数理解析研究所講究録
第811巻 1992年 188-193

代数知識のデータベース化の試み

電子技術総合研究所 **元吉文男** (Fumio Motoyoshi)
電子技術総合研究所 **秋葉澄孝** (Sumitaka Akiba)

1 はじめに

これまでに，数学公式集をデータベース化しようとする試みはあったが，定義や定理などについてはまだ行なわれていない．一方，定義や定義間の関係などを知りたいときに電子化されたデータベースがあれば計算機の能力を生かして様々な検索を行なうことができると考えられる．そこで，最初の試みとして代数知識をデータベース化しようと実験を行なったので報告する．

データベース化といっても，辞書のように文章で(自然言語で)記述するのでは，(すでに電子化は行なわれており)特に「数学知識」に関して実現する必要もない．計算機を利用できるのであるから，自然言語のように平坦ではなく，何らかの構造を持った内部表現で実現した方が，後での利用に有利である．

ここでは，記述に一階述語論理を使用することにした．ただし，単純な一階述語では記述力が弱いので記述に便利なようにsyntax sugar を導入して，テキストとして読んだ場合にも意味を理解し易いようにした．また，一階述語論理で記述してあるので，定理の証明などに従来の証明法を使用することが可能であるので，データベース上の知識を用いた推論などへの応用が考えられる．

2 記述言語

代数知識の記述は，一階述語論理で集合論の用語を用いて行なうことにした．これは，

1. なるべく少ない基本述語で記述する，
2. 高階の論理を使用しない

ように方針を決めて，いろいろ試みた結果である．

2.1 無定義で使用する記号

基本述語は以下の 2 つとする.

- $=$　　等号
- $\in$　　メンバーシップ

両方ともに構文的にはインフィクスで使用し，等号は集合の要素の同一性を決定する述語として，メンバーシップは集合に要素が含まれているかどうかを判定する述語として使用する.

基本的な関数記号としては，集合の直積を表すためと，直積集合の要素を表すために以下の記号を用いる.

- $\times$　　直積
- $(\ ,\ )$　　tuple

2.2 Syntax sugar

基本述語と基本関数記号だけでは，記述にも不便であるし，読みにくい．そこで以下で述べる syntax sugar を使用した.

型付き限量子

限量子を使用するときに，そこでの変数の値域が (ある集合の要素であるというように) 限定されていることが多いのでそのために次のような記法を使用する.

$$\exists x \in S.P(x) \quad \Leftrightarrow \quad \exists x.(x \in S \wedge P(x))$$
$$\forall x \in S.P(x) \quad \Leftrightarrow \quad \forall x.(x \in S \rightarrow P(x))$$

ここで，P は任意の論理式である．この記法は限量子の自然な拡張であり，通常の意味とも一致している.

また，「一つだけ存在する」という言明はよく現れるのでそのための記法を導入し，次のような記述の短縮形であると定義する.

$$\exists! x.P(x) \quad \Leftrightarrow \quad \exists x.\Big(P(x) \wedge \forall x, y.(P(x) \wedge P(y) \rightarrow x = y)\Big)$$

関数記号

関数は直積集合の部分集合として表現するが，通常使用する関数表記の方が便利であるので，関数表記された記述は syntax sugar であるとして，それを一階述語論理での記述に変換するための規則を以下のように定義する．

$$P(f(x,y)) \Leftrightarrow \exists z.((x,y,z) \in f \wedge P(z))$$

syntax sugar 上での関数記号は，一階述語表現では単に集合を示すための記号として扱われる．(f が算法であるかどうか (x, y から z がユニークに決まるかどうか) は別に定義する．) また，＋や＊などの中置関数記号も，関数記号と同じく，以下のように直積集合での簡易表現とする．

$$P(x+y) \Leftrightarrow \exists z.((x,y,z) \in + \wedge P(z))$$

3 代数知識の記述

前節での記法を使用して代数知識の最初の部分を記述してみたものを以下に示す．

```
empty(S) ⇔ ¬∃ x.x ∈ S ;
composition(+,S) ⇔ + ⊂ S × S × S ∧ ∀ x,y ∈ S. ∃ !z ∈ S.(x,y,z) ∈ + ;
AlgebraicSystem(S,+) ⇔ ¬ empty(S) ∧ composition(+,S) ;
associative(+,S) ⇔ ∀ a,b,c ∈ S. (a+b)+c = a+(b+c) ;
Semigroup(S,+) ⇔ AlgebraicSystem(S,+) ∧ associative(+,S) ;
unit(e,+,S) ⇔ e ∈ S ∧ ∀ a ∈ S.( e+a = a ∧ a+e = a ) ;
Monoid(S,+) ⇔
  UnitarySemigroup(S,+) ⇔ Semigroup(S,+) ∧ ∃ e.unit(e,+,S) ;
leftInverse(x,a,+,S) ⇔ x ∈ S ∧ a ∈ S ∧ unit(x+a,+,S) ;
rightInverse(x,a,+,S) ⇔ x ∈ S ∧ a ∈ S ∧ unit(a+x,+,S) ;
invertible(a,+,S) ⇔
  ∃ x.leftInverse(x,a,+,S) ∧ ∃ x.rightInverse(x,a,+,S));
inverse(x,a,+,S) ⇔ x ∈ S ∧ a ∈ S ∧ unit(a+x,+,S) ∧ unit(x+a,+,S) ;
Group(S,+) ⇔ Monoid(S,+) ∧ ∀ a ∈ S.invertible(a,+,S) ;
commutative(+,S) ⇔ ∀ a,b ∈ S. a+b = b+a ;
CommutativeGroup(S,+) ⇔ Group(S,+) ∧ commutative(+,S) ;
distributive(+,*,S) ⇔ ∀ a,b,c ∈ S.(a*(b+c)=a*b+a*c ∧ (b+c)*a=b*a+c*a) ;
```

```
Ring(S,+,*) ⇔
  CommutativeGroup(S,+) ∧ Semigroup(S,*) ∧ distributive(+,*,S) ;
UnitaryRing(S,+,*) ⇔ Ring(S,+,*) ∧ ∃ e.unit(e,*,S) ;
ZeroRing(S,+,*) ⇔ Ring(S,+,*) ∧ #S=1 ;
CommutativeRing(S,+,*) ⇔ Ring(S,+,*) ∧ commutative(*,S) ;
CommutativeUnitaryRing(S,+,*) ⇔ UnitaryRing(S,+,*) ∧ commutative(*,S) ;
zeroDivisor(a,+,*,S) ⇔ ¬ unit(a,+,S) ∧ ∃ b.(¬ unit(b,+,S) ∧
(unit(a+b,+,S) ∨ unit(b+a,+,S))) ;
IntegralDomain(S,+,*) ⇔
  CommutativeUnitaryRing(S,+,*) ∧ #S>1 ∧ ∀ a. ¬ zeroDivisor(a,+,*,S) ;
SkewField(S,+,*) ⇔
  UnitaryRing(S,+,*) ∧ #S>1 ∧
  ∀ a ∈ S.(¬ unit(a,+,S) → invertible(a,*,S) ) ;
Field(S,+,*) ⇔ SkewField(S,+,*) ∧ commutative(*,S) ;
```

4 定理の手動確認

以上で記述された述語が妥当なものであることを確かめるために，あるいは，自動の定理証明の参考とするために，定理を上の定義を使用して手動で証明してみた．なお，証明には融合原理を用いることにし，そのためのツールを作成して，紙の上ではなく計算機上で証明した．

例として次の式を証明する．

```
∀ s,+.(AlgebraicSystem(s,+) →
        ∀ x,y ∈ s.(∀ a ∈ s.a+x=a ∧ ∀ a ∈ s.y+a=a → x=y)) ;
```

この式の否定をとって，冠頭標準形にする．このとき skolem 関数を大文字で書く (+ も 0 引数の skolem 関数になる).

```
∀ u,v,s1,s2,a1,a2.
  (Z ∈ S ∧ + ⊂ S × S × S ∧
   (¬ u ∈ S ∨ ¬ v ∈ S ∨ W(u,v) ∈ S) ∧
   (¬ u ∈ S ∨ ¬ v ∈ S ∨ (u,v,W(u,v)) ∈ +) ∧
   (¬ u ∈ S ∨ ¬ v ∈ S ∨ ¬ (u,v,s1) ∈ + ∨ ¬ (u,v,s2) ∈ + ∨ s1=s2 )
```

X ∈ S ∧ Y ∈ S ∧ (¬ a1 ∈ S ∨ (a1,X,a1) ∈ +) ∧
(¬ a2 ∈ S ∨ (Y,a2,a2) ∈ +) ∧¬ X=Y)

これを節ごとに書くと

1) Z ∈ S
2) + ⊂ S × S × S
3) ¬ u ∈ S ∨ ¬ v ∈ S ∨ W(u,v) ∈ S
4) ¬ u ∈ S ∨ ¬ v ∈ S ∨ (u,v,W(u,v)) ∈ +
5) ¬ u ∈ S ∨ ¬ v ∈ S ∨ ¬ (u,v,s1) ∈ + ∨ ¬ (u,v,s2) ∈ + ∨ s1=s2
6) X ∈ S
7) Y ∈ S
8) ¬ a1 ∈ S ∨ (a1,X,a1) ∈ +
9) ¬ a2 ∈ S ∨ (Y,a2,a2) ∈ +
10) ¬ X=Y

となり，これに対して融合を行なう．

6: 9=11) (Y,X,Y) ∈ +
7:10=12) (Y,X,X) ∈ +
5: 7=13) ¬ v ∈ S ∨ ¬ (Y,v,s1) ∈ + ∨ ¬ (Y,v,s2) ∈ + ∨ s1=s2
6:13=14) ¬ (Y,X,s1) ∈ + ∨ ¬ (Y,X,s2) ∈ + ∨ s1=s2
12:14=15) ¬ (Y,X,s2) ∈ + ∨ X=s2
11:15=16) X=Y
10:16=17)

となり 7 ステップで証明できる．

また，次の証明には約 90 ステップかかった．

$$\forall s,+.(Composition(+,a) \wedge associative(+,s) \wedge \exists e.unit(e,s,+) \wedge$$
$$\forall a \in s.\exists x.leftInverse(x,a,+,s)$$
$$\rightarrow \forall a,b,c \in s.(a+b=a+c \rightarrow b=c))$$

以上のことから，これまでの定義には大きな誤りがないものと思われる．

以下に述べるように，定義は，証明すべきものとは独立に前以って処理しておくことができる．すなわち，一般に定理は以下の形をしていると考えられる．

$$\Gamma \rightarrow (P \rightarrow Q)$$

ここで，Q は結論を，P は前提を，Γ は定義の集まりを表しているとする．融合原理で証明するために，この式の否定をとって変形すると

$$\begin{aligned} & \neg(\Gamma \to (P \to Q)) \\ \Leftrightarrow\ & \neg(\neg\Gamma \vee (P \to Q)) \\ \Leftrightarrow\ & \Gamma \wedge \neg(P \to Q) \\ \Leftrightarrow\ & \Gamma \wedge \neg(\neg P \vee Q) \\ \Leftrightarrow\ & \Gamma \wedge P \wedge \neg Q \end{aligned}$$

ここで Γ の部分はすべての定義の連言 (and) であり，それぞれには全称限量子が付いているものと解釈するが，この部分を前以って冠頭標準形にしておくことにより，各定理での処理を省略できる．

5 今後の予定 (希望)

今後の予定としては以下のことが考えられる．

- 知識の充実
  代数の知識をさらに述語化してデータベース化する．さらには 検索の手段も追加して「電子化数学辞典」としての機能を持たせる，
- 定理の自動確認
  定理の証明を手動で行なうのではなく，自動で，あるいはそれが無理であれば サブゴールの証明を自動で行なう程度のプログラムを作成する，
- 定理の抽出
  証明過程を分析して，共通して使用されるパターンを あらたな定理として自動的に抽出する，
- 定理の利用
  抽出された，あるいは与えられた定理を利用して自動証明の効率化をはかる，
- コンサルタントシステム
  定義間の関係などの質問について応答できる代数コンサルタントシステム を作成する