq i \wedge i \leq b \Rightarrow \phi(i)$.

In the case of a flexary disjunction $\phi(a) \vee \cdots \vee \phi(b)$ the formula is $\exists_{i \in \mathbb{N}} : a \leq i \wedge i \leq b \wedge \phi(i)$.

Computation of the bounds a and b is based on terms occurring in formulas $\phi(a)$ and $\phi(b)$. The terms a and b represent the minimum difference between forms of terms of $\phi(a)$ and $\phi(b)$.

$\phi(n) \wedge \cdots \wedge \phi(n+5)$ results in $a = n$ and $b = n + 5$

$\phi(n+0) \wedge \cdots \wedge \phi(n+5)$ gives $a = 0$ and $b = 5$

Ellipsis – proof skeletons

```
P[a] & ... & P[b]
proof
  let i be natural number;
  assume a <= i & i <= b;
  thus P[i];
end;
```

```
P[a] or ... or P[b]
proof
  take i = example;
  thus a <= i & i <= b;
  thus P[i];
end;
```

Ellipsis – bounds

$$\frac{\alpha(1)\ \&\ \ldots\ \&\ \alpha(4)}{\alpha(1)\ \&\ \alpha(2)\ \&\ \alpha(3)\ \&\ \alpha(4)}$$

The actual implementation is rather restricted:

- bounds a and b must be numerals (or 0 that technically in Mizar is a nullary functor),
- the difference $b - a$ must be small. We put $b - a \leq 100$, but it is a subject of further experiments and discussions.

Ellipsis – example

```
for n being Nat st n <= 1 holds n = 0 or n = 1;

for n being Nat st n <= 2 holds n = 0 or n = 1 or n = 2;

for n being Nat st n <= 3 holds
    n = 0 or n = 1 or n = 2 or n = 3;



for m,n being Nat st n <= m holds n = 0 or ... or n = m;
```

Ellipsis – profits

```
let k;
assume k <= 2;
then k = 0 or ... or k = 2 by NAT_1:60;
then per cases;
suppose k = 0;
  thus .....;
end;
suppose k = 1;
  thus .....;
end;
suppose k = 2;
  thus .....;
end;
```

Order of adjectives

Previous

Brackets used for grouping adjectives.

```
let R be commutative associative (non empty multMagma);
```

Recent

Order plays a role, later adjectives are processed first. Brackets are not allowed.

```
let R be commutative associative non empty multMagma;
let R be commutative non empty associative multMagma;
```

HIDDEN (a part)

```
definition
  mode object;
end;

definition
  mode set -> object;
end;

definition let x,y be object;
  pred x = y;
end;

definition let x be object, X be set;
  pred x in X;
end;
```

TARSKI (a part)

```
theorem :: TARSKI:1
  for x being object holds x is set;
```

```
definition let x,y be object;
  func [x,y] -> object equals
:: TARSKI:def 5
  { { x,y } , { x } };
end;
```

Other changes

- Communication between Parser and Analyzer – XML
- Pragmas
  - `@proof` removed
    - `::$P- ::$P+` – switch off-on proofs
    - `::$V- ::$V+` – switch off-on verification
    - `::$EOF` – virtual end of file
  - `canceled` removed
    - `::$CT` – canceled theorems
    - `::$CD` – canceled definitions
    - `::$CS` – canceled schemes
  - Comments
    - `::$N` – name for a theorem
    - `::$MR` – name for a main result

References


Korniłowicz, A.: On Rewriting Rules in Mizar.
Journal of Automated Reasoning, 50(2), pp. 203–210, 2013


Korniłowicz, A.: Tentative Experiments with Ellipsis in Mizar.
In: J. Jeuring et al. (Eds.), LNAI 7362, pp. 453–457, 2012


Byliński, C., Alama, J.: New developments in parsing Mizar.
In: J. Jeuring et al. (Eds.), LNAI 7362, pp. 427–431, 2012


http://mizar.org

Thank you very much
for your attention!