4K-4

Lazy Reduction による HA の Normalization

萩谷昌己
(東大・理・情報科学)

プログラム合成の理論的基礎に、theoryに関する次の性質がある。

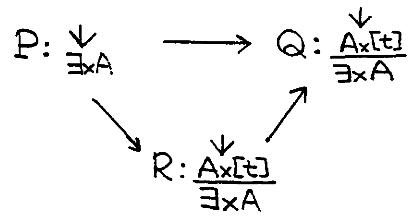
(EP) $\vdash \underset{(\text{closed})}{\exists x A} \Rightarrow (\exists t) \vdash A_x[t]$

(DP) $\vdash \underset{(\text{closed})}{A \vee B} \Rightarrow \vdash A$ or $\vdash B$

一般に、constructive(またはintuitionistic)なtheoryでは、EPやDPが成り立つ。たとえば、HA(Heyting Arithmetic)は、典型的な例だ。EPやDPをいうには、いくつかの方法があるが、Prawitzがやった、Natural Deduction(ND)のnormalizationによるのが、最も基本的で、実現も容易だ。

Jervellは、HAでは、任意のreductionの列は有限で、唯一のnormal formに至るということを証明した。これから、EPとDPは簡単に出る。しかし、我々は、normal formを必要としているわけではなく、EP, DPが目的だ。

いま、P⊢∃xAとしよう。P→Qで、PからQへのreductionの列があることを示す。さて、下図のように、P→Qとなる任意のQに対し、



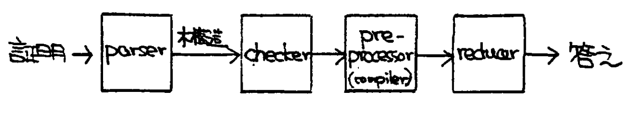
P→R→QとなるRが(Qに関係なく)存在するだろうか。(ただし、Q, Rは、上図のような形。)また、RをPから求めるreduction procedureは、存在するであろうか。

上の答えは、Yesである。求めるprocedureは、割りと簡単に構成することができた。これを、Lazy Evalとのanalogyから、Lazy Reductionと呼ぶことにした。このprocedureは、左のような意味で最も効率のよいものだ。

NDのnormalizationについては、実現も含めて、Goodがよく研究している。彼のreduction procedureは、call by valueとある。このためか、彼は公理をHarrop formulaに限定した。我々の場合、公理は何でもよい。reductionに行き詰まると、procedureはerrorを出すが、それはもともと、左図のようなQが存在しないということを意味する。

reducerを計算機上に実現する際に、最も問題になるのが、代入という操作だ。(alistなどにより)代入なしですませられれば、一番よいのだが、そのために、余分の計算がふえる恐れがある。ここでは、各nodeに、変数の依存情報(bit table)を与えて、代入を高速化する試みをした。このため、処理系はLISPで書かれていない。(Cで書いてある。)

処理系は、VAX/UNIXで動いている。全体の構成は、次のようだ。



GBCはまだない。全体の8割ぐらいを、parserとcheckerが占めている。

証明を記述するのに、新しく言語を設計した。(裏参照。)

(*)

D. Prawitz (1970), Ideas and results in proof theory,
H. Jervell (1970), A normalform in first order arithmetic, Proc. Second Scandinavian Logic Symposium, North-Holland, 1972
C. A. Good (1980), Computational Uses of the Manipulation of Formal Proofs, Stanford University

```
% cat eqdecide
{
  [Theorem]
        `A(x)A(y){x = y | ~ x = y}`
        IND(x)
        BASE
                `A(y){0 = y | ~ 0 = y}`
                IND(y)
                BASE {
                  [1]
                        `A(x) x = x` id;
                  [2]
                        | LEFT 1(0)
                } STEP() {
                  [1]
                        `~ 0 = y'` (`A(x) ~ 0 = x'` succ monotonic)(y);
                  [2]
                        | RIGHT 1
                }
        STEP(`A(y){x = y | ~ x = y}` IH)
                `A(y){x' = y | ~ x' = y}`
                IND(y)
                BASE {
                  [1]
                        `A(x)A(y){x = y -> y = x}` tr;
                  [2]
                        succ monotonic(x);
                  [3]
                        `~ x' = 0`
                        ASSUME(so) `FALSE` 2 ^ (1(x')(0) ^ so);
                  [4]
                        | RIGHT 3
                } STEP()
                        WHETHER
                                `x = y | ~ x = y` IH(y)
                        LEFT(a) {
                          [1]
                                `A(x)A(y){x = y -> x' = y'}` succ equality;
                          [2]
                                `x' = y'` 1(x)(y) ^ a;
                          [3]
                                | LEFT 2
                        } RIGHT(a) {
                          [1]
                                `A(x)A(y){x' = y' -> x = y}` succ injective;
                          [2]
                                `~ x' = y'`
                                ASSUME(so) `FALSE` a ^ (1(x)(y) ^ so);
                          [3]
                                | RIGHT 2
                        };
  [7 not eq 10]
        Theorem(7)(10)
}
% lr < eqdecide
parse end(459)
check end(659)
compile end(659)
reduction starts
main symbol is ORir
1821 words used
%
```