'ライプニッツの夢'について

上村義明　京都産業大学 理学部

Leibnitz は普遍記号学を夢想して日常生活での推論までもこれを 用いて処理される様な未来を期待したと言はれる。
将来computer の普及が夢想を現実にするかも知れない。 それはそれとして、bool 値の第一市民権がなかなか認められないのは、如何したことか? 先ずは、この辺りが改善を要する。ところで、 プログラム意味論研究のマイナー なグループのなかで、ある必要から'Leibnitzの夢'を実現させる羽目になった人達が出てきた。　　　いきさつは割愛して本論に入ろう。

定理をboolean expression と見做せば bool 値を保存する変換を次々に施してその値が trueになることを示せばそれは証明といえる。
そこで 1) 証明したいことを boolean expressionの形で捉えて
　　　　2)この方式ですべての証明をかくことにしよう。
1) については" 等式"をboolean expressionと見做す工夫がいる。a = b においてa がスカラーなら問題はない。
a がベクトルや場であるときはcolumwiseとかpointwiseとかのandを取つてbool値を計算する事にする。これはeverywhere operatorとよばれ[ ]によって明示される。　[a = b]
2)について次のような工夫がなされる。

[a] ≡ [b]
[b] ≡ [c]
[c] ≡ true
　より [a] ≡ true が推論されるのは ≡ の推移律によるが
表現方式は次のようにする。

[a]
=　{ hint why [a] ≡ [b] }
[b]
=　{hint why　[b]　≡ [c]
[c]
=　{hint why　[c] ≡ true }
true

今 a が a = d の形をしている時は
[a = d]
= { hint why [a = b] }
[b = d]
= { hint why [b = c] }
[c = d]
= { hint why [c = d] }
true

が推論可能である。

これは
狭義の Frege's principle
expression が subexpression の関数である事　と
Leibnitz's Rule
[a = b] -> [f.a = f.b]　(f は関数記号)
より導かれる。
それ故 表現方式は更にくだけて、これを
a
= {hint why [a = b] }
b
= {hint why [b = c] }
c
= { hint why [c = d] }
d

と書いてもよい事にする。
即ち[a = b] &[b = c]& [c = d] が成り立てば [a = d] が結論されるのであるから equality を中心に両辺を[と]で括ったつもりで読めばよい。
もはや、等価変換などではないがそれは見てくれの事で裏は取れているのである。

[A ≡ D]を証明したい場合には、時として

A
= { hint }
......
= { hint }
A v D
= { hint }
......
= { hint }
D

となるが、勿論 [A ≡ A v D] & [A v D ≡ D] のお蔭である。
当然A v D のところがA&Dとなる場合もある。

再び Leibniz's Rule　により

[X ≡ Y] & [Y -> Z] -> [X -> Z]
[X -> Y] & [Y ≡ Z] -> [X -> Z]

含意の推移律から

[X -> Y] & [Y -> Z] -> [X -> Z]

よって[A -> D] の証明は

A
-> { hint why [A -> B] }
B
= { hint why [B ≡ C] }
C
-> { hint why [C -> D] }
D

と書くことにする。[A -> B] & [B ≡ C] & [C -> D] が裏にあるのは言うまでもない。

-> imply が 許さるなら <- follow from も許される。

たとえば

```
D
<-  { hint why [ D <- C] }
C
=   { hint why [ C ≡ B ] }
B
<-  { hint why [ B <- A ] }
A
```

のような形も許そう。また [E] -> [A ≡ D]を証明するときは

```
A
=   { hint why  [ A ≡ B ] }
B
=   { hint why [ E ] -> [ B ≡ C ] }
C
=   { hint why [ C ≡ D ] }
D
```

のように条件が用いられる所を明示するのがよい。
これは [[E] & A ≡ [E] & D] と同じだから前に述べたcaseに当てはまる。
以上のような表現形式は極めてありふれた発想ではあるが式の変形をちゃんと証明として裏付けて大袈裟にいえば計算を証明として再生させている。 あるいは証明のcaluculus化といえる。
そこでhigh shool やcollegeの教科書などをこの流儀で書き換えて面目を一新しようというあたりが如何にもLeibniz's Dreamの雰囲気を醸す所以である。

彼等があげている例題を引用しておこう。

定理　conjunctive な predicate transformer は 単調である。

ここに　1) boolean structure から boolean structure への関数は predicate transformer とよばれる。

2) (f is conjunctive) ≡ (A x,y :: [f.(x & y) ≡ f.x & f.y])

3) (f is monotonic) ≡ (A P,Q : : [P -> Q] -> [f. P -> f. Q])

Proof

[f.P -> f.Q]

= { predicate calculus }

[f.P & f.Q ≡ f.P]

= { f is conjuctive }

[f. (P & Q) ≡ f. P]

<- { Leibniz }

[P & Q ≡ P]

= { predicate calculus }

[P -> Q]　　(End of Proof)

話が equivalence を 中心にしているので ≡ の性質をしらべておかねばならない。
equivalence は equalty の性質とoperator の性質を兼ね備えている。
従来、後者が軽視され過ぎている。先ずはoperatorとして associative 且つ symmetric である事を強調する。
即ち

$$[((X \equiv Y) \equiv Z) \equiv (X \equiv (Y \equiv Z))]$$
$$[(X \equiv Y) \equiv (Y \equiv X)]$$

これらは公理である。だから括弧はいらないし、$[X \equiv (Y \equiv Y \equiv X)]$ から $[Y \equiv Y] \equiv$ true が単位元である事もすぐわかる。

$$[X \equiv \text{true} \equiv X]$$

分りきった事といはれるかも知れないが X は スカラーとは限らないから自明と片付ける訳にも行くまい。
equivalence を軸にすべてが展開される。このあたりが普通の論理学の教科書とは趣を異にしている。
disjunction は
associative 且つ symmetric,idempotent と distribute over equivalence で characterize される。
conjuction は

$$[X \,\&\, Y \equiv X \equiv Y \equiv X \vee Y] \qquad (\text{Golden Rule})$$

によって導入される。
imply は

$$[X \rightarrow Y \equiv X \vee Y \equiv Y]$$

によって
negation は

$$[\neg(X \equiv Y) \equiv \neg X \equiv Y]$$

によって導入される。

勿論 排中律 は公理である。

$$[X \vee \neg X]$$

古典論理の諸定理は以上の公理達から容易に導かれる。

例えば次のように $[(X \rightarrow Y) \vee (Y \rightarrow Z)]$ が証明される。

$[(X \rightarrow Y) \vee (Y \rightarrow Z)]$
= { relation between -> and V }
$[(X \vee Y \equiv Y) \vee (Y \vee Z \equiv Z)]$
= { v distributes over ≡ }
$[X \vee Y \ \vee Y \vee Z \equiv X \vee Y \vee Z \equiv Y \vee Y \vee Z \equiv Y \vee Z]$
= { idempotence of v; identity element of ≡ }
true