数理解析研究所講究録
第236巻 1975年 8-14

A NATURAL DEDUCTION SYSTEM FOR ASSERTIONS

by

Shigeru Igarashi (Kyoto U.)

ABSTRACT: First-order logic is extended so as to include formulas expressing the termination as well as the correctness of algorithmic statements. The intended formal system is a generalization of the natural deduction system NK postlated by Gentzen. The termination of recursive procedures is proved by mathematical induction and the correctness formulas introduced by Hoare are expressed and proved conveniently in our system, thereby describing the semantics of statements.

Let T be an axiomatized extension of Peano arithmetic which is denoted by N, and L(T) the first-order language of T.

The statements over L(T) are defined inductively as follows.

There are symbols called (m,n)-ary procedure symbols for each m and n. A (0,0)-ary procedure symbol is called a statement constant. Moreover there are symbols called (m,n)-ary constructors for each m and n where m or n is

Intended Interpretation of Assertions.

Let A(x,y), or A in short, be a statement whose left-occurring variables are x and other variables are y. We consider A realizes a partial transformation $\underline{A}$ of the assignments of natural numbers (as the universe of T) to the variables designated by x and y. For each such assignment i, i(a), i(b), ... shall denote the numerals (name of number) assigned to the variables a, b, ..., respectively. We may suppose that

a) for each i, ($\underline{A}$i) (a) is defined either for all a within x or y, or for no variable; and that

b) ($\underline{A}$i) (a)=i(a) for a within y, if $\underline{A}$i is defined.

We extend $\underline{A}$ for the assignments $\underline{i}$ involving variables other than x or y, by simply letting ($\underline{A}$i)(a)=$\underline{i}$(a) for such a variable. Whenever we evaluate the truth of a subexpression of an assertion we use the assignments to all the variables occurring in that assertion. The truth of af's is defined by induction as follows.

1. We know the truth of an instance of a formula F of L(T). Therefore we let F(a,b,...) be <u>true for the assignment i</u> if the instance F(i(a),i(b),...) of F is true; and let it be false otherwise.

2. We let $A \vdash F$ be true for i if $\underline{A}$i is defined and F is true for the assignment $\underline{A}$i; and be false otherwise.

3. The truth of $\sim A$ and $A \to B$ are defined simply by the truth table. Thus $\sim A$ is true for i if and only if A is false for i.

4. We let (a)A(a) be true for i if A(a) is true for every j such that i(b)=j(b) for each b except a; and be false otherwise.

An assertion formula A is <u>true</u> if A is true for every assignment i.

Thus $A \vdash 0=0$ means A terminates (for every initial assignment.) $A \vdash 0=1$ is a contradiction for any A. The correctness formula of the form

$F \{ A \} G$ [Hoare (1969). Igarashi, London, and Luckham (1973)]

not 0.

a) For an (m,n)-ary procedure symbol f and terms $t_1, \ldots, t_n$ (abbreviated by t with n understood) and arbitrary but distinct variables $x_1, \ldots, x_m$ (abbreviated by x with m understood),

$$f(x_1, \ldots, x_m ; t_1, \ldots, t_n), \text{ namely, } f(x;t)$$

is a statement.

b) For an (m,n)-ary constructor symbol K, quantifier-free formulas $F_1, \ldots, F_m$, and statements $A_1, \ldots, A_n$,

$$K(F_1, \ldots, F_m, A_1, \ldots, A_n)$$

is a statement.

Assertion formulas, or af's in short, are the formulas inductively defined as follows.

a) For a statement A and a formula F over L(T), $A \vdash F$ is an atomic assertion formula, or aaf in short. An aaf is an af.

b) If A is an af, then $\sim A$ is an af.

c) If A and B are af's, then $A \to B$ is an af.

d) If A is an af and x does not occur left*) in A, then $(x)A$ is an af.

Henceforce a formula shall mean only a formula over L(T). An assertion formula will be called an assertion on occasions.

---

*) A variable x_i is said to occur left in $f(x_1,\ldots,x_m;t)$, $1 \leq i \leq m$.
A variable x is said to occur left in a statement or an assertion if it occurs left in a statement of the form f(y;t), within that statement or assertion.

IV. Constructors. $\dfrac{A \vdash F \qquad [\,F\,]\ B \vdash G}{AB \vdash G}$

$\dfrac{F \qquad A \vdash G}{F\,|A,B| \vdash G}$ †† $\dfrac{\neg F \qquad B \vdash G}{F\,|A,B| \vdash G}$

V. Primitive Procedures. $x \leftarrow y \vdash x=y$. $\dfrac{t=u \qquad x \leftarrow t \vdash F}{x \leftarrow u \vdash F}$

$\Lambda \vdash 0=0$.

(A coded procedure is regarded as a primitive procedure, e.g.

$x=a\&y=b \rightarrow Swap(x,y;\ ) \vdash x=b\&y=a.$)

VI. Specification and Defined Procedures.

$\dfrac{t=u \qquad f(x;s,t) \vdash F}{f(x;s,u) \vdash F}$ **) $\dfrac{A(x,t) \vdash F}{f(x;t) \vdash F}$ ***)

EXAMPLE A. Mutually recursive procedures computing the greatest common divisor of two numbers.

Declaration: p(z;x,y) proc if x=y then z:=x else q(z;x,y).

q(z;x,y) proc if y<x then p(z;x-y,y) else q(z;y,x).

To prove: (x)(y)(p(z;x,y) $\vdash$ z=gcd(x,y)). (Size of proof = 225% of EXAMPLE C).

Induction hypothesis: (x)(y)(h(x,y)$\leq$n $\rightarrow$ p(z;x,y) $\vdash$ z=gcd(x,y)), where

h(x,y)=(x+y)/gcd(x,y); or=0,(x=y=0).

EXAMPLE B. Procedure translated from a while program computing n!

Declaration: f(x,y;a,b) proc if a<n then [x:=a+1;y:=bx;f(x,y;x,y)] else [x:=a;y:=b].

Specification: f(x,y;a,b) value a,b. (Size of proof = 125% of EXAMPLE C).

To prove: f(x,y;0,1) $\vdash$ y=n!

Key assertion: f(x,y;n-k,b) $\vdash$ x=n&y=b·n!/(n-k)!

---

††) $F\,|A,B|$ denotes if F then A else B in ALGOL60.

**) f is specified by f(w;y,z) value z.

***) f is declared by f(w;y) proc A(w,y).

is expressed in our language by

$$(F \to A \vdash 0=0) \to (F \to A \vdash G) \ ,$$

which describes that if A terminates for every initial assignment of values to variables satisfying the formula F then the resulting assignment (corresponding to such initial assignment) satisfies G.

Summary of Natural Deduction system for Assertions

The following inference rules are used for assertion formulas.

I. Logical Inference.

Propositional connectives. All the rules of inference in NK except the quantifier rules are used.

Quantifiers. A bound variable must not occur left within its scope, so that all the quantifier rules except the universal specification are used with this restriction.

Universal specification. $\dfrac{(x)A(x)}{A(t)}$

No variable in the term t occurs left in $A(t)$.

II. Equality and Induction. $\dfrac{t=u \quad A(t)}{A(u)}$

No variable in the terms t or u occurs left in $A(t)$, nor in $A(u)$.

$$\frac{A(0) \qquad \overset{[\,A(a)\,]}{A(a+1)}}{(x)A(x)}$$

a does not occur left in $A(a)$, besides a satisfies the condition of eigenvariable.

III. Formulas and Atomic Assertions. $\dfrac{A \vdash F \quad A \vdash G}{A \vdash F\&G}$

$$\frac{A \vdash F \qquad T \vdash F \to G}{A \vdash G} \qquad \frac{F \qquad T \vdash F \to G}{G}$$

$$\frac{A \vdash 0=0 \qquad F}{A \vdash F} \ *) \qquad \frac{A \vdash 0=1}{0=1}$$

† $T \vdash F$ denotes that F is a theorem in the first-order theory T.

*) No free variable in F occurs left in A.

EXAMPLE C. Recursive procedure computing n!

Declaration: f(y;x) proc if x=0 then y:=1 else [f(y;x-1);y:=xy].

To prove: f(y;x) ⊢ y=x! (Size of this proof $\stackrel{def}{=}$ 100%)

$$\dfrac{\dfrac{\dfrac{}{\vdash 0=0} \qquad \dfrac{\dfrac{}{\vdash 1=1}}{y\leftarrow 1 \vdash y=1}}{0=0 \mid y\leftarrow 1,\ S' \mid \vdash y=1}}{f(y;0) \vdash y=1 \quad \vdash \quad y=1\rightarrow y=0!} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{f(y;a) \vdash y=a!} \qquad \dfrac{\dfrac{}{\vdash a=a'-1} \qquad \dfrac{}{a'y=a'!}\ (dis)}{y=a!}}{f(y;a'-1) \vdash y=a! \qquad y\leftarrow a'y \vdash y'a'!}}{\vdash a'\neq 0 \qquad f(y;a'-1);y\leftarrow a'y \vdash y=a'!}}{a'=0 \mid y\leftarrow 1,\ f(y;a'-1);y\leftarrow a'y \mid \vdash y=a'!}}{f(y;a') \vdash y=a'!}$$

*) *)

$$\dfrac{(x)(f(y;x) \vdash y=x!)}{f(y;x) \vdash y=x!}$$

Remark. The following rule is a derived rule.

$$\dfrac{F(t)}{x\leftarrow t \quad \vdash \quad F(x)}$$

*) The above rule is used.

Discussions

The language of the assertion formulas can be generalized in the following manner. This definition is much simpler than the original both for the formation of formulas and the truth valuation. However one usually doesn't need this generalized assertion formula in proving the correctness of programs.

Assertion formulas, or af's, in the generalized sense are defined inductively as follows.

a) An atomic formula over L(T) is an af.

b) If A is an af, then $\sim A$ is an af.

c) If A and B are af's, then $A \vee B$ is an af.

d) If A is an af, and x is a variable, then $(x)A$ is an af.

e) If A is a statement, and A is an af, then $A \vdash A$ is an af.

The truth of the assertion formulas, in the generalized sense, is defined inductively as follows.

1. If A is a formula over L(T), then A is said to be true for i if it is true for the assignment i in the sense of predicate calculus.
2. We let $A \vdash A$ be true for i if $\underline{A}i$ is defined and A is true for $\underline{A}i$; and let it be false otherwise, (etc.)

As shown as examples, the constructor while-do can be uniformly translated into recursive procedures, so that we can give simple inference rules for this constructor as the relatively sound rules of our formalism.

Cf. "Automatic program verification I" Stanford AIM 200.

"Admissibility of fixed-point induction in first-order logic of typed theories" Stanford AIM 168, Proc. Symp. Th. Prog. (USSR Academy of Sciences, in Russian), or Lecture Notes in Computer Science. (Springer Verlag, English edition).

ACKNOWLEDGMENTS

C. Hoare, D. Luckham, R. London, and N. Wirth have done much in the study of the correctness formulas and assertions. Hoare obtained a new induction principle for recursive procedures. The recursive procedures have been studied by J. de Bakker who based his study on the induction principle given by D. Scott. (After my talks at IRIA and at WG2.2 (Stuttgart, March 1974), Z.Manna communicated with me about an independent formalism by him and A.Pnueli on the same subject. During my stay at IRIA R.Burstall taught me his treatment of the similar problems.)