Mizar: An Impression

Freek Wiedijk

Nijmegen University

<freek@cs.kun.nl>

アブストラクト（Abstract）

このノートは Mizar system の紹介に続いて Mizar と Coq の短い比較を提示します。 付録は Mizar の文法と Mizar article の注釈つきの完全な例（annotated example）です。

日本語訳 l'Hospitalier
2008

第 2.4a 校

オリジナル（英文）<http://www.cs.ru.nl/~freek/mizar/mizarintro.pdf>
訳（日本語）<http://std.int-univ.com/~r060002/CGI/MizarAnImpression_JP.pdf>

目　次

信州大学IT大学院の勉強のため個人的に翻訳しました。　訳者はMizarの知識が不十分なので現行のMizarとの差異について<http://mizar.org/>を参照し、読者の責任において御使用ください。　誤訳、改良等についての読者のコメントをお待ちします。　原著者は日本語を解さないので、文責はすべてl'Hospitalierにあります。　Mizar学習のための個人的使用を想定して公開します。　以下はDr. Freek Wiedijkからの翻訳許可（部分）です。

2008 年　晩秋　　　l'Hospitalier

Yes, of course you have my permission.　Although I would appreciate it if you would include a comment that I myself don't know Japanese and therefore can't judge whether the translation is accurate.

Freek　　　2008/10/24 21:42

1　やあ、こんにちは（What It Is）

Mizar はコンピュータで数学的証明を行うためのシステムで、プログラムで正当性をチェックすることができます。　1973 年頃以来、ポーランドのビアリストク（Bialystok）において Andrzej Trybulec と彼のチームにより開発が続けられてきました。　Mizar 言語は略式の（informal）数学の言葉（‘数学の特有の表現（vernacular）’）にとても良く似ています。　Mizar は古典的な一階の（述語）論理を持つ ZF 類似の（訳注：Zermelo-Fraenkel like）集合論を基礎にしています。 Mizar プロジェクトの一部は大規模な数学データベースの開発であり、これは現在 587 の article で 41M バイト（6M バイトの証明を除いて）を要しています。　現在進行中の主なプロジェクトは（Grzegorz Bancerek 指導の）実際の数学の本「連続束論概論」（‘A Compendium of Continuous Lattice’）の Mizar への翻訳です。

Mizar には以前は複数の方言がありました。　‘Mizar MSE’（baby Mizar とも呼ばれます）は実際への応用を意図していない、おもちゃの言語です。　現行の Mizar 言語の完全版は ‘PC Mizar’と呼ばれています。　これは Turbo/Borland PASCAL で記述され DOS/Windows で実行できます。　機能としてはもともと一つのプログラムである ‘**mizf**’ は非対話的に Mizar file の正当性をチェックします。

Mizarシステムには十分なドキュメンテーションがありません。　最もマニュアル[1] に近いのはMichał Muzalewski [3] による ‘An Outline of PC Mizar’でしょう。　意外にうまくいくMizar言語の探究方法はMizar文法を研究することです（文脈に依存しない文法については 26 頁のAppendix Bを見てください）。個々の構文はMizar library <http://www.mizar.org/library/>を検索してください。

[1] このマニュアルの電子版はWorld Wide WebのURL:
<http://www.cs.kun.nl/~freek/mizar/mizarmanual.ps.gz>で入手できます。
（訳注:Windowsの場合は<http://www.cs.ru.nl/~freek/mizar/mizarmanual.pdf>のPDFファイルが便利のようです。　ページ頭部が切れて読めない方は英文については<http://std.int-univ.com/~r060002/CGI/MM-manual.pdf>を、日本語訳は<http://std.int-univ.com/~r060002/CGI/MM-manual_JP.pdf>を参照してください）

２　定義(Definition)

Mizar article は Mizar library から引用する他の article を指定する 1 つの **‘environ’** ヘッダーと、それに続く定義と定理のひと続き（sequence）からなります。

　定義は一般的に：

definition
　let *arguments*;
　assume *preconditions*;
　func *pattern* -> *type*　**means** :*label* : *statement*;
　correctness proof
end;

の形を持ちます。　*pattern* というのは演算が書かれているやり方を表します（通常の関数記法（normal function notation）と演算子記法（operator notation）の両方が可能です；　そしてすべての引数が *pattern* のなかに現れる必要はありません：Mizar は暗黙の引数をサポートします)。　*定義ステートメント（defining statement）*の中では、定義されたオブジェクトは **‘it’** で参照できます。　正当性の証明（correctness proof）は、前提条件（preconditions）を与えられて定義されたオブジェクトの、存在性、唯一性を示さなければなりません。

func *pattern* **->** *type*　**means** :*label* :
　it = *expression*;

の形の定義は：

func *pattern* **->** *type*　**equals** :*label* : *expression*;

と短縮できます。

　ここに定義の例をあげます、(article **‘POWER’** から）**‘log’** の定義です。

```
reserve a,b for Real;
definition
  let a,b;
  assume A1: a>0 & a<>1 and A2 : b>0;
  func log(a,b) -> Real means
 :Def3: a to_power it = b;
```

```
    existence
  251 lines of existence proof omitted
    uniqueness by A1,Th57;
end;
```

（この **‘reserve’**ステートメントは（複数の）変数をある（１つの）型に予約します：予約された変数の型を（訳注：使用時に再度）与える必要はありません。そこで最初の一行が **‘reserve’**ステートメントなので **‘let a,b’** は **‘let a,b be Real’**を意味します。

ここに引き算の定義があります（**‘REAL_1’**から）：

```
reserve x,y for Real;
definition
  let x,y;
  func x-y -> Real equals :Def 3: x+(-y);
  correctness;
end;
```

（この最後の **‘correctness’** は証明されるべく残されている正当性証明の全ての要素の短縮形です： そして **‘means’** を使った定義については、これらは **‘existence’** と **‘uniqueness’** となります；そして **‘equals’** を使った定義に対してはそれの **‘coherence’** で、それらはオブジェクトが正しい型を持っていると言っています。 この例では正当性（correctness）は明白ですので **‘by’** による justification の必要性はありません。）

関数のための **‘func’** の定義を別にして、Mizar は述語のための **‘pred’** 定義、型のための **‘mode’** 定義と **‘attr’** 定義を持ちます（Mizar の型については後のほうの「5 型（Types）」を見てください）。 これらはかなり構文的に **‘func’** の定義に類似しており、詳細な説明はされません。

関数と述語の定義においては、同意語（synonym）と反意語（antonym）をあげることができ、可換性（commutativity）や対称性（symmetry）、反射性（reflexivity）、非反射性（irreflexivity）のような特性（properties）を指摘できます。 Mizar system は魔法のようにこれらのこと（things）を‘知る’でしょう。 そして同じ表現上での変動は、まるでそれらが単なる構文上の変種（variants）であるかのように振舞うのです。 例えば **≤** の定義は（**‘ARYTM’** の中の）：

```
definition
  let x,y be Element of REAL;
```

```
  pred x ≤ y means
  6 lines of definition
  89 lines of correctness proof
  synonym y ≥ x;
  antonym y < x;
  antonym x > y;
 end;
```

なので、**x<y** と書いても **y>x** と書いてもかまいません。　さらに、それは **<** についての定理がしばしば **≥** についてのステートメントを証明するのに使用できるのを意味します（多分、含意の対偶に関して）。

3　定理（Theorems）

Mizar article の大部分は（ちょうどコンピュータプログラムのソースの大部分が procedures / functions からできているように）定理（theorems）からできています。　定理は：

```
theorem label: statement
  proof
    proof steps
  end;
```

という形をしています。　定理の例（ '**IRRAT_1**'から）は：

```
theorem T2:
  ex x, y st x is irrational & y is irrational &
    x.^.y is rational
proof
  set w = √2;
  H1: w is irrational by INT_2:44,T1;
  w>0 by AXIOMS:22,SQUARE_1:84;
  then (w.^.w).^.w = w.^.(w•w) by POWER:38
    .= w.^.(w^2) by SQUARE_1:58
    .= w.^.2 by SQUARE_1:88
    .= w^2 by POWER:53
    .= 2 by SQUARE_1:88
  then H2: (w.^.w).^.w is rational by RAT_1:8;
```

```
   per cases;
   suppose H3: w.^.w is rational;
      take w, w;
      thus thesis by H1,H3;
     suppose H4: w.^.w is irrational;
      take w.^.w, w;
      thus thesis by H1,H2,H4;
 end;
```

となります。 （Mizarは‘√’や‘●’や‘²’のような‘高位 ASCII キャラクタ’を使用するのに注意してください、それらは DOS のキャラクタ・セットのなかで251, 249そして253の ASCII コードを持ちます）。 この例のいろいろな要素については以下で議論しましょう。

Mizar のステートメントは一階の述語論理の言語ステートメントです（キーワード **‘contradiction’**、**‘not’**、**‘&’**、**‘or’**、**‘implies’**、**‘iff’**、**‘for … holds’**、 そして **‘ex … st’** を使います ）。 この言語の原子式は述語のインスタンスか（しばしば **‘=’** を持つインフィックス演算子として書かれ、最も重要なものの一つです）、あるいは表現は型、あるいは形容詞である（**‘is’**）と述べているステートメントのどちらかです。

これからのちょっとした逸脱は、全称量化子と含意の組み合わせ（universal implication）があるだけで、それは通常：

for *variable* **holds** (*statement* **implies** *statement*)

のように書かれることはなく:

for *variable* **st** *statement* **holds** *statement*

と書かれます。

混乱の原因はおそらく Mizar が **‘&’** に対して **‘and’** を、 **‘st’** に対して **‘such that’** の両方を持つことでしょう。 最初の変種（variants）は一階の式の文法（syntax）で、後の変種は Mizar ステートメントの一部であるキーワードです。

他の‘重複’(‘duplication’)は Mizar では **‘func’** 機能を定義しており(Mizarではそれらの定義された演算（operations）を ‘functors’ と呼びます）、これに対して集合論の基礎をなす（underlying）function（クラトウスキ対（Kuratowski pairs）の集合）を持っていることです。 functionの一番目の種類の応用は **f(x)** と書かれ、これに対して２番目の種類は **f.x**と（ドットオペレータのインフィックス記述で）書かれます。

３番目のその手の‘重複’は **‘x∈X’** と **‘x is Element of X’** の関係にあります。　最初のほうは集合論からの原始的な２項述語で、後のほうは（**‘x’** は **‘Element of X’** という型を持つ）と言う、型付けステートメントです。　各々の集合で **‘Element of’** はその要素の型を与えます。　例えば：**‘REAL’** は集合です、しかし **‘Real’** は、**‘Element of REAL’** と定義されていて、これは型です。　そこで、**‘x ∈ REAL’** であって、**‘x is Real’** あるいは **‘x is Element of REAL’** ではありません。

Mizar の証明は proof ステップのリストから成ります。　そういう proof ステップは基本的に：

label **:** *statement* **by** *labels*;

の形をもちます。　例えば、Mizar の proof ステップの例は：

```
H1: w is irrational by INT_2:44, T1;
```

となります。　ここでステートメントは **‘w is irrational’** でそれは **‘H1’** とラベルされており、それは **‘INT_2:44’** と **‘T1’** とラベルされたステートメントの帰結であって、その内容は：

```
:: INT_2:44
  2 is prime
```

（article **‘INT_2’** の44番目の定理）そして：

```
T1: p is prime implies sqrt p is irrational
```

です（これは同じファイルの前の方で証明されています）。

リストの **‘by’** キーワードの後にあるラベルは先行する定理からのラベルか、あるいは同じファイルの中の proof ステップのラベルか、あるいは別の Mizar article の定理の参照のうちのどれかです。　後のケースではそれは：

article name **:** *sequence number*

という風になります。　（そう、‘ローカル’参照は名前によって（by name）、‘外部’参照（outside references）は番号によって（by number）ということです）。　ライブラリの‘アブストラクト’・ファイルをブラウズすることで連続番号（sequence number）を見つけ出すことができます、アブストラクト・ファイルとは自動的に proof を取り除かれ、連続番号を挿入された article のことです。

直前のステップのラベルを参照する代わりに、証明のステップの前に **‘then’**

というキーワードをつけることもできます。　そうすると：

A: *statement* **by** *labels*;
B: *next statement* **by A,** *more labels*;

は：

statement **by** *labels*;
then B: *next statement* **by** *more labels*;

と書くことができます。

この種類の（'diffuse' reasoning stepsと呼ばれる）ステップを別にして、証明されるステートメントに関するステップ（'skeleton' steps）があります。　あらゆる時に証明されるべき'カレント'（'current'）ステートメントが存在します:　スケルトン・ステップはこのカレント・ステートメントを変形（transform）します。　例として証明すべきステートメントが以下のような形だとしましょう：

A implies B & C

さて続くスケルトンは'**assume**'と'**thus**'のスケルトン・ステップを使って、それを証明します：

assume *label*: **A;**
diffuse steps
thus B by *labels*;
more diffuse steps
thus C by *labels*;

'**assume**'ステップの後で、証明されるべきステートメント（訳注：カレント・ステートメント）は **B & C** に縮小されます。　そして最初の'**thus**'の後で、それは **C** になってしまいます。

'**then thus**'の組み合わせは構文的に（syntactically）正しくありません：それは'**hence**'と書かれます。　'**hence**'はそのステートメントが前のステップからの続きであり（'**then**'）、証明されるべきものの一部である（'**thus**'）ことを意味します。

そこでMizarの証明の各瞬間に証明されるべく残されているステートメントが存在することになります。　このステートメントはキーワード'**thesis**'で参照されます。　しばしば証明（プルーフ）や副証明（サブプルーフ）は'**hence thesis**'をもって終了します。(Mizarライブラリはこの構文を36885回含みま

す。）

　‘assume’と**‘thus’**ステップは、自然の演繹（deduction）システムのなかでは、含意（implication）の導入と論理積（conjunction, 合接）の導入に相当します。　その他のスケルトン・ステップ には：

- **‘let’**は全称命題の導入（universal introduction）：

  let *variable* **be** *type*;

- **‘consider’**は存在文の消去（existential elimination）：

  consider *variable* **being** *type* **such that** *properties*
  　by *labels*;

  （ラベルで参照されるステートメントは適切な存在ステートメントを justifyしないといけません。）

- **‘take’**は存在命題の導入：

  take *expression*;

- **‘per cases’**は論理和（disjunction、離接）の消去：

  per cases by *labels*;
  suppose *label: statement*;
  　proof for the first case
  suppose *label: another statement*;
  　proof for the second case
  more cases

などがあります。

すべての自然の演繹規則がそれに相当する（counterpart）Mizar のスケルトン・ステップを持っているわけではありません：いくつかは diffuse step として扱われます。

Mizar ではプルーフはサブプルーフを含むことができます。　プルーフ・ステップの一部で：

　. . . **by** *labels*;

という形はその justification と呼ばれます。　それは完全な一階述語論理の証明能力（可能性）（full first order provability）（訳注：を持つの）ではなく、いくらか弱い変種で、迅速な決定ができます（概略で：仮定（premise）の全称量

化子（universal quantor、訳注：quantor＝quantifierの使用例あり）のうちの一つだけが実例化（instantiate）されるようです； 他方で、それは型の情報からの推論、等式の適用、そして存在のステートメントの演繹などをとても上手にこなします）。 この **'by'** justification は時々十分ではありません： そこでステートメントはまた：

label : *statement* _
proof
proof steps
end;

の形で full proof によって justify されることもあります。 *ステートメント*の後にセミコロンがないことに注意してください（訳注：上記アンダーライン'_'の場所）（セミコロンは'空の' justification の意味があります）。 この場合のように full proof が与えられる場合には、**'now . . . end'** 構文によりステートメントを省略 (omitted) することができます（訳注：下記アンダーライン'_'の場所）：

label : ________
now
proof steps
end;

その場合、（ラベルが参照する）証明されるステートメントは証明から'計算'されます。

Mizar はまた等式型の推論（equational reasoning）をサポートします。

label : *expression* = *expression* **by** *labels*
.= *expression* **by** *labels*
more steps
.= *expression* **by** *labels*;

と書くことができます。 これらの等式の推移律（transitivity）は自動的にハンドリングされます：*ラベル*は最初と最後の等式の表現を参照します。

そして Mizar は高階のステートメント（higher order statement）をサポートします。 そのような **'スキーム（scheme）'**を呼び出す（invoke）ときは、スキームの名前は **'by'** というキーワードではなく **'from'** というキーワードの後に書き、それは引数（複数）をとります。 これらの引数はスキームの定義のなかのカッコに囲まれた'高階のパラメータ'では*なく*（それらは自動的に決定されます）、**'provided'** のあとの'条件（conditions）'です。

自然数の数学的帰納法をおこなうための **scheme** は（article '**NAT_1**' から）：

```
scheme Ind { P[Nat] }:
  for k holds P[k]
provided
   A1: P[0] and
   A2: for k st P[k] holds P[k+1]
```

16 lines of proof omitted

で定義されます。（角カッコは **P** が述語であることを意味します：関数は丸カッコを用いて書かれます；その種の parameter function がたとえ単一の定数であっても、それらのカッコは書かれなくてはなりません）。　その '**Ind**' スキームは：

label: *statement* **from Ind**(*label*, *label*);

のように応用され、その中で２つのラベルはスキームが必要とする２つの '**provided**' ステートメントのインスタンスを参照します（それらは、もちろん帰納の基点ステップと帰納ステップの場合です）。

４　構文（Syntax）

Mizar テキストの表現はかなり数学的に見えます。　これには２つの原因があります：Mizar ライブラリは DOS のキャラクタ・セットの高位 ASCII 部分を使用します（そこには多数の '数学' のシンボルがあります）。　そして Mizar - 表現では演算子の多様なスタイルが許されています（プリフィックス、ポストフィックス、インフィックス、そして 'カッコ様（bracket-like）'）。

Mizar の語彙の構文（lexical syntax）は固定されていません。　実際、Mizar article は一般的には 2 つのファイルからなります:（'**.miz**' で終わる名前の）article file と（'**.voc**' で終わる名前を持ち、システムがそれを見つけられるように '**dict**' と呼ばれるサブディレクトリの中に置かなければならない）vocabulary file です。　最後の種類であるボキャブラリ・ファイルは語彙の要素（lexical element）を提供します。

それは定義からの '識別子'（'identifiers'）と同様にオペレータ・シンボルの両方を持ちます（contains）。　そのボキャブラリ・ファイルは各語彙の要素について、そのシンボルの構文上のカテゴリーを与える大文字で始まる行（関数、述語、モード、などのシンボルです。　これらはすべて*異なる*構文上のカテゴリーに属します）、そしてシンボル自身、場合によっては（possibly）優先

度も保持しています。
　というわけで、Mizar ライブラリは、実に多くの article と多くのボキャブラリが一緒になって成立しているのです。　article はソース・ファイルのフォーマット（訳注：ASCII テキストファイル）でブラウズできますが、ボキャブラリは‘コンパイルされた’形でだけ存在します。　**‘findvoc -w’** プログラムは指定したシンボルがどのボキャブラリ由来かを見つけるために使用されます。
例えばコマンド：

```
findvoc –w .
```

は

```
vocabulary: FUNC
O. 100
```

を出力し、それは私達に‘.’はボキャブラリ‘**FUNC**’由来の演算子シンボルで（構文レベルではインフィックス、プリフィックス、ポストフィックスのオペレータの区別はないので、３つの形式はすべて使用可能です）、優先度（priority）100 を持っていると教えてくれます（もし‘.’演算子が多重定義されているなら、それらは全てその優先度を持ちます）。
　表現：

```
f.x
```

の解析として３つの面（aspect）を識別できます：語彙的解析（lexical analysis）（３つのトークン（表象）：‘**f**’、‘.’、‘**x**’があります）、表現が構文解析されるはずのやり方（以下の文：

```
reserve x for set;
reserve f for Function;
definition
  let f, x;
  func f.x -> set means
:: FUNC_1:def 4
  [x, it] ∈ f if x ∈ dom f otherwise it = 0;
end;
```

からのパターンはここに応用できるものです）、そしてそれが参照する‘概念’（notion）（‘.’演算子の‘意味’）の３つです。　これらの３つの面は厳密にその article の **‘environ’** ヘッダーのなかの **‘vocabulary’**、**‘notation’**、

そして **‘constructors’** 指令に対応しています。 そこでこの **‘f.x’** という表現を正確に処理するためには指令：

```
vocabulary FUNC;
notation FUNCT_1;
constructors FUNCT_1;
```

が存在しないといけません。 それが属する article とボキャブラリの名前がしばしば同一であるにもかかわらず、名前の一致は必須でないことに注意してください（ここではそれらは **FUNC** に対して **FUNCT_1** であるように）：article の名前とボキャブラリの名前の‘名前空間’は分離しているのです。

article の正しい **‘environ’** ヘッダーをつくるのは全く困難なことですが、これらの３つの指令を理解するのも極めて困難なのです。 **‘theorems’**、**‘schemes’**、**‘clusters’** 指令は直截です： article からの定理、スキーム、クラスタを使えるようにするには、その article の名前が適切な指令の中に存在しないといけません。

‘definitions’ 指令は定義拡張について*のみの*指令です（それは型理論のなかでは‘デルタ・リダクション’（delta reduction）と呼ばれます)。 この指令は Mizar システムが、リストアップされた article からのすべての定義を‘展開（unfold)’するのを許されていると言っています。 定義から派生した（stem from）定理（番号の前に **‘def’** と記されている）は‘*theorems*’です、そして該当する指令は‘**theorems**’指令です。 **‘definitions’** 指令はめったに使われません。

現在のところ **‘requirements’** 指令の可能な事例（possible instance）は：

```
requirements ARYTM;
```

の一つだけです。 それは Mizar システムが自然数と自然数の間に成り立つ恒等式（identities）と不等式（inequality）を‘知る’であろうという意味です。

例えば不等式

```
1<>0;
```

は **ARYTM** requirement があれば、いかなる justification も必要としません。

Mizar 演算子は２つ以上の引数をとることができます、しかしその場合カッコがその両側にあり（引数の）間にはカンマがないといけません。 そこで合法的なポストフィックス演算子は:

```
 x f
(x) f
```

```
(x,y) f
 ...
```

となり、例えば合法的なインフィックス演算子は:

```
(x,y) f (z,v,w)
```

となります。

‘通常の（ordinary）’関数記法の応用はこのパラダイムに適合するのに注意してください。

関数の識別子と演算子のシンボルはボキャブラリ・ファイルの中で型 ‘**O**’ を持ちます。 Mizar ではさらにカッコ様 (bracket-like) 記法も許されています：そのためにボキャブラリ型 ‘**K**’ と ‘**L**’ が存在します。 ‘**<***’ は **K** 型を ‘***>**’ は **L** 型を持つので（両方ともボキャブラリ ‘**FINSEQ**’ から取りました)、表現：

```
<* x *>
```

は合法的なパターンです（それは一つの元を持つ有限数列を示します)。

５　型（Types）

Mizar の意味論（semantics）は型なしの集合論なのですが（Mizar の公理は ZF 公理＋恣意的巨大到達不可能基数の存在（existence of arbitrarily large unreachable cardinals）についてのかなり強い公理 －選択公理を含意する－です)、言語自体は型つき（typed）です。 しかし型は言語の表現の特性（property）であって、表現が参照するオブジェクト（集合）の特性ではありません、ですから言語は‘型理論’に基礎を置いていません。 型は表現の曖昧さをなくす (disambiguate) ためと推論 (reasoning) のために使われます (演算子はオーバーロードすることができ、引数の型で決定されます)。

Mizar の型づけ（typing）は：

variable **be** *type*

あるいは：

variable **being** *type*

と書かれます。 （多分 ‘**let**’ のなかでは最初の形を使用し、‘**for**’ のなかでは２番目の形を使用するでしょう：しかしこの２つの変種はどこで使ってもよいと思います)

Mizar の型は‘モード（mode）’と‘属性（attributes）’から組み立てられ

ます。 Mizar の型はモードのインスタンス（事例）で（それはパラメータ化された型（parameterised type））です)、場合により（possibly）多数の形容詞（adjectives）が前につきます（その形容詞は属性あるいは属性の否定です)。
例えば、型：

non empty Subset of NAT

のなかで **‘Subset’** というモードは表現 **‘NAT’**（それは自然数の集合を示します）にかかり（is applied to） **‘Subset of NAT’** という型を与えます。 この型に対して形容詞 **‘non empty’** が与えられ、それは属性 **‘empty’** の否定です。
どの Mizar の型にも先祖型（ancestor type）があります。 これはその原型（根の型）が **‘Any’**（あるいはその同義語 **‘set’**）である木型の階層構造があるということです。 この木型構造のあらゆるノードに形容詞により与えられる‘ブール代数類似の構造’が存在します。
Mizar システムはこの木型構造に従って型を‘広げる’方法を知ります。 Mizar ではある表現に明白な型を与えることが可能です。 この目的で **‘reconsider’** 構文が存在しますが、それは **‘set’** ステートメントの変異型（variant）です。 Mizar で表現に局所的な名前をつける方法は：

set *variable* = *expression*;

と書きます。 この後では、どの場所においても、その*変数*（*variable*）は*表現*（*expression*）がそれに置き換えられてしまったかのように振舞います。
‘reconsider’ はそれに似ていますが、*表現*が別の型を獲得するだけであるという点が異なります。 それは：

reconsider *variable* = *expression* **as** *type*
by *labels*;

（‘consider’ と ‘reconsider’ ステートメントは関係がないことに注意してください：最初のものはある特性（properties）を持つ新しいオブジェクトを見つけるために存在のステートメントを使います、二番目のほうは表現により与えられ、すでに知られているオブジェクトをある型に **‘casts’** します。
Mizar はかなり良く発達した型機構を持っています。 特に Mizar は特別な型情報（extra type information）の自動的な推論（deduce）を可能にする、いわゆる‘クラスタ（clusters）’を持ちます：クラスタは Mizar システムによる形容詞の集合の操作を可能にします。 ３種類のクラスタ定義があります：最初の種類のものはある形容詞の組み合わせが non-empty 型を与えると述べています（これはシステムに知らされている必要があります、なぜなら Mizar

のすべての型は non-empty であることが立証可能でなければなりません（have to be provably non-empty））、２番目の種類のものはある形容詞の組み合わせは他の形容詞を含意すると述べ、３番目の種類のものはある形の表現はある形容詞を持つことを述べています。

最初の種類のクラスタの例は（article ‘**HIDDEN**’ から）：

```
definition
  cluster non empty set;
end;
```

です（証明はありません、なぜなら ‘**HIDDEN**’ と ‘**TARSKI**’ は、Mizar の公理系（axiomatics）を提供する２つの‘特別な（special）’ article ですから）。 それは ‘**non empty set**’ 型が存在する（inhabited）と述べています。 もしこのクラスタ型が知られていないと：

```
let x be non empty set;
```

のようなものは決して許されないでしょう、なぜならその場合システムは ‘**non empty set**’ が正しく non-empty type であることを決して知ることができないからです。

２番目の種類のクラスターステートメントの例は（‘**BINTREE1**’ から）：

```
definition
  cluster binary -> finite-order Tree;
28 lines of proof omitted
end;
```

です。

これは ‘**binary Tree**’ 型を持つすべての表現が同様に形容詞 ‘**finite-order**’ も持つと言っています（実際それはより情報の多い（informative）‘**binary finite-order Tree**’ 型を持ちます）。

３番目の種類のクラスタは（ ‘**ABIAN**’ から）

```
definition
   let i be even Integer;
   cluster i+1 -> odd;
5 lines of proof omitted
end;
```

です。

最後の種類のクラスタは非常にパワフルです、なぜならそれによって表現の多くの特性（properties）を自動的に導出できるからです、しかし時々それはシステムを著明にスローダウンします。

Mizar は‘**redefine**’ステートメントを持っています、それは一つのオペレーションは複数の定義を持つことができ、そのなかで最初の一つを除くすべてにキーワード‘**redefine**’が付加されます。 これらの再定義は中途半端に言い表されている（underspecified）のかも知れません（すべての他の情報はオリジナルの定義からコピーされています)、それらは引数の比較的小さな集合上に定義されているのでしょうが、しかしそれらはオリジナル定義と‘互換性がある（compatible)’ことが証明されねばなりません。

これは同じオペレーションを複数の型に与えることにむいています（can be used to)。 例えば（‘**NAT_1**’から）:

```
reserve k,n for Nat;
definition
  let n,k;
  redefine func n+k -> Nat;
18 lines of proof omitted
end;
```

があるので、加算という演算は本来実数について定義されたにもかかわらず（その定義では **Real** 型を持ちます）、Mizar システムは２つの自然数の和が自然数であることを知っています（なぜなら統語構文（syntactically）上の理由で２つの引数が自然数の場合は‘再定義された’バージョンの演算が選択されるからです：このためには‘**notation**’指令の中の article の順序が正しい必要があります）。 しかし‘**redefine**’のため、再定義は型 **Nat** を持つにもかかわらず、オリジナルの型 **Real** の演算についての現行のすべての定理はそれに（訳注: 自然数に）同様に適用できるのです。

6　意味論（Semantics）

Mizar の意味論はかなり直截（fairly straightforward）です。 Mizar はだいたいのところ一階記号論理を持つ ZF スタイルの集合論といったところです。

しかしながら、未定義表現の巧妙さ（subtlety）があります。 Mizar の意味論は未定義の表現の問題を、ある予期しない特性（unexpected properties）というやり方で ‘解決’します。 第１に、型は‘シンタックス・シュガー（syntactic sugar)’でコーディングされた述語というだけでは***ない***ことを意

味します（そうではなく、それらは意味論レベルの何かを‘意味’します。）。そして第２に選択公理は Mizar が一階述語論理を実装した方法で証明可能であるということを意味します。

Mizar の **‘func’** 演算は前提条件（precondition）を持つことができます。 演算が明確に定義されている証明では、これらの前提条件を使います。 しかし、そのような演算を*適用（applying）*するときは、前提条件の成立を証明する必要はありません：それらは間違っているかも知れません。 そこで、例えば除算の前提条件は分母がゼロでないことですが、**1/0** のような表現を書くことが許されてしまいます。

Mizar の意味論は前提条件が成立しない場合の演算を適用した場合の結果については何も語りません。 もし前提条件が真ならば、定義からの definition ステートメントの成立は保証されます；しかし前提条件が偽の場合、あなたはなにも知ることができません。 そこで **1/0** は未知のオブジェクト（unknown object）ということになります。 これを別の方法で見ると、Mizar の表現をそれの意味にマッピングする関数はユニークではないということです：‘未定義’の表現に関してはどんな値をとることもできます。

この規則の一つの例外は未定義表現の場合でも*型*は依然として適用されなければならないということです。 そこで、**1/0** が何であるか知らないにもかかわらず、それが実数でなければならないことを*知る（do know）*のです（除算の演算は **‘Real’** 型を持つので）。 そこで定義するステートメント（defining statement）（定義されるオブジェクトに関する述語です）は前提条件が成立しないときは該当しないのですが（not relevant）、型については該当するのです。

この意味論の取り扱いにおける単項（unary）述語とそれらの述語のコーディングの型の差異は次の例でも明らかです。 **‘something’** をある（関連性のない）ステートメントとしましょう。 それから：

```
reserve X for set;

definition
  let X;
  assume A: contradiction;
  func choice(X) -> Element of X means
:Def_choice: something;
  correctness by A;
end;

definition
let X;
```

```
   assume A: contradiction;
   func choice1(X) means
 :Def_choice1: it is Element of X & something;
   correctness by A;
 end;
```

と書きます。（前提条件はもちろん **'contradiction'** なので、これらの演算は前提条件が成立するときはいつでも明確に定義されています。） **'choice'** と**'choice1'** 関数はお互いのマイナー・バリエーションのように見えます。 しかし、意味論が未定義表現を取り扱うやりかたにより：

```
   theorem AC:
     choice(X) is Element of X;
```

は証明できます。 しかし:

```
   theorem AC1:
     choice1(X) is Element of X
   proof
     thus thesis by Def_choice1;
     ::>            *4
   end;
```

は証明が *できません*。（***4** の行は **'mizf'** チェッカーがファイルに挿入したエラー・メッセージで、ナンバー４は **'by'** による推論が成立しないことを意味します。 それは Mizar チェッカーで **'This inference is not accepted'**（「この推論は受け入れられません」と説明されます。））

定理 **'AC'** は'同型（uniform）'の選択公理（axiom of choice）を与えることに注意してください(選択公理はまた Mizar の集合理論公理(set-theoretical axioms）からも得られます、しかしそれはすでに Mizar が一階の述語論理を取り扱う方法の中に'ハードワイヤード'で組み込まれているのは明らかです。)

X が空集合のときの型 **'Element of X'** の意味は、ある種の未知である non-empty クラスになります（すべての Mizar 型は non-empty です）。Mizar ではステートメント：

```
     ex x st x is Element of Ø
```

（ある **'Element of Ø'** が存在すると述べています）は証明可能です（**'consider x being Element of Ø; take x;'**）。 しかし一見これと矛盾したステートメント：

```
not ex x st x ∈ ∅
```

も同様に証明可能です、なぜならそれは ZF の定理だからです（これは Mizar の意味論が健全でないことを意味するのではなく、**'Element of'** がちょっと（just）奇妙な解釈を持つということです）。　人はその **'Element of'** のモード（**'HIDDEN'** にあります）の定義が前提条件 **'X is non empty'** を持っているのを期待するでしょう、しかしこの場合はあてはまりません。

数学的な構造体を'構築'（'build'）するために、Mizar は **'struct'**（構造体）型を持っています。　それらは：

struct(*ancestor struct*) *struct name* **(#**
field name -> *type*,
field name -> *type*,
more fields
field name -> *type*
#)

のように定義されます。（**'(#'** と **'#)'**、これは高位の ASCII 文字で '≪' と '≫' に見える文字に置き換えられることもあります：実際、こちらの記法がより一般的です。）

そのような **struct** 型のオブジェクトは：

struct name **(#** *value, value, ... value* **#)**

のように記述され、そしてフィールドは **struct** から：

the *field name* **of** *struct expression*

により選択されます。

構造体（**struct**）の一例として、Mizar に位相空間を導入するやり方がここにあります（article **'STRUCT_0'** と **'PRE_TOPC'** から）

```
struct 1-sorted (#
  carrier -> set
#);

struct(1-sorted) TopStruct (#
  carrier -> set,
  topology -> Subset-Family of the carrier
#);
```

```
definition
  let IT be TopStruct;
  attr IT is TopSpace-like means
    the carrier of IT ∈ the topology of IT &
    (for a being Subset-Family of the carrier of IT
    st a c= the topology of IT
    holds union a ∈ the topology of IT) &
   (for a,b being Subset of the carrier of IT st
    a ∈ the topology of IT & b ∈ the topology of IT
    holds a ∩ b ∈ the topology of IT);
  end;

  definition
    mode TopSpace is TopSpace-like TopStruct;
  end;
```

背景にある集合論という観点からは構造体***特有***の意味論というものは実際あまり面白いものではありません：ある種の ad hoc（その場限りの）なコーディングでその仕事はできるでしょう（フィールドの名前としてある集合を選択し、構造体（**struct**）をその集合からの部分的関数と解釈すれば良いでしょう）。もし定義で要求されたフィールド数*以上*のフィールドを持つなら、**struct** 型を拡張する方法を利用するために、オブジェクトもまた **struct** 型にします。形容詞 '**strict**' はそういう余分のフィールドが存在しないことを意味します。

7　Coqとの比較（Comparison To Coq）

（訳注：Coqはフランス語でオンドリ（rooster），研究開発言語に動物の名前をつけるフランスの慣例で名付けられた。）

Mizar システムは Coq のような LCF（訳注：Logic for Computable Functions）の伝統から生まれたシステムとは全く類似性がありません。　ここでは Mizar を特に Coq と比較しようと思います、しかし同様の比較は他の HOL、Nuprl, Isabelle, その他の LCF 類似システムとの間でも成立します。

Mizar と Coq の間の最も印象的な違いは Mizar が一時にファイルの全体をチェックするバッチ・チェッカーであることです（ '**@proof**' キーワードは使用者に特定の proof のチェックの抑制を許すことで、この面でのある程度のコントロールを提供します）、一方 Coq はインタラクティブ・システムです（これは概念的にはコンパイル型とインタープリタ型のプログラム言語の違いに似ています）。　この違いが現れる点のひとつは公式化（formalization）の読み

やすさです： Coq ファイルは単なる‘コマンド’の長いリストで直線的な構造を持ち、コマンドは読みやすいようにデザインされていません、一方 Mizar テキストはより多くの構造を持ちファイルを‘実行’することなく（訳注：汎用エディタで）完全にアクセス可能です。

もう一つの違いは Mizar が ‘proof object’ を持たないことです：システムはその正当性（correctness）を、いくつかの原始的要素（primitives）についてだけを知っている、小さな‘核（kernel）’の正当性への縮小（reduce）を行いません。　さらに２つのシステムの間には論理（logic）の種類の違いがあります： Coq カーネルは自然に構文論理（constructive logic）を持つことになる‘型理論’（‘type theory’）を基礎としています。　一方 Mizar は非常に古典的なシステムです。

Mizar プロジェクトは Coq よりも***多数の***自動操作（more automation）を備えていて、Coq には***少し***しかありません。　Mizar でもっとも使われるステップは **‘by’** による推論（inference）で； Coq の共通の戦術はもっと初歩的（elementary）です。　その **‘by’** による推論は型を論理的に取り扱う（reason with）ことを知っています、それは恒等式を使うことができ、多数のステートメントからの情報を組み合わせられます（最大：**‘GENEALG1’** の22ステートメント）。　一方それは完全な一階述語論理による推論ではなく、そのパワーは人間がやる推論のステップに近い種類のものです（Mizar システムは余計なステップを除去するための **‘relinfer’** プログラムを持ち、それを走らせると **‘by’** は予想よりさらにパワフルな傾向があるのがわかります）。　大多数のステップで Mizar は Coq に比べよりパワフルです。 他方 Coq システムは‘閉じて’いません、それはおもいのままに（arbitrarily）強力な‘戦術’を使って拡張することができます。　これらの‘**Omega**’や **‘Ring’** のような戦術は、関連する定義域に特異的（involved domain specific）なタスクを解決することができ、Mizar はこれに相当するもの（counterpart）を持ちません。　Coq システムはまたアルゴリズムを‘反省（‘reflect’ on）’することができます：それは定理が正しいと証明するために、アルゴリズムをシステムの***中***に記述することができ、そして実行しシステムを内側から（from within）拡張することが可能です。　これが、ある意味で Coq が Mizar よりもパワフルだと言う理由です。

Mizar と Coq の他の差異は Coq が証明されるべきステートメントをおもに後方から推論し、一方 Mizar は前方から推論します（reasons forward）。基本的には Mizar の‘スケルトン’ステップは Coq の戦術（tactics）に相当し、証明されるべきステートメントを縮小（reduce）させます、一方 ‘diffuse’ ステップはすでに知られているステートメントから順に（forward）推論（reason）

します。　Mizar の証明のステップの大部分は diffuse ステップです。

言語やシステムの違いほど大きくはない差異はプロジェクトの違いです、それは Mizar プロジェクトでは巨大な組織化されたライブラリの開発に高い優先度を置いています。　Mizar システムは意図的に数個以上のファイルからなるプロジェクトを作るのを困難にしています（可能ですが、システムは遅くなります）、そして Mizar ユーザーをして彼らの仕事を Mizar ライブラリに統合するような投稿をするように激励（stimulate）しています。　そして Mizar ライブラリは Mizar グループ（‘ライブラリ委員会’）により持続的に再構成され、変更され続けていて、ライブラリを全体として構造的で一貫したものに変貌させ続けています。　これはもし彼ら（訳注：ライブラリ委員会）がファイルを‘所有’していなければ不可能でしょう。

ここにある時 Mizar ライブラリに適用された変化の一例があります。Mizar ライブラリでは実数は有理数からデデキントの切断として構成されます。しかし集合 **‘REAL’** の構成で、それは有理数のコピーを‘切り出し’て‘オリジナル’の有理数の‘中に貼りつけ（glue in）’ます（同様に整数は有理数の中に貼りつけられ、自然数は整数の中に貼りつけます）。　その方法で自然数は実数の部分集合であるだけでなく自然数は*‘自然な（natural）’*自然数となります、すなわち（i.e.）、有限順序数（finite ordinals）です。　そこで実数 0 は自然数の 0 で空集合なのです。　この数の集合のカットアンドペースト（cutting-and-pasting）は‘その場限りのやっつけ（hack）’ですが、とてもうまくいきます。　まだ複素数はこのやり方の実数を含んでいません、しかしこのような ‘複素数の中に実数を仕込む’ 変更はすでに計画されています。

そういうわけで、Mizar ライブラリの開発は Coq ライブラリに比べて明らかに著しく良好です。　例えば、それはすでに証明された多数の属性を持つ実数の完全な構文含んでいます（Coq ライブラリは公理としての実数を含んでいるに過ぎません）。　（他方、Mizar ライブラリは現在ではかなり初歩的である多項式の概念(notion)を含んでいません、まだあまりリッチではないのです)。さらに Mizar は Coq と較べてより以上に数学的です。　Mizar は主に抽象数学についての正当性証明であるのに対し、Coq は関数型プログラムの正当性を証明するといったコンピュータ科学のトピックに多くのフォーカス持ちます。

別のもう一つの差異は： Coq は Mizar よりも‘主流の科学（mainstream science）’です。　Mizar はあまり多くはない人しか知らない懐かしい（an old-fashioned）システムです（もし個々の Mizar ユーザーが Mizar ライブラリのために一つ article を書いているという公平な仮定を採用すると、世界中に117人の Mizar を知っている人がいることになります）、それについてのドキュメンテーションはほとんど無く、この種のシステムでの通常のプラット・

フォーム（それは Unix の走る SUN ですが）で実行できません（訳注：現在では状況が異なるのでhttp://www.mizar.org/system/index.html#downloadを参照してください）。 他方 Coq は積極的にタイプ理論のコミュニティーからの発展をフォローしており、広く知られていて、多くのユーザーを持ち、良いドキュメンテーションを持ち、そしてすべてのメジャーなプラット・ホームで利用できます。

John Harrison は HOL のために 'Mizar mode' を書きました。 HOL や Coq のような LCF スタイルの証明プログラム（prover）に Mizar インターフェースを作り出すためには **'by'** による推論を可能にする tactic（戦術）を書かねばなりませんし（Harrisonは完全な一階述語論理の provability（証明能力／証明可能性）を実装することに決めました、それはよりパワフルですが非効率です）、Mizar 構文のためのパーサー（構文解析ツール）も書かねばなりません。 Mizar と Coq の意味論と型システムは同じではないので、２つの間には橋渡しをしなければならない差異もあります。 Mizar と Coq を合わせた強さを持つシステムを持つのは素晴らしいでしょう（それは多分 proof object を持つ 'Mizar' でしょう）しかしそれを実現するのに Coq を'エンハンス'するのはかなりエレガントでなく非効率と思われます（機能性（functionality）の重複を伴うでしょう）。 実際、数学への志向を持つユーザーの視点から見れば、 Mizar はすでに必要なものを全て持っているのです。

文献（References）

[1] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *A Compendium of Continuous Lattices.* Springer-Verlag, Berlin, Heidelberg, New York, 1980.

[2] J. Kotowicz, B. Madras, and M. Korolkiewicz. Basic notation of universal algebra. *Journal of Formalized Mathematics,* 4, 1992.

[3] M. Muzalewski. *An Outline of PC Mizar.* Fondation Philippe le Hodey, Brussels, 1993.

A　Mizar の実行方法（How To Run Mizar）

（訳注：2008年10月の段階では、少なくとも Windows（32bit）に関してはディストリビューションファイル mizar-7.9.03_4.110.1033-i386-win32.exe をダウンロードして適当なディレクトリで実行すれば、そのディレクトリ内に .zip ファイル、install.bat、その他のファイルが展開されます。　このinstall.bat を実行すれば ドライブ名：¥mizarの下に ¥abstr, ¥bib, ¥doc, ¥mml, ¥prelのディレクトリが自動的に作成されて mizar の実行環境が整います。　従ってこのＡ　Mizar の実行方法（How To Run Mizar）の記述は無視したほうが良いとおもいます。）

ディレクトリを３つ選びます：

- *distdir* : Mizar ディストリビューションを unzip する場所として
- *mizardir* : インストールされた Mizar ファイル用
- *workdir* : あなたの article を置くため

mizardir としては **'C:¥MIZAR'** の選択を推薦します、他の場所に置いてもすべてうまく動作しますが。

Mizar システムをインストールするには ZIP で圧縮されたインストレーションファイルを：

<ftp://ftp.mizar.org/ver*version*/disks.zip/>

から手にいれてください（このノートに書かれたシステムの '*version*'は '5-3.07' です（訳注：<http://www.mizar.org/system/index.html#download>をアクセスするほうが良いと思います。）。　すべてを *distdir* のなかに UNZIP します。それから **install** コマンドを実行します：

distdir¥**install** *distdir*¥ *mizardir*

これで '*mizardir*¥**MML**' にインストールされる Mizar ライブラリのソースを除くすべてがインストールされます。　これらのソースは実際あまり重要ではありません（ライブラリの参照においてさえも：'*mizardir*¥**ABSTR**' にあるアブストラクトの参照がより適切です）。　Mizar ライブラリのソースをアンパックするためには、'*mizardir*¥**MML**' のなかで'**FM***number*.**EXE**' ファイルを実行してください。

さて：

set misfiles = *mizardir*

を適切な **'AUTOEXEC.BAT'** ファイルに追加してください、 *mizardir* を DOS

のサーチ **PATH** に書き足し、それでインストレーションは完了します。

article を書くためにはサブディレクトリを２つ作ってください：

- *workdir*¥**TEXT**
- *workdir*¥**DICT**

あなたが作業をしようとしている article の名前が **‘FOO’** だとしましょう。 それでは：

- *workdir*¥**TEXT¥FOO.MIZ**
- *workdir*¥**DICT¥FOO.VOC**

を作成しましょう、あなたの article とボキャブラリ・ファイルです。

article が正しいかどうかチェックするためには、*workdir* へ（訳注：ワーキング）ディレクトリをチェンジして（**cd** *workdir*）、コマンド：

mizf text¥foo

を実行してください（**mizf** チェッカーは自分で **‘.miz’** サフィックスを追加します）。 これはファイルをチェックして、エラー・メッセージをチェックしたファイルの‘なか’に追加します（そこで **‘mizf’** コマンドを走らせる前にエディタが編集中のファイルをクローズする必要があるでしょう）。

B　Grammar （文法）

Mizar-Article =
  'environ'
   {'**vocabulary**' *File-Name-List* ';' |
    ('**notation**' |
    '**constructors**' |
    '**clusters**' |
    '**definitions**' |
    '**theorems**' |
    '**schemes**') File-Name-List ';' |
   '**requirements**' File-Name-List ';' }
 ('**begin**' { Text-Item }) { . . . } .
Text-Item =
 '**reserve**' *Identifier-List* '*for*' *Type-Expression-List* ';' |
 '**definition**'
   { *Definition-Item* }
   ['**redefine**' { *Definition-Item* }]
   '**end**' ';' |
Structure-Definition |
'**theorem**' *Proposition Justification* ';' |
['**scheme**'] *Identifier*
  '{' (*Identifier-List* '[' *Type-Expression-List* ']' |
      Identifier-List '(' *Type-Expression-List* ')' '->' *Type-Expression*)
      { ',' . . . } '}'
':' *Formula-Expression*
    ['**provided**' *Proposition* { '**and**' . . . }]
    Justification ';' |
   Auxiliary-Item |
   '**canceled**' [*Numeral*] ';' .
Definition-Item =
  Assumption |
  Auxiliary-Item |
  Structure-Definition |
  '**mode**' *M-Symbol* ['**of** *Identifier-List*]
    (['->' *Type-Expression*] ['**means**' *Definiens*] ';'

Correctness-Conditions |
'**is**' *Type-Expression* ';')
{ '**synonym**' *M-Symbol* ['**of**' *Identifier-List*] ';' } |
'**func**' *Functor-Pattern* ['->' *Type-Expression*]
[('**means**' | '**equals**') *Definiens*] ';'
Correctness-Conditions
{ '**commutativity**' *Justification* ';' }
{ '**synonym**' *Functor-Pattern* ';' } |
'**pred**' *Predicate-Pattern* ['***means***' *Definiens*] ';'
Correctness-Conditions
{ '**symmetry**' *Justification* ';' |
'**connectedness**' *Justification* ';' |
'**reflexivity**' *Justification* ';' |
'**irreflexivity**' *Justification* ';' }
{ ('**synonym**' | '**antonym**') *Predicate-Pattern* ';' } |
'**attr**' *Identifier* '**is**' *V-Symbol* '**means**' *Definiens* ';'
[*Correctness-Conditions*]
{ ('**synonym**' | '**antonym**')
(*Identifier* '**is**' *V-Symbol* | *Predicate-Pattern*) ';' } |
'**canceled**' [*Numeral*] |
'**cluster**' *Adjective-Cluster Type-Expression* ';' *Correctness-Conditions* |
'**cluster**' *Adjective-Cluster* '->' *Adjective-Cluster Type-Expression* ';'
Correctness-Conditions |
'**cluster**' *Term-Expression* '->' *Adjective-Cluster* ';'
Correctness-Conditions .

Structure-Definition =
'**struct**' ['(' *Type-Expression-List* ')'] *G-Symbol* ['*over*' *Identifier-List*]
'**(#**' (*U-Symbol* { ',' . . . } '->' *Type-Expression*) { ',' . . . } '**#)**' ';' .

Definiens =
[':' *Identifier* ':'] (*Formula-Expression* | *Term-Expression*) |
[':' *Identifier* ':']
((*Formula-Expression* | *Term-Expression*) '**if**' *Formula-Expression*)
{ ',' . . . } ['**otherwise**' (*Formula-Expression* | *Term-Expression*)] .

Functor-Pattern =
[*Functor-Loci*] **O**-*Symbol* [*Functor-Loci*] |
K-*Symbol Identifier-List L-Symbol* .

Functor-Loci =
    Identifier |
    '(' *Identifier-List* ')' .
Predicate-Pattern = [*Identifier-List*] **R**-*Symbol* [*Identifier-List*] .
Correctness-Conditions =
    { '**existence**' *Justification* ';' |
    '**uniqueness**' *Justification* ';' |
    '**coherence**' *Justification* ';' |
    '**compatibility**' | *Justification* ';' |
    '**consistency**' *Justification* ';' }
    ['**correctness**' *Justification* ';'] .
Justification =
    Simple-Justification |
    ('**proof**' | '**@proof**') *Reasoning* '**end**' .
Reasoning =
    { *Reasoning-Item* }
    ['**per**' '**cases**' *Simple-Justification* ';'
    (('**case**' (*Proposition* | *Conditions*) ';' { *Reasoning-Item* })
    { . . . } |
    ('**suppose**' (*Proposition* | *Conditions*) ';' { *Reasoning-Item* })
    { . . . })] .
Reasoning-Item =
    Auxiliary-Item |
    Assumption |
    ('**thus**' | '**hence**') *Statement* |
    '**take**' (*Term-Expression* | *Identifier* '=' *Term-Expression*) { ',' . . . } ';' .
Auxiliary-Item =
    ['**then**'] *Statement* |
    '**set**' (*Identifier* '=' *Term-Expression*) { ',' . . . } ';' |
    '**deffunc**' *Identifier* '(' [*Type-Expression-List*] ')' '=' *Term-Expression* |
    '**defpred**' *Identifier* '[' [*Type-Expression-List*] ']' '**means**'
    Formula-Expression .
Assumption =
    ('**let**' | '**given**') *Qualified-Variables* ['**such**' *Conditions*] ';' |
    '**assume**' (*Proposition* | *Conditions*) ';' .
Statement =

[‘**then**’]

(*Proposition Justification* ‘;’ |

‘**consider**’ *Qualified-Variables* [‘**such**’ *Conditions*]

Simple-Justification ‘;’ |

‘**reconsider**’

(*Identifier* ‘=’ *Term-Expression* | *Identifier*) { ‘,’ . . . }

‘**as**’ *Type-Expression Simple-Justification* ‘;’ |

Term-Expression ‘=’ *Term-Expression Simple-Justification*

‘.=’ (*Term-Expression Simple-Justification*) { ‘.=’ . . . }) |

[*Identifier* ‘:’] ‘**now**’ *Reasoning* ‘*end*’ ‘;’ .

Simple-Justification =

[‘**by**’ *Reference* { ‘,’ . . . }] |

‘**from**’ *Identifier* [‘(‘ *Reference* { ‘,’ . . . } ‘)’] .

Reference =

Identifier |

File-Name ‘:’ (*Numeral* | ‘**def**’ *Numeral*) { ‘,’ . . . } .

Conditions = ‘**that**’ *Proposition* { ‘**and**’ . . . } .

Proposition = [*Identifier* ‘:’] *Formula-Expression* .

Formula-Expression =

‘(‘ *Formula-Expression* ‘)’ |

[*Term-Expression-List*] **R**-*Symbol* [*Term-Expression-List*] |

Identifier [‘[‘ *Term-Expression-List* ‘]’] |

Term-Expression ‘**is**’ { [‘*non*’] **V**-*Symbol* } |

Term-Expression ‘**is**’ *Type-Expression* |

Quantified-Formula-Expression |

Formula-Expression ‘**&**’ *Formula-Expression* |

Formula-Expression ‘**or**’ *Formula-Expression* |

Formula-Expression ‘**implies**’ *Formula-Expression* |

Formula-Expression ‘**iff**’ *Formula-Expression* |

‘**not**’ *Formula-Expression* |

‘**contradiction**’ |

‘**thesis**’ .

Quantified-Formula-Expression =

‘**for**’ *Qualified-Variables* [‘**st**’ *Formula-Expression*]

(‘**holds**’ *Formula-Expression* | *Quantified-Formula-Expression*) |

‘**ex**’ *Qualified-Variables* ‘**st**’ *Formula-Expression* .

Qualified-Variables =
  Identifier-List |
  (*Identifier-List* ('**being**' | '**be**') *Type-Expression*) { ',' . . . }
  [',' *Identifier-List*] .
Type-Expression =
  '(' *Type-Expression* ')' |
  Adjective-Cluster **M**-*Symbol* ['**of**' *Term-Expression-List*] |
  Adjective-Cluster **G**-*Symbol* ['**over**' *Term-Expression-List*] .
  Adjective-Cluster = { ['**non**'] **V**-*Symbol* } .
  Term-Expression =
    '(' *Term-Expression* ')' |
    [*Arguments*] **O**-*Symbol* [*Arguments*] |
    K-*Symbol* *Term-Expression-List* **L**-*Symbol* |
    Identifier '(' [*Term-Expression-List*] ')' |
    G-*Symbol* '**(#**' *Term-Expression-List* '**#)**' |
    Identifier |
    '{' *Term-Expression*
      [('*where*' *Identifier-List* '**is**' *Type-Expression*) { ',' . . . }]
      ':' *Formula-Expression* '}' |
    Numeral |
    Term-Expression '**qua**' *Type-Expression* |
    '**the**' **U**-*Symbol* '**of**' *Term-Expression* |
    '**the**' **U**-*Symbol* |
    '**$1**' | '**$2**' | '**$3**' | '**$4**' | '**$5**' | '**$6**' | '**$7**' | '**$8**' |
    '**it**' .
  Arguments =
  Term-Expression |
  '(' *Term-Expression-List* ')' .

File-Name-List = File-Name { ',' . . . } .
Identifier-List = Identifier { ',' . . . } .
Type-Expression-List = Type-Expression { ',' . . . } .
Term-Expression-List = Term-Expression { ',' . . . } .

C　公理（Axioms）

ここには未定義の概念と Mizar システムの公理があります： Mizar ライブラリのすべてのものは定義され、これらからのみで証明がおこなわれます。（これは article ‘**HIDDEN**’ と ‘**TARSKI**’ の内容です）　公理 ‘**TARSKI:8**’ は削除されました、これは ‘**TARSKI:def 5**’ から導出できます。

```
definition mode Any; synonym set; end;
reserve x,y,z,u for Any, N,M,X,Y,Z for set;
definition let x,y; pred x = y; reflexivity; symmetry; antonym x <> y; end;
definition let x,X; pred x ∈ X; antisymmetry; end;
definition let X; attr X is empty; end;
definition cluster empty set; cluster non empty set; end;
definition func ∅ -> empty set; end;
definition let X; mode Element of X; end;
definition let X; func bool X -> non empty set; end;
definition let X; mode Subset of X is Element of bool X; end;
definition let X be non empty set; cluster non empty Subset of X; end;
definition let X,Y; pred X c= Y; reflexivity; end;
definition let D be non empty set, X be non empty Subset of D;
  redefine mode Element of X -> Element of D;
end;

theorem (for x holds x ∈ X iff x ∈ Y) implies X = Y;
definition let y; func {y} -> set means x ∈ it iff x = y;
  let z; func {y,z} -> set means x ∈ it iff x = y or x = z;
  commutativity;
end;
definition let y; cluster {y} -> non empty;
  let z; cluster {y,z} -> non empty;
end;
definition let X,Y; redefine pred X c= Y means x ∈ X implies x ∈ Y; end;
definition let X;
  func union X -> set means x ∈ it iff ex Y st x ∈ Y & Y ∈ X;
end;
theorem X = bool Y iff for Z holds Z ∈ X iff Z c= Y;
theorem x ∈ X implies ex Y st Y ∈ X & not ex x st x ∈ X & x ∈ Y;
```

```
scheme Fraenkel {A()->set, P[Any,Any]}:
   ex X st for x holds x ∈ X iff ex y st y ∈ A() & P[y,x]
  provided
   for x,y,z st P[x,y] & P[x,z] holds y = z;
definition let x,y; func [x,y] equals {{x,y},{x}}; end;
definition let X,Y;
  pred X ≈ Y means ex Z st
    (for x st x ∈ X ex y st y ∈ Y & [x,y] ∈ Z) &
    (for y st y ∈ Y ex x st x ∈ X & [x,y] ∈ Z) &
     for x,y,z,u st [x,y] ∈ Z & [z,u] ∈ Z holds x = z iff y = u;
end;

:: Axiom der unerreichbaren Mengen
theorem ex M st N ∈ M &
  (for X,Y holds X ∈ M & Y c= X implies Y ∈ M) &
  (for X holds X ∈ M implies bool X ∈ M) &
  (for X holds X c= M implies X ≈ M or X ∈ M);
```

D　完全な例（Complete Example）

ここに完全な Mizar text の例があります：それは '**UNIALG_1**' という article で Mizar ライブラリの Jarosław Kotowicz,　Beata Madras そして Małgorzata Korolkiewicz [2]による article の番号303から取りました。 そこでは '**Universal_Algebra**' 型を定義しています（一緒に '**arity**'（項数：引数の数）と '**signature**' の概念も)、それは普遍代数（universal algebra）の理論からの 'one-sorted algebras' の Mizar への実装（インプリメンテーションです)。

ここで 'アブストラクト' ファイル '**UNIALG_1.ABS**' と、完全な Mizar article '**UNIALG_1.MIZ**' の両方を同様に提示します。 行の間に散らばっているのは説明です。 時々説明は他の article からの Mizar テキストを含みます： article '**UNIALG_1**' のメイン・テキストは字下げ（indented）されていなくて、左マージンに行番号を持つのでそれと分かります（訳注：行番号のあとで indent してあります)。

D.1 アブストラクト（Abstract）

Mizar ライブラリの article の中に何があるかを見つけるために、一般的にはそのアブストラクト・ファイルだけを見ます。 これは full article から自動的に作成されます。 そのファイルでは、すべての証明は取り除かれ 定理に対しては ‘**UNIALG_1:5**’のように、定義に対しては ‘**UNIALG_1:def 11**’のように‘sequence numbers’が自動的に挿入されます。

```
:: Basic Notation of Universal Algebra
::   by Jaros{\l}aw Kotowicz, Beata Madras and Ma{\l}gorzata Korolkiewicz
::
:: Received December 29, 1992
:: Copyright (c) 1992 Association of Mizar Users

environ

vocabulary UNIALG, PFUNC1, FINSEQ, FUNC_REL, FUNC, FINITER2, PBOOLE,
  UNIALG_D;
notation ARYTM, NAT_1, STRUCT_0, TARSKI, RELAT_1, FUNCT_1, FINSEQ_1,
  FUNCOP_1,
    PARTFUN1, ZF_REFLE;
clusters TARSKI, FINSEQ_1, RELSET_1, STRUCT_0, ARYTM, PARTFUN1,
  FUNCOP_1;
constructors FINSEQ_4, STRUCT_0, ZF_REFLE, FUNCOP_1, PARTFUN1;
requirements ARYTM;
```

これは article の ‘**environ**’ ヘッダーです。 それは article の使う Mizar ライブラリからの、種々のボキャブラリ（ ‘**vocabulary**’ 指令のなかで）、article（‘**notations**’、 ‘**clusters**’、そして ‘**constructors**’ 指令のなかで）について述べます。 このヘッダーの唯一特別なアイテムはボキャブラリ ‘**UNIALG**’ で、それはこの article に‘特有（special）’なものです、それから ‘**ARYTM**’ ラベルで、それはボキャブラリの名前でも article の名前でもありません。

ボキャブラリ・ファイル ‘**UNIALG.VOC**’ は Mizar の配布ファイルの中に明示的には存在せず、‘コンパイルされた’ボキャブラリ・ライブラリ ‘**MML.VCB**’ の一部として存在します。 しかし、それはコマンド ‘**listvoc UNIALG**’ を使って：

```
GUAStr
Ucharact
Vhomogeneous
Vquasi_total
Vpartial
OOpers 128
MUniversal_Algebra
Oarity 128
Osignature
```

とプリントアウトできます。　これらは **‘func’(‘O’)**, **‘mode’(‘M’)**, **‘attr’(‘V’)**, **‘struct’(‘G’)**, **‘struct field’(‘U’)** のシンボルで、この article で‘新しく’出てきました。　これらのシンボルはアブストラクトの 62, 63, 75, 82, 105, /, 142, 150, そして161 行に対応し、full article の153, 154, 176, 182, 257, /, 343, 351そして395行に対応します。 **‘Opers’** という操作はこの article には現れません、しかし article **‘UNIALG_2’** に定義してあります（両方の article はボキャブラリを共用します）。

```
17 begin
```

Mizar article は一つあるいはそれ以上の ‘sections’ から構成され、それらはいずれも **‘begin’** で始まります（対応する **‘end’** はありません）。　この article の持つ section は一つだけです。

```
20 reserve A for set,
21         a for Element of A,
22         x,y for FinSequence of A,
23         h for PartFunc of A*,A ,
24         n,m for Nat,
25         z for set;
```

これらの予約は直截（straight-forward）です：それらのうちのいくつかは **‘A’** がスコープの中にあるときだけ使用されます。
このリストで最も興味のある型は **‘h’** の型です：

```
PartFunc of A*,A
```

ポストフィックス・オペレータ **‘*’** はarticle **‘FINSEQ_1’** で：

```
definition let D be set;
```

```
  func D* -> set means x ∈ it iff x is FinSequence of D;
end;
```

と定義されています（これを決める一番簡単な方法は webバージョンのMizar ライブラリ[2]でその（訳注：シンボルを）クリックすることです）。（ここに提示した定義はアブストラクトから取りました、ですから証明を含みません）、オペレータ・シンボル '*' は vocabulary 'FINSEQ' に：

```
O* 128
```

と定義があります。　一方、モード 'FinSequence' は同じ article から：

```
definition let n;
  func Seg n -> set equals { k : 1 ≤ k & k ≤ n };
end;
definition let IT be Relation;
  attr IT is FinSequence-like means ex n st dom IT = Seg n;
end;
definition
  mode FinSequence is FinSequence-like Function;
end;
definition let D be set;
  mode FinSequence of D -> FinSequence means rng it c= D;
end;
```

と出てきます。　'PartFunc' モードはarticle 'PARTFUN1' で：

```
definition let X,Y;
  mode PartFunc of X,Y is Function-like Relation of X,Y;
end;
```

と定義されています。　そこで '**PartFunc of A*, A**' は '**partial function from A* to A**' を意味します。　モードに名前を付ける一般的な方法は：

mode-name of parameter, parameter,. . .

で、それはすこし不自然な構文記法を説明します。

```
27 definition let A;
```

[2]<http://www.mizar.org/JFM/mmlident.html>

```
28    let IT be PartFunc of A*,A;
29 attr IT is homogeneous means
30   :: UNIALG_1:def 1
31   for x,y st x ∈ dom IT & y ∈ dom IT holds len x = len y;
32 end;

34 definition let A;
35   let IT be PartFunc of A*,A;
36 attr IT is quasi_total means
37  :: UNIALG_1:def 2
38   for x,y st len x = len y & x ∈ dom IT holds y ∈ dom IT;
39 end;

41 definition let A be non empty set;
42 cluster homogeneous quasi_total non empty PartFunc of A*,A;
43 end;
```

このクラスタは型：

```
homogeneous quasi_total non empty PartFunc of A*,A
```

が存在（inhabited）すると述べています。（Mizar の型は non-empty でないといけないので）形容詞 ‘**homogeneous**’、‘**quasi_total**’、そして ‘**non-empty**’ を（単独で、あるいは組み合わせて）‘**PartFunc of A*, A**’ の形の型と一緒に使用することが許されている必要があります。 それがないと、例えば、このアブストラクトの行 50, 149 そして行 165-166 の型：

```
homogeneous quasi_total non empty PartFunc of A*,A
homogeneous non empty PartFunc of A*,A
homogeneous non empty
    PartFunc of (the carrier of U)*,the carrier of U
```

は正しくないことになります（証明の中でこのような型がさらに１１種類出現します）。

```
45 theorem :: UNIALG_1:1
46 h is non empty iff dom h <> Ø;

48 theorem :: UNIALG_1:2
49 for A being non empty set, a being Element of A
50 holds {<Ø>A} -->a is homogeneous quasi_total non empty PartFunc of A*,A;
```

演算子 **<∅>A** は **A** 型の要素を持つ空の有限数列を意味します。　それは article **‘FINSEQ_1’** に：

```
definition redefine func ∅; synonym <∅>; end;
definition let D be set;
    func <∅>(D) -> empty FinSequence of D equals <∅>;
end;
```

と定義されています。　ここでオペレータ・シンボル **‘<∅>’** はボキャブラリ **‘FINSEQ’** に：

```
O<∅> 254
```

とあります。　このなかで、**‘∅’** は article **‘HIDDEN’** で：

```
definition
  func ∅ -> empty set;
end;
```

として導入され、そしてボキャブラリ **‘HIDDEN’** で：

```
O∅ 128
```

と導入された空集合の名前です。（**‘HIDDEN’** は Mizar システムの公理を与える２つのファイルの一つなのでこの定義の中に **‘means’** や **‘equal’** はありません。　さらには **‘HIDDEN’** articleと **‘HIDDEN’** ボキャブラリは常に存在するので article の **‘environ’** ヘッダーのリストには書く必要はありません。）

中カッコ (braces) **‘{ … }’** は元一つを持つ集合を示します。　それは article **‘TARSKI’** で：

```
definition let y;
  func {y} -> set means x ∈ it iff x = y;
end;
```

と定義されています。　シンボル **‘{ ’** と **‘ }’** は Mizar 構文 (Mizar syntax) の中に、独立に (on their own) 出現しますが、ボキャブラリには現れません。

インフィックス演算子 **‘-->’** は集合についての定数機能（constant function）を作成します。　それは article **‘FUNCOP_1’** で：

```
definition let A, a be set;
  func A --> a -> set equals [:A, {a}:];
```

```
end;
```

と、それからボキャブラリ‘**FINITER2**’で：

```
O--> 16
```

と定義されています。　ここで[: … :]は article ‘**ZFMISC_1**’ からの直積（Cartesian product、デカルト積）です：

```
definition let X1,X2;
  func [: X1,X2 :] means
    z ∈ it iff ex x,y st x ∈ X1 & y ∈ X2 & z = [x,y];
  end;
```

そして‘**[…]**’ は article ‘**TARSKI**’ からのクラトウスキ対です：

```
definition
  func [x,y] equals {{x,y},{x}};
end;
```

```
theorem :: UNIALG_1:3
for A being non empty set, a being Element of A
 holds {<Ø>A} -->a is Element of PFuncs(A*,A);
```

func ‘**PFuncs**’ はarticle ‘**PARTFUN1**’ で：

```
definition let X,Y;
  func PFuncs(X,Y) -> set means
    x ∈ it iff
      ex f being Function st x = f & dom f c= X & rng f c= Y;
end;
```

と定義されています。

```
definition let A;
 mode PFuncFinSequence of A -> FinSequence of PFuncs(A*,A) means
:: UNIALG_1:def 3
 not contradiction;
end;
```

公式の Mizar の文法の提案（appendix B, ページ16）にもかかわらず、冗長な性格付け（characterization） ‘**means not contradiction**’ は除去さ

れないでしょう。

```
struct (1-sorted) UAStr ≪ carrier -> set,
        charact -> PFuncFinSequence of the carrier≫;
```

Mizar での 'algebra' の概念の実装の根底となる構造体 (**struct**) **'UAStr'** には２つのフィールドがあります：**'carrier'** それは algebra の '種類（sort）' ですが、それと **'charact'** それは algebra の 'function' の一連の続き（sequence）です。　祖先型（ancestor）である構造体 **struct '1-sorted'** は article **'STRUCT_0'** で定義されています：

```
definition
  struct 1-sorted ≪carrier -> set≫;
end;
```

```
definition
  cluster non empty strict UAStr;
end;
```

このクラスタは **non empty strict UAStr** が存在すると述べています。それはこの article では使用されていません。

型 **'UAStr'** は型 **'1-sorted'** へ拡張されるので、そしてそれは **'set'** 型よりは狭いのですが、形容詞 **'non empty'** は article **'STRUCT_0'** の定義：

```
definition let S be 1-sorted;
  attr S is empty means the carrier of S is empty;
end;
```

を参照し、article **'HIDDEN'** の定義：

```
definition let X be set;
  attr X is empty;
end;
```

は参照しないことに注意してください。

```
definition let D be non empty set, c be PFuncFinSequence of D;
  cluster UAStr ≪D,c ≫ -> non empty;
end;
```

このクラスタは形容詞 **'non empty'** を獲得 (to gain) するために形容詞 'non empty' をもつ **D** とともに **'UAStr≪D,c≫'** の形の表現を引き起こし（cause）

ます。　それは article の323行で使用されます。

```
definition let A;
 let IT be PFuncFinSequence of A;
attr IT is homogeneous means
:: UNIALG_1:def 4
   for n,h st n ∈ dom IT & h = IT.n holds h is homogeneous;
end;

definition let A;
 let IT be PFuncFinSequence of A;
attr IT is quasi_total means
:: UNIALG_1:def 5
   for n,h st n ∈ dom IT & h = IT.n holds h is quasi_total;
end;

definition let F be Function;
redefine attr F is non-empty means
:: UNIALG_1:def 6
 for n being set st n ∈ dom F holds F.n is non empty;
end;
```

その **attr** '**non empty**' は article '**ZF_REFLE**' で：

```
definition let F be Function;
  attr F is non-empty means not Ø ∈ rng F;
end;
```

と定義されていました。

ここに与えられた再定義（redefinition）はこれと等価（equivalent）でなければなりません（それはもっと特異的なパラメータ型を持つことを許されていますが、ここはそのケースではありません）。

オリジナルの定義と再定義の両方とも厳密に同じ方法で使用できます。この新しい定義は full article の 242-250 行で使用されています（ここでの証明は定義の'拡張'型（'expanded' form）だからです： Mizar は定義を article 自身から拡張します、そして **'definitions' environ** 指令の中の article からも）。　さらにその 'definitional theorem'（定義定理）**('Def6')** は full article の 301, 413 そして 452 行から参照されます.

```
93 definition let A be non empty set; let x be Element of PFuncs(A*,A);
94  redefine
95    func <*x*> -> PFuncFinSequence of A;
96 end;
```

これは article ‘**`FINSEQ_1`**’ からの演算子 ‘**`<* ... *>`**’ を：

```
definition let x;
  func <*x*> -> set equals { [1,x] };
end;
```

に再定義します。　この演算子は長さ 1 の一連の続きを書くのに使われます。再定義はこの func に新しい性格付けを与えるのではありません（それは元のそれを継承（inherit）します）、しかしそれはその型を変化させます。　それなしには ‘homogeneous’ や ‘quasi_total’ のような形容詞は、例えばアブストラクトの123行のように、 ‘**`<*x*>`**’ の形の表現に適用することは決してできません。

```
98 definition let A be non empty set;
99 cluster homogeneous quasi_total non-empty PFuncFinSequence of A;
100 end;
```

このクラスタはこの article では使用されません。

```
102 reserve U for UAStr;

104 definition let IT be UAStr;
105 attr IT is partial means
106 :: UNIALG_1:def 7
107  the charact of IT is homogeneous;
108 attr IT is quasi_total means
109 :: UNIALG_1:def 8
110  the charact of IT is quasi_total;
111 attr IT is non-empty means
112 :: UNIALG_1:def 9
113 the charact of IT <> <Ø> & the charact of IT is non-empty;
114 end;

116 reserve A for non empty set,
117          h for PartFunc of A*,A ,
```

```
118         x,y for FinSequence of A,
119         a for Element of A;
```

これらの予約は 20-23 行のそれと、これの後に ‘**A**’ が形容詞 ‘**non-empty**’ を持つことを除き、まったく同じものです。

```
121 theorem :: UNIALG_1:4
122 for x be Element of PFuncs(A*,A) st x = {<Ø>A} --> a holds
123    <*x*> is homogeneous quasi_total non-empty;
125 definition
126  cluster quasi_total partial non-empty strict non empty UAStr;
127 end;
```

このクラスタは型の正当性を確立するため、この article で５回使用されます。　例えばこのアブストラクトの 142 行でモード（**mode**）‘**Universal_Algebra**’ の定義の中で使用されています。

```
129 definition let U be partial UAStr;
130  cluster the charact of U -> homogeneous;
131 end;

133 definition let U be quasi_total UAStr;
134  cluster the charact of U -> quasi_total;
135 end;

137 definition let U be non-empty UAStr;
138  cluster the charact of U -> non-empty non empty;
139 end;
```

‘**non-empty**’ は algebra の中に少なくとも一つの関数があることを意味します、‘**non-empty**’ はこれらの関数に全部が空でないことを意味します。

```
141 definition
142  mode Universal_Algebra is quasi_total partial non-empty non empty
      UAStr;
143 end;

145 reserve U for partial non-empty non empty UAStr;

147 definition
148  let A;
```

```
 let f be homogeneous non empty PartFunc of A*,A;
func arity(f) -> Nat means
:: UNIALG_1:def 10
 x ∈ dom f implies it = len x;
end;
```

この **func** の引数 **'A'** は暗黙の引数であるのに注意してください：それは **'arity f'** の表記（notation）のなかに存在しません。

```
theorem :: UNIALG_1:5
for U holds for n st n∈dom the charact of(U) holds
   (the charact of(U)).n is
   PartFunc of (the carrier of U)*,the carrier of U;
definition let U;
func signature(U) ->FinSequence of NAT means
:: UNIALG_1:def 11
 len it = len the charact of(U) &
 for n st n ∈ dom it holds
   for h be homogeneous non empty
   PartFunc of (the carrier of U )*,the carrier of U
     st h = (the charact of(U)).n
     holds it.n = arity(h);
end;
```

D.2 アーティクル（Article）

```
:: Basic Notation of Universal Algebra
:: by Jaros{\l}aw Kotowicz, Beata Madras and Ma{\l}gorzata Korolkiewicz
::
:: Received December 29, 1992
:: Copyright (c) 1992 Association of Mizar Users

environ

vocabulary UNIALG, PFUNC1, FINSEQ, FUNC_REL, FUNC, FINITER2, PBOOLE,
 UNIALG_D;
constructors FINSEQ_4, STRUCT_0, ZF_REFLE, FUNCOP_1, PARTFUN1;
requirements ARYTM;
```

```
12 notation ARYTM, NAT_1, STRUCT_0, TARSKI, RELAT_1, FUNCT_1, FINSEQ_1,
   FUNCOP_1,
13     PARTFUN1, ZF_REFLE;
14 clusters TARSKI, FINSEQ_1, RELSET_1, STRUCT_0, ARYTM, PARTFUN1,
   FUNCOP_1;
15 definitions TARSKI, STRUCT_0, ZF_REFLE;
16 theorems TARSKI, FUNCT_1, PARTFUN1, FINSEQ_1, FUNCOP_1, RELAT_1,
   RELSET_1,
17       FINSEQ_3, ZF_REFLE;
18 schemes MATRIX_2;
```

この **'definitions'**, **'theorems'** そして **'schemes'** 指令はアブストラクトにはありませんでした、それらは証明だけに関連するからです。 それらはまたライブラリからの article の名前を含みます。 最後の２つはそのなかから **theorems** と **schemes** が使用される article をリストアップします。

15 行の **'definitions'** 指令はこの article の9ヶ所に応用されます。

- 50-54, 56-60, 93-97, 99-103, 131-135そして137-141行の証明。

これらの proof は包含の演算子 **'c='** を含むステートメントです、しかし proof は全称量化子式を証明します。 **'c='** 演算子は article **'TARSKI'** で定義されています（実際は再定義です、なぜなら **'c='** 演算子はもともと定義なしで article **'HIDDEN'** で）：

```
definition let X,Y;
  redefine pred X c=Y means x∈X implies x∈Y;
end;
```

と導入されていますから。 そこで、**'definitions TARSKI;'** 指令によりその **'c='** ステートメントは適切な全称式に拡張されるでしょう。

- 161と170 行の **'thus the carrier of UAStr ≪D,c≫ is non empty;'**。

ここで証明されるステートメントは **sturct** についてです、しかし供給されるステートメントはその **struct** の **the carrier** についてです。 これが正しくなるためには **'definitions STRUCT_0;'** が structs ステートメントに関する **'empty'** の定義を透過（transparent）にする必要があります：

```
definition let S be 1-sorted;
  attr S is empty means the carrier of S is empty;
```

```
end;
```

- 199-201 行の証明 **‘assume Ø ∈ rng F; ... hence contradiction by A2;’** 。

それは **‘not Ø ∈ rng F’** を証明します、しかし実際に証明すると考えられるステートメントは **‘F is non empty’** です。　そこで指令 **‘definitions ZF_REFLE;’** が定義：

```
definition let F be Function;
  attr F is non-empty means not Ø ∈ rng F;
end;
```

を透過（transparent）にするために必要です。
これらの定義の拡張を必要とするのは（Mizar checkerの型チェック・フェーズである）**‘Analyzer’** であり、（推論の論理的正当性をチェックする）**‘Checker’** ではありません。

```
20 begin

23 reserve A for set,
24          a for Element of A,
25          x,y for FinSequence of A,
26          h for PartFunc of A*,A ,
27          n,m for Nat,
28          z for set;

30 definition let A;
31  let IT be PartFunc of A*,A;
32 attr IT is homogeneous means       :Def1:
33  for x,y st x ∈ dom IT & y ∈ dom IT holds len x = len y;
34 end;
```

この **definition** は（32行のラベルのために）この article の **‘Def1’** として参照されるのに注意してください（364行）、しかし（このアブストラクトの30行のラベル）他の article では **‘UNIALG_1:def 1’** として参照されます（具体的には： article **‘ALG_1’**, **‘FREEALG’** そして **‘PRALG_1’** です）。　同様にこの article（121, 357, そして 372行）では83-84行からの定理（**theorem**）は **‘Th1’** として参照されます、しかし他の（**‘ALG_1’**, **‘MSSUBLAT’**, **’PRALG_1’** そして **‘UNIALG_2’** ）（訳注：の article では）**‘UNIALG_1:1’** （このアブストラクトの 45行）として参照されます。

```
36 definition let A;
37   let IT be PartFunc of A*,A;
38 attr IT is quasi_total means
39   for x,y st len x = len y & x ∈ dom IT holds y ∈ dom IT;
40 end;
```

以下の定義とその証明は、それ以外の部分（remainder）の Mizar text よりさらに詳細な注釈を付けることにしましょう。　証明（45-80行）は実際、**theorem**‘**Th2**’（58ページの89-122行）の証明と同一です、それでその部分は障害（interruption）なしに読むことができます。

```
42 definition let A be non empty set;
43 cluster homogeneous quasi_total non empty PartFunc of A*,A;
```

このクラスタは型：

```
homogeneous quasi_total non empty PartFunc of A*,A
```

は存在している（inhabited）と述べています。　それはもし誰かが形容詞‘**homogeneous**’、‘**quasi_total**’あるいは‘**non empty**’を‘**PartFunc of A*,A;**’の形の型と一緒に使用すると望めば、存在するのです。

```
44 existence
```

この種類の‘クラスタ’はある型の non-emptiness を述べているので、その正当性条件は ‘**existence**’ステートメントです。　このステートメントは：

```
ex f being PartFunc of A*,A st
   f is homogeneous & f is quasi_total & f is non empty
```

です。

```
45 proof
46   consider a be Element of A;
```

‘**consider**’ステップには存在のステートメントが justification として必要です。　このケースでは、それは：

```
ex a be Element of A
```

です（これは Mizar の完全な式ではありません：それを完成するには‘**st**

not contradiction' を追加してください）。 Mizar の型はすべて non-empty なので、この種の存在証明は明白です：ですから '**consider**' は justification を必要としません。

'**A**' の：

```
A is non empty
```

という特性（42行の）は、ここでは使用されません：'**consider**' ステップは '**A**' がうまい具合に（just）'**set**' であったならば有効であったかも知れません。 しかし、その場合は：

```
a ∈ A
```

は正しくなかったでしょう（'**A is non empty**' の時、それ（'a∈A'）は '**a is Element of A**' から得ることができ、'a∈A' は Mizar システムに組み込まれています）、そして59行目の '**hence**' ステップは失敗していたでしょう。

```
47  set f = {<Ø>A} -->a;
```

これはローカルな定数 '**f**' の定義です：これ以降出現するの全ての '**f**' は '**{<Ø>A} -->a;**' に拡張されます。

ここで定義された '**f**' は、集合（set）'**{<Ø>A}**' を定義域として持つ関数で、定数 '**a**' にマッピングされています。 この定義域は長さゼロの数列であり：集合 $\mathbf{A}^0$ です。 そこで：

$$\mathbf{f}: \mathbf{A}^0 \to \mathbf{A}, \quad \langle\rangle \mapsto \mathbf{a}$$ （訳注：LaTeXでは↦は¥mapsto）

となります。 そこで、この **f** は **A** に関する'無引数の（nullary）'の関数でそれは定数 **a** を表します。

```
48  A1: dom f = {<Ø>A} & rng f = {a} by FUNCOP_1:14;
```

このステートメントは：

```
A1: dom ({<Ø>A}-->a) = {<Ø>A} & rng ({<Ø>A}-->a) = {a}
```

に拡張され、それは42行の '**A**' の型：

```
A is non empty set
```

article '**HIDDEN**' からの '**Ø**' の型：

```
definition
  func Ø -> empty set;
```

```
end;
```

そして:

```
theorem :: FUNCOP_1:14
  A <> ∅ implies dom (A --> x) = A & rng (A --> x) = {x};
```

から得られます。

```
49  A2: dom f c= A*
```

15行目の‘**definitions TARSKI**’指令のゆえに、これは：

```
definition let X,Y; redefine pred X c= Y means
:: TARSKI:def 3
  x ∈ X implies x ∈ Y;
end;
```

に従い

```
A2: for z being set st z ∈ dom f holds z ∈ A*
```

に拡張されます。

```
50    proof
```

ここからステートメント‘**A2**’の‘**local**’サブプルーフが始まります。

```
51      let z; assume z ∈ dom f; then
```

‘**let**’と‘**assume**’のスケルトン・ステップは、ステートメントの‘**for** …’と‘**st** … **holds**’の部分に対応します。　これらのステップの後に：

```
z ∈ A*
```

が証明されるものとして残されます。

```
52      z = <∅>A by TARSKI:def 1,A1;
```

ステートメント：

```
z = <∅>A
```

は（51行の最後の‘**then**’のゆえに）：

```
z ∈ dom f
```

そして：

```
A1: dom f = {<Ø>A} & rng f = {a}
```

そして：

```
definition let y; func {y} -> set means
:: TARSKI:def 1
   x ∈ it iff x = y;
end
```

から得ることができます。

```
     hence thesis by FINSEQ_1:65;
```

証明されるべく残されているステートメントは：

```
z ∈ A*
```

さて（‘**hence**’ が ’**then**’ を含むので）：

```
z = <Ø>A
```

から得ることができます。　そして型 **<Ø>A** は：

```
definition let D be set;
   func <Ø>(D) -> empty FinSequence of D equals
   :: FINSEQ_1:def 6
     <Ø>;
end;
```

と

```
theorem :: FINSEQ_1:65
   x ∈ D* iff x is FinSequence of D;
```

から与えられます。

```
   end;
```

… そしてサブプルーフは完結します。

```
rng f c= A
   proof
     let z; assume z ∈ rng f; then
```

```
58      z = a by A1,TARSKI:def 1;
59      hence thesis;
60    end; then
```

このサブプルーフは前のものと類似のものです。　行59のステップは：

```
a ∈ A
```

を使用し、これは46行の **‘consider’** に続くテキストで説明されます。

```
61  reconsider f as PartFunc of A*,A by RELSET_1:11,A2;
```

この後では reconsider **‘f’** は同じ意味を維持するでしょう（それは依然として **‘{<Ø>A}-->a’** に拡張されます）、しかしその*型*は **‘PartFunc of A*,A’** になってしまいます。　これを justify するには、まず：

```
f is PartFunc of A*,A
```

を証明せねばなりません、というのは **‘f’** の定義とモード **‘PartFunc’** の定義：

```
definition let X,Y;
  mode PartFunc of X,Y is Function-like Relation of X,Y;
end;
```

は

```
{<Ø>A}-->a is Function-like Relation of A*,A
```

に‘拡張’されるからです。　これは：

```
A2: dom f c= A*
```

そして（60行の‘**then**’）：

```
rng f c= A
```

そして（article ‘**FUNCOP_1**’ から）：

```
definition let A, z be set;
   cluster A --> z -> Function-like Relation-like;
end;
```

そして（article **‘RELAT_1**’ から）：

```
definition
   mode Relation is Relation-like set;
end;
```

そして：

```
theorem :: RELSET_1:11
   for R being Relation st dom R c= X & rng R c= Y holds
      R is Relation of X,Y;
```

から得られます。

```
62    A3: f is homogeneous
```

このステートメントは、30-34行の定義により：

```
A3: for x,y being FinSequence of A st x ∈ dom f & y ∈ dom f holds
  len x = len y
```

に拡張されます。

```
63    proof
64      let x,y be FinSequence of A; assume
65      x ∈ dom f & y ∈ dom f; then
```

この後、証明されるべく残されているステートメントは：

```
len x = len y
```

です。

```
66    x = <∅>A & y = <∅>A by TARSKI:def 1,A1;
```

ステートメント：

```
x = <∅>A & y = <∅>A
```

は（65行目の‘**then**’）

```
x ∈ dom f & y ∈ dom f
```

と：

```
A1: dom f = {<∅>A} & rng f = {a}
```

と：

```
definition let y; func {y} -> set means
:: TARSKI:def 1
   x ∈ it iff x = y;
end
```

から得られます。

```
67   hence thesis;
```

（**'hence'** は前のステートメントを参照している）ので：

```
x = <Ø>A & y = <Ø>A
```

証明されるべく残されている命題（thesis）は：

```
len x = len y
```

で、それは：

```
len <Ø>A = len <Ø>A
```

と同等（equivalent）で、反射律（reflexivity）により（等式型の推論（equational reasoning は Mizar に組み込まれています）正しいのです。

```
68   end;
69 A4: f is quasi_total
```

このステートメントは、36-40行の定義により：

```
for x,y being FinSequence of A st len x = len y & x ∈ dom f holds
  y ∈ dom f
```

に拡張されます。

```
70   proof
71     let x,y be FinSequence of A; assume
72     A5: len x = len y & x ∈ dom f; then
```

これの後で、証明されるべく残されているステートメントは：

```
y ∈ dom f
```

です。

```
73 x = <Ø>A by TARSKI:def 1,A1; then
```

ステートメント：

```
x = <∅>A
```

は：

```
A1: dom f = {<∅>A} & rng f = {a}
```

と(72行の‘**then**’)：

```
len x = len y & x ∈ dom f
```

と:

```
definition let y; func {y} -> set means
:: TARSKI:def 1
   x ∈ it iff x = y;
end
```

から得られます。

```
74     len x = 0 by FINSEQ_1:32; then
```

ステートメント:

```
len x = 0
```

は：

```
x = <∅>A
```

と（73行の‘**then**’）：

```
theorem :: FINSEQ_1:32
  p=<∅>(D) iff len p = 0;
```

から得られます。

```
75     y = <∅>A by FINSEQ_1:32,A5;
```

ステートメント：

```
y = <∅>A
```

は：

```
A5: len x = len y & x ∈ dom f
```

と（74行の **'then'**）：

```
len x = 0
```

と：

```
theorem :: FINSEQ_1:32
  p=<Ø>(D) iff len p = 0;
```

から得られます。

```
76    hence thesis by A1,TARSKI:def 1;
```

証明されるべき命題は：

```
y ∈ dom f
```

は（**'hence'**）：

```
y = <Ø>A
```

と：

```
A1: dom f = {<Ø>A} & rng f = {a}
```

と：

```
definition let y; func {y} -> set means
  :: TARSKI:def 1
x ∈ it iff x = y;
end
```

から得られます。

```
77   end;
78 f is non empty by RELAT_1:60, FUNCOP_1:14;
```

ステートメント：

```
f is non empty
```

は：

```
{<Ø>A}-->a is non empty
```

に拡張され、それは：

```
theorem :: RELAT_1:60
  dom Ø = Ø & rng Ø = Ø;
```

と：

```
theorem :: FUNCOP_1:14
  A <> Ø implies dom (A --> x) = A & rng (A --> x) = {x};
```

とクラスタ（article **'TARSKI'** から）の：

```
definition let y;
   cluster {y} -> non empty;
end;
```

から得られます。　知見（knowledge）：

```
A is empty implies A = Ø
```

は Mizar の reasoner（訳注：証明のスケルトンに責任があるプロセッサ（言語処理系）のモジュール）に組み込まれています。

```
79    hence thesis by A3,A4;
```

証明すべき命題は：

```
ex f being PartFunc of A*,A st
  f is homogeneous & f is quasi_total & f is non empty
```

で、これは47-61行で'定義された'　特定の **f** に対して、われわれは：

```
f is homogeneous & f is quasi_total & f is non empty
```

ということを知っていて、それは：

```
A3: f is homogeneous
```

と：

```
A4: f is quasi_total
```

と（**'hence'**）：

```
f is non empty
```

という事実から得ることができます。　Mizar は、**'f'** が証言された（witnessing）オブジェクトであると言われなくても、自分自身で存在量化

子 **ex** の導入を解決（figure out）できることに注意してください。

```
80   end;
```

これで注釈つきの証明は終了です。

```
81 end;

83 theorem Th1:
84 h is non empty iff dom h <> Ø by RELAT_1:64, RELAT_1:60;

86 theorem Th2:
87 for A being non empty set, a being Element of A
88 holds {<Ø>A} -->a is homogeneous quasi_total non empty PartFunc of A*,A
```

この定理はすでにクラスタのなかで証明されています(47,61,62,69と78行)、しかしクラスタはその定理を述べるために提示されなければならないのに注意してください。

```
89   proof let A be non empty set, a be Element of A;
90       set f = {<Ø>A} -->a;
91       A1: dom f = {<Ø>A} & rng f = {a} by FUNCOP_1:14;
92       A2: dom f c= A*
93        proof
94         let z; assume z ∈ dom f; then
95         z = <Ø>A by TARSKI:def 1,A1;
96         hence thesis by FINSEQ_1:65;
97        end;
98     rng f c= A
99        proof
100         let z; assume z ∈ rng f; then
101         z = a by A1,TARSKI:def 1;
102         hence thesis;
103        end; then
104       reconsider f as PartFunc of A*,A by RELSET_1:11,A2;
105       A3: f is homogeneous
106          proof
107           let x,y be FinSequence of A; assume
108           x ∈ dom f & y ∈ dom f; then
109           x = <Ø>A & y = <Ø>A by TARSKI:def 1,A1;
```

```
        hence thesis;
       end;
     A4: f is quasi_total
        proof
           let x,y be FinSequence of A; assume
           A5: len x = len y & x ∈ dom f; then
           x = <∅>A by TARSKI:def 1,A1; then
           len x = 0 by FINSEQ_1:32; then
           y = <∅>A by FINSEQ_1:32,A5;
           hence thesis by A1,TARSKI:def 1;
        end;
     thus thesis by A3,A4,A1,Th1;
  end;

theorem Th3:
 for A being non empty set, a being Element of A
  holds {<∅>A} -->a is Element of PFuncs(A*,A)
  proof let A be non empty set, a be Element of A;
      set f = {<∅>A} -->a;
      A1: dom f = {<∅>A} & rng f = {a} by FUNCOP_1:14;
      A2: dom f c= A*
        proof
          let z; assume z ∈ dom f; then
          z = <∅>A by TARSKI:def 1,A1;
          hence thesis by FINSEQ_1:65;
       end;
     rng f c= A
       proof
         let z; assume z ∈ rng f; then
         z = a by A1,TARSKI:def 1;
         hence thesis;
      end; then
    reconsider f as PartFunc of A*,A by RELSET_1:11,A2;
    f ∈ PFuncs(A*,A) by PARTFUN1:119;
    hence {<∅>A} -->a is Element of PFuncs(A*,A);
 end;
```

```
147 definition let A;
148  mode PFuncFinSequence of A -> FinSequence of PFuncs(A*,A) means
149 :Def3: not contradiction;
150  existence;
```

モード定義（mode definition）の正当性条件は定義された型が空でない（non-emptiness）ことで、それは **'existence'** ステートメントになります。このケースではステートメントは：

```
ex c being FinSequence of PFuncs(A*,A) st not contradiction
```

になります。

Mizar の型は全て空では無い（non-empty）ので、適切な型の **'c'** の存在は明白で、もちろんそれは **'not contradiction'** を満たします。　そこでこのステートメントは justification を必要としません。

```
151 end;

153  struct (1-sorted) UAStr ≪ carrier -> set,
154       charact -> PFuncFinSequence of the carrier≫;

156  definition
157  cluster non empty strict UAStr;
158  existence
159  proof consider D being non empty set, c being PFuncFinSequence of D;
160    take UAStr ≪D,c ≫;
161    thus the carrier of UAStr ≪D,c≫ is non empty;
162    thus thesis;
163  end;
164 end;
166 definition let D be non empty set, c be PFuncFinSequence of D;
167  cluster UAStr ≪D,c≫ -> non empty;
168  coherence
```

この種のクラスタの正当性条件は **'coherence'** と呼ばれます。
ここで、それはもちろん：

```
UAStr ≪D,c≫ is non empty
```

です。　15行目の **'definitions STRUCT_0;'** 指令により、これは：

```
the carrier of UAStr≪D,c≫ is non empty
```

に拡張され、それは166行の **'D'**の型から得られる：

```
D is non empty
```

に縮小されます。

```
169  proof
170    thus the carrier of UAStr≪D,c≫ is non empty;
171  end;
172 end;

174 definition let A;
175  let IT be PFuncFinSequence of A;
176 attr IT is homogeneous means :Def4:
177    for n,h st n ∈ dom IT & h = IT.n holds h is homogeneous;
178 end;

180 definition let A;
181  let IT be PFuncFinSequence of A;
182 attr IT is quasi_total means :Def5:
183    for n,h st n ∈ dom IT & h = IT.n holds h is quasi_total;
184 end;

186 definition let F be Function;
187 redefine attr F is non-empty means :Def6:
188  for n being set st n                    ∈ dom F holds F.n is non
empty;
189  compatibility
```

'pred' あるいは **'attr'** の再定義に対する正当性条件は **'compatibility'** と呼ばれます。　このケースでは、それは：

```
F is non empty iff
   for n being set st n ∈ dom F holds F.n is non empty
```

となります。　このステートメントとその証明で **'non empty'** は依然として'古い'定義：

```
definition let F be Function;
 attr F is non-empty means
```

```
:: ZF_REFLE:def 4
    not Ø ∈ rng F;
end;
```

を保持しています。

```
proof
  hereby assume F is non-empty; then
```

‘hereby’ というキーワードは **‘thus’** と **‘now’** の組み合わせのように振舞います（この構文は 26 ページの appendix B にある公式 Mizar 文法にはありません）。

```
A1:     not Ø ∈ rng F by ZF_REFLE:def 4;
     let i be set;
     assume i ∈ dom F;
     hence F.i is non empty by A1,FUNCT_1:11;
   end;
   assume
A2:     for n being set st n ∈ dom F holds F.n is non empty;
   assume Ø ∈ rng F;
     then ex i being set st i ∈ dom F & F.i = Ø; by FUNCT_1:11;
   hence contradiction by A2;
  end;
end;

definition let A be non empty set; let x be Element of PFuncs(A*,A);
  redefine
    func <*x*> -> PFuncFinSequence of A;
   coherence
```

‘func’ の再定義に対する‘正当性条件（**correctness** conditions）’は、型が変化した場合の **’coherence’** と、‘定義’が変化した場合の **‘compatibility’** です。　ここでは最初の場合だけで、条件はもちろん：

`<*x*> is PFuncFinSequence`　です。

```
    proof
     <*x*> is FinSequence of PFuncs(A*,A);
    hence thesis by Def3;
```

```
  end;
end;
definition let A be non empty set;
cluster homogeneous quasi_total non-empty PFuncFinSequence of A;
 existence
   proof
     consider a being Element of A;
     reconsider f = {<Ø>A} -->a as PartFunc of A*,A by Th2;
     reconsider f as Element of PFuncs(A*,A) by PARTFUN1:119;
     take <*f*>;
     thus <*f*> is homogeneous
       proof
        let n; let h be PartFunc of A*,A; assume
        A1: n ∈ dom <*f*> & h =<*f*>.n;
        then n ∈ {1} by FINSEQ_1:4,FINSEQ_1:def 8; then
         h = <*f*>.1 by A1,TARSKI:def 1;
        then h = f & f is homogeneous PartFunc of A*,A by Th2,FINSEQ_1:def 8;
        hence thesis;
       end;
     thus <*f*> is quasi_total
       proof
         let n; let h be PartFunc of A*,A; assume
           A2: n ∈ dom <*f*> & h =<*f*>.n;
           then n ∈ {1} by FINSEQ_1:4,FINSEQ_1:def 8; then
             h = <*f*>.1 by A2,TARSKI:def 1;
           then h = f & f is quasi_total PartFunc of A*,A by Th2,FINSEQ_1:def 8;
          hence thesis;
        end;
      thus <*f*> is non-empty
        proof
          let n be set; assume
          A3: n ∈ dom <*f*>;
          then reconsider n as Nat;
          n ∈ {1} by FINSEQ_1:4,A3,FINSEQ_1:def 8; then
```

```
         n = 1 by TARSKI:def 1; then
       <*f*>.n=f by FINSEQ_1:def 8;
         hence thesis by Th2;
     end;
   end;
 end;

reserve U for UAStr;

definition let IT be UAStr;
attr IT is partial means :Def7:
 the charact of IT is homogeneous;
attr IT is quasi_total means :Def8:
 the charact of IT is quasi_total;
attr IT is non-empty means :Def9:
 the charact of IT <> <Ø> & the charact of IT is non-empty;
end;

reserve A for non empty set,
          h for PartFunc of A*,A ,
          x,y for FinSequence of A,
          a for Element of A;

theorem Th4:
for x be Element of PFuncs(A*,A) st x = {<Ø>A} --> a holds
   <*x*> is homogeneous quasi_total non-empty
  proof let x be Element of PFuncs(A*,A) such that
   A1: x = {<Ø>A} --> a;
```

'let ... such that' 構文はステートメントの中の **'for ... st'** に対応します、それで **'assume'** が後に続く **'let'** と同等になります。

```
reconsider f=x as PartFunc of A*,A by PARTFUN1:121;
A2: for n,h st n ∈ dom <*x*> & h = <*x*>.n holds h is homogeneous
 proof let n,h; assume
   A3: n ∈ dom <*x*> & h =<*x*>.n;
   then n ∈ {1} by FINSEQ_1:4,FINSEQ_1:def 8; then
   h = <*x*>.1 by A3,TARSKI:def 1;
   then h = x & f is homogeneous PartFunc of A*,A by Th2,A1,FINSEQ_1:def
```

```
    8;
   hence thesis;
  end;
  A4: for n,h st n ∈ dom <*x*> & h = <*x*>.n holds h is quasi_total
   proof let n,h; assume
     A5: n ∈ dom <*x*> & h =<*x*>.n;
     then n ∈ {1} by FINSEQ_1:4,FINSEQ_1:def 8; then
       h = <*x*>.1 by A5,TARSKI:def 1;
     then h = x & f is quasi_total PartFunc of A*,A by
    Th2,A1,FINSEQ_1:def 8;
   hence thesis;
  end;
  for n being set st n ∈ dom <*x*> holds <*x*>.n is non empty
   proof let n be set; assume
     n ∈ dom <*x*>;
     then n ∈ {1} by FINSEQ_1:4,FINSEQ_1:def 8; then
       <*x*>.n = <*x*>.1 by TARSKI:def 1;
     then <*x*>.n = x & f is non empty PartFunc of A*,A by Th2,A1
  ,FINSEQ_1:def 8;
     hence thesis;
   end;
  hence thesis by A2,A4,Def6,Def5,Def4;
end;

definition
  cluster quasi_total partial non-empty strict non empty UAStr;
  existence
     proof
       consider A be non empty set;
       consider a be Element of A;
       set f = {<∅>A} --> a;
       reconsider w = f as Element of PFuncs(A*,A) by Th3;
       set U = UAStr ≪ A, <*w*> ≫;
       take U;
       A1: the charact of(U) is quasi_total &
         the charact of(U) is homogeneous & the charact of(U) is
          non-empty
```

```
      by Th4;
    the charact of(U) <> <∅>
      proof assume A2: the charact of(U) = <∅>;
        A3: len(the charact of(U)) = 1 by FINSEQ_1:56;
        len (<∅>) = 0 by FINSEQ_1:25;
        hence contradiction by A3,A2;
      end;
      hence thesis by A1,Def9,Def8,Def7;
    end;
  end;

definition let U be partial UAStr;
  cluster the charact of U -> homogeneous;
  coherence by Def7;
end;

definition let U be quasi_total UAStr;
  cluster the charact of U -> quasi_total;
  coherence by Def8;
end;

definition let U be non-empty UAStr;
  cluster the charact of U -> non-empty non empty;
  coherence by Def9;
end;

definition
mode Universal_Algebra is quasi_total partial non-empty non empty UAStr;
end;

reserve U for partial non-empty non empty UAStr;

definition
  let A;
  let f be homogeneous non empty PartFunc of A*,A;
func arity(f) -> Nat means
  x ∈ dom f implies it = len x;
  existence
```

func の定義の正当性条件は '**existence**' と '**uniqueness**' の部分か

らなります。　このケースでは **‘existence’** 条件は：

```
ex n st
    for x st x ∈ dom f holds n = len x
```

となります。

```
proof
  ex n st for x st x ∈ dom f holds n = len x
    proof
A1:      dom f <> Ø by Th1;
        consider x being Element of dom f;
        dom f c= A* by RELSET_1:12; then
        x ∈ A* by A1,TARSKI:def 3; then
        reconsider x as FinSequence of A by FINSEQ_1:65;
        take n = len x;
        let y; assume y ∈ dom f;
        hence n = len y by Def1;
      end;
    hence thesis;
end;
uniqueness
```

‘uniqueness’ 条件は：

```
for n,m st
  (for x st x ∈ dom f holds n = len x) &
  (for x st x ∈ dom f holds m = len x) holds
  n = m
```

です。

```
proof
  let n,m such that A2: (for x st x ∈ dom f holds n = len x) &
             for x st x ∈ dom f holds m = len x;
A3:   dom f <> Ø by Th1;
      consider x being Element of dom f;
      dom f c= A* by RELSET_1:12; then
      x ∈ A* by A3,TARSKI:def 3; then
      reconsider x as FinSequence of A by FINSEQ_1:65;
      n = len x & m = len x by A3,A2;
```

```
378        hence thesis;
379      end;
380  end;

382 theorem Th5:
383 for U holds for n st n ∈ dom the charact of(U) holds
384    (the charact of(U)).n is
385    PartFunc of (the carrier of U)*,the carrier of U
386  proof let U,n;
387    set pu = PFuncs((the carrier of U)*, the carrier of U),
388        o = the charact of(U); assume
389      n ∈ dom o; then
390    o.n ∈ rng o & rng o c= pu by FUNCT_1:12,FINSEQ_1:def 4;
391  hence thesis by PARTFUN1:121;
392    end;

394 definition let U;
395 func signature(U) ->FinSequence of NAT means
396    len it = len the charact of(U) &
397    for n st n ∈ dom it holds
398      for h be homogeneous non empty
399      PartFunc of (the carrier of U )*,the carrier of U
400        st h = (the charact of(U)).n
401      holds it.n = arity(h);
402    existence
403  proof
404    defpred P[Nat,set] means
405    for h be homogeneous non empty
406      PartFunc of (the carrier of U )*,the carrier of U
407        st h = (the charact of(U)).$1
408          holds $2 = arity(h);
```

'defpred' は'ローカルな'述語を定義します:それは **'set'** に似ていますが、表現ではなく述語を定義します。 それはいろいろなところで拡張されます(これは415と419行で使用されています)。 引数は **'$1'**, **'$2'**, **. . .** という風に参照される **'body'** の中にあります。 許されるもっとも高いインデックスは **'$8'** なので **'deffunc'** あるいは **'defpred'** は最大で8個の引数をとります。

```
A1:  now let m; assume
       m∈ Seg len the charact of(U); then
       m∈ dom the charact of(U) by FINSEQ_1:def 3; then
       reconsider H=(the charact of(U)).m as homogeneous non empty
         PartFunc of (the carrier of U )*,the carrier of U by
          Th5,Def4,Def6;
       take n=arity(H);
       thus P[m,n];
     end;
    consider p be FinSequence of NAT such that
    A2: dom p = Seg(len the charact of(U)) and
    A3: for m st m ∈ Seg(len the charact of(U)) holds P[m,p.m] from
   SeqDEx(A1);
```

article **‘MATRIX_2’** からのスキーム **‘SeqDEx’** は：

```
scheme SeqDEx{D()->non empty set,A()->Nat,P[set,set]}:
   ex p being FinSequence of D() st dom p = Seg A() &
     for k st k ∈ Seg A() holds P[k,p.k]
provided
   for k st k ∈ Seg A() ex x being Element of D() st P[k,x];
```

と定義されています。（スキームへの参照は番号によってではなく、名前によって行われます:Mizar ライブラリは 546 のスキームを持ち､それは article の数よりも少数です。　ここではそれは：

```
  ex p be FinSequence of NAT st
   dom p = Seg(len the charact of(U)) &
     for m st m ∈ Seg(len the charact of(U)) holds P[m,p.m]
```

を：

```
A1: for m st m∈Seg len the charact of(U) holds ex n st P[m,n]
```

から派生する（derive）のに使われます（これは **‘consider’** ステートメントのために必要です）。　そこで、このケースではスキームのインスタンシエーション（実例化）は：

```
    D() → NAT
    A() → len the charact of U
 P[k,x] → P[k,x]
```

となります。　この代入演算は明示的には与えられていません、しかし、スキームの中で‘引数’‘A1’と‘provided’のあとの‘条件（condition）’、そしてスキームのなかの‘結論’を持つ証明すべきステートメント、をマッチング（照らし合わせて適合）させることで見つけられます

```
   take p;
 Seg len the charact of(U) = dom the charact of(U) by FINSEQ_1: def 3;
   hence A4: len p = len the charact of(U) by A2,FINSEQ_3:31;
   let n; assume
   n ∈ dom p; then
   A5: n ∈ Seg(len the charact of(U)) by FINSEQ_1:def 3,A4;
   let h be homogeneous non empty
         PartFunc of (the carrier of U )*,the carrier of U; assume
   h = (the charact of U).n;
   hence p.n = arity(h) by A5,A3;
  end;
 uniqueness
  proof
     let x,y be FinSequence of NAT; assume that
     A6: len x = len the charact of(U) and
     A7: for n st n ∈ dom x holds for h be homogeneous non empty
       PartFunc of (the carrier of U )*,the carrier of U
       st h = (the charact of(U)).n
       holds x.n = arity(h) and
     A8: len y = len the charact of(U) and
     A9: for n st n ∈ dom y holds for h be homogeneous non empty
       PartFunc of (the carrier of U )*,the carrier of U
         st h = (the charact of(U)).n
         holds y.n = arity(h);
       now let m; assume
        1≤m & m≤len x; then
        m ∈ Seg len x by FINSEQ_1:3; then
     m ∈ dom x by FINSEQ_1:def 3;
       then A10: m ∈ dom the charact of(U) & m∈dom x & m∈dom y
                                  by A6,A8,FINSEQ_3:31;
       then reconsider h=(the charact of(U)).m
         as homogeneous non empty
```

```
          PartFunc of (the carrier of U )*,the carrier of U by
          Th5,Def4,Def6;
      x.m=arity(h) & y.m=arity(h) by A7,A9,A10;
      hence x.m=y.m;
    end;
  hence thesis by A6,A8,FINSEQ_1:18;
 end;
end;
```