Integrity of MML - Mizar Mathematical Library

Piotr Rudnicki
University of Alberta
Edmonton, Canada

Andrzej Trybulec
University of Białystok
Białystok, Poland

We can not offer a succinct definition of **integrity**. (Coherence? Cohesion?)

A similar problem: left-hand or right-hand traffic (or both)?

We had a paper on integrity at MKM '03, but many rough edges have been ironed out by now.

Main integrity concerns

- **Compatibility** required for new developments to avoid translations from one formulation to another
  - cf. three different formalization of possibly infinite lists in AFP
  - two different formalizations of finite sequences in MML (0 or 1?)
  - MML has five different formalizations of basic graph theory and I am just adding a new one
- **Uniformity** forced by searching needs
  - for theorems: ability to foresee how it was formulated.
  - for notions: ability to foresee its appearance and the definiens.

We need pragmatic rules guiding the growing body of authors.

In everyday mathematical practice we see abundance of various notations, similar notions, repetitions, etc.

The initial years of MML development (1989—2002) resulted in *creative mess*. Integrity was not an issue, Mizar authors were sought to contribute to MML.

- Mizar is a relatively rich language permitting a variety of ways in expressing the same meaning.
- Mizar authors can extend the lexicon and notations which renders parsing and searching a challenge.
- Current MML ver 4.160.1126 is quite large (but minuscule from the viewpoint of entire mathematics): 1122 articles, 236 authors (over the years), 63000 theorems and facts, 9993 definitions, registrations.
- MML is evolving, some revisions aim at restoring integrity.
- Although MML is maintained in a centralized fashion it is not clear what mechanisms could assure integrity of a growing repository of this size.
- Work toward distributed development but centered around one official MML version.

Handbook vs Archive

The *archival* part of MML contains all past contributions which are maintained verifiable as the system evolves. Very liberal acceptance policy. Multiple versions of proofs, formulations of theorems, overlapping notions are OK.

The *handbook* part of MML is the environment for creating new articles and it seems desirable for it to be

- free of repetitions
- free of overlapping notions

The handbook material is spread all over MML.

The handbook material is migrated and re-organized into *Encyclopedia of Mathematics in Mizar*, articles starting with `X`. 11 such articles about set operations and arithmetic (complex and real). A lot of migrations remain.

Some small examples

- `i < j` vs `i+1 <= j`, for integers
- `A meets B` vs `A /\ B <> {}`
- `A in bool B` vs `A c= B`
- `i in dom p` vs `1 <= i & i <= len p`, for finite sequences
- Indexing finite sequences from 0 or from 1?

`i < j` vs `i+1 <= j`

Should both versions be used or only one?

For naturals one has to use

```
theorem :: NAT_1:13
  i < j + 1 iff i <= j;
```

to move from one to the other; this suffices as `x <= y` for reals is defined with antonyms `y < x` and `x > y`, thus `NAT_1:13` can also be seen as

```
  j + 1 <= i iff j < i;
```

Somewhat surprisingly, for integers we have only

```
theorem :: INT_1:20
  i0 < i1 implies i0 + 1 <= i1;
```

and the other direction has to be proven when needed.

`A meets B` **vs** `A /\ B <> {}`

that is, using the definiens instead of the notion

```
pred X misses Y means                                  :: XBOOLE_0:def 7
 X /\ Y = {};
symmetry;
...
antonym X meets Y for X misses Y;
```

`A in bool B` **vs** `A c= B`

that is, using equivalent formulae from the definiens

```
let X be set;
 func bool X means                                     :: ZFMISC_1:def 1
  Z in it iff Z c= X;
```

All this causes substantial inconvenience when searching for needed facts, inflates the data base and hinders writing new proofs.

How to say that `i` is a valid index for finite sequence `p`?

`i in dom p` or `i in Seg len p` or `1 <= i & i <= len p`

with the definitions

```
let n be Nat;
func Seg n -> set equals                                    :: FINSEQ_1:def 1
    { k where k is Element of NAT: 1 <= k & k <= n };
...
let p;
synonym len p for card p;
...
let p;
redefine func len p -> Element of NAT means                 :: FINSEQ_1:def 3
    Seg it = dom p;
```

The following translation fact

```
theorem :: FINSEQ_3:27
 n in dom p iff 1 <= n & n <= len p;
```

is referenced 5239 times in MML (out of ~1M references).

The problem is of some size.

Indexing finite sequences from 0 or from 1?

```
let IT be Relation;
attr IT is FinSequence-like means                          :: FINSEQ_1:def 2
    ex n st dom IT = Seg n;
...
mode FinSequence is FinSequence-like Function;

let IT be set;
attr IT is T-Sequence-like means                           :: ORDINAL1:def 7
    proj1 IT is ordinal;
...
mode T-Sequence is T-Sequence-like Function;

mode XFinSequence is finite T-Sequence;                    :: from AFINSEQ_1
```

In `FinSequence p`

`i in dom p iff 1 <= i <= len p`, for natural `i`.

In `XFinSequence r`

`i in dom r iff i < len r`, for natural `i`.

Which is better? There are strong sentiments for both.

Various graphs

- ORDERS_1, 1989

```
struct(1-sorted) RelStr (#
  carrier -> set,
  InternalRel -> Relation of the carrier                    #);
```

- GRAPH_1, 1990

```
 struct(2-sorted) MultiGraphStruct (#
   carrier, carrier' -> set,
   Source, Target -> Function of the carrier', the carrier  #);
```

- SGRAPH1, 1994

```
struct (1-sorted) SimpleGraphStruct (#
  carrier -> set,
  SEdges -> Subset of TWOELEMENTSETS(the carrier)           #);
```

- ALTCA_1, 1995

```
struct(1-sorted) AltGraph (#
  carrier -> set,
  Arrows -> ManySortedSet of [: the carrier, the carrier:]  #);
```

- GLIB_000, 2005

```
mode GraphStruct is finite NAT-defined Function;

let G be GraphStruct;
 attr G is [Graph-like] means                          :: GLIB_000:def 11
VertexSelector in dom G & EdgeSelector in dom G &
SourceSelector in dom G & TargetSelector in dom G &
the_Vertices_of G is non empty set &
the_Source_of G is Function of the_Edges_of G, the_Vertices_of G &
the_Target_of G is Function of the_Edges_of G, the_Vertices_of G;
```

- 2010

```
mode SimpleGraph is 1-at_most_dimensional subset-closed (finite-membered set);
```

Possible solutions

- Some policy enforced by the library committee for changing the undesired formulations when accepting an article.

  Is it possible to automate this process?

- Revisions of the past. This is the current practice, done mainly by hand.

- Strengthening the Mizar processor such that such differences in formulations are transparent.

  Danger: blow up and substantial slow down in processing.

Lexicon

New lexical symbols are defined in *vocabularies* and then used in newly defined notations. Bad choice of symbols may cause a lot of grief.

In the past, the letter `U` was used as a symbol of binary set union. Thus `U` could not have been used as an identifier. Even experienced users frequently tripped over this. Now it is `\/`.

The symbol `c=` (vocabulary `HIDDEN`) is a single token (small `c` followed by `=`) used for the subset relationship as in `A c= B`. (The symbol can be reused.)

`a=b`	equality of `a` and `b`
`c=b`	syntax error
`c =b`	equality of `c` and `b`
`ac=b`	equality of `ac` and `b`
`a c=b`	`a` is subset of `b`

All the above depends on context, if vocabulary `HIDDEN` is not in the environment then the above would not be true. (Butt `HIDDEN` is in default).

Syntax

Not just the context-free part but also identification of objects.

A definition defines a new constructor and gives its syntax and meaning.

Format and pattern of a constructor

- format: symbol of the constructor, place and number of arguments
- pattern: format + types of the arguments

A constructor may have several patterns: synonyms and antonyms.

Notation: pattern + the identified constructor

The analyzer problem is in finding the constructor from a recognized pattern. This is complicated due to overloading and order of imports.

Insisting on a unique constructor for each pattern seems overly restrictive.

Example:

```
let f be Function;
 func "f -> Function means                                  :: FUNCT_3:def 2
  dom it = bool rng f & for Y st Y in bool rng f holds it.Y = f"Y;

let X, Y be set, f be Function of X, Y;
 func "f -> Function of bool Y, bool X means              :: MEASURE6:def 7
  for y being Subset of Y holds it.y = f"y;
```

Identical formats: `"` is the functor symbol with one postfix argument.

The patterns are different as the types of arguments differ.

Problem: `Function of X, Y` widens to `Function` and when we have

`g` of type `Function of X, Y`

then which notation we mean when writing `"g` ?

This depends on the order of imported notations. If `FUNCT_3` is followed by `MEASURE6` then `"g` will be identified as the latter.

One can use `"(g qua Function)` which forces the former. This solution still feels unsatisfactory.

Integrity of notions: different incarnations

There are three notions of an ordered pair in MML

- the Kuratowski pair, `TARSKI:def 5`

  `[x,y]` `{ {x, y}, {x} }`

- `FinSequence` of length 2, `FINSEQ_1:def 9`

  `<*x, y*>` `{ [1, x], [2, y] }`

- `finite T-Sequence` of length 2, in `AFINSQ_1:def 6`

  `<%x, y%>` `{ [0, x], [1, y] }`

Only one of them should be used. Which?

Similar problems with products and finite sequences.

A lot of variations for more complicated notions.

Repetition of similar notions

Gluing catenation: use when the last item in p equals the first item in q.

```
  let p, q be FinSequence;
  func p ^' q -> FinSequence equals                    :: GRAPH_2:def 2
   p^(2, len q)-cut q;
```

```
let p,q be FinSequence;
func p$^q -> FinSequence means                        :: REWRITE1:def 1
 it = p^q if p = {} or q = {}
   otherwise ex i being Element of NAT, r being FinSequence
              st len p = i+1 & r = p|Seg i & it = r^q;
```

Coincide when the last element in p is the same as the first element in q.

```
let p,q be FinSequence;
func p%^q -> FinSequence means          :: Freek proposed
 it = p^q if p.len p <> q.1
          otherwise p^(2, len q)-cut q;
```

Formulating theorems

Theorems frequently take the form of an equivalence

```
for x being set st C[x] holds P[x] iff Q[x]
```

but frequently a better formulation is with two separate implications.

```
for x being set st Cpq[x] & P[x] holds Q[x]
```

```
for x being set st Cqp[x] & Q[x] holds P[x]
```

The conditions for each direction are different.

This phenomenon can be automatically analyzed.

More import anomalies

We mentioned troubles with importing notations for `"f`.

Another trouble with importing definitions. We can refer explicitly to the definitional theorem behind a definition.

Directive `definitions` imports

- definientia introduced by `means` to be tacitly used in proofs based on definitional expansion.
- definientia introduced by `equals` to tacitly add equalities.

Article `SCMBSORT` runs in 2:50 with 12 explicit references to `SCMFSA6A`.

With implicit import through the `definitions` we save 185 bytes (out of 137616) but the checking time increases to 5:31 min.

Similarly, for article `SCPQSORT` and the definitions from `SCMPDS_4`, we gain 76 bytes (out of 229116) while increasing checking time from 8:34 to 20:45.

Reason: a lot of irrelevant terms are added.

Various appearances in various contexts

- Discovery context i.e. constructing the proof by a human

  Automated provers. heuristics, long checking time are OK.

  Incremental checking of Mizar text is still far into the future.

  Many enhancers and utilities are used before inclusion into MML.

- Justification context i.e. just re-checking the repository

  After any change, revision, reorganization or experiment, we have to recheck the entire MML, preferably in no time.

  The proofs from the discovery context better be converted to sequences of basic, immediate to check inference steps.

  In MML: trying to step back from a quite restricted resolution.

- Presentation context (for proofs)

  When browsing MML little control of the level of detail displayed.

  We need synthesis from the justification level up.

  The result of the the discovery context is usually too messy to be shown.

Restricted resolution adventure

Mizar uses only pattern matching. And the following is not accepted.

```
A: for x holds P[x] implies Q[x];
B: for x hlds Q[x] implies R[x];
  P[x0] implies R[x0] by A,B;
```

If we add a new 'fact', then it is accepted.

```
A: for x holds P[x] implies Q[x];
B: for x hlds Q[x] implies R[x];
C: Q[x0] or not Q[x0];
  P[x0] implies R[x0] by A,B,C;
```

For fast checking resolution causes problems in the justification context.

- unstable (non monotonic): adding a new premise can cause rejection
- unreadable to humans
- can run long on incorrect inferences (hundreds of disjunctions to add)

When MML is revised this 'resolution' steps result in long running time.

There are ca. 17000 such steps and trying to eliminate them is a headache.

Conclusions